

Deckhouse Kubernetes Platform

Руководство пользователя

Версия v1.76

Дата генерации: 15.09.2026

Содержание

Содержание	2
Настройка аутентификации для приложений	10
Обзор	10
Интерфейс	10
Смена и сброс пароля локального пользователя	11
Смена пароля по требованию политики или администратора	11
Самостоятельный сброс пароля	11
Включение аутентификации в веб-приложении	11
Аутентификация через прокси (без поддержки OIDC)	11
Аутентификация для приложений с поддержкой OIDC	13
Дополнительные ресурсы	14
Настройка вертикального автомасштабирования (VPA) для приложений	14
Как работает вертикальное масштабирование (VPA)	14
Работа VPA с лимитами	14
Мониторинг VPA в Grafana	16
Примеры настройки VPA	17
Классы приоритета	17
Использование класса приоритета в Deployment	17
Пошаговая демонстрация вытеснения	18
Разделение окружений по приоритетам	20
Защита Stateful-приложений	20
Демонстрация защиты Stateful-приложения	21
Шаг 1. Создание защищённого StatefulSet с PDB	21
Шаг 2. Симуляция нехватки ресурсов	22
Шаг 3. Наблюдение за работой механизмов защиты	22
Как работают механизмы защиты в критической ситуации	23
Эксплуатация и диагностика	23
Под не запускается из-за нехватки ресурсов	23
Практическая диагностика невозможности вытеснения	24
Практическая проверка отсутствия подходящих подов	24
Практический пример «Лимит подов на узле»	25
Полезные команды для мониторинга приоритетов	27
Обработка входящего трафика	28
Особенности и назначение NLB	28
Особенности и назначение ALB	28
Использование Network Load Balancer (NLB)	29
Примеры настроек для объектов Service	29
Общий IP-адрес для нескольких сервисов	29
Принудительное назначение IP-адреса	29
Назначение IPAddressPool (режим BGP)	30
Использование Application Load Balancer (ALB)	30
Сравнение вариантов ALB	30
Ingress NGINX	30
Kubernetes Gateway API	30
Istio	31
Как понять, что доступно в кластере	31
Следующие шаги	31
Дополнительные ресурсы	31
Публикация приложений средствами Kubernetes Gateway API	32
Публикация приложений средствами Kubernetes Gateway API	32
Публикация приложения с ListenerSet и HTTPRoute	32
Проверка публикации	33
Работа с объектами GRPCRoute, TLSRoute, TCPRoute и UDPRoute	34
GRPCRoute для gRPC-трафика	34
Сквозная передача TLS с TLSRoute	35
TLS-терминация на шлюзе с TCPRoute	37
Дополнительные TCP- и UDP-порты	38
Перевод приложения на публикацию через другой Gateway	40
Привязка маршрута к ListenerSet в другом неймспейсе	40
Настройка параметров TLS через BackendTLSPolicy	41

Поддерживаемые аннотации HTTPRoute	43
Публикация приложения при включённом Istio-сайдкапе	44
WAF на HTTPRoute	45
Дополнительные ресурсы	47
Публикация приложений средствами Ingress NGINX Controller	47
Публикация приложений средствами Ingress NGINX Controller	47
Проверка	48
Балансировка gRPC	49
Дополнительные ресурсы	49
Публикация приложений средствами Istio	49
Публикация приложений средствами Istio	49
Публикация приложений с использованием Ingress NGINX	49
Публикация приложений с использованием ресурса Istio Ingress Gateway	50
Canary-развёртывание через VirtualService	51
Дополнительные ресурсы	52
Дополнительные сети для использования в прикладных подах	52
Создание сети проекта (пользовательской сети)	53
Подключение дополнительных сетей к подам	53
IPAM для дополнительных сетей	54
Особенности использования IPAM в кластере DKP	54
Выделение пула IP-адресов для сети проекта и включение IPAM	55
Назначение IP-адресов на сетевые интерфейсы подов, подключаемых к дополнительной сети	55
Автоматическая выдача IP-адресов	56
Ручное (явное) создание IPAddress с типом Auto	56
Ручное назначение статического IP-адреса на дополнительный интерфейс пода	57
Проверка назначения IP-адреса интерфейсу	58
Дополнительные ресурсы	59
Подключение физических сетевых интерфейсов к подам DPDK-приложений	59
Подключение физических сетевых интерфейсов к подам	59
Создание пода с устройством из Underlay-сети	59
Внутрикластерное взаимодействие	62
Стандартный балансировщик	62
Ограничения стандартного балансировщика Service	63
Пример сервиса для стандартного балансировщика	63
Пример стандартного балансировщика для работы с PostgreSQL-кластером	63
Создание StatefulSet для PostgreSQL	63
Расширенный балансировщик	64
Внутреннее устройство балансировщика	64
Настройка балансировщика	65
Пример конфигурации расширенных балансировщиков ServiceWithHealthchecks	65
Размещение двух независимых балансировщиков на одной виртуальной машине	66
Создание виртуальной машины	66
Манифесты расширенных балансировщиков для веб-сервиса и SMTP	67
Дополнительные ресурсы	68
Управление авторизацией и доступом к нагрузке с Istio	68
Авторизация	68
Ресурс AuthorizationPolicy	68
Дополнительные ресурсы	69
Настройка маршрутизации запросов с Istio	69
Ресурс VirtualService	69
Дополнительные ресурсы	69
Активация Istio для приложений	69
Дополнительные ресурсы	70
Управление балансировкой запросов между эндпоинтами сервиса с Istio	70
Ресурс DestinationRule	70
Дополнительные ресурсы	70
Настройка Locality failover с Istio	70
Дополнительные ресурсы	71
Настройка Retry для запросов с Istio	71
Ресурс VirtualService	72
Дополнительные ресурсы	72

Настройка ресурсов для istio-proxy сайдкаров	72
Поддерживаемые аннотации	72
Примеры конфигурации	73
Дополнительные ресурсы	73
Настройка Circuit Breaker	73
Пример настройки Circuit Breaker	74
Дополнительные ресурсы	74
Настройка Canary deployment	74
Примеры настроек Canary deployment средствами Ingress NGINX	75
Создание Deployment и Service для стабильной версии	75
Создание Deployment и Service для Canary-версии	75
Настройка Ingress для Canary deployment	76
Постепенное увеличение трафика на Canary-версию	77
Откат или завершение Canary deployment	77
Дополнительные аннотации для Canary deployment	77
Примеры настроек Canary deployment средствами Istio	78
Распределение по наличию cookie	78
Распределение по вероятности	79
Дополнительные ресурсы	79
Настройка сканирования контейнерных образов на уязвимости	79
Доступ к результатам сканирования	80
Просмотр отчёта сканирования своего приложения	80
Просмотр результатов CIS compliance-проверки	80
Просмотр результатов сканирования	80
Описание ресурсов с результатами сканирования	81
Безопасность на уровне ресурсов	81
VulnerabilityReport	81
ConfigAuditReport	83
ExposedSecretReport	84
SbomReport	85
Безопасность на уровне кластера	88
RbacAssessmentReport	88
ClusterComplianceReport	91
Дополнительные ресурсы	92
Настройка использования TLS-сертификатов	92
Работа с сертификатами	92
Получение информации о сертификатах	93
Автоматический заказ сертификата	93
Заказ wildcard-сертификата с DNS в Cloudflare	93
Заказ wildcard-сертификата с DNS в AWS Route53	94
Заказ wildcard-сертификата с DNS в Google	95
Заказ самоподписанного сертификата	96
Создание самоподписанного сертификата	96
Пример создания сертификата	97
Защита учётных данных	98
Поддержка аннотации tls-acme	99
Дополнительные ресурсы	100
Параметры безопасности подов и контейнеров	101
runAsUser	101
Расположение параметра	102
Доступные значения параметра	102
Пример конфигурации	102
runAsNonRoot	102
Расположение параметра	103
Доступные значения параметра	103
Пример конфигурации	103
runAsGroup	103
Расположение параметра	104
Доступные значения параметра	104
Пример конфигурации	104
readOnlyRootFilesystem	104
Расположение параметра	105

Доступные значения параметра	105
Пример конфигурации	105
fsGroup	105
Расположение параметра	106
Доступные значения параметра	106
Пример конфигурации	106
supplementalGroups	106
Расположение параметра	107
Доступные значения параметра	107
Пример конфигурации	107
fsGroupChangePolicy	108
Расположение параметра	108
Доступные значения параметра	109
Пример конфигурации	109
appArmorProfile	109
Расположение параметра	110
Доступные значения параметра	110
Пример конфигурации	110
seccompProfile	111
Расположение параметра	111
Доступные значения параметра	112
Пример конфигурации	112
seLinuxOptions	112
Расположение параметра	113
Доступные значения параметра	113
Пример конфигурации	113
capabilities	113
Расположение параметра	114
Доступные значения параметра	114
Примеры конфигурации	114
allowPrivilegeEscalation	115
Расположение параметра	115
Доступные значения параметра	115
Пример конфигурации	115
privileged	116
Расположение параметра	116
Доступные значения параметра	116
Пример конфигурации	116
procMount	117
Расположение параметра	117
Доступные значения параметра	117
Пример конфигурации	117
sysctls	117
Расположение параметра	118
Доступные значения параметра	118
Пример конфигурации	118
hostNetwork	119
Расположение параметра	119
Доступные значения параметра	119
Пример конфигурации	119
hostPID	120
Расположение параметра	120
Доступные значения параметра	120
Пример конфигурации	120
hostIPC	121
Расположение параметра	121
Доступные значения параметра	121
Пример конфигурации	121
hostPath	122
Расположение параметра	122
Доступные значения параметра	122
Пример конфигурации	122

automountServiceAccountToken	123
Расположение параметра	123
Доступные значения параметра	124
Пример конфигурации	124
Дополнительные ресурсы	124
Веб-интерфейс Deckhouse Kubernetes Platform	124
Доступ к веб-интерфейсу	124
Deckhouse	124
Подраздел «Обзор»	124
Подраздел «Обновления»	125
Подраздел «Модули»	126
Подраздел «Глобальные настройки»	126
Подраздел «Kubernetes»	127
Управление узлами	128
Подраздел «Группы узлов»	128
Подраздел «Классы машин»	134
Подраздел «Узлы»	135
Подраздел «Статические машины»	136
Мультиотенантность	138
Подраздел «Проекты»	138
Подраздел «Шаблоны проектов»	139
Подраздел «Пользовательские неймспейсы»	141
Доступ	143
Подраздел «Операции над пользователями»	143
Сеть	144
Подраздел «Ингресс-контроллеры»	144
Хранилище	145
Подраздел «Классы хранилищ»	145
Подраздел «Persistent volume»	145
Подраздел «Локальные тома»	146
Безопасность	148
Подраздел «Сканер CVE»	148
Мониторинг	148
Подраздел «Обзор»	148
Подраздел «Обработка метрик»	150
Подраздел «Отправка метрик»	150
Подраздел «Источники для Grafana»	151
Подраздел «Дашборды для Grafana»	152
Подраздел «Активные алерты»	153
Журналирование	153
Подраздел «Отправка логов»	153
Подраздел «Сбор логов»	155
Просмотр информации об API	156
Дополнительные ресурсы	157
Веб-интерфейс Grafana	157
Веб-интерфейс системы мониторинга	158
Главный экран	158
Работа с дашбордами	158
Фильтрация информации	159
Работа с данными	160
Описание дашбордов	163
Просмотр журналов событий безопасности	176
Дополнительные ресурсы	179
Веб-интерфейс управления алертами	179
Доступ к веб-интерфейсу	179
Веб-интерфейс генерации kubeconfig	181
Доступ к веб-интерфейсу	181
Веб-интерфейс загрузки CLI-утилит	183
Доступ к веб-интерфейсу	183
Дополнительные ресурсы	183
Веб-интерфейс управления секретами	184
Доступ к веб-интерфейсу	184

Работа с механизмами секретов	184
Просмотр информации о механизме секретов	184
Управление доступом к данным и функциям Stronghold	189
Работа с методами аутентификации	190
Работа с группами пользователей	190
Работа с сущностями и псевдонимами	191
Управление временными правами доступа к секретам и ресурсам (Leases)	192
Работа с политиками контроля доступа	193
Работа с дополнительными инструментами	194
Мониторинг состояния кластера Raft Stronghold	196
Мониторинг активности и оценка нагрузки на Stronghold	196
Запечатывание и распечатывание хранилища секретов	197
Работа со Stronghold CLI	197
Дополнительные ресурсы	198
Веб-интерфейс визуализации сетевого стека кластера	198
Доступ к веб-интерфейсу	198
Фильтрация данных для отображения	199
Работа со схемой сетевых потоков	200
Работа с таблицей сетевых потоков и событий	201
Настройка набора столбцов таблицы	201
Просмотр информации о записях таблицы	201
Использование мультитенантности	202
Ограничения	202
Создание проекта	203
Автоматическое создание проекта для неймспейса	203
Дополнительные ресурсы	204
Обзор	204
Как это работает	204
Managed PostgreSQL	204
Проверка доступных ресурсов	205
Проверка хранилища	205
Основной пример: создание Postgres	206
Выбор PostgresClass	207
Ограничения размещения	208
Настройка ресурсов	208
Изменение ресурсов существующего Postgres	208
Проверка ограничения CPU и памяти через PostgresClass	209
Выбор режима работы	210
Режим Cluster	210
Режим Standalone	210
Настройка топологии и режима репликации	211
Настройка топологии	211
Размещение без выбора зоны	211
Размещение в одной зоне	212
Настройка репликации	212
Проверка режима репликации	212
Режим Availability	213
Режим Consistency	214
Режим ConsistencyAndAvailability	215
Изменение режима репликации существующего кластера	215
Создание логической базы данных и пользователя	216
Пользователь PostgreSQL	216
Декларативное управление пользователями	217
Логическая база данных	218
Подключение к PostgreSQL	218
Подключение из кластера	219
Внешнее подключение к PostgreSQL	219
Публикация PostgreSQL для внешнего доступа	219
Подключение по IP-адресу	221
Подключение с проверкой TLS	222
Настройка параметров PostgreSQL	223
Изменение разрешённого параметра	223

Ограничения на переопределение параметров	224
Проверка значений параметров	224
Настройка TLS	225
Режим K8s	225
Мониторинг и алерты	226
Резервное копирование и восстановление	227
Создание снимка	227
Восстановление из PostgresSnapshot	229
Проверка состояния	230
Дополнительные ресурсы	231
Мониторинг приложений и инфраструктуры	231
Возможности по умолчанию	231
Дашборды	231
Алерты	231
Настройки мониторинга	232
Дополнительные ресурсы	232
Дашборды мониторинга	232
Виды дашбордов	232
Предустановленные дашборды	232
Ingress Nginx	233
Потребление ресурсов (Main)	233
Security	233
Пользовательские дашборды	233
С помощью модуля observability	233
Разграничение прав доступа	234
Конвертация дашбордов из GrafanaDashboardDefinition	235
С помощью GrafanaDashboardDefinition	236
Дополнительные ресурсы	237
Настройка мониторинга приложений	237
Настройка сбора метрик через лейблы и аннотации	238
Пример настройки сбора метрик с Service	239
Пример настройки сбора метрик с Deployment	239
Дополнительные аннотации для тонкой настройки	240
Настройка сбора метрик с помощью ресурса PodMonitor или ServiceMonitor	240
Добавление пользовательских лейблов в метрики	241
Настройка сбора метрик через scrape_configs с помощью ресурса ScrapeConfig	242
Настройка сбора метрик с помощью blackbox-exporter	243
Дополнительные ресурсы	244
Доставка приложений с помощью Argo CD	245
Предварительные условия	245
Проекты AppProject	245
Подготовка неймспейса	245
Развёртывание приложения	246
Создание приложения с помощью объекта Application	246
Создание приложения через веб-интерфейс Argo CD	247
Создание приложения с помощью CLI-утилиты argocd	247
Дополнительные ресурсы	248
Marketplace	248
Предварительные условия	248
Дополнительные ресурсы	248
Установка и управление приложениями	248
Просмотр доступных версий пакетов	248
Установка приложения	249
Ограничения на имена	249
Проверка статуса приложения	250
Условия (Conditions)	250
Summary	250
Несколько экземпляров	251
Обновление приложения	251
Удаление приложения	251
FAQ	252
Можно ли обновлять приложение автоматически?	252

Может ли Application зависеть от другого Application?	252
Можно ли установить одно приложение в разных неймспейсах?	252
Дополнительные ресурсы	252
Диагностика	252
Проверка наличия CRD Marketplace	252
Чтение summary приложения	252
Чтение отдельных условий	253
Просмотр логов контроллера DKP	253
Просмотр логов подов приложения	254
Частые условия и их значение	254
Дополнительные ресурсы	254
Настройка сборки логов приложений	254
Настройка сбора логов из приложения	254
Примеры	255
Создание source в пространстве имён и чтение логов всех подов в нём с направлением их в Loki	256
Чтение подов в указанном пространстве имён с определенным лейблом	256
Дополнительные ресурсы	256

Настройка аутентификации для приложений

Обзор

Аутентификация — это процесс проверки подлинности пользователя. Deckhouse Kubernetes Platform (DKP) поддерживает сквозную аутентификацию пользователей при доступе к интерфейсам платформы и ресурсам кластера. Этот механизм также может использоваться для включения аутентификации в собственных приложениях, развёрнутых в кластере.

DKP позволяет настраивать аутентификацию как на основе внутренней базы данных, так и с использованием внешних провайдеров — например, LDAP, GitLab или GitHub. Это даёт возможность использовать единую аутентификацию сразу 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.

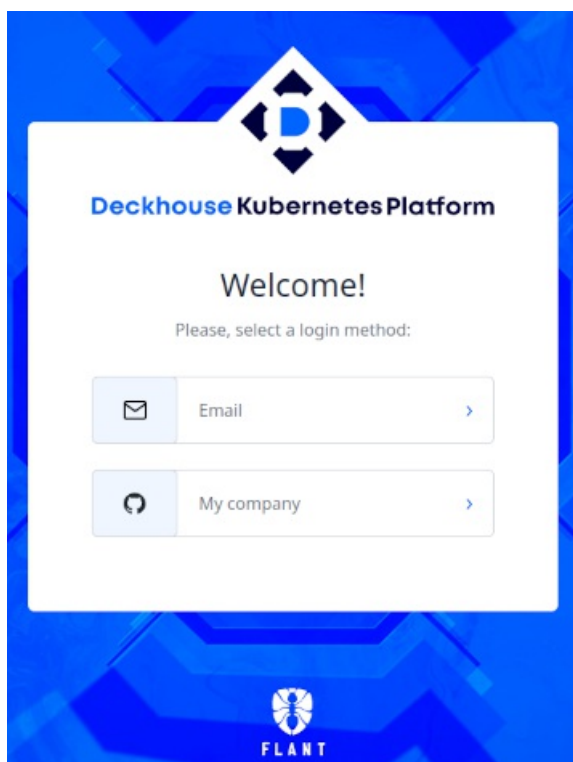


Чтобы использовать аутентификацию в DKP, необходима настройка.

Интерфейс

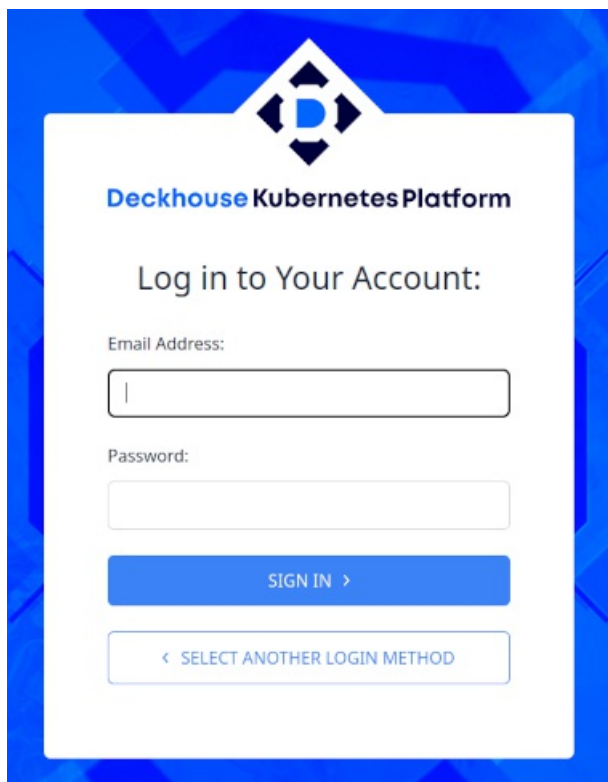
Интерфейс аутентификации открывается при первом обращении к ресурсу, для которого включена аутентификация — DKP перенаправляет пользователя на страницу аутентификации. Если пользователь уже прошёл аутентификацию (например во внешнем провайдере аутентификации), то DKP перенаправит запрос обратно к ресурсу, к которому первоначально обращался пользователь, обогатив запрос данными аутентификации. Если аутентификация не пройдена, пользователь увидит интерфейс аутентификации.

Пример интерфейса аутентификации в DKP:



Интерфейс аутентификации предлагает выбрать метод аутентификации, если их настроено несколько. Если настроен только один внешний провайдер аутентификации, то пользователь сразу попадет на страницу аутентификации этого провайдера. Если в DKP созданы локальные пользователи, то DKP предложит ввести логин и пароль.

Пример интерфейса аутентификации в DKP с вводом логина и пароля:



Смена и сброс пароля локального пользователя

Если в кластере настроена локальная аутентификация, пользователь может изменить или сбросить пароль самостоятельно.

Смена пароля по требованию политики или администратора

При входе DKP может перенаправить локального пользователя на форму смены пароля, если:

- сложность пароля не соответствует настроенной политике;
- истёк срок действия пароля согласно настроенной политике;
- администратор инициировал принудительную смену пароля.

В форме нужно указать текущий пароль, новый пароль и подтверждение. Новый пароль должен соответствовать парольной политике кластера.

Самостоятельный сброс пароля

Локальный пользователь может сбросить свой пароль в интерфейсе аутентификации. Операция выполняется через ресурс `UserOperation` с `initiatorType: self`.

Самостоятельный сброс доступен только для локальных учётных записей. Если вы входите через LDAP, GitHub или другой внешний провайдер, обратитесь к администратору соответствующей системы.

После сброса пароля активные сессии завершаются — потребуется повторный вход с новым паролем.

Включение аутентификации в веб-приложении

i

Для работы аутентификации в приложении, аутентификация должна быть настроена на уровне Deckhouse Kubernetes Platform.

В DKP можно включить аутентификацию для приложения двумя способами. В зависимости от того, умеет приложение обрабатывать запросы на аутентификацию (выступать OIDC-клиентом) или нет, в DKP можно включить аутентификацию для приложения двумя способами. Оба способа рассматриваются далее.

Аутентификация через прокси (без поддержки OIDC)

Аутентификация в приложении, которое не умеет самостоятельно обрабатывать запросы на

аутентификацию, реализуется с помощью специального прокси-сервера. Он обрабатывает запросы на аутентификацию, также выполняет функции авторизации, скрывая от приложения детали этих процессов.

Чтобы включить аутентификацию для приложения, развернутого в DKP выполните следующие шаги:

1. Создайте объект DexAuthenticator в пространстве имен приложения.

После появления объекта DexAuthenticator, в пространстве имен будет создан набор компонентов, необходимых для работы аутентификации:

- Deployment, содержащий контейнеры с прокси-сервером аутентификации/авторизации и хранилищем данных Redis;
- Сервис (Service), ведущий на прокси-сервер аутентификации/авторизации;
- Ingress-ресурс (Ingress), который принимает запросы по адресу `https://<applicationDomain>/dex-authenticator` и отправляет их в сторону сервиса;
- Секреты (Secret), необходимые для доступа к системе аутентификации DKP.

Пример 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-c5cjpg`.
  name: app-name
  # Namespace, в котором будет развернут Dex authenticator.
  namespace: app-ns
spec:
  # Домен вашего приложения. Запросы на него будут перенаправляться для
  # прохождения аутентификацию в Dex.
  applicationDomain: "app-name.kube.my-domain.com"
  # Отправлять ли `Authorization: Bearer` header приложению. Полезно в связке с
  # auth_request в NGINX.
  sendAuthorizationHeader: false
  # Имя Secret'a 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
```

Обратите внимание на следующие возможности при настройке аутентификации:

- В параметре `applicationDomain` DexAuthenticator указывается основной домен приложения. Дополнительные домены можно указать в параметре `additionalApplications.domain`;
 - Параметры `whitelistSourceRanges` и `additionalApplications.whitelistSourceRanges` позволяют открыть возможность аутентификации в приложении только для указанного списка IP-адресов;
- О настройке авторизации читайте в разделе Авторизация документации.

2. 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://<NAME>-dex-authenticator.<NS>.svc.<CLUSTER_DOMAIN>/dex-authenticator/auth`, где:
 - `<NAME>` — значение параметра `metadata.name` `DexAuthenticator`;
 - `<NS>` — значение параметра `metadata.namespace` `DexAuthenticator`;
 - `<CLUSTER_DOMAIN>` — домен кластера (параметр `clusterDomain` `ClusterConfiguration`, по умолчанию — `cluster.local`).

Пример (для `DexAuthenticator` с именем `app-name`, в пространстве имен `app-ns`):

```
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
```

Аутентификация для приложений с поддержкой OIDC

Приложения, которые умеют самостоятельно обрабатывать запросы на аутентификацию и выступать OIDC-клиентом, могут напрямую взаимодействовать с системой аутентификации DKP. В этом случае приложение самостоятельно перенаправляет пользователя на страницу входа и обрабатывает полученные OIDC-токены.

Чтобы включить аутентификацию для такого приложения, выполните следующие шаги:

1. Создайте объект `DexClient` в пространстве имен приложения.

После создания объекта `DexClient`, Deckhouse выполнит следующие действия:

- В системе аутентификации DKP будет зарегистрирован OIDC-клиент с идентификатором (`clientID`) вида: `dex-client-<NAME>@<NAMESPACE>` где `<NAME>` и `<NAMESPACE>` — это `metadata.name` и `metadata.namespace` объекта `DexClient`;
 - Будет автоматически сгенерирован `clientSecret` и сохранён в виде секрета `dex-client-<NAME>` в том же пространстве имён;
 - Пользователь сможет использовать этот `clientID` и `clientSecret` в своём приложении для настройки OIDC.
2. Укажите допустимые `redirect-URI`. Эти URI определяют, куда провайдер (Dex) может перенаправить пользователя после успешной аутентификации.
 3. Ограничьте доступ по группам, если необходимо. Используйте параметр `allowedGroups`, чтобы указать, какие группы пользователей имеют право входа в приложение через этот клиент.
 4. (Опционально). Укажите список доверенных клиентов (`trustedPeers`), если вы хотите разрешить делегирование аутентификации между приложениями.

Пример объекта `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

```

5. Получите `clientSecret`. Секрет будет создан автоматически:

```

apiVersion: v1
kind: Secret
metadata:
  name: dex-client-myname
  namespace: mynamespace
type: Opaque
data:
  clientSecret: c2VjcmV0

```

6. Настройте своё приложение как OIDC-клиент. Используйте `clientId`, `clientSecret`, `redirectURIs`, а также адрес Dex как провайдера. Адрес Dex (`https://dex.<publicDomainTemplate>`) можно получить с помощью команды:

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

Дополнительные ресурсы

- Глобальные кастомные ресурсы
- Кастомные ресурсы модуля `user-authn`

Настройка вертикального автомасштабирования (VPA) для приложений

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.

Работа VPA с лимитами

VPA управляет запросами ресурсов (requests) контейнера, но не управляет лимитами (limits), если это явно не указано в его политике.

VPA рассчитывает рекомендуемые значения на основе данных о потреблении ресурсов контейнером. Это поведение может повлиять на соотношение запросов ресурсов и лимитов:

- Если запросы ресурсов и лимиты равны, то VPA изменяет только запросы, оставляя лимиты неизменными.
- Если лимиты не указаны, VPA обновляет только запросы ресурсов.
- Если лимиты заданы, но не управляются VPA, соотношение запросов ресурсов и лимитов может измениться.

1. Пример 1. В кластере имеется:

◦ Объект VPA:

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"

```

◦ Под с ресурсами:

```

resources:
limits:
  cpu: 2
requests:
  cpu: 1

```

Если контейнер будет потреблять 1 CPU, VPA порекомендует 1,168 CPU. В данном случае, соотношение между запросами ресурсов и лимитами будет равно 100%. При пересоздании пода VPA изменит ресурсы на следующие:

```

resources:
limits:
  cpu: 2336m
requests:
  cpu: 1168m

```

2. Пример 2. В кластере имеется:

◦ VPA:

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: test2
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: test2
  updatePolicy:
    updateMode: "Initial"

```

◦ Под с ресурсами:

```

resources:
limits:
  cpu: 1
requests:
  cpu: 750m

```

В данном случае соотношение запросов ресурсов и лимитов будет 25%. Если VPA порекомендует 1,168 CPU, ресурсы контейнера будут изменены на:

```
resources:
  limits:
    cpu: 1557m
  requests:
    cpu: 1168m
```

Если не ограничивать ресурсы, VPA может выставить слишком большие ресурсы, что может привести к проблемам.

Для предотвращения этого можно:

- Использовать параметр `maxAllowed` в VPA:

```
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
```

- Настроить `Limit Range` в кластере:

```
apiVersion: autoscaling.k8s.io/v1
kind: LimitRange
metadata:
  name: my-app-vpa
spec:
  limits:
    - default:
        cpu: 2
        memory: 4Gi
      defaultRequest:
        cpu: 500m
        memory: 256Mi
      type: Container
```

Мониторинг VPA в Grafana

Для эффективного управления ресурсами с помощью Vertical Pod Autoscaler (VPA) рекомендуется использовать Grafana-дашборды. Они позволяют отслеживать текущий статус VPA, его настройки, а также процент подов, на которых он активирован.

Grafana предоставляет несколько уровней детализации информации о VPA. Основные дашборды:

- Main / Namespace — отображает информацию об использовании VPA на уровне пространства имен.
- Main / Namespace / Controller — детализирует данные VPA для конкретных контроллеров.
- Main / Namespace / Controller / Pod — самый детальный уровень, отображает информацию о каждом отдельном поде.

При мониторинге VPA в Grafana используются следующие ключевые столбцы:

- VPA type — показывает текущее значение `updatePolicy.updateMode`, которое определяет режим работы VPA для данного пода и отображается в дашбордах:
 - Main / Namespace
 - Main / Namespace / Controller
 - Main / Namespace / Controller / Pod
- VPA % (Процент подов с включенным VPA) — показывает процент подов в пространстве имен, на которых включен VPA. Позволяет быстро определить, сколько подов в кластере автоматически масштабируются с помощью VPA.

Примеры настройки VPA

1. Пример минимального ресурса VerticalPodAutoscaler:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: StatefulSet
    name: my-app
```

2. Пример полного ресурса 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
```

Классы приоритета

Класс приоритета задаёт, какие поды важнее при нехватке ресурсов на узле. Можно назначить класс в манифесте Deployment, проверить вытеснение и понять, почему под застрял в `Pending`.

Использование класса приоритета в Deployment

В Deckhouse Kubernetes Platform (DKP) уже есть набор классов приоритета. Далее приведён пример использования класса приоритета в шаблоне пода в Deployment.

Создайте файл `deployment-with-priority.yaml`, чтобы развернуть приложение с предустановленным в DKP классом `production-high` (значение `9000`, в разделе «Доступные классы приоритета»):

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      # Указание имени класса приоритета.
      priorityClassName: production-high
      containers:
      - name: app
        image: nginx:latest
        ports:
        - containerPort: 80

```

Примените манифест в кластере:

```
d8 k apply -f deployment-with-priority.yaml
```

Проверьте приоритеты созданных подов:

```

d8 k get pods -l app=my-app -o custom-
columns=NAME:.metadata.name,CLASS:.spec.priorityClassName,PRIORITY:.spec.priority

```

Пример вывода:

NAME	CLASS	PRIORITY
my-app-7d8f9b5c4f-2xq9t	production-high	9000
my-app-7d8f9b5c4f-4p7k2	production-high	9000
my-app-7d8f9b5c4f-9w5r8	production-high	9000



Параметр `priorityClassName` нельзя изменить у работающего пода. Это поле является неизменяемым. Чтобы сменить приоритет, обновите шаблон в Deployment и пересоздайте поды.

Пошаговая демонстрация вытеснения

В этом примере показано, как планировщик вытесняет под с более низким приоритетом, когда на узле не хватает ресурсов для нового пода с более высоким приоритетом. Сначала на worker-узле запускается под с классом `develop` (1000), который занимает большую часть свободных ресурсов. Затем создаётся под с классом `production-high` (9000) с такими же `requests` — места на узле уже нет, и планировщик вытесняет под с низким приоритетом в пользу нового.

В примерах вместо `worker-0` укажите имя своего worker-узла.



Пример рассчитан на один worker-узел с 4 CPU и 8 Gi памяти. Если узлов несколько, поды могут разместиться на разных узлах, и вытеснения не будет. Подберите `requests` так, чтобы под с низким приоритетом занял большую часть свободных ресурсов узла.

Проверьте список узлов в кластере:

```
d8 k get nodes
```

Пример вывода:

NAME	STATUS	ROLES	AGE	VERSION
master-0	Ready	control-plane,master	14d	v1.34.9
worker-0	Ready	worker	14d	v1.34.9

Проверьте ресурсы на worker-узле:

```
d8 k describe node worker-0 | grep -E "Capacity|Allocatable|Allocated" -A 5
```

Пример вывода:

```
Capacity:
  cpu:          4
  memory:       8174932Ki
Allocatable:
  cpu:          3800m
  memory:       7174932Ki
Allocated resources:
  cpu:          1200m (31%)
  memory:       2Gi (28%)
```

Создайте под с низким приоритетом, который займёт значительную часть свободных ресурсов, а затем под с высоким приоритетом, которому не хватит оставшегося места.

Создайте файл `low-priority-pod.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: low-priority-pod
spec:
  priorityClassName: develop
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        cpu: "2"
        memory: "4Gi"
```

Примените манифест:

```
d8 k apply -f low-priority-pod.yaml
```

Дождитесь, пока под перейдёт в статус `Running`:

```
d8 k get pods low-priority-pod
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
low-priority-pod	1/1	Running	0	10s

Создайте файл `high-priority-pod.yaml`, чтобы запустить под с высоким приоритетом, который запросит больше ресурсов, чем осталось свободно на узле:

```

apiVersion: v1
kind: Pod
metadata:
  name: high-priority-pod
spec:
  priorityClassName: production-high
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        cpu: "2"
        memory: "4Gi"

```

Примените манифест:

```
d8 k apply -f high-priority-pod.yaml
```

Проверьте статус подов:

```
d8 k get pods | grep priority
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
high-priority-pod	1/1	Running	0	5s

Под `high-priority-pod` находится в статусе `Running`. Под `low-priority-pod` может отсутствовать в выводе, так как планировщик вытеснил его за считанные секунды. В некоторых случаях можно увидеть `low-priority-pod` в статусе `Pending`.

Проверьте события вытеснения:

```
d8 k get events -A --field-selector reason=Preempted
```

Пример вывода:

NAMESPACE	LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
default	68s	Normal	Preempted	pod/low-priority-pod	Preempted by pod d9d25b95-4a7d-4214-8a30-8ce1fd616f67 on node worker-0

Разделение окружений по приоритетам

Неймспейс сам по себе не влияет на вытеснение: планировщик сравнивает только классы приоритета подов, независимо от того, в каких неймспейсах они находятся. Чтобы защитить `production`-нагрузку, в тестовых и `develop`-окружениях указывайте более низкий класс приоритета (`develop`, `staging`), а в `production` — более высокий (`production-low`, `production-medium`, `production-high`).

Защита Stateful-приложений

Stateful-приложения (с сохранением состояния, например базы данных, очереди сообщений) хранят данные в памяти или в постоянных томах (PVC). Их защита требует особого подхода, так как внезапное уничтожение пода без корректного завершения работы может повредить данные, а массовое вытеснение реплик приводит к потере доступности сервиса.

Подробнее о механизмах защиты от вытеснения смотрите в разделе «Защита пода от вытеснения». В подразделе «Демонстрация защиты Stateful-приложения» показан практический сценарий, в котором эти механизмы работают вместе.

Для защиты Stateful-приложений в данном примере используется комбинация трёх механизмов:

- Высокий `priorityClassName` — в примере у `StatefulSet` указан `production-medium` (6000). Для реальных критичных Stateful-приложений обычно берут `production-high` (9000) и

выше, см. раздел «Доступные классы приоритета». Так поды реже становятся кандидатами на вытеснение.

- PodDisruptionBudget (PDB) гарантирует минимальное количество работающих реплик (например, `minAvailable: 5`).
- Параметр `terminationGracePeriodSeconds` задаёт время на запись данных на диск и закрытие транзакций перед завершением пода (рекомендуется 30–60 секунд).

Демонстрация защиты Stateful-приложения

В этом примере разворачивается учебное Stateful-приложение: ему назначается класс `production-medium` (6000) вместе с PodDisruptionBudget и `terminationGracePeriodSeconds`, чтобы ограничить масштаб вытеснения и дать подам время корректно завершить работу. Затем создаётся под `emergency-task` с более высоким классом `production-high` (9000) и большим запросом памяти — чтобы смоделировать критическую нехватку ресурсов на узле и проверить, как срабатывают механизмы защиты.

Шаг 1. Создание защищённого StatefulSet с PDB

Создайте файл `stateful-protect.yaml`:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mock-stateful
spec:
  serviceName: mock-stateful
  replicas: 7
  selector:
    matchLabels:
      app: mock-stateful
  template:
    metadata:
      labels:
        app: mock-stateful
    spec:
      priorityClassName: production-medium
      terminationGracePeriodSeconds: 30
      containers:
        - name: app
          image: busybox
          command:
            - sh
            - -c
            - |
              trap 'echo ">>> НАЧАЛО: Сохранение данных на диск..."; sleep 10; echo ">>>
КОНЕЦ: Данные сохранены, выход."' TERM
              echo "Приложение запущено и работает..."
              while true; do sleep 1; done
          resources:
            requests:
              cpu: "100m"
              memory: "256Mi"
  ---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: mock-stateful-pdb
spec:
  minAvailable: 5
  selector:
    matchLabels:
      app: mock-stateful
```

Примените конфигурацию:

```
d8 k apply -f stateful-protect.yaml
```

Дождитесь статуса `Running` для всех 7 подов:

```
d8 k get pods -l app=mock-stateful -w
```

Шаг 2. Симуляция нехватки ресурсов

Создайте файл `emergency-task.yaml`, чтобы запустить под с предустановленным классом `production-high` (значение `9000`), который выше, чем у `Stateful`-приложения (`production-medium`, значение `6000`):

```
apiVersion: v1
kind: Pod
metadata:
  name: emergency-task
spec:
  priorityClassName: production-high
  containers:
  - name: task
    image: busybox
    command: ["sleep", "infinity"]
    resources:
      requests:
        cpu: "1"
        memory: "5Gi"
```

Примените манифест:

```
d8 k apply -f emergency-task.yaml
```

Шаг 3. Наблюдение за работой механизмов защиты

Проверьте статус подов:

```
d8 k get pods | grep -E 'mock-stateful|emergency-task'
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
emergency-task	0/1	Pending	0	5s
mock-stateful-0	1/1	Terminating	0	55s
mock-stateful-1	1/1	Terminating	0	53s
mock-stateful-2	1/1	Running	0	51s
mock-stateful-3	1/1	Running	0	49s
mock-stateful-4	1/1	Running	0	47s
mock-stateful-5	1/1	Running	0	45s
mock-stateful-6	1/1	Terminating	0	43s



Логи вытесняемых подов доступны только пока под находится в статусе `Terminating`. Чтобы увидеть процесс корректного завершения работы, необходимо успеть выполнить команду `d8 k logs` до полного удаления пода.

Проверьте логи завершающегося пода:

```
d8 k logs mock-stateful-0 --tail=20
```

Пример вывода:

```

Приложение запущено и работает...
>>> НАЧАЛО: Сохранение данных на диск...
>>> КОНЕЦ: Данные сохранены, выход.

```

Как работают механизмы защиты в критической ситуации

Поскольку приоритет созданного в «Демонстрации защиты Stateful-приложения» пода `emergency-task` (9000, класс `production-high`) выше, чем у Stateful-приложения (6000, класс `production-medium`), и других кандидатов на вытеснение нет, планировщик вынужден выбрать Stateful-приложение для вытеснения. При этом срабатывают защитные механизмы:

1. В данном случае `PodDisruptionBudget` пытается ограничить масштаб ущерба, но поскольку запрос экстремально велик, планировщик нарушает PDB, хоть и пытается минимизировать количество удаляемых подов.
2. Параметр `terminationGracePeriodSeconds` гарантирует сохранность данных, даже в случае полного вытеснения.

Эксплуатация и диагностика

В этом разделе приведены практические команды для проверки состояния подов и событий планировщика. Описание причин проблем и возможных действий на уровне кластера приведено в административном разделе «Эксплуатация и диагностика».

В «Пошаговой демонстрации вытеснения» под `high-priority-pod` переходит в `Running`. В следующих подразделах тот же под разбирается в двух случаях, когда он остаётся в `Pending`: не хватает CPU или памяти, и планировщик не находит поды для вытеснения.

Под не запускается из-за нехватки ресурсов

Если в кластере нет свободных ресурсов и вытеснение невозможно, под останется в состоянии `Pending`.

Проверьте статус пода:

```
d8 k get pod high-priority-pod
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
high-priority-pod	0/1	Pending	0	2m

Проверьте события пода:

```
d8 k describe pod high-priority-pod | grep -A10 "Events:"
```

Ищите в разделе `Events` сообщения вида:

```

Events:
  Type     Reason             Age   From                  Message
  ----     -
Warning   FailedScheduling   2m    default-scheduler    0/3 nodes are available: 2
Insufficient cpu, 1 Insufficient memory. preemption: 0/3 nodes are available: 3
Preemption is not helpful for scheduling.

```

Что означают эти сообщения:

- `Insufficient cpu` или `Insufficient memory` — на узлах не хватает запрошенных ресурсов, соответственно CPU и память.
- `Preemption is not helpful for scheduling` — вытеснение существующих подов не освободит достаточно ресурсов (например, все поды имеют равный или более высокий приоритет).

Проверьте доступные ресурсы на узлах:

```
d8 k describe nodes | grep -A 5 "Allocated resources"
```

Пример вывода:

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests    Limits
-----
cpu                3800m (95%) 4200m (105%)
memory            7Gi (87%)   8Gi (100%)
```

Возможные решения:

- Уменьшите `requests` в манифесте пода.
- Добавьте ресурсы в кластер (новые узлы или увеличение существующих).
- Удалите неиспользуемые поды.

Практическая диагностика невозможности вытеснения

Если вытеснение подов с низким приоритетом не происходит, хотя ресурсы свободны, проверьте события с причиной `FailedPreemption`:

```
d8 k get events -A --field-selector reason=FailedPreemption --sort-by='.metadata.creationTimestamp'
```

Пример вывода:

NAMESPACE	LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
default	30s	Warning	FailedPreemption	pod/high-priority-pod	no preemption victims found for pod

Сообщение `no preemption victims found for pod` обычно значит, что на узле нет подов с более низким приоритетом — их нельзя вытеснить ради нового пода. Разбор такого случая — в разделе «Практическая проверка отсутствия подходящих подов».

Под может не запуститься и по причинам, не связанным с классом приоритета: например, нет подходящих узлов из-за `taint` и `tolerations`. Тогда в событиях будет `FailedScheduling` с сообщениями вроде `untolerated taint(s)`, а не `FailedPreemption`.

Возможные действия на уровне кластера смотрите в разделе «Эксплуатация и диагностика».

Практическая проверка отсутствия подходящих подов

В этом примере используется состояние кластера после «Демонстрации защиты Stateful-приложения». Под `mock-stateful-0` имеет приоритет `6000`. На целевом узле остальные поды `mock-stateful` имеют такой же приоритет, `emergency-task` — более высокий (`9000`), а системные поды — ещё более высокий. Поэтому планировщик не может вытеснить ни один из них ради запуска `mock-stateful-0`: подходящих подов с более низким приоритетом на узле нет.

Если в событиях есть сообщение `No preemption victims found for incoming pod`, проверьте приоритеты подов на узле:

```
d8 k get pods --all-namespaces -o wide --field-selector spec.nodeName=worker-0 -o custom-columns=NAME:.metadata.name,NODE:.spec.nodeName,PRIORITY:.spec.priority --sort-by=.spec.priority
```

Пример вывода:

NAME	NODE	PRIORITY
mock-stateful-5	worker-0	6000
mock-stateful-2	worker-0	6000
mock-stateful-3	worker-0	6000
mock-stateful-4	worker-0	6000
emergency-task	worker-0	9000
multitenancy-manager-5968799d76-ktjgl	worker-0	
2000000000		
csi-node-s82x4	worker-0	
2000001000		
agent-wqzxq	worker-0	
2000001000		
early-oom-6tkzg	worker-0	
2000001000		
safe-agent-updater-ntrzq	worker-0	
2000001000		
kubernetes-api-proxy-worker-0	worker-0	
2000001000		
node-exporter-cddjm	worker-0	
2000001000		
oom-kills-exporter-pplfk	worker-0	
2000001000		

Проверьте точную причину, почему `mock-stateful-0` не смог вытеснить другие поды:

```
d8 k describe pod mock-stateful-0 | grep -A10 "Events:"
```

Пример сообщения из событий пода:

```
Events:
  Type            Reason              Age   From                  Message
  ----            -
  Warning         FailedScheduling    13s   default-scheduler    0/2 nodes are available:
preemption: 0/2 nodes are available: 1 No preemption victims found for incoming pod
```

Сообщение	Значение
0/2 nodes are available	В кластере есть 2 узла, но ни один не подходит для размещения пода.
1 No preemption victims found for incoming pod	На узле с нехваткой памяти нет подов с более низким приоритетом, которые можно было бы вытеснить.

Что делать на уровне кластера, смотрите в разделе «Эксплуатация и диагностика».

Практический пример «Лимит подов на узле»

Даже если освободились CPU и память, лимит на максимальное количество подов на узле может помешать запуску пода с высоким приоритетом.

Проверьте лимит подов на узле:

```
d8 k describe node worker-0 | grep pods -A2
```

Пример вывода:

```
Capacity:
  pods: 120
```

Проверьте текущее количество подов на узле:

```
d8 k get pods --all-namespaces -o wide | grep worker-0 | wc -l
```

Пример вывода:

```
64
```

Сейчас на узле 64 пода из 120. Свободное место ещё есть.

Создайте файл `pod-filler.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pod-filler
spec:
  replicas: 110
  selector:
    matchLabels:
      app: filler
  template:
    metadata:
      labels:
        app: filler
    spec:
      priorityClassName: develop
      containers:
      - name: filler
        image: busybox
        command: ["sleep", "infinity"]
        resources:
          requests:
            cpu: "1m"
            memory: "5Mi"
```

Примените манифест:

```
d8 k apply -f pod-filler.yaml
```

Дождитесь, пока Deployment заполнит узел:

```
d8 k get pods -l app=filler -o wide | grep worker-0 | wc -l
```

Создайте файл `high-priority-limit-pod.yaml`, чтобы запустить под с высоким приоритетом:

```
apiVersion: v1
kind: Pod
metadata:
  name: high-priority-limit-pod
spec:
  priorityClassName: production-high
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        cpu: "100m"
        memory: "256Mi"
```

Примените манифест:

```
d8 k apply -f high-priority-limit-pod.yaml
```

Проверьте статус пода с высоким приоритетом:

```
d8 k get pod high-priority-limit-pod
```

Пример вывода:

NAME	READY	STATUS	RESTARTS	AGE
high-priority-limit-pod	0/1	Pending	0	11s

Посмотрите причину в событиях:

```
d8 k describe pod high-priority-limit-pod | grep -A10 "Events:"
```

Пример вывода:

```
Events:
  Type            Reason             Age   From              Message
  ----            -
  Warning         FailedScheduling   11s   default-scheduler  0/2 nodes are available: 1 Too many pods, 1 node(s) had intolerated taint(s).
```

Проверьте, что вытеснения не произошло:

```
d8 k get events -A --field-selector reason=Preempted
```

Пример вывода:

```
No resources found
```

Суть проблемы: лимит подов на узле уже достигнут, поэтому даже под с высоким приоритетом не может запуститься. Вытеснение существующих подов не помогает, так как количество подов останется прежним (вытесненный под будет заменён новым).

Полезные команды для мониторинга приоритетов

Подсчёт количества подов по классам приоритета:

```
d8 k get pods -A -o jsonpath='{range .items[*]}{.spec.priorityClassName}{"\n"}{end}' | sort | uniq -c | sort -rn
```

Пример вывода:

```
34 cluster-medium
30 cluster-low
26 system-cluster-critical
18 system-node-critical
6 production-high
```

Просмотр событий вытеснения во всех неймспейсах:

```
d8 k get events -A --field-selector reason=Preempted -O custom-columns=NAMESPACE:.metadata.namespace,POD:.involvedObject.name,MESSAGE:.message
```

Пример вывода:

NAMESPACE	POD	MESSAGE
d8-chrony	chrony-master-9wzbl	Preempted by
pod ac651aed-... on node master-0		
d8-console	backend-58f9989c9d-4svjw	Preempted by
pod ac651aed-... on node master-0		
d8-monitoring	prometheus-main-0	Preempted by
pod 91f6e071-... on node worker-0		
default	log-collector-dlxpv	Preempted by
pod 91f6e071-... on node worker-0		

i События хранятся ограниченное время (обычно около часа). Если вытеснения давно не было, эти команды могут ничего не вернуть — повторите демонстрацию вытеснения и выполните команды сразу после неё.

Поды, которые вытеснялись чаще всего:

```
d8 k get events -A --field-selector reason=Preempted -o jsonpath='{range .items[*]}{.involvedObject.name}{"\n"}{end}' | sort | uniq -c | sort -rn | head -10
```

Пример вывода:

```
2 prometheus-main-0
2 memcached-0
1 user-api-77494dc777-jzp7p
1 upmeter-dex-authenticator-7f54c8dfb4-wwv22
1 upmeter-dex-authenticator-7f54c8dfb4-wqfn5
1 upmeter-dex-authenticator-7f54c8dfb4-h6784
1 upmeter-dex-authenticator-7f54c8dfb4-bxsvk
1 upmeter-dex-authenticator-7f54c8dfb4-28fgt
1 upmeter-agent-4chrw
1 status-dex-authenticator-786c6cc554-mfsdw
```

Обработка входящего трафика

Балансировщики Network Load Balancer (NLB) и Application Load Balancer (ALB) используются для предоставления внешнего доступа к приложениям, развернутым в кластере под управлением Deckhouse Kubernetes Platform.

Особенности и назначение NLB

NLB работает на транспортном уровне. Он балансирует TCP и UDP трафик на уровне IP и портов.

Основные преимущества:

- высокая производительность;
- минимальные задержки при передаче трафика;
- простая конфигурация.

NLB подходит для приложений, которые используют TCP/UDP-протоколы, например, для баз данных.

Особенности и назначение ALB

ALB работает на прикладном уровне. Он анализирует содержимое входящих запросов (например, HTTP-заголовки, пути URL, cookies) и может выполнять маршрутизацию на их основе.

Преимущества ALB:

- поддержка HTTP(S)-протоколов и gRPC;
- гибкая маршрутизация (path-based, host-based);
- возможность терминации SSL/TLS;
- интеграция с механизмами аутентификации и авторизации.

ALB подходит для веб-приложений, API и других сервисов, где важны интеллектуальная

маршрутизация и работа с HTTP-запросами.

Использование Network Load Balancer (NLB)

NLB обеспечивается за счет использования сервисов с типом LoadBalancer.

Примеры настроек для объектов Service

Общий IP-адрес для нескольких сервисов

Для того, чтобы сервисы использовали одни и те же IP-адреса, добавьте к ним аннотацию `network.deckhouse.io/load-balancer-shared-ip-key`:

```
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
```

Принудительное назначение IP-адреса

Чтобы принудительно задать адрес для сервиса, добавьте 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

```

Назначение IPAddressPool (режим BGP)

В режиме BGP LoadBalancer получение IP-адреса возможно из определённого пула адресов через аннотацию `metallb.universe.tf/address-pool`. Для режима L2 LoadBalancer необходимо использовать настройки `MetalLoadBalancerClass`.

Пример:

```

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

```

Использование Application Load Balancer (ALB)

Публикация приложений и балансировка трафика на прикладном уровне в Deckhouse Kubernetes Platform (DKP) могут выполняться средствами:

- Ingress NGINX Controller (модуль `ingress-nginx`);
- Kubernetes Gateway API (модуль `alb`);
- Istio (модуль `istio`).

Сравнение вариантов ALB

Ниже описаны особенности каждого варианта ALB и типичные сценарии их использования.

Ingress NGINX

ALB средствами Ingress NGINX Controller построена на базе веб-сервера `nginx` и реализуется модулем `ingress-nginx`. Этот вариант подходит для:

- базовой маршрутизации трафика на основе доменов или URL;
- использования SSL/TLS для защиты трафика.

Kubernetes Gateway API

ALB средствами Kubernetes Gateway API реализуется модулем `alb`. Шлюзы работают на Envoy Proxy. Приём и маршрутизация описываются стандартными объектами API (Gateway, ListenerSet, HTTPRoute и при необходимости GRPCRoute, TLSRoute, TCPRoute, UDPRoute, BackendTLSPolicy). Контроллер разворачивает инфраструктуру входа и проверяет конфигурацию, чтобы не допускать конфликтующих обработчиков.

Модель Gateway API разделяет ответственность между администратором кластера (ClusterALBInstance), администратором неймспейса (ALBInstance и ListenerSet — hostname, TLS, порты) и разработчиками приложения (HTTPRoute и другие ресурсы маршрутизации).

Используйте этот вариант для:

- публикации приложений в модели Gateway API вместо классического Ingress;
- общекластерной точки входа или отдельного шлюза для приложения или команды в своём неймспейсе;
- маршрутизации HTTP/HTTPS, gRPC, TCP, UDP, а также термации или сквозной передачи TLS;
- WAF на уровне маршрута или Istio-сайдкара на прокси шлюза;
- GeoIP и OpenTelemetry на шлюзе (настраивает администратор — «Использование GeoIP и GeoLite2» и «Настройка трассировки OpenTelemetry»);
- параметров маршрута, которых нет в спецификации, через аннотации HTTPRoute.

Сравнение с `ingress-nginx` и пояснения по терминологии — в разделе «Сравнение возможностей модулей `ingress-nginx` и `alb`».

Istio

ALB на основе модуля `istio` поддерживает управление трафиком в service mesh. Используйте этот вариант для:

- маршрутизации для canary deployment и аналогичных сценариев;
- распределения трафика между версиями приложения и микросервисами;
- mTLS для шифрования трафика между подами;
- трассировки запросов.

Как понять, что доступно в кластере

Перед публикацией приложения проверьте, какие механизмы ALB включены и настроены:

1. Убедитесь, что нужный модуль включён — в колонке `STATE` должно быть значение `Enabled`:

```
d8 k get moduleconfig ingress-nginx alb istio
```

2. Для Ingress NGINX — посмотрите ресурсы IngressNginxController и имя IngressClass:

```
d8 k get ingressnginxcontrollers
d8 k get ingressclass
```

3. Для Gateway API — убедитесь, что есть готовый ClusterALBInstance или ALBInstance в состоянии `Ready`, затем найдите управляемый Gateway и ListenerSet:

```
d8 k get clusteralbinstances,albinstances --all-namespaces
d8 k get gateway,listenerset --all-namespaces
```

4. Для Istio — проверьте IngressIstioController и класс ingress gateway, который сообщит администратор кластера:

```
d8 k get ingressistiocontrollers
```

Запросите у администратора кластера `ingressClass`, имя и неймспейс Gateway или класс Istio ingress для манифестов приложения.

Следующие шаги

После настройки инфраструктуры администратором можно выполнить публикацию приложения:

- средствами Kubernetes Gateway API (модуль `alb`);
- средствами Ingress NGINX Controller (модуль `ingress-nginx`);
- средствами Istio (модуль `istio`).

Дополнительные ресурсы

- Публикация приложений средствами Kubernetes Gateway API
- Публикация приложений средствами Ingress NGINX Controller

- Публикация приложений средствами Istio
- Балансировка входящего трафика
- Документация модуля `alb`
- Документация модуля `ingress-nginx`
- Документация модуля `istio`

Публикация приложений средствами Kubernetes Gateway API

Публикация приложений средствами Kubernetes Gateway API

Публикация приложения возможна через общекластерный шлюз (`ClusterALBInstance`, создаёт администратор кластера) или через отдельный шлюз в неймспейсе приложения (`ALBInstance`).

Создание управляемого Gateway (`ClusterALBInstance` или `ALBInstance`, инлеты, включение модуля) — задача администратора. Настройка инфраструктуры описана в разделе «Включение модуля и создание Gateway».

Этот сценарий предполагает, что объект `ClusterALBInstance` или `ALBInstance` уже создан и перешёл в состояние `Ready`. Запросите у администратора имя и неймспейс управляемого Gateway или получите имя Gateway из поля `status.gateway` инстанса:

```
d8 k get clusteralbinstance <CLUSTER_ALB_INSTANCE_NAME> \
-o jsonpath='{.status.gateway}{"\n"}'
d8 k -n <NAMESPACE> get albinstance <ALB_INSTANCE_NAME> \
-o jsonpath='{.status.gateway}{"\n"}'
```

Для `ClusterALBInstance` управляемый Gateway обычно находится в неймспейсе `d8-alb`. Для `ALBInstance` — в том же неймспейсе, что и объект `ALBInstance`. Описание полей статуса — в `status`.

Администратор неймспейса создаёт `ListenerSet`, привязанный к этому Gateway (`spec.parentRef`). Разработчики приложения создают объекты `HTTPRoute`, привязанные к `ListenerSet`.

Слушатели `d8-http` и `d8-https` предназначены для служебных задач — например, для проверки доступности шлюза или запросов HTTP-01 от `cert-manager`. Не привязывайте к ним маршруты приложений — для публикации приложений используйте `ListenerSet`.

Публикация приложения с `ListenerSet` и `HTTPRoute`

Пример `ListenerSet` и `HTTPRoute` для публикации приложения через управляемый Gateway:

```
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: app-listeners
  namespace: prod
spec:
  parentRef:
    name: public-gw # Имя объекта Gateway из status ClusterALBInstance,
    # предоставленное администратором.
    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 # Secret с TLS-сертификатом (выпустите через cert-manager
или создайте вручную).
        namespace: prod
    ---
# Маршрут для HTTP-трафика
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-http-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-http
      port: 80
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc # Наименование сервиса приложения.
          port: 8080
    ---
# Маршрут для HTTPS-трафика
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-https-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc # Наименование сервиса приложения.
          port: 8080

```

Проверка публикации

После применения ListenerSet и HTTPRoute проверьте статус объектов:

```

d8 k -n <NAMESPACE> get listenerset
d8 k -n <NAMESPACE> describe listenerset <LISTENERSET_NAME>
d8 k -n <NAMESPACE> get httproute
d8 k -n <NAMESPACE> describe httproute <HTTPROUTE_NAME>

```

Проверьте следующее:

- В статусе ListenerSet слушатели в состоянии Programmed или Accepted.
- У HTTPRoute в parentRefs указан нужный ListenerSet и условия Accepted выполнены.
- Адрес точки входа и DNS для hostname согласованы с администратором (для инлета LoadBalancer адрес обычно берётся из Service в неймспейсе d8-alb).

Проверьте доступность приложения с клиента, подставив адрес точки входа и hostname (успешный ответ — код 200 или другой ожидаемый код приложения):

```
curl -vk \
  --resolve app.example.com:443:<ENTRY_POINT_ADDRESS> \
  https://app.example.com/
```

Если маршрут не принимается, проверьте имя Gateway из `status` инстанса, неймспейс `ListenerSet`, порт и `sectionName` в `parentRefs`. Конфликты hostname и порта с другими объектами `ListenerSet` видны в выводе команды `describe listenerset` из предыдущего шага — в статусе конфликтующего слушателя появляется условие с описанием причины.

Работа с объектами `GRPCRoute`, `TLSRoute`, `TCPRoute` и `UDPRoute`

Помимо `HTTPRoute`, для публикации приложений можно использовать `GRPCRoute` (gRPC-трафик), `TLSRoute` (сквозная передача TLS) и `TCPRoute`/`UDPRoute` (произвольный TCP и UDP-трафик).

`GRPCRoute` для gRPC-трафика

Объект `GRPCRoute` предназначен для маршрутизации gRPC-трафика. Для него создаётся объект `ListenerSet` со слушателем HTTPS, а затем добавляется объект `GRPCRoute`:

```
apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: grpc-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # Имя объекта Gateway из поля status ALBInstance.
    namespace: prod
  listeners:
    - name: grpc-https
      port: 443 # В ListenerSet для HTTPS указывайте порт 443. Это не параметр
      hostPort.httpsPort инлета HostPort на узле.
      protocol: HTTPS
      hostname: grpc.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: grpc-tls # Наименование объекта Secret, содержащего необходимый
            TLS-сертификат.
            namespace: prod
  ---
apiVersion: gateway.networking.k8s.io/v1
kind: GRPCRoute
metadata:
  name: grpc-route
  namespace: prod
spec:
  parentRefs:
    - name: grpc-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: grpc-https
      port: 443
  hostnames:
    - grpc.example.com
  rules:
    - backendRefs:
        - name: grpc-svc # Наименование сервиса приложения.
          port: 9090
```

Сквозная передача TLS с TLSRoute

Для сквозной передачи TLS (passthrough), когда расшифровка трафика должна выполняться на стороне приложения, можно использовать либо слушателя TLS, либо слушателя HTTPS.

Дополнительные порты задаются в `spec.inlet.additionalPorts` объекта, который владеет шлюзом:

- для общекластерного Gateway — в `ClusterALBInstance` (это делает администратор кластера);
- для шлюза в неймспейсе — в `ALBInstance` (администратор неймспейса или вы, если есть права на объект).

Пример для `ClusterALBInstance` — в разделе «Открытие дополнительного TCP/UDP-порта на общекластерном Gateway». Ниже показан пример для `ALBInstance`.

- Слушатель TLS
- Слушатель HTTPS

Поскольку в этом примере слушатель TLS использует дополнительный порт, попросите администратора добавить `additionalPorts` в объект, владеющий шлюзом (`ClusterALBInstance` или `ALBInstance`), или добавьте параметр сами, если у вас есть права:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    additionalPorts:
      - port: 8443    # Дополнительный TCP-порт для TLS-трафика.
        protocol: TCP
```

Далее настройте объекты `ListenerSet` и `TLSRoute`:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: tls-pass-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # Имя объекта Gateway из поля status ALBInstance.
    namespace: prod
  listeners:
    - name: tls-pass
      port: 8443 # В данном примере для TLS-трафика переиспользуется порт 8443.
      protocol: TLS
      hostname: pass.example.com
      tls:
        mode: Passthrough # Режим TLS – сквозной.
---
apiVersion: gateway.networking.k8s.io/v1
kind: TLSRoute
metadata:
  name: tls-pass-route
  namespace: prod
spec:
  parentRefs:
    - name: tls-pass-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: tls-pass
      port: 8443 # В данном примере для TLS-трафика переиспользуется порт 8443.
  hostnames:
    - pass.example.com
  rules:
    - backendRefs:
        - name: tls-pass-svc # Наименование сервиса приложения.
          port: 8443

```

Вариант со слушателем HTTPS удобен, когда нужно использовать стандартный обработчик на порту 443. Дополнительный порт для сквозной передачи TLS открывать не требуется.


```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: https-pass-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # Имя объекта Gateway из поля status ALBInstance.
    namespace: prod
  listeners:
    - name: https-pass
      port: 443 # В данном примере для TLS-трафика переиспользуется порт 443.
      protocol: HTTPS
      hostname: pass.example.com
      tls:
        mode: Passthrough # Режим TLS – сквозной.
---
apiVersion: gateway.networking.k8s.io/v1
kind: TLSRoute
metadata:
  name: https-pass-route
  namespace: prod
spec:
  parentRefs:
    - name: https-pass-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: https-pass
      port: 443 # В данном примере для TLS-трафика переиспользуется порт 443.
  hostnames:
    - pass.example.com
  rules:
    - backendRefs:
        - name: tls-pass-svc # Наименование сервиса приложения.
          port: 8443

```

TLS-терминация на шлюзе с TCPRoute

Если TLS нужно терминировать на шлюзе, а затем передать трафик дальше как TCP-поток к бэкенду (например, когда приложение принимает уже расшифрованный TCP, а не HTTPS), создайте объект `ListenerSet` со слушателем TLS и режимом `Terminate`, после чего подключите объект `TCPRoute`:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: tls-term-listeners
  namespace: prod
spec:
  parentRef:
    name: app-gw # Имя объекта Gateway из поля status ALBInstance.
    namespace: prod
  listeners:
    - name: tls-term
      port: 443 # В данном примере для TLS-трафика переиспользуется порт 443.
      protocol: TLS
      hostname: term.example.com
      tls:
        mode: Terminate # Режим TLS – терминация.
        certificateRefs:
          - name: term-tls # Наименование объекта Secret, содержащего необходимый TLS-
            сертификат.
            namespace: prod
    ---
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: tls-term-route
  namespace: prod
spec:
  parentRefs:
    - name: tls-term-listeners # Имя ListenerSet.
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: tls-term
      port: 443
  rules:
    - backendRefs:
        - name: tls-svc # Наименование сервиса приложения.
          port: 8080

```

Дополнительные TCP- и UDP-порты

Для портов TCP и UDP из `additionalPorts` маршрут привязывается напрямую к слушателю управляемого Gateway, без отдельного ListenerSet. Иначе контроллер отклонит конфигурацию из-за пересечения обработчиков.

Дополнительные порты задаются в объекте, который владеет шлюзом: в ClusterALBInstance для общекластерного Gateway или в ALBInstance для шлюза в неймспейсе. Для ClusterALBInstance используйте пример в разделе «Открытие дополнительного TCP/UDP-порта на общекластерном Gateway».

- TCP
- UDP

Для публикации TCP-сервиса попросите администратора открыть дополнительный TCP-порт в объекте, владеющем шлюзом (ClusterALBInstance или ALBInstance), или откройте его сами, если у вас есть права:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    additionalPorts:
      - port: 9000
        protocol: TCP

```

Контроллер модуля `alb` автоматически создаёт на управляемом объекте Gateway TCP-слушатель на основе `spec.inlet.additionalPorts`. Привяжите объект TCPRoute к этому слушателю:

```

apiVersion: gateway.networking.k8s.io/v1alpha2
kind: TCPRoute
metadata:
  name: tcp-route
  namespace: prod
spec:
  parentRefs:
    - name: app-gw # Имя объекта Gateway из поля status ALBInstance.
      namespace: prod
      kind: Gateway
      group: gateway.networking.k8s.io
      sectionName: tcp-port-9000
      port: 9000
  rules:
    - backendRefs:
        - name: tcp-svc # Наименование сервиса приложения.
          port: 9000

```

Для публикации UDP-сервиса попросите администратора открыть дополнительный UDP-порт в объекте, владеющем шлюзом (ClusterALBInstance или ALBInstance), или откройте его сами, если у вас есть права:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: ALBInstance
metadata:
  name: app-gw
  namespace: prod
spec:
  gatewayName: app-gw
  inlet:
    type: LoadBalancer
    additionalPorts:
      - port: 5353
        protocol: UDP

```

Контроллер модуля `alb` автоматически создаёт на управляемом объекте Gateway UDP-слушатель на основе `spec.inlet.additionalPorts`. Привяжите объект UDPRoute к этому слушателю:

```

apiVersion: gateway.networking.k8s.io/v1
kind: UDPRoute
metadata:
  name: udp-route
  namespace: prod
spec:
  parentRefs:
    - name: app-gw # Имя объекта Gateway из поля status ALBInstance.
      namespace: prod
      kind: Gateway
      group: gateway.networking.k8s.io
      sectionName: udp-port-5353
      port: 5353
  rules:
    - backendRefs:
        - name: udp-svc # Наименование сервиса приложения.
          port: 5353

```

Перевод приложения на публикацию через другой Gateway

Если приложение нужно опубликовать через другой объект Gateway, выполните следующие шаги:

1. Получите у администратора кластера новый управляемый Gateway (новый ClusterALBInstance или ALBInstance), чтобы контроллер создал новый объект Gateway. Создание управляемых Gateway описано в разделе «Включение модуля и создание Gateway».
2. Создайте объект ListenerSet с теми же именами хостов, портами и TLS-настройками. В `spec.parentRef` укажите новый объект Gateway.
3. В существующий объект HTTPRoute, в `parentRefs` добавьте ещё один объект, который указывает на новый объект ListenerSet.
4. Проверьте доступность приложения через новый шлюз.
5. После проверки удалите из `parentRefs` объекта HTTPRoute ссылку на неактуальные ListenerSet.

Привязка маршрута к ListenerSet в другом неймспейсе

Gateway API по умолчанию запрещает маршрутам ссылаться на объекты в чужих неймспейсах — такую привязку нужно разрешить явно. Если HTTPRoute и целевой ListenerSet находятся в разных неймспейсах, добавьте ReferenceGrant рядом с ListenerSet.

В примере ниже — общий ListenerSet в `shared-gw`, прикладной HTTPRoute в `prod` и ReferenceGrant в `shared-gw`, разрешающий такую привязку:

```

apiVersion: gateway.networking.k8s.io/v1
kind: ListenerSet
metadata:
  name: shared-listeners
  namespace: shared-gw
spec:
  parentRef:
    name: public-gw
    namespace: d8-alb
  listeners:
    - name: app-https
      port: 443
      protocol: HTTPS
      hostname: app.example.com
      tls:
        mode: Terminate
        certificateRefs:
          - name: app-tls
            namespace: shared-gw
---
apiVersion: gateway.networking.k8s.io/v1
kind: ReferenceGrant
metadata:
  name: allow-prod-httproute-to-shared-listeners
  namespace: shared-gw
spec:
  from:
    - group: gateway.networking.k8s.io
      kind: HTTPRoute
      namespace: prod
  to:
    - group: gateway.networking.k8s.io
      kind: ListenerSet
      name: shared-listeners
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: prod
spec:
  parentRefs:
    - name: shared-listeners
      namespace: shared-gw
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc
          port: 8080

```

Настройка параметров TLS через BackendTLSPolicy

Если трафик от шлюза к бэкенду должен идти по TLS, создайте BackendTLSPolicy в неймспейсе бэкенд-объекта Service. В примере ниже — HTTPRoute, бэкенд-объект Service с именованным портом, ConfigMap с CA bundle и BackendTLSPolicy с TLS-валидацией для этого бэкенда:

```

apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app-route
  namespace: prod
spec:
  parentRefs:
    - name: app-listeners
      namespace: prod
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - app.example.com
  rules:
    - backendRefs:
        - name: app-svc
          port: 8443
---
apiVersion: v1
kind: Service
metadata:
  name: app-svc
  namespace: prod
spec:
  selector:
    app: app
  ports:
    - name: https
      port: 8443
      targetPort: 8443
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-backend-ca
  namespace: prod
data:
  ca.crt: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
---
apiVersion: gateway.networking.k8s.io/v1
kind: BackendTLSPolicy
metadata:
  name: app-svc-tls
  namespace: prod
spec:
  targetRefs:
    - group: ""
      kind: Service
      name: app-svc
      sectionName: https
  validation:
    hostname: app.internal.example.com
    caCertificateRefs:
      - group: ""
        kind: ConfigMap
        name: app-backend-ca

```

GeoIP и трассировку OpenTelemetry настраивает администратор на ClusterALBInstance или ALBInstance, по инструкциям «Использование GeoIP и GeoLite2» и «Настройка трассировки OpenTelemetry».

Поддерживаемые аннотации HTTPRoute

Так как текущая спецификация Gateway API пока не покрывает все возможности, нужные для работы кластера Deckhouse Kubernetes Platform (DKP), модуль предоставляет аннотации HTTPRoute для недостающих параметров. Контроллер читает эти ключи из `HTTPRoute.metadata.annotations`.

Аннотация	Описание
<code>alb.network.deckhouse.io/tls-disable-protocol</code>	Отключает протокол обработчика для маршрута с указанным именем хоста (например, значение <code>http2</code>). Может быть необходимо в редких случаях, когда используется общий сертификат с несколькими DNS-именами в сочетании с перенаправлением запросов
<code>alb.network.deckhouse.io/whitelist-source-range</code>	Список подсетей в формате CIDR через запятую. Задаёт фильтр по IP на уровне маршрута и переопределяет глобальный <code>whitelist</code> (например, <code>10.1.1.10/32</code> , <code>10.2.2.2/32</code>)
<code>alb.network.deckhouse.io/response-headers-to-add</code>	JSON-объект дополнительных заголовков ответа (например, <code>{"Strict-Transport-Security": "max-age=31536000; includeSubDomains"}</code>)
<code>alb.network.deckhouse.io/session-affinity</code>	JSON для закрепления сессии (session affinity) с режимом cookie (<code>mode</code> , <code>path</code> , <code>cookieName</code> , <code>ttl</code> и др.); не все поля обязательны (например, <code>{"mode": "cookie", "path": "/path", "cookieName": "mycookie", "ttl": 0}</code>)
<code>alb.network.deckhouse.io/hash-key</code>	Ключ консистентного хеширования Envoy для выбора бэкенда Service у объекта HTTPRoute; например, <code>source-ip</code> — хеширование по IP-адресу клиента
<code>alb.network.deckhouse.io/service-upstream</code>	<code>"true"</code> : трафик к upstream идёт через соответствующий сервис, а не напрямую к подам
<code>alb.network.deckhouse.io/basic-auth-secret</code>	<code>namespace/secret</code> с данными <code>htpasswd</code> для HTTP Basic Auth на этом маршруте
<code>alb.network.deckhouse.io/satisfy</code>	<code>all</code> или <code>any</code> : определяет, нужно ли пройти обе проверки (<code>whitelist</code> и <code>basic-auth</code>) или достаточно одной (по умолчанию <code>all</code>)
<code>alb.network.deckhouse.io/auth-url</code>	Определяет URL внешнего сервиса аутентификации
<code>alb.network.deckhouse.io/auth-signin</code>	Определяет URL редиректа для авторизации в случае получения <code>401</code> от внешней аутентификации
<code>alb.network.deckhouse.io/auth-response-headers</code>	Список через запятую: дополнительные заголовки из ответа сервиса аутентификации для передачи в upstream (поверх стандартного <code>allowlist</code>)
<code>alb.network.deckhouse.io/mod-security</code>	JSON-конфигурация для WAF ModSecurity/Coraza на уровне маршрута
<code>alb.network.deckhouse.io/rewrite-target</code>	Позволяет переопределять URL-путь для правил с типом <code>RegularExpression</code> с использованием совпавших частей регулярного выражения (например, <code>/my-path/\1</code>)

Аннотация	Описание
<code>alb.network.deckhouse.io/buffer-max-request-bytes</code>	Максимальный размер буфера для буферизации запроса, в байтах (целое число). По умолчанию Envoy Proxy не буферизует запросы
<code>alb.network.deckhouse.io/limit-rps</code>	Лимит RPS на маршрут
<code>alb.network.deckhouse.io/backend-tls-settings</code>	Например, <code>{"mode": "SIMPLE", "insecureSkipVerify": true, "clientCertificate": "", "privateKey": "", "caCertificates": "", "sni": "example.com", "secret": "<NAMESPACE>/<SECRET_NAME>"}</code> ; позволяет явно указать параметры TLS подключения к upstream. <code><NAMESPACE></code> — неймспейс Secret; <code><SECRET_NAME></code> — имя Secret
<code>alb.network.deckhouse.io/idle-timeout</code>	Устанавливает <code>idle_timeout</code> Envoy на уровне маршрута, в секундах. Аналогично таймаутам <code>proxy-read-timeout/proxy-send-timeout</code> в <code>ingress-nginx</code> . Это тайм-аут неактивности, а не тайм-аут общей длительности запроса
<code>alb.network.deckhouse.io/proxy-buffer-size</code>	Задаёт максимальный размер заголовков ответа в настройках upstream-кластера; при превышении этого значения Envoy возвращает 503. Аналогично <code>nginx.ingress.kubernetes.io/proxy-buffer-size</code>

Публикация приложения при включённом Istio-сайдкаре

Если для прокси шлюза включён Istio-сайдкар с помощью параметра `istioSidecar` объекта `ALBInstance` или `ClusterALBInstance`, трафик к бэкенду должен попадать в сайдкар через Service и содержать FQDN этого Service в заголовке `Host`.

Настройте HTTPRoute следующим образом:

- Добавьте аннотацию `alb.network.deckhouse.io/service-upstream: "true"`, чтобы трафик шёл через объект Service, а не напрямую к подам. Это эквивалент аннотации `nginx.ingress.kubernetes.io/service-upstream: "true"` из `ingress-nginx`.
- Добавьте фильтр `URLRewrite`, который задаёт в поле `hostname` FQDN объекта бэкенд-сервиса. Он заменяет аннотацию `nginx.ingress.kubernetes.io/upstream-vhost` из `ingress-nginx`.

Пример HTTPRoute для шлюза с Istio-сайдкаром:


```

apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: myservice
  namespace: myns
  annotations:
    alb.network.deckhouse.io/service-upstream: "true" # Трафик идёт через объект
Service, чтобы его мог обработать Istio-сайдкар.
spec:
  parentRefs:
    - name: app-listeners # Имя объекта ListenerSet.
      namespace: myns
      kind: ListenerSet
      group: gateway.networking.k8s.io
      sectionName: app-https
      port: 443
  hostnames:
    - myservice.example.com
  rules:
    - filters:
        - type: URLRewrite
          urlRewrite:
            hostname: myservice.myns.svc # FQDN объекта backend-сервиса, чтобы сайдкар
определил назначение.
      backendRefs:
        - name: myservice
          port: 80

```

WAF на HTTPRoute

Аннотация `alb.network.deckhouse.io/mod-security` позволяет включить WAF (Web Application Firewall) на базе ModSecurity/Coraza для отдельного HTTPRoute. Конфигурация WAF применяется только к маршруту, для которого указана аннотация, и не влияет на остальные маршруты.

Аннотация поддерживает следующие поля:

Поле	Описание
<code>mode</code>	Режим работы WAF: <code>on</code> , <code>off</code> , либо любое другое значение для <code>DetectionOnly</code>
<code>preset</code>	Необязательный набор правил. Сейчас поддерживается только <code>owasp-crs</code> . Если поле не указано, набор правил не загружается
<code>paranoiaLevel</code>	Необязательный уровень CRS <code>paranoia</code> от 1 до 4. Применяется только если <code>preset</code> равен <code>owasp-crs</code>
<code>configRef.namespace</code>	Необязательный неймспейс ConfigMap с пользовательскими правилами. По умолчанию используется неймспейс HTTPRoute
<code>configRef.name</code>	Имя ConfigMap с пользовательскими правилами
<code>configRef.key</code>	Необязательный ключ в ConfigMap. Если не указан, читаются все ключи в отсортированном порядке
<code>directives</code>	Необязательный список директив ModSecurity/Coraza, заданных непосредственно в аннотации; они добавляются после правил из набора и ConfigMap

Директивы и правила применяются в следующем порядке:

1. Базовые директивы, поставляемые модулем (`@coraza.conf`, `SecRuleEngine`, `SecResponseBodyAccess Off`).
2. Правила из набора, заданного в `preset`.
3. Пользовательские правила из ConfigMap, указанного в `configRef`.
4. Директивы из поля `directives` аннотации.

Директивы из аннотации применяются последними и могут переопределять набор правил или правила из ConfigMap.

Ниже — примеры аннотации `alb.network.deckhouse.io/mod-security` для HTTPRoute.

- `mode: on`
- OWASP CRS
- ConfigMap

Минимальная конфигурация: WAF включён без набора правил и без пользовательских директив.

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app
  namespace: prod
  annotations:
    alb.network.deckhouse.io/mod-security: |
      {
        "mode": "on"
      }
spec:
  hostnames:
    - app.example.com
  parentRefs:
    - group: gateway.networking.k8s.io
      kind: ListenerSet
      name: app-listeners
      namespace: prod
      sectionName: app-https
      port: 443
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /
      backendRefs:
        - name: app-svc
          port: 8080
```

WAF с предустановленным набором правил OWASP CRS и уровнем `paranoiaLevel`:

```
metadata:
  annotations:
    alb.network.deckhouse.io/mod-security: |
      {
        "mode": "on",
        "preset": "owasp-crs",
        "paranoiaLevel": 1
      }
```

WAF с OWASP CRS, пользовательскими правилами из ConfigMap и дополнительными директивами в аннотации:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: waf-rules
  namespace: prod
data:
  rules.conf: |
    SecRule ARGS:test "@streq block" \
      "id:1000001,phase:2,deny,status:403,msg:'test waf block'"
  ---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: app
  namespace: prod
annotations:
  alb.network.deckhouse.io/mod-security: |
    {
      "mode": "on",
      "preset": "owasp-crs",
      "paranoiaLevel": 1,
      "configRef": {
        "name": "waf-rules",
        "key": "rules.conf"
      },
      "directives": [
        "SecResponseBodyAccess Off"
      ]
    }

```

Подробнее о синтаксисе правил и директив в документации Coraza и ModSecurity:

- синтаксис правил Coraza и директива `SecRule`;
- переменные ModSecurity;
- операторы ModSecurity;
- директива `SecRuleEngine` и другие директивы ModSecurity.

Текущие особенности и ограничения WAF:

- Поддерживается только набор правил `owasp-crs`.
- Параметр `paranoiaLevel` применяется только при использовании `preset: owasp-crs`. Если параметр `preset` не указан или имеет другое значение, параметр `paranoiaLevel` игнорируется.
- Допустимые значения `paranoiaLevel`: от 1 до 4. На практике рекомендуется начинать со значения 1.
- WAF проверяет только входящие запросы к приложению и при необходимости блокирует их. Ответы приложения клиенту не анализируются.
- Правила, заданные через ConfigMap, могут быть многострочными: строки, завершающиеся символом `\`, автоматически объединяются.

Дополнительные ресурсы

- Использование Application Load Balancer (ALB)
- Документация модуля `alb`

Публикация приложений средствами Ingress NGINX Controller

Публикация приложений средствами Ingress NGINX Controller

Для публикации приложений администратор кластера должен создать Ingress-контроллер. Порядок настройки описан в разделе «Примеры настройки балансировки». Запросите у администратора имя `ingressClass` контроллера и укажите его в манифесте ресурса Ingress, который маршрутизирует входящий трафик к приложению.

- HTTP

- HTTPS

Пример базового Ingress для публикации приложения по HTTP:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: prod
spec:
  ingressClassName: nginx # Имя IngressClass, предоставленное администратором
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: productpage
            port:
              number: 80
```

Для HTTPS укажите Secret с сертификатом в секции `tls`. Secret должен существовать в том же неймспейсе, что и Ingress. Сертификат может выпустить либо `cert-manager`, либо администратор вручную.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: prod
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - app.example.com
    secretName: application-tls # Secret с tls.crt и tls.key в неймспейсе prod.
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: productpage
            port:
              number: 80
```

Дополнительные параметры маршрутизации задаются аннотациями Ingress. Полный список поддерживаемых аннотаций — в описании модуля `ingress-nginx`. Частые примеры:

- `nginx.ingress.kubernetes.io/rewrite-target` — перезапись пути;
- `nginx.ingress.kubernetes.io/whitelist-source-range` — список разрешённых CIDR;
- `nginx.ingress.kubernetes.io/backend-protocol: "GRPC"` — протокол к бэкенду для gRPC, если имя порта Service не указывает на gRPC.

Проверка

После применения Ingress проверьте объект и адрес контроллера:

```
d8 k -n <NAMESPACE> get ingress
d8 k -n <NAMESPACE> describe ingress <INGRESS_NAME>
```

Убедитесь, что в статусе Ingress есть адрес и что DNS для `host` указывает на точку входа контроллера, которую сообщил администратор.

Проверьте доступность приложения, подставив адрес точки входа (успешный ответ — код `200` или другой ожидаемый код приложения):

```
curl -vk \
  --resolve app.example.com:443:<ENTRY_POINT_ADDRESS> \
  https://app.example.com/
```

Балансировка gRPC

Раздел относится к публикации gRPC через Ingress NGINX Controller (`ingress-nginx`). Для Gateway API используйте GRPCRoute.

Чтобы балансировка gRPC-сервисов за Ingress NGINX заработала автоматически, присвойте порту соответствующего объекта Service имя с префиксом или значением `grpc`. Если имя порта другое, добавьте аннотацию `nginx.ingress.kubernetes.io/backend-protocol: "GRPC"` к Ingress.

Дополнительные ресурсы

- Документация модуля `ingress-nginx`

Публикация приложений средствами Istio

Публикация приложений средствами Istio

Публикация средствами Istio настраивается в два слоя. Администратор кластера разворачивает `IngressIstioController` (и связанную инфраструктуру), как описано в разделе «Istio Ingress Gateway». Разработчики приложения создают ресурсы Gateway и VirtualService, как показано ниже.

При публикации приложения средствами Istio выберите один из способов: «Публикация приложений с использованием Ingress NGINX» или «Публикация приложений с использованием ресурса Istio Ingress Gateway».

Публикация приложений с использованием Ingress NGINX

Для публикации приложения средствами Ingress NGINX администратор кластера должен настроить Ingress-контроллер, добавив к нему сайдкар от Istio.

Для публикации приложения подготовьте Ingress-ресурс, который ссылается на сервис. Укажите `ingressClassName` контроллера с Istio-сайдкаром (значение сообщит администратор). 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 отправлять их напрямую в поды приложения. Сайдкар-контейнер `istio-proxy` перехватывает только этот трафик, остальные запросы отправляются напрямую;
- `nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc` — с этой аннотацией сайдкар сможет идентифицировать прикладной сервис, для которого предназначен запрос.

Примеры:

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: productpage
  namespace: bookinfo
  annotations:
    # Включает проксирование трафика через nginx на ClusterIP вместо собственных IP
    # подов.
    nginx.ingress.kubernetes.io/service-upstream: "true"
    # В Istio вся маршрутизация осуществляется на основе `Host:` заголовка запросов.
    # Это позволяет не заводить в Istio внешний домен `productpage.example.com` –
    # вместо него используется внутренний домен, уже известный Istio.
    nginx.ingress.kubernetes.io/upstream-vhost: productpage.bookinfo.svc
spec:
  # Имя IngressClass контроллера с Istio-сайдкаром.
  ingressClassName: nginx
  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

```

Публикация приложений с использованием ресурса Istio Ingress Gateway

Для публикации приложения средствами Istio Ingress Gateway администратор кластера должен создать ресурс `IngressIstioController`.

Для публикации приложения с использованием ресурса Istio Ingress Gateway создайте ресурс `Gateway`. В поле `spec.selector` укажите лейбл, ссылающийся на `ingressGatewayClass`, и имя `Secret`, полученные от администратора кластера:

```

apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: gateway-app
  namespace: app-ns
spec:
  selector:
    # Значение ingressGatewayClass, полученное от администратора кластера.
    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
        # Secret с сертификатом и ключом, созданный администратором в неймспейсе d8-
        ingress-istio.
        # Поддерживаемые форматы Secret: https://istio.io/latest/docs/tasks/traffic-
        management/ingress/secure-ingress/#key-formats.
        credentialName: app-tls-secret
      hosts:
        - app.example.com

```

Затем определите правила маршрутизации с помощью VirtualService, который связывает Gateway и обслуживаемый им сервис:

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  # Имя ресурса Gateway, созданного выше.
  gateways:
    - gateway-app
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            # Имя сервиса, на который нужно направить трафик.
            host: app-svc

```

Canary-развёртывание через VirtualService

Общий обзор canary в Deckhouse Kubernetes Platform (DKP) — в разделе «Canary-развёртывание». Ниже — пример на VirtualService и DestinationRule.

Для постепенного переключения трафика между версиями приложения используйте DestinationRule с подмножествами (subsets) и веса в VirtualService. Пример направляет 90% трафика на стабильную версию и 10% на canary:

```

apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: app-svc
  namespace: app-ns
spec:
  host: app-svc
  subsets:
    - name: stable
      labels:
        version: v1
    - name: canary
      labels:
        version: v2
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: vs-app
  namespace: app-ns
spec:
  gateways:
    - gateway-app
  hosts:
    - app.example.com
  http:
    - route:
        - destination:
            host: app-svc
            subset: stable
          weight: 90
        - destination:
            host: app-svc
            subset: canary
          weight: 10

```

Поды версий должны иметь лейблы `version: v1` и `version: v2`, совпадающие с `subsets` в `DestinationRule`. Перед изменением весов проверьте, что обе версии обслуживаются сервисом `app-svc`.

После применения манифестов отправьте несколько запросов на `app.example.com` и проверьте, что трафик распределяется в заданной пропорции — примерно 90/10. Чтобы определить, какая версия ответила, используйте признак, отличающий `stable` от `canary` в самом приложении, например, значение в теле ответа или отдельный HTTP-заголовок:

```

for i in $(seq 1 20); do curl -S https://app.example.com/ | grep version; done \
  | sort | uniq -c

```

При необходимости скорректируйте веса и повторите проверку.

Дополнительные ресурсы

- Использование Application Load Balancer (ALB)
- Документация модуля `istio`

Дополнительные сети для использования в прикладных подах

В DKP реализована возможность использования дополнительных программно-определяемых сетей (далее — дополнительные сети) для прикладных нагрузок (поды, виртуальные машины). Вы можете использовать сети следующих типов:

- Кластерная (общедоступная) — сеть, общедоступная в каждом проекте, настраивается и

управляется администратором. Пример — публичная WAN-сеть или shared-сеть обмена трафиком между проектами. Для создания такой сети и ее использования для прикладных подов обратитесь к администратору кластера.

- Сеть проекта (пользовательская сеть) — сеть, доступная в рамках неймспейса, создается и управляется пользователем с использованием предоставленного администратором манифеста NetworkClass.

Подробнее о дополнительных программно-определяемых сетях — в разделе «Настройка и подключение дополнительных виртуальных сетей для использования в прикладных подах».

Создание сети проекта (пользовательской сети)

Для создания сети для проекта используйте кастомные ресурсы Network и NetworkClass (предоставляется администратором):

1. Создайте и примените манифест объекта Network, указав в поле `spec.networkClass` имя NetworkClass, полученное у администратора:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  name: my-network
  namespace: my-namespace
spec:
  networkClass: my-network-class # Имя NetworkClass, полученное от администратора.
```

i Поддерживается статическое определение номера VLAN ID из пула, выданного администратором кластера. Если значение поля `spec.vlan.id` не указано, VLAN ID будет назначен динамически.

2. После создания объекта Network проверьте его статус:

```
d8 k -n my-namespace get network my-network -o yaml
```

Пример статуса объекта Network:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  ...
status:
  bridgeName: d8-br-600
  conditions:
    - lastTransitionTime: "2025-09-29T14:51:26Z"
      message: All node interface attachments are ready
      reason: AllNodeInterfaceAttachmentsAreReady
      status: "True"
      type: AllNodeAttachmentsAreReady
    - lastTransitionTime: "2025-09-29T14:51:26Z"
      message: Network is operational
      reason: NetworkReady
      status: "True"
      type: Ready
  nodeAttachmentsCount: 1
  observedGeneration: 1
  readyNodeAttachmentsCount: 1
  vlanID: 600
```

После создания сети ее можно подключать к подам.

Подключение дополнительных сетей к подам

Вы можете подключать к подам кластерные сети и сети проекта. Для этого используйте аннотацию пода, в которой укажите параметры подключаемых дополнительных сетей.

Пример манифеста пода с добавлением двух дополнительных сетей (кластерной `my-cluster-network` и сети проекта `my-network`):

i В поле `ifName` (опционально) задается имя TAP-интерфейса внутри пода. В поле `mac` (опционально) задается MAC-адрес, который следует назначить TAP-интерфейсу.

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-additional-networks
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "veth_mynet",
          "mac": "aa:bb:cc:dd:ee:ff"
        },
        {
          "type": "ClusterNetwork",
          "name": "my-cluster-network",
          "ifName": "veth_public"
        }
      ]
spec:
  containers:
    - name: app
      # остальные параметры...
```

IPAM для дополнительных сетей

Механизм IPAM (IP Address Management) позволяет выделять IP-адреса (поддерживаются IPv4-адреса) из пулов и назначать их на дополнительные сетевые интерфейсы подов, подключаемых к кластерным сетям и сетям проекта.

Выделением IP-адресов для подключения к кластерным сетям занимается администратор кластера. Он включает, настраивает IPAM для сетей и определяет пул IP-адресов для них. Назначать адреса и настраивать IPAM в сетях проекта могут пользователи.

Особенности использования IPAM в кластере DKP

IPAM в кластере DKP имеет следующие особенности использования:

- IPAM включается **на уровне сети** через параметр `spec.ipam.ipAddressPoolRef` объекта `Network` или `ClusterNetwork` (для `ClusterNetwork` IPAM включает администратор кластера).
- Назначение IP-адреса на интерфейс пода описывается в добавляемой к поду аннотации `network.deckhouse.io/networks-spec` через поля:
 - `ipAddressNames` — список объектов `IPAddress`, которые нужно назначить на данный интерфейс (если параметр не указан, `IPAddress` может создаваться автоматически).
 - `skipIPAssignment` — управление резервированием/отслеживанием `IPAddress`. Если `skipIPAssignment: true`, включается резервирование/отслеживание `IPAddress`, но IP-адрес **не назначается** на интерфейс внутри пода (вариант для продвинутого использования).
- Поддерживается назначение **только IPv4-адресов** на дополнительные сетевые интерфейсы подов.



Если в одном поде подключено несколько дополнительных сетей с включенным IPAM, рекомендуется явно задавать `ipAddressNames` для каждого интерфейса (создавая отдельные `IPAddress`). Автоматически создаваемый `IPAddress` привязан к поду и может не подходить для нескольких IPAM-сетей одновременно.

Выделение пула IP-адресов для сети проекта и включение IPAM



Чтобы выделить пул адресов для сети проекта, создайте ресурс `IPAddressPool` **в том же неймспейсе**, что и сеть проекта (поды, подключаемые к сети).

Для выделения пула адресов и их назначения на сетевые интерфейсы подов, подключаемых к сети проекта, выполните следующие действия:

1. Создайте пул адресов. Для этого используйте ресурс `IPAddressPool`.

Пример:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddressPool
metadata:
  name: my-net-pool
  namespace: my-namespace
spec:
  leaseTTL: 1h
  pools:
    - network: 192.168.10.0/24
      ranges:
        - 192.168.10.50-192.168.10.200
      routes:
        - destination: 10.10.0.0/16
          via: 192.168.10.1
```



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 дополнительной сети. Для этого в параметре `spec.ipam.ipAddressPoolRef` ресурса `Network` укажите параметры созданного на предыдущем шаге `IPAddressPool`.

Пример:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: Network
metadata:
  name: my-network
  namespace: my-namespace
spec:
  networkClass: my-network-class
  ipam:
    ipAddressPoolRef:
      kind: IPAddressPool
      name: my-net-pool
```

После выделения пула IP-адресов для сети проекта их можно назначать на сетевые интерфейсы подов, подключаемых к этой сети.

Назначение IP-адресов на сетевые интерфейсы подов, подключаемых к дополнительной сети

В DKP реализована автоматическая выдача IP-адресов для дополнительных интерфейсов подов, а также возможность ручного назначения конкретных статических IP-адресов на дополнительные интерфейсы подов.

Автоматическая выдача IP-адресов

Чтобы IP-адрес для дополнительного сетевого интерфейса пода был выбран автоматически из пула, добавьте к поду аннотацию `network.deckhouse.io/networks-spec`. В этой аннотации укажите параметры сети с включенным IPAM.

Пример (IP-адрес будет выбран автоматически из пула, созданного для сети `my-network` и назначен на интерфейс `net1`):

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-ipam
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1"
        }
      ]
spec:
  containers:
    - name: app
      image: nginx
```

В таком случае будет автоматически создан объект `IPAddress` (тип `Auto`) и из прикрепленного к дополнительной сети (в примере — `my-network`) пула будет автоматически выбран IP-адрес и назначен на сетевой интерфейс пода.

Ручное (явное) создание `IPAddress` с типом `Auto`

Также можно **вручную** создать объект `IPAddress` с `spec.type: Auto` (без указания параметра `static.ip`). В этом случае контроллер выделит свободный адрес из пула прикрепленного к дополнительной сети (в примере — `my-network`), а вы сможете привязать его к конкретному интерфейсу пода через параметр `ipAddressNames` в аннотации `network.deckhouse.io/networks-spec`.

Пример:

1. Создайте объект `IPAddress`:

```
apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddress
metadata:
  name: app-net1-auto
  namespace: my-namespace
spec:
  networkRef:
    kind: Network
    name: my-network
  type: Auto
```

2. Назначьте IP-адрес из пула на интерфейс пода:

```

apiVersion: v1
kind: Pod
metadata:
  name: app-with-manual-auto-ip
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1",
          "ipAddressNames": ["app-net1-auto"]
        }
      ]
spec:
  containers:
    - name: app
      image: nginx

```

Ручное назначение статического IP-адреса на дополнительный интерфейс пода

Чтобы назначить конкретный статический IP-адрес на дополнительный интерфейс пода, выполните следующие шаги:

1. Создайте IPAddress в неймспейсе пода и укажите, для какой сети он предназначен, и какой IP-адрес требуется:

```

apiVersion: network.deckhouse.io/v1alpha1
kind: IPAddress
metadata:
  name: app-net1-static
  namespace: my-namespace
spec:
  networkRef:
    kind: Network
    name: my-network
  type: Static
  static:
    ip: 192.168.10.50

```

2. Подключите сеть к поду и укажите созданный IPAddress в параметре `ipAddressNames`:

```

apiVersion: v1
kind: Pod
metadata:
  name: app-with-static-ip
  namespace: my-namespace
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "Network",
          "name": "my-network",
          "ifName": "net1",
          "ipAddressNames": ["app-net1-static"]
        }
      ]
spec:
  containers:
    - name: app
      image: nginx

```

Проверка назначения IP-адреса интерфейсу

Чтобы проверить, что IP-адрес назначен на интерфейс, выполните следующие шаги:

1. Проверьте выделенный адрес и фазу у `IPAddress` (фаза должна быть `Allocated`):

```
d8 k -n my-namespace get ipaddress app-net1-static -O yaml
```

Пример вывода:

NAME	PHASE	AGE	TYPE	KIND	NAME	ADDRESS	NETWORK
ipaddress-auto-1	Allocated	4d1h	Auto	Network	mynet	192.168.12.1	192.168.12.0/24
ipaddress-auto-2	Allocated	4d1h	Auto	Network	mynet	192.168.12.2	192.168.12.0/24

2. Проверьте аннотацию пода `network.deckhouse.io/networks-status` (включая `ipAddressConfigs` и маршруты):

```

d8 k -n my-namespace get pod app-with-static-ip -O
jsonpath='{.metadata.annotations.network\.deckhouse\.io/networks-status}' |
jq

```

Пример вывода:

```
[
  {
    "type": "Network",
    "name": "mynet",
    "ifName": "aabbcc",
    "mac": "ae:1c:68:7a:00:8f",
    "vlanID": 0,
    "ipAddressConfigs": [
      {
        "name": "ipaddress-auto-1",
        "address": "192.168.12.1",
        "network": "192.168.12.0/24"
      }
    ],
    "conditions": [
      {
        "type": "Configured",
        "status": "True",
        "lastTransitionTime": "2026-02-26T10:06:49Z",
        "reason": "InterfaceConfiguredSuccessfully",
        "message": ""
      },
      {
        "type": "Negotiated",
        "status": "True",
        "lastTransitionTime": "2026-02-26T10:06:49Z",
        "reason": "Up",
        "message": ""
      }
    ]
  }
]
```

Дополнительные ресурсы

- Кастомные ресурсы модуля `sdn`

Подключение физических сетевых интерфейсов к подам DPDK-приложений

Если в вашем неймспейсе (проекте) размещаются высокопроизводительные рабочие нагрузки, требующие прямого доступа к оборудованию (например, приложения DPDK), можно использовать прямое подключение физических сетевых интерфейсов (Physical Functions и Virtual Functions) к подам через Kubernetes Dynamic Resource Allocation (DRA).

Физические сетевые интерфейсы могут подключаться в поды в одном из двух режимов:

- **Shared** — создается Virtual Functions (VF) из Physical Functions (PF) с использованием SR-IOV, несколько подов могут совместно использовать одно и то же оборудование.
- **Dedicated** — каждый под получает эксклюзивный доступ к полному PF.

Подробнее о возможностях и особенностях работы с Underlay-сетями в DKP — в разделе «Настройка и подключение underlay-сетей для проброса аппаратных устройств».

Подключение физических сетевых интерфейсов к подам

Для подключения физических сетевых интерфейсов (PF/VF) напрямую в поды для DPDK-приложений необходимо:

1. Убедиться, что администратор добавил на ваш неймспейс лейбл для использования Underlay-сетей.
2. Создать под с аннотацией, запрашивающей устройство из Underlay-сети.

Создание пода с устройством из Underlay-сети

Создайте под, который запрашивает устройство из Underlay-сети. В аннотации пода `network.deckhouse.io/networks-spec` укажите параметры:

- `type: "UnderlayNetwork"` — указывает, что это запрос физического устройства;
- `name: "underlay-network-name"` — имя ресурса UnderlayNetwork, созданного администратором;
- `bindingMode` — режим привязки устройства (VFIO-PCI, DPDK или NetDev).

Пример конфигурации пода для режима DPDK (универсальный режим, который автоматически выбирает подходящий драйвер для вендора сетевого адаптера):

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: mydpdk
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK"
        }
      ]
spec:
  containers:
    - name: dpdk-container
      image: dpdk-app:latest
      securityContext:
        privileged: false
        capabilities:
          add:
            - NET_ADMIN
            - NET_RAW
            - IPC_LOCK
      volumeMounts:
        - mountPath: /hugepages
          name: hugepage
      resources:
        limits:
          hugepages-2Mi: 4Gi
          memory: 4Gi
          cpu: 4
        requests:
          cpu: 4
          memory: 4Gi
        command: ["/bin/sh", "-c", "sleep infinity"]
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages
```



Для DPDK-приложений важно:

- Настроить `capabilities` (`NET_ADMIN`, `NET_RAW`, `IPC_LOCK`) для запуска в непривилегированном режиме вместо использования `privileged: true`;
- Подключить `volumes` с `hugepages`, так как DPDK требует `hugepages` для эффективного управления памятью.



Для VF устройств в режиме Shared можно дополнительно указать `vlanID` в аннотации для настройки VLAN-тегирования на VF:

```
network.deckhouse.io/networks-spec: |
[
  {
    "type": "UnderlayNetwork",
    "name": "dpdk-shared-network",
    "bindingMode": "VFIO-PCI",
    "vlanID": 100
  }
]
```

После создания пода убедитесь, что устройство было выделено, проверив аннотацию `network.deckhouse.io/networks-status`:

```
d8 k -n mydpdk get pod dpdk-app -O
jsonpath='{.metadata.annotations.network\.deckhouse\.io/networks-status}' | jq
```

Вы также можете проверить ResourceClaim, который был автоматически создан:

```
d8 k -n mydpdk get resourceclaim
```

Пример статуса пода с выделенным устройством UnderlayNetwork:

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: mydpdk
  annotations:
    network.deckhouse.io/networks-spec: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK"
        }
      ]
    network.deckhouse.io/networks-status: |
      [
        {
          "type": "UnderlayNetwork",
          "name": "dpdk-shared-network",
          "bindingMode": "DPDK",
          "netDevInterfaces": [
            {
              "name": "ens1f0",
              "mac": "00:1b:21:bb:aa:cc"
            }
          ],
          "conditions": [
            {
              "type": "Configured",
              "status": "True",
              "reason": "InterfaceConfiguredSuccessfully",
              "message": "",
              "lastTransitionTime": "2025-01-15T10:35:00Z"
            },
            {
              "type": "Negotiated",
              "status": "True",
              "reason": "Up",
              "message": "",
              "lastTransitionTime": "2025-01-15T10:35:00Z"
            }
          ]
        }
      ]
status:
  phase: Running
  ...

```

Внутрикластерное взаимодействие

Для организации внутрикластерного взаимодействия в Deckhouse Kubernetes Platform рекомендуется использовать сервисы вместо обращения напрямую к подам. Они обеспечивают балансировку нагрузки между подами, стабильное сетевое взаимодействие и интеграцию с DNS для удобного обнаружения сервисов. Также сервисы поддерживают различные сценарии доступа и обеспечивают изоляцию и безопасность сетевого трафика.

При необходимости можно использовать стандартный балансировщик на базе сервисов или расширенный балансировщик на базе модуля `service-with-healthchecks`.

Стандартный балансировщик

В Kubernetes за внутреннюю и внешнюю балансировку запросов отвечает ресурс типа `Service`. Этот ресурс:

- распределяет запросы между рабочими подами приложения;
- исключает из балансировки повреждённые экземпляры подов.

Для проверки способности пода обрабатывать входящие запросы применяются readiness-пробы, которые указываются в спецификации контейнеров, входящих в этот под.

Ограничения стандартного балансировщика Service

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 считается 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.

Пример сервиса для стандартного балансировщика

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  namespace: bookinfo
spec:
  ports:
    - name: http
      port: 9080
  selector:
    app: productpage
  type: ClusterIP
```

Пример стандартного балансировщика 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
  namespace: 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

```

Расширенный балансировщик

Расширенный балансировщик реализуется с помощью ресурсов ServiceWithHealthcheck модуля `service-with-healthchecks`.

В отличие от стандартного балансировщика, где `readiness`-пробы привязаны к состоянию контейнеров, ресурс `ServiceWithHealthcheck`, который используется в балансировщике, позволяет настраивать активные пробы на отдельные TCP-порты. Таким образом, каждый балансировщик, обслуживающий один и тот же под, может работать независимо от других.

Внутреннее устройство балансировщика

Балансировщик состоит из двух компонентов:

- контроллер — работает на master-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 объектов 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 объект EndpointSlice со списком проверенных IP-адресов и портов. Данный объект EndpointSlice привязан к дочернему ресурсу Service, созданному выше.
- CNI сопоставит все объекты 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 `healthcheck`. Во время жизненного цикла ресурса ServiceWithHealthchecks создается одноименный сервис в том же пространстве имён, чтобы привычным способом (`kube-proxy` или `cni`) направить трафик на рабочие нагрузки в кластере.

Настройка балансировщика



Для использования расширенных балансировщиков:

- настройте сетевые политики проекта, где будут использоваться балансировщики ServiceWithHealthchecks;
- если есть необходимость, перейдите вручную на использование ServiceWithHealthchecks вместо стандартных балансировщиков (замените используемые объекты Service на ServiceWithHealthchecks).

Подробнее — в документации модуля `service-with-healthchecks`.

Настроить данный способ балансировки можно при помощи ресурса ServiceWithHealthchecks:

- Его спецификация идентична стандартному ресурсу 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.

Пример конфигурации расширенных 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()"

```

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: host
  cpu:
    cores: 1
  memory:
    size: 1Gi
  provisioning:
    type: UserData
    userData: |
      #cloud-config
      ssh_pwauth: True
      users:
        - name: cloud
          passwd:
'$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo1RquzS0
R6GgdIDlvS.kbUwDablGZKZcTP/'
      shell: /bin/bash
      sudo: ALL=(ALL) NOPASSWD:ALL
      lock_passwd: False
  blockDeviceRefs:
    - kind: VirtualDisk
      name: linux-disk

```

Манифесты расширенных 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

```

Дополнительные ресурсы

- Документация модуля `service-with-healthchecks`

Управление авторизацией и доступом к нагрузке с Istio

Для управления авторизацией и контролем доступа к workload можно использовать модуль `istio`. Перед настройкой авторизации убедитесь, что модуль включен в кластере.

Авторизация

Управление авторизацией осуществляется с помощью ресурса `AuthorizationPolicy` от Istio. Когда для сервиса создается этот ресурс, применяются следующие правила принятия решения о запросах:

- 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 можно использовать следующие аргументы:

- идентификаторы сервисов и wildcard на их основе (`mycluster.local/ns/myns/sa/myapp` или `mycluster.local/*`);
- пространство имен;
- диапазоны IP;
- HTTP-заголовки;
- JWT-токены из прикладных запросов.

Ресурс `AuthorizationPolicy`

Подробнее ознакомиться с `AuthorizationPolicy` можно в документации Istio.

Ресурс `AuthorizationPolicy` включает и определяет контроль доступа к workload. Поддерживает как `ALLOW`-, так и `DENY`-правила, описанные выше.

Аргументы для принятия решения об авторизации:

- `source`:
 - `namespace`;
 - `principal` (идентификатор юзера, полученный после аутентификации);
 - IP.
- `destination`:
 - `method` (`GET`, `POST` и т. д.);
 - `host`;
 - `port`;

- URI.
- `conditions`:
 - HTTP-заголовки;
 - аргументы `source`;
 - аргументы `destination`;
 - JWT-токены.

Дополнительные ресурсы

- Документация модуля `istio`

Настройка маршрутизации запросов с Istio

Для маршрутизации HTTP- и TCP-запросов в Deckhouse Kubernetes Platform можно использовать модуль `istio`.

Основной ресурс для управления маршрутизацией — `VirtualService` от Istio, он позволяет настраивать маршрутизацию HTTP- или TCP-запросов.

Ресурс `VirtualService`

Подробнее ознакомиться с `VirtualService` можно в документации `istio`.

Использование `VirtualService` опционально, классические сервисы продолжают работать, если их функционала достаточно. С помощью этого ресурса можно настроить маршрутизацию запросов:

- Аргументы для принятия решения о маршруте:
 - `host`;
 - `uri`;
 - `weight` (вес).
- Параметры итоговых направлений:
 - новый `host`;
 - новый `uri`;
 - если `host` определен с помощью `DestinationRule` можно направлять запросы на subset'ы;
 - таймаут и настройки `retry` (повторных попыток).



Для корректной работы `destination` в Istio необходимо его указать. Если вы используете внешний API, укажите его с помощью `ServiceEntry`.

Дополнительные ресурсы

- Документация модуля `istio`

Активация Istio для приложений

Активация Istio для приложений возможна, если в кластере включен и настроен модуль `istio`. За это отвечает администратор кластера.

Суть активации — добавить сайдкар-контейнер к подам приложения, после чего Istio сможет управлять трафиком.

Рекомендованный способ добавления сайдкаров — использовать сайдкар-injector. Istio умеет «подсаживать» к подам приложения сайдкар-контейнер с помощью механизма Admission Webhook. Для добавления сайдкаров используются лейблы и аннотации:

- Лейбл `namespace` — обозначает ваше пространство имён для компонента сайдкар-injector. После применения лейбла к новым подам будут добавлены сайдкар-контейнеры:
 - `istio-injection=enabled` — использует глобальную версию Istio (`spec.settings.globalVersion` в ресурсе `ModuleConfig`);
 - `istio.io/rev=v1x16` — использует конкретную версию Istio для этого пространства имён.
- Аннотация к поду `sidecar.istio.io/inject` (`"true"` или `"false"`) позволяет локально переопределить политику `sidecarInjectorPolicy`. Эти аннотации работают только в пространствах имён, обозначенных лейблами из списка выше.

Также существует возможность добавить сайдкар к определённому поду в пространстве имён без установленных лейблов `istio-injection=enabled` или `istio.io/rev=vXxYZ` путем установки лейбла `sidecar.istio.io/inject=true`.

Istio-proxy, который работает в качестве сайдкар-контейнера, тоже потребляет ресурсы и добавляет накладные расходы:

- Каждый запрос DNAT'ится в Envoy, который обрабатывает это запрос и создает еще один. На принимающей стороне — аналогично.
- Каждый Envoy хранит информацию обо всех сервисах в кластере, что требует памяти. Больше кластер — больше памяти потребляет Envoy. Решение — кастомный ресурс Sidecar.

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-проху. Используйте эту опцию, только если у вашего сервиса есть ClusterIP;
 - `nginx.ingress.kubernetes.io/upstream-vhost: "myservice.myns.svc"` — сайдкар-проху Ingress-контроллера принимает решения о маршрутизации на основе заголовка `Host`. Без этой аннотации контроллер оставит заголовок с адресом сайта, например `Host: example.com`.

Дополнительные ресурсы

- Кастомные ресурсы модуля `ingress-nginx`
- Документация модуля `istio`

Управление балансировкой запросов между эндпоинтами сервиса с Istio

Для управления балансировкой запросов между эндпоинтами сервиса можно использовать модуль `istio`. Перед настройкой балансировки убедитесь, что модуль включен в кластере.

Основной ресурс для управления балансировкой запросов — `DestinationRule` от `istio.io`. Этот ресурс предоставляет возможности настройки следующих параметров:

- лимиты и таймауты для TCP;
- алгоритмы балансировки между эндпоинтами;
- правила определения проблем на стороне эндпоинта для выведения его из балансировки;
- нюансы шифрования.



Все настраиваемые лимиты работают для каждого пода клиента по отдельности. Например, если настроить для сервиса ограничение на одно TCP-соединение, а клиентских подов — три, то сервис получит три входящих соединения.

Ресурс DestinationRule

Подробнее ознакомиться с `DestinationRule` можно в документации `istio`. Используйте этот ресурс, чтобы:

- Определить стратегию балансировки трафика между эндпоинтами сервиса:
 - алгоритм балансировки (`LEAST_CONN`, `ROUND_ROBIN`, и т. д.);
 - определение и исключение неработающих эндпоинтов;
 - лимиты TCP-соединений и запросов для эндпоинтов;
 - поддержка `Sticky Sessions`;
 - настройка `Circuit Breaker`.
- Задать альтернативные группы эндпоинтов для обработки трафика (применимо для `Canary Deployments`). Каждая группа может иметь свои стратегии балансировки.
- Настроить TLS для исходящих запросов.

Дополнительные ресурсы

- Документация модуля `istio`

Настройка Locality failover с Istio

В Deckhouse Kubernetes Platform можно реализовать механизм `Locality failover` средствами модуля `istio`. Перед настройкой механизма убедитесь, что модуль включен в кластере.

Механизм `Locality failover` управляет маршрутизацией трафика и направляет его на

приоритетный фейловер в случае недоступности определённых экземпляров сервисов.



При необходимости ознакомьтесь с документацией [Locality failover](#).

С использованием Istio настраивается приоритетный географический фейловер между эндпоинтами. Для определения зоны применяются лейблы узлов с соответствующей иерархией:

- `topology.istio.io/subzone`;
- `topology.kubernetes.io/zone`;
- `topology.kubernetes.io/region`.

Это полезно для межкластерного фейловера при использовании совместно с мультикластером.



Для активации Locality Failover используется ресурс `DestinationRule`, в котором также необходимо указать параметр `outlierDetection`.

Пример:

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
spec:
  host: helloworld
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true # Включение LF.
    outlierDetection: # Обязательное включение outlierDetection.
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m
```

Дополнительные ресурсы

- Документация модуля `istio`

Настройка Retry для запросов с Istio

Для повторных попыток (Retry) для запросов можно использовать модуль `istio`. Перед настройкой убедитесь, что модуль включен в кластере.

Чтобы настроить Retry для запросов, используйте ресурс `VirtualService` от `istio.io`.



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

```

Ресурс VirtualService

При необходимости ознакомьтесь с документацией VirtualService.

Использование VirtualService опционально, классические сервисы продолжают работать, если их функционала достаточно. С помощью этого ресурса можно настроить маршрутизацию запросов:

- Аргументы для принятия решения о маршруте:
 - `host`;
 - `uri`;
 - `weight` (вес).
- Параметры итоговых направлений:
 - новый `host`;
 - новый `uri`;
 - если `host` определен с помощью DestinationRule можно направлять запросы на subset'ы;
 - таймаут и настройки `retry` (повторных попыток).



Для корректной работы `destination` в Istio необходимо его указать. Если вы используете внешний API, укажите его с помощью ServiceEntry.

Дополнительные ресурсы

- Документация модуля `istio`

Настройка ресурсов для istio-proxy сайдбаров

При использовании модуля `istio` в кластере вы можете управлять ресурсами, выделяемыми для `istio-proxy` сайдбаров в отдельных рабочих нагрузках. Для этого используются аннотации.

Поддерживаемые аннотации

Для переопределения глобальных ограничений ресурсов для `istio-proxy` сайдбаров в отдельных рабочих нагрузках поддерживаются аннотации:

Аннотация	Описание	Пример значения
<code>sidecar.istio.io/proxyCPU</code>	Запрос CPU для сайдбара	<code>200m</code>
<code>sidecar.istio.io/proxyCPULimit</code>	Лимит CPU для сайдбара	<code>"1"</code>
<code>sidecar.istio.io/proxyMemory</code>	Запрос памяти для сайдбара	<code>128Mi</code>
<code>sidecar.istio.io/proxyMemoryLimit</code>	Лимит памяти для сайдбара	<code>512Mi</code>



Все аннотации из таблицы должны быть указаны в манифесте рабочей нагрузки одновременно. Частичная конфигурация не поддерживается.

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
```

Дополнительные ресурсы

- Документация модуля `istio`

Настройка Circuit Breaker

В Deckhouse Kubernetes Platform механизм Circuit Breaker реализуется средствами Istio (модуль istio) и обеспечивает следующие возможности:

- временное исключение эндпоинта из балансировки, если превышен лимит ошибок;
- настройка лимитов на количество TCP-соединений и количество запросов в сторону одного эндпоинта;

- выявление зависших запросов и обрывание их с кодом ошибки (HTTP request timeout).

Пример настройки 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 # Допускается 7 ошибок (включая `5xx`, TCP-таймауты и
      HTTP-таймауты)
      interval: 5m # в течение 5 минут,
      baseEjectionTime: 15m # после которых эндпоинт будет исключен из балансировки
      на 15 минут.
```

Также для настройки HTTP-таймаутов используется ресурс `VirtualService`. Эти таймауты учитываются и при подсчёте 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
```

Дополнительные ресурсы

- Документация модуля `istio`

Настройка Canary deployment

Canary deployment — стратегия развертывания приложений, которая позволяет постепенно внедрять в production новую версию приложения. Этот подход даёт возможность тестировать новые версии на небольшой части трафика, минимизируя риски и обеспечивая плавный переход. С помощью Canary deployment возможно переключение трафика на новую версию по мере уверенности в её стабильности, с возможностью быстрого отката на старую версию при возникновении проблем. В Deckhouse Kubernetes Platform Canary deployment может быть реализован средствами ingress-nginx или istio (рекомендуемый способ).

Примеры настроек Canary deployment средствами Ingress NGINX

Для реализации Canary deployment средствами Ingress NGINX используются аннотации и правила, которые определяют направление части трафика на новую версию приложения.

Создание Deployment и Service для стабильной версии

Пример манифеста для стабильной версии:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-v1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
      version: v1
  template:
    metadata:
      labels:
        app: my-app
        version: v1
    spec:
      containers:
        - name: app
          image: app:v1
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 80
```

Создание Deployment и Service для Canary-версии

Пример манифеста для Canary-версии:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
      version: v2
  template:
    metadata:
      labels:
        app: my-app
        version: v2
    spec:
      containers:
        - name: app
          image: app:v2
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-canary-service
spec:
  selector:
    app: my-app
    version: v2
  ports:
    - port: 80
      targetPort: 80

```

Настройка Ingress для Canary deployment

Для реализации Canary deployment с использованием Ingress NGINX используются специальные аннотации:

- `nginx.ingress.kubernetes.io/canary` — включает Canary-режим для Ingress.
- `nginx.ingress.kubernetes.io/canary-weight` — указывает процент трафика, который будет направлен на Canary-версию.

Пример манифеста для Ingress (10% трафика будет направлено на Canary-версию (`app-canary-service`), 90% трафика — на стабильную версию (`app-service`)):


```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "10" # 10% трафика на Canary.
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-canary-service
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress-main
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app-service
            port:
              number: 80

```

Постепенное увеличение трафика на Canary-версию

Вы можете постепенно увеличивать процент трафика на Canary-версию, изменяя значение аннотации `nginx.ingress.kubernetes.io/canary-weight`. Например, чтобы направить 50% трафика на Canary-версию, обновите аннотацию следующим образом:

```
nginx.ingress.kubernetes.io/canary-weight: "50"
```

Откат или завершение Canary deployment

Если Canary-версия работает стабильно, вы можете полностью переключить трафик на новую версию, удалив Canary-аннотации и обновив основной Ingress. Если возникли проблемы, вы можете уменьшить процент трафика на Canary-версию или полностью отключить ее, установив `nginx.ingress.kubernetes.io/canary-weight: "0"`.

Дополнительные аннотации для Canary deployment

- `nginx.ingress.kubernetes.io/canary-by-header` — направляет трафик на Canary-версию на основе значения HTTP-заголовка.
- `nginx.ingress.kubernetes.io/canary-by-cookie` — направляет трафик на Canary-версию на основе значения cookie.

Пример использования заголовка:

```
nginx.ingress.kubernetes.io/canary-by-header: "canary"
nginx.ingress.kubernetes.io/canary-by-header-value: "true"
```

В этом случае трафик будет направлен на Canary-версию, если запрос содержит заголовок `canary: true`.

Примеры настроек Canary deployment средствами Istio

i 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 пространстве имён размещены 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'ы);
- 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'ы доступны только при обращении к хосту через VirtualService из пода под
  # управлением Istio.
  # Эти subset'ы 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

```

Дополнительные ресурсы

- Документация модуля `ingress-nginx`
- Документация модуля `istio`

Настройка сканирования контейнерных образов на уязвимости

Deckhouse Kubernetes Platform (DKP) соответствует рекомендациям CIS Kubernetes Benchmark, обеспечивая безопасность как на уровне отдельных компонентов, так и всей платформы в целом.

Для постоянного мониторинга соответствия требованиям CIS в каждом кластере DKP автоматически выполняются проверки. Результаты этих проверок представлены в виде отчётов и доступны в Grafana на дашборде `Security / CIS Kubernetes Benchmark`.

DKP предоставляет встроенное средство для автоматического поиска уязвимостей в контейнерных образах на основе проекта Trivy.

Ниже приведены команды для просмотра и фильтрации отчётов о результатах сканирования уязвимостей и CIS compliance-проверок, выполненных в кластере.

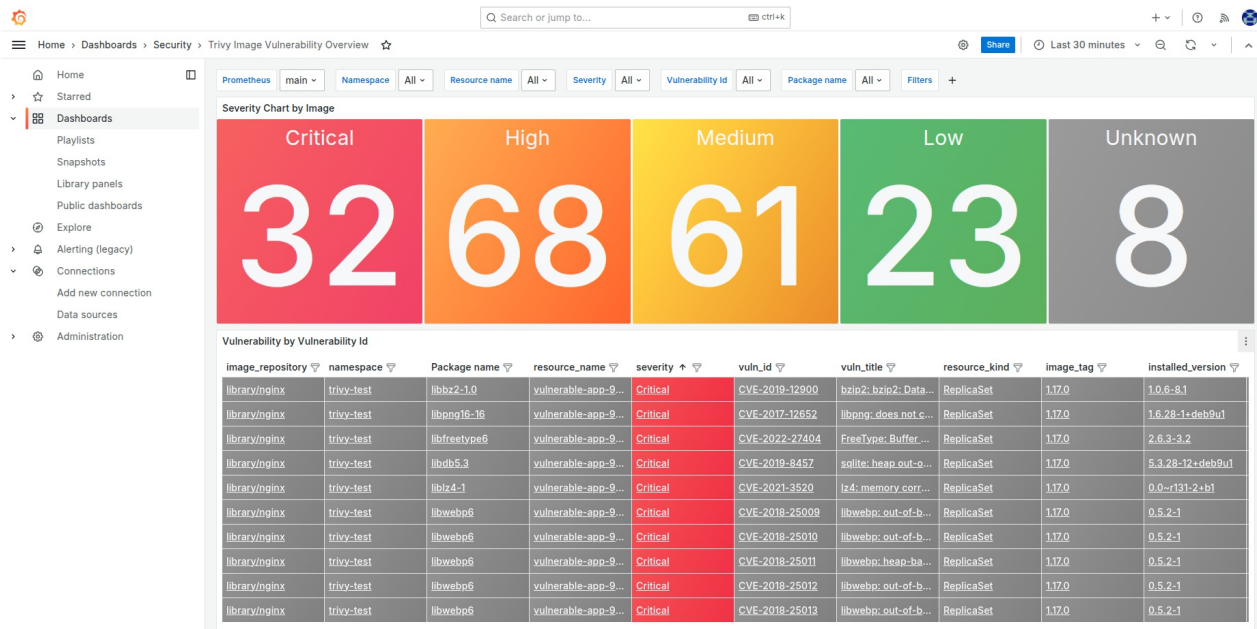
Доступ к результатам сканирования

Доступ к результатам сканирования, в том числе возможность просматривать ресурсы с результатами предоставляется пользователям, обладающим следующими ролями доступа:

- `d8:manage:networking:viewer` или выше;
- `d8:manage:permission:module:operator-trivy:view`.

Просмотр отчёта сканирования своего приложения

Для просмотра результатов сканирования вашего приложения воспользуйтесь Grafana-дашбордом `Security / Trivy Image Vulnerability Overview`. Вы можете отфильтровать результаты по нужному пространству имён и ресурсу.



Просмотр результатов CIS compliance-проверки

- Чтобы вывести все ресурсы, которые не прошли проверку:

```
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq '.status.detailReport.results | map(select(.checks | map(.success) | all | not))'
```

- Чтобы выполнить поиск по идентификатору конкретной проверки:

```
check_id="5.7.3"
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq --arg check_id "$check_id" '.status.detailReport.results | map(select(.id == $check_id))'
```

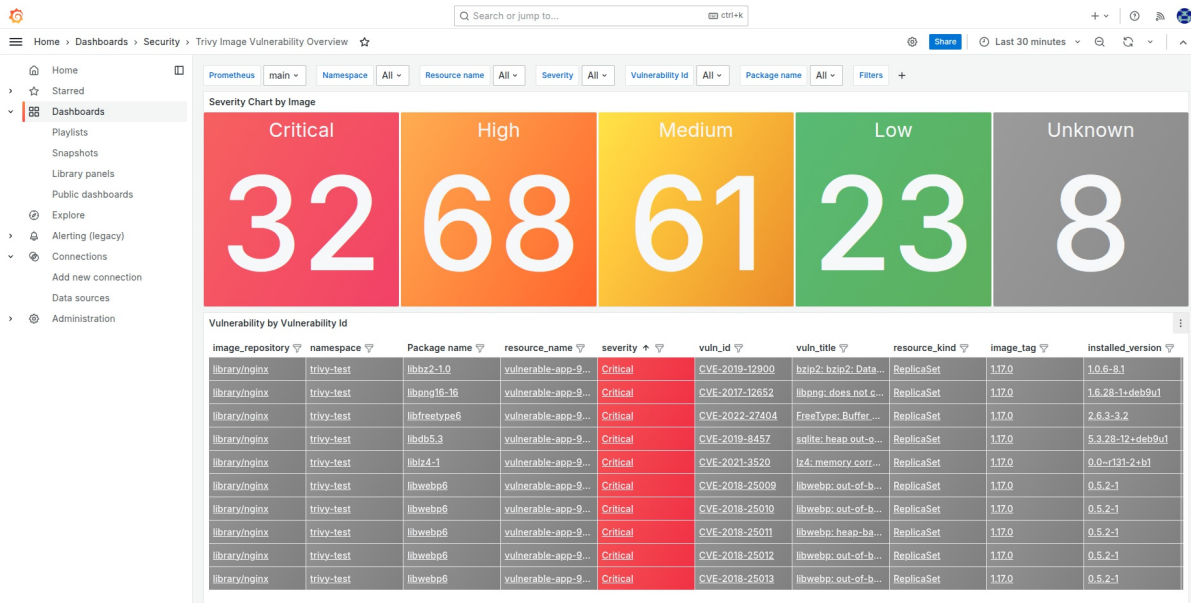
- Чтобы выполнить поиск по описанию проверки:

```
check_desc="Apply Security Context to Your Pods and Containers"
d8 k get clustercompliancereports.aquasecurity.github.io cis -ojson |
jq --arg check_desc "$check_desc" '.status.detailReport.results |
map(select(.description == $check_desc))'
```

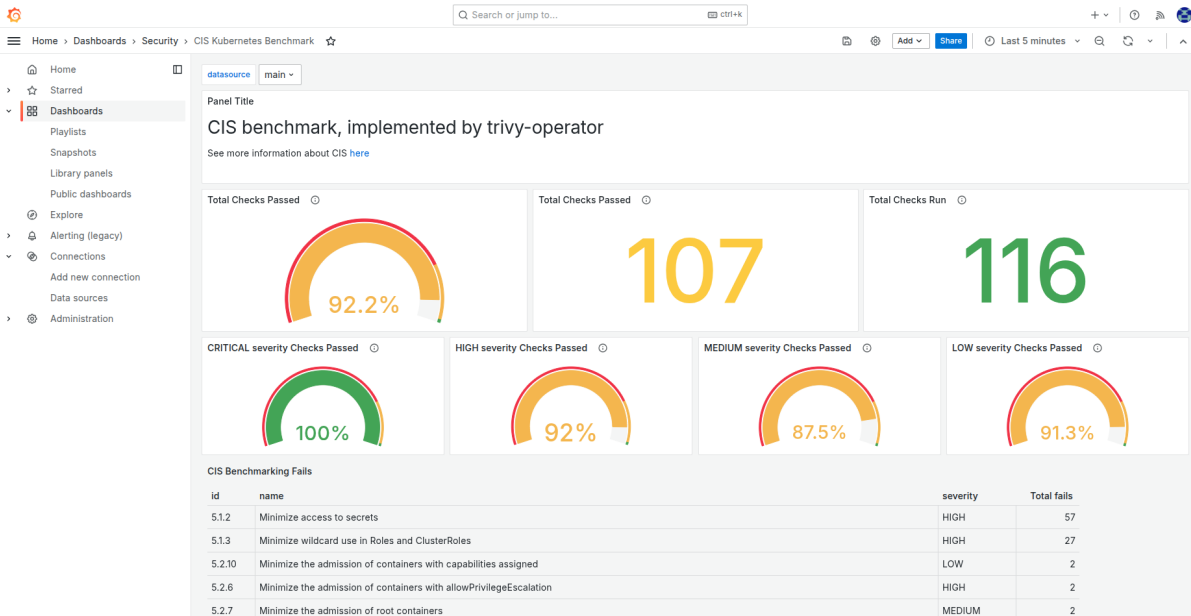
Просмотр результатов сканирования

Дашборды в Grafana:

- `Security/Trivy Image Vulnerability Overview` — сводный обзор уязвимостей в образах контейнеров, запущенных в кластере.



- **Security/CIS Kubernetes Benchmark** — результаты проверки соответствия кластера требованиям CIS Kubernetes Benchmark.



В кластере:

- Отчеты о безопасности кластера:
 - ClusterComplianceReport
 - RbacAssessmentReport
- Отчеты о безопасности нагрузки в кластере:
 - VulnerabilityReport — уязвимости в образах контейнеров;
 - SbomReport — состав ПО в образах (SBOM);
 - ConfigAuditReport — ошибки конфигурации Kubernetes-объектов;
 - ExposedSecretReport — утечки секретов в контейнерах.

DKP использует набор кастомных ресурсов, разработанных проектом Trivy Operator от Aqua Security, для представления результатов сканирования уязвимостей, анализа конфигурации и проверки соответствия кластера требованиям безопасности.

Ниже приведено описание ключевых CRD, создаваемых operator-trivy, с примерами и ссылками на официальную документацию.

Описание ресурсов с результатами сканирования

Безопасность на уровне ресурсов

VulnerabilityReport

VulnerabilityReport — ресурс, содержащий отчёт об уязвимостях, обнаруженных в контейнерном

образе, используемом в рабочей нагрузке Kubernetes.

Отчёт включает список известных уязвимостей в пакетах операционной системы и приложениях, сгруппированных по уровням серьёзности (Critical, High, Medium и т.д.).

Для каждого контейнера в многоконтейнерной нагрузке operator-trivy создаёт отдельный ресурс VulnerabilityReport в пространстве имён соответствующей нагрузки.

Связь с ресурсом Kubernetes устанавливается через поле `ownerReference`.

Имена ресурсов формируются по шаблону: `<тип рабочей нагрузки>-<имя рабочей нагрузки>-<имя контейнера>`.

Пример:

```

apiVersion: aquasecurity.github.io/v1alpha1
kind: VulnerabilityReport
metadata:
  name: replicaset-nginx-6d4cf56db6-nginx
  namespace: default
  labels:
    trivy-operator.container.name: nginx
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: nginx-6d4cf56db6
    trivy-operator.resource.namespace: default
    resource-spec-hash: 7cb64cb677
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: nginx-6d4cf56db6
      uid: aa345200-cf24-443a-8f11-ddb438ff8659
      controller: true
      blockOwnerDeletion: false
report:
  artifact:
    repository: library/nginx
    tag: '1.16'
  os:
    family: debian
    name: '10.3'
  registry:
    server: index.docker.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.35.0
  summary:
    criticalCount: 2
    highCount: 0
    mediumCount: 0
    lowCount: 0
    unknownCount: 0
  vulnerabilities:
    - vulnerabilityID: CVE-2019-20367
      resource: libbsd0
      installedVersion: 0.9.1-2
      fixedVersion: 0.9.1-2+deb10u1
      severity: CRITICAL
      score: 9.1
      target: library/nginx:1.21.6
      primaryLink: https://avd.aquasec.com/nvd/cve-2019-20367
    - vulnerabilityID: CVE-2018-25009
      resource: libwebp6
      installedVersion: 0.6.1-2
      fixedVersion: ''
      severity: CRITICAL
      score: 9.1
      target: library/nginx:1.16
      title: 'libwebp: out-of-bounds read in WebPMuxCreateInternal'
      primaryLink: https://avd.aquasec.com/nvd/cve-2018-25009

```

ConfigAuditReport

ConfigAuditReport — ресурс, содержащий результаты проверки конфигурации Kubernetes-объекта с помощью инструментов аудита, таких как Trivy.

Отчёт включает список замечаний к конфигурации, сгруппированных по категориям (например, Security) и уровням серьёзности (Critical, High и т.д.).

Примеры проверок:

- запуск контейнера от имени пользователя без прав root;
- наличие у контейнера лимитов ресурсов;
- конфигурация сетевого доступа (hostNetwork, hostPID и др.);
- наличие флагов безопасности, запрещающих эскалацию привилегий.

ConfigAuditReport создаётся для любого ресурса в пространстве имён, включая:

- рабочие нагрузки (Pod, Deployment, StatefulSet и др.);
- вспомогательные ресурсы (Service, ConfigMap, Role, RoleBinding и др.).

Каждый отчёт связан с проверяемым объектом через поле `ownerReference` и сохраняется в том же пространстве имён.

Имена ресурсов формируются по шаблону: `<тип рабочей нагрузки>-<имя рабочей нагрузки>`.

Пример:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: ConfigAuditReport
metadata:
  name: replicaset-nginx-6d4cf56db6
  namespace: default
  labels:
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: nginx-6d4cf56db6
    trivy-operator.resource.namespace: default
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: nginx-6d4cf56db6
report:
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: '0.22.0'
  summary:
    criticalCount: 2
    highCount: 0
    mediumCount: 0
    lowCount: 9
  checks:
    - checkID: hostPIDSet
      severity: CRITICAL
      messages: ["Host PID is not configured"]
      success: true
    - checkID: notReadOnlyRootFilesystem
      severity: LOW
      messages: ["Filesystem should be read only"]
      success: false
      scope:
        type: Container
        value: nginx
```

ExposedSecretReport

ExposedSecretReport — ресурс, содержащий отчёт с результатами проверки потенциальных секретов, обнаруженных в контейнерном образе, используемом в рабочей нагрузке Kubernetes.

В отчёте перечислены строки, содержащие чувствительные данные (например, токены, ключи, пароли), найденные в файлах внутри образа. Каждое обнаружение сопровождается категорией, правилом, уровнем серьёзности и указанием пути к файлу.

Для каждого контейнера в многоконтейнерной рабочей нагрузке operator-trivy создаёт отдельный ExposedSecretReport в пространстве имён рабочей нагрузки. Связь с ресурсом Kubernetes осуществляется через `ownerReference`.

Именованное ресурса соответствует шаблону: `<тип рабочей нагрузки>-<имя рабочей нагрузки>-<имя контейнера>`.

Пример:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: ExposedSecretReport
metadata:
  name: replicaset-app-67b77f5965-app
  namespace: default
  labels:
    trivy-operator.container.name: app
    trivy-operator.resource.kind: ReplicaSet
    trivy-operator.resource.name: app-67b77f5965
    trivy-operator.resource.namespace: default
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: app-67b77f5965
report:
  artifact:
    repository: myimagewithsecret
    tag: v0.22.0
  registry:
    server: index.docker.io
  scanner:
    name: Trivy
    vendor: Aqua Security
    version: 0.35.0
  secrets:
    - category: Stripe
      ruleID: stripe-access-token
      severity: HIGH
      target: "/app/config/secret.yaml"
      match: "publishable_key: *****"
      title: Stripe
    - category: Stripe
      ruleID: stripe-access-token
      severity: HIGH
      target: "/app/config/secret.yaml"
      match: "secret_key: *****"
      title: Stripe
  summary:
    criticalCount: 0
    highCount: 2
    mediumCount: 0
    lowCount: 0
  updateTimestamp: "2022-06-29T14:29:37Z"
```

SbomReport

SbomReport — ресурс, содержащий отчёт Software Bill of Materials (SBOM) для контейнерного образа, используемого в рабочей нагрузке Kubernetes.

Он представляет собой перечень всех компонентов программного обеспечения, включая системные пакеты и зависимости приложений, найденные в контейнере. Такая информация полезна для анализа состава образа, аудита безопасности и соответствия требованиям поставщиков.

Для многоконтейнерной рабочей нагрузки operator-trivy создаёт отдельный ресурс SbomReport

для каждого контейнера.

Отчёт создаётся в том же пространстве имён, где размещена рабочая нагрузка, и связан с ней через `ownerReference`.

Именование ресурса следует шаблону: `<тип рабочей нагрузки>-<имя рабочей нагрузки>-<имя контейнера>`.

Пример:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: SbmReport
metadata:
  creationTimestamp: "2023-07-10T09:37:21Z"
  generation: 1
  labels:
    resource-spec-hash: 796669cd5d
    trivy-operator.container.name: kube-apiserver
    trivy-operator.resource.kind: Pod
    trivy-operator.resource.name: kube-apiserver-kind-control-plane
    trivy-operator.resource.namespace: kube-system
  name: pod-kube-apiserver-kind-control-plane-kube-apiserver
  namespace: kube-system
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: false
    controller: true
    kind: Pod
    name: kube-apiserver-kind-control-plane
    uid: 732b4aa7-91f8-40a3-8b21-9627a98a910b
  resourceVersion: "6148"
  uid: 2a5000fe-b97e-46d0-9de7-62fb5fbc6555
report:
  artifact:
    repository: kube-apiserver
    tag: v1.21.1
  components:
    bomFormat: CycloneDX
    components:
    - bom-ref: 9464f5f9-750d-4ea0-8705-c8d067b25b29
      name: debian
      properties:
      - name: aquasecurity:trivy:Class
        value: os-pkgs
      - name: aquasecurity:trivy:Type
        value: debian
      supplier: {}
      type: operating-system
      version: "10.9"
    - bom-ref: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
      licenses:
      - expression: GPL-3.0
        license: {}
      name: base-files
      properties:
      - name: aquasecurity:trivy:LayerDiffID
        value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
      - name: aquasecurity:trivy:LayerDigest
        value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
      - name: aquasecurity:trivy:PkgID
        value: base-files@10.3+deb10u9
      - name: aquasecurity:trivy:PkgType
        value: debian
      - name: aquasecurity:trivy:SrcName
```

```

    value: base-files
  - name: aquasecurity:trivy:SrcVersion
    value: 10.3+deb10u9
  purl: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
  supplier:
    name: Santiago Vila <sanvila@debian.org>
  type: library
  version: 10.3+deb10u9
- bom-ref: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  licenses:
  - expression: GPL-2.0
    license: {}
  name: netbase
  properties:
  - name: aquasecurity:trivy:LayerDiffID
    value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
  - name: aquasecurity:trivy:LayerDigest
    value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
  - name: aquasecurity:trivy:PkgID
    value: netbase@5.6
  - name: aquasecurity:trivy:PkgType
    value: debian
  - name: aquasecurity:trivy:SrcName
    value: netbase
  - name: aquasecurity:trivy:SrcVersion
    value: "5.6"
  purl: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  supplier:
    name: Marco d'Itri <md@linux.it>
  type: library
  version: "5.6"
- bom-ref: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  name: tzdata
  properties:
  - name: aquasecurity:trivy:LayerDiffID
    value: sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b
  - name: aquasecurity:trivy:LayerDigest
    value: sha256:5dea5ec2316d4a067b946b15c3c4f140b4f2ad607e73e9bc41b673ee5ebb99a3
  - name: aquasecurity:trivy:PkgID
    value: tzdata@2021a-0+deb10u1
  - name: aquasecurity:trivy:PkgType
    value: debian
  - name: aquasecurity:trivy:SrcName
    value: tzdata
  - name: aquasecurity:trivy:SrcRelease
    value: 0+deb10u1
  - name: aquasecurity:trivy:SrcVersion
    value: 2021a
  purl: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  supplier:
    name: GNU Libc Maintainers <debian-glibc@lists.debian.org>
  type: library
  version: 2021a-0+deb10u1
dependencies:
- dependsOn:
  - pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
  - pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  - pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  ref: 9464f5f9-750d-4ea0-8705-c8d067b25b29
- dependsOn: []
  ref: pkg:deb/debian/base-files@10.3+deb10u9?arch=amd64&distro=debian-10.9
- dependsOn: []

```

```

    ref: pkg:deb/debian/netbase@5.6?arch=all&distro=debian-10.9
  - dependsOn: []
    ref: pkg:deb/debian/tzdata@2021a-0+deb10u1?arch=all&distro=debian-10.9
  - dependsOn:
    - 9464f5f9-750d-4ea0-8705-c8d067b25b29
    ref: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
  metadata:
    component:
      bom-ref: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
      name: k8s.gcr.io/kube-apiserver:v1.21.1
      properties:
        - name: aquasecurity:trivy:DiffID
          value:
sha256:417cb9b79adeec55f58b890dc9831e252e3523d8de5fd28b4ee2abb151b7dc8b, sha256:b5013
1762317bbe47def2d426d5c78a353a08b966d36bed4a04aee99dde4e12b, sha256:1e6ed7621dee7e03d
d779486ed469a65af6fb13071d13bd3a89c079683e3b1f0
        - name: aquasecurity:trivy:ImageID
          value:
sha256:771ffcf9ca634e37cbd3202fd86bd7e2df48ecba4067d1992541bfa00e88a9bb
        - name: aquasecurity:trivy:RepoDigest
          value: k8s.gcr.io/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9
        - name: aquasecurity:trivy:RepoTag
          value: k8s.gcr.io/kube-apiserver:v1.21.1
        - name: aquasecurity:trivy:SchemaVersion
          value: "2"
      purl: pkg:oci/kube-
apiserver@sha256:53a13cd1588391888c5a8ac4cef13d3ee6d229cd904038936731af7131d193a9?
repository_url=k8s.gcr.io%2Fkube-apiserver&arch=amd64
      supplier: {}
      type: container
      timestamp: "2023-07-10T09:37:21+00:00"
      tools:
        - name: trivy
          vendor: aquasecurity
      serialNumber: urn:uuid:50dbce86-28c5-4caf-9d08-a4aadf23233e
      specVersion: 1.4
      version: 1
    registry:
      server: k8s.gcr.io
    scanner:
      name: Trivy
      vendor: Aqua Security
      version: 0.52.2
    summary:
      componentsCount: 5
      dependenciesCount: 5
      updateTimestamp: "2023-07-10T09:37:21Z"

```

Безопасность на уровне кластера

RbacAssessmentReport

RbacAssessmentReport — ресурс, содержащий отчёт, созданный на основе анализа настроек RBAC (Role-Based Access Control) в кластере Kubernetes.

Он содержит результаты проверок, выполненных средствами аудита конфигурации, таких как Trivy.

Примеры проверок включают в себя выявление ролей, которые:

- предоставляют чрезмерные привилегии (например, полный доступ к секретам всех групп API);
- нарушают принципы минимально необходимого доступа.

Каждый отчёт связан с конкретной ролью (Role или ClusterRole) и размещается в том же пространстве имён, что и проверяемый объект.

Именование ресурса следует шаблону: `<Role|ClusterRole>-<имя роли>`.

Пример:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: RbacAssessmentReport
metadata:
  name: role-868458b9d6
  namespace: kube-system
report:
  checks:
    - category: Kubernetes Security Check
      checkID: KSV051
      description: Check whether role permits creating role bindings and associating
        to privileged role/clusterrole
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow role binding creation and association with privileged
role/clusterrole
    - category: Kubernetes Security Check
      checkID: KSV056
      description: The ability to control which pods get service traffic directed to
        them allows for interception attacks. Controlling network policy allows for
        bypassing lateral movement restrictions.
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow management of networking resources
    - category: Kubernetes Security Check
      checkID: KSV041
      description: Check whether role permits managing secrets
      messages:
        - Role permits management of secret(s)
      severity: CRITICAL
      success: false
      title: Do not allow management of secrets
    - category: Kubernetes Security Check
      checkID: KSV047
      description: Check whether role permits privilege escalation from node proxy
      messages:
        - ""
      severity: HIGH
      success: true
      title: Do not allow privilege escalation from node proxy
    - category: Kubernetes Security Check
      checkID: KSV045
      description: Check whether role permits wildcard verb on specific resources
      messages:
        - ""
      severity: CRITICAL
      success: true
      title: No wildcard verb roles
```

```

- category: Kubernetes Security Check
  checkID: KSV054
  description: Check whether role permits attaching to shell on pods
  messages:
    - ""
  severity: HIGH
  success: true
  title: Do not allow attaching to shell on pods
- category: Kubernetes Security Check
  checkID: KSV044
  description: Check whether role permits wildcard verb on wildcard resource
  messages:
    - ""
  severity: CRITICAL
  success: true
  title: No wildcard verb and resource roles
- category: Kubernetes Security Check
  checkID: KSV050
  description: An effective level of access equivalent to cluster-admin should
not
  be provided.
  messages:
    - ""
  severity: CRITICAL
  success: true
  title: Do not allow management of RBAC resources
- category: Kubernetes Security Check
  checkID: KSV046
  description: Check whether role permits specific verb on wildcard resources
  messages:
    - ""
  severity: CRITICAL
  success: true
  title: No wildcard resource roles
- category: Kubernetes Security Check
  checkID: KSV055
  description: Check whether role permits allowing users in a rolebinding to add
  other users to their rolebindings
  messages:
    - ""
  severity: LOW
  success: true
  title: Do not allow users in a rolebinding to add other users to their
rolebindings
- category: Kubernetes Security Check
  checkID: KSV052
  description: Check whether role permits creating role ClusterRoleBindings and
  association with privileged cluster role
  messages:
    - ""
  severity: HIGH
  success: true
  title: Do not allow role to create ClusterRoleBindings and association with
privileged
  role
- category: Kubernetes Security Check
  checkID: KSV053
  description: Check whether role permits getting shell on pods
  messages:
    - ""
  severity: HIGH
  success: true

```

```

    title: Do not allow getting shell on pods
  - category: Kubernetes Security Check
    checkID: KSV042
    description: Used to cover attacker's tracks, but most clusters ship logs
quickly
    off-cluster.
  messages:
    - ""
  severity: MEDIUM
  success: true
  title: Do not allow deletion of pod logs
  - category: Kubernetes Security Check
    checkID: KSV049
    description: Some workloads leverage configmaps to store sensitive data or
configuration
    parameters that affect runtime behavior that can be modified by an attacker
    or combined with another issue to potentially lead to compromise.
  messages:
    - ""
  severity: MEDIUM
  success: true
  title: Do not allow management of configmaps
  - category: Kubernetes Security Check
    checkID: KSV043
    description: Check whether role permits impersonating privileged groups
  messages:
    - ""
  severity: CRITICAL
  success: true
  title: Do not allow impersonation of privileged groups
  - category: Kubernetes Security Check
    checkID: KSV048
    description: Check whether role permits update/create of a malicious pod
  messages:
    - ""
  severity: HIGH
  success: true
  title: Do not allow update/create of a malicious pod
scanner:
  name: Trivy
  vendor: Aqua Security
  version: '0.22.0'
summary:
  criticalCount: 1
  highCount: 0
  lowCount: 0
  mediumCount: 0
updateTimestamp: null

```

ClusterComplianceReport

ClusterComplianceReport — ресурс, содержащий сводный отчёт о соответствии кластера требованиям информационной безопасности.

На текущий момент поддерживается проверка на соответствие CIS Kubernetes Benchmark — набора рекомендаций по безопасной настройке компонентов Kubernetes.

Структура отчёта:

- `spec.compliance.controls` — описание проверяемых критериев.
- `status` — результаты выполнения проверок, соответствующих описанию в `spec`. Результаты формируются на основе отчётов, собранных от различных сканеров безопасности.



Ознакомиться с результатами отчёта можно также в Grafana-дашборде `Security / CIS Kubernetes Benchmark`.

Пример:

```
apiVersion: aquasecurity.github.io/v1alpha1
kind: ClusterComplianceReport
metadata:
  name: cis
spec:
  compliance:
    controls:
      - checks:
          - id: AVD-KCV-0048
            commands:
              - id: CMD-0001
                description: Ensure that the API server pod specification file has permissions
                  of 600 or more restrictive
                id: 1.1.1
                name: Ensure that the API server pod specification file permissions are set
                  to 600 or more restrictive
                severity: HIGH
            - checks:
                - id: AVD-KCV-0049
                  commands:
                    - id: CMD-0002
                      description: Ensure that the API server pod specification file ownership is
                        set to root:root
                      id: 1.1.2
                      name: Ensure that the API server pod specification file ownership is set to
                        root:root
                      severity: HIGH
          ...
        summary:
          failCount: 9
          passCount: 107
          updateTimestamp: "2025-07-29T06:00:00Z"
```

Дополнительные ресурсы

- Документация модуля `operator-trivy`

Настройка использования TLS-сертификатов

Deckhouse Kubernetes Platform (DKP) предоставляет встроенные средства управления TLS-сертификатами, упрощающие настройку и управление шифрованием трафика в приложениях, работающих в кластере.

На этой странице описаны следующие аспекты использования сертификатов в DKP:

- как вручную заказывать TLS-сертификаты с помощью ресурсов `Certificate` и `ClusterIssuer`;
- как безопасно хранить и использовать учётные данные для доступа к удостоверяющим центрам (CA);
- как автоматически получать сертификаты с помощью аннотации `tls-acme` в ресурсах `Ingress`.



Общее описание порядка управления сертификатами в DKP, список поддерживаемых издателей, а также рекомендации по их настройке приведены на странице «Управление сертификатами».

Работа с сертификатами

Получение информации о сертификатах

- Чтобы вывести список всех сертификатов в кластере, используйте следующую команду:

```
d8 k get certificate --all-namespaces
```

- Чтобы проверить статус конкретного сертификата, воспользуйтесь следующей командой:

```
d8 k -n <NAMESPACE> describe certificate <CERTIFICATE-NAME>
```

Автоматический заказ сертификата

Чтобы заказать выпуск сертификата `letsencrypt`, выполните следующие шаги:

1. Создайте ресурс Certificate, опираясь на документацию `cert-manager`. Сверяйтесь с примером ниже:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com          # Имя сертификата.
  namespace: default
spec:
  secretName: example-com-tls # Имя Secret, в котором будет сохранён приватный
                              # ключ и сертификат.
  issuerRef:
    kind: ClusterIssuer      # Данные об издателе сертификата.
    name: letsencrypt
  commonName: example.com    # Основной домен сертификата.
  dnsNames:                  # Опциональные дополнительные домены сертификата (как
                              # минимум одно DNS-имя или IP-адрес).
    - www.example.com
    - admin.example.com
```

2. Модуль `cert-manager` автоматически запустит проверку владения доменом (challenge) с использованием метода, указанного в ресурсе ClusterIssuer — например, `HTTP-01` или `DNS-01`.
3. Модуль `cert-manager` автоматически создаст временный ресурс Ingress для проверки владения доменом. Временный ресурс не влияет на работу основного Ingress-ресурса.
4. После успешной проверки выпущенный сертификат будет сохранён в Secret, указанный в поле `secretName`.



Если в процессе заказа сертификата выводится ошибка `CAA record does not match issuer`, проверьте DNS-записи домена, для которого заказывается сертификат. Для использования сертификата `letsencrypt` у домена должна быть следующая CAA-запись: `issue "letsencrypt.org"`.

Подробнее про CAA-записи можно почитать в документации Let's Encrypt.

Заказ wildcard-сертификата с DNS в Cloudflare

1. Получите `GlobalAPIKey` и `Email`:
 - зайдите на страницу `dash.cloudflare.com/profile`;
 - ваша почта указана наверху под **Email Address**;
 - для просмотра API-ключа нажмите **View** напротив **Global API Key** внизу страницы.
2. Отредактируйте настройки модуля `cert-manager`, добавив следующую секцию:

```
settings:
  cloudflareGlobalAPIKey: APIkey
  cloudflareEmail: some@mail.somedomain
```

или указав API-токен вместо ключа (рекомендуемый вариант):

```
settings:
  cloudflareAPIToken: some-token
  cloudflareEmail: some@mail.somedomain
```

После этого DKP автоматически создаст ClusterIssuer и Secret для Cloudflare в пространстве имён `d8-cert-manager`.

3. Создайте ресурс Certificate с проверкой с помощью провайдера Cloudflare. Данная возможность появится только при указании настройки `cloudflareGlobalAPIKey` и `cloudflareEmail` в DKP:

```
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. Создайте ресурс 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 AWS Route53

1. Создайте пользователя с необходимыми правами:
 - зайдите на страницу управления политиками и создайте политику со следующими 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": "*"
    }
  ]
}
```

- о зайдите на страницу управления пользователями и добавьте пользователя с созданной ранее политикой.

2. Отредактируйте настройки модуля `cert-manager`, добавив следующую секцию:

```
settings:
  route53AccessKeyID: <ACCESS_KEY_ID>
  route53SecretAccessKey: <SECRET_ACCESS_KEY>
```

После этого Deckhouse автоматически создаст ClusterIssuer и Secret для Route53 в пространстве имён `d8-cert-manager`.

3. Создайте ресурс Certificate с проверкой с помощью провайдера Route53. Данная возможность появится только при указании настроек `route53AccessKeyID` и `route53SecretAccessKey` в DKP:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespase
spec:
  secretName: tls-wildcard
  issuerRef:
    name: route53
    kind: ClusterIssuer
  commonName: "*.domain.com"
  dnsNames:
    - "*.domain.com"
```

Заказ wildcard-сертификата с DNS в Google

1. Создайте ServiceAccount с необходимой ролью:

- о зайдите на страницу управления политиками;
- о выберите нужный проект и создайте ServiceAccount с желаемым названием, например `dns01-solver`;
- о зайдите в созданный ServiceAccount и создайте ключ, нажав на **Добавить ключ**. Будет скачан JSON-файл с данными ключа;
- о закодируйте полученный файл в строку формата Base64:

```
base64 project-209317-556c656b81c4.json
```

2. Сохраните полученную Base64-строку в параметре `cloudDNSServiceAccount`.

После этого Deckhouse автоматически создаст `ClusterIssuer` и `Secret` для CloudDNS в пространстве имён `d8-cert-manager`.

3. Создайте ресурс `Certificate` с валидацией через CloudDNS:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: domain-wildcard
  namespace: app-namespase
spec:
  secretName: tls-wildcard
  issuerRef:
    name: clouddns
    kind: ClusterIssuer
  dnsNames:
    - "*.domain.com"
```

Заказ самоподписанного сертификата

Чтобы заказать самоподписанный сертификат, укажите `selfsigned` в качестве имени издателя в поле `issuerRef.name`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-com          # Имя сертификата.
  namespace: default
spec:
  secretName: example-com-tls # Имя Secret, в котором будет сохранён приватный ключ
                              # и сертификат.
  issuerRef:
    kind: ClusterIssuer      # Данные об издателе сертификата.
    name: selfsigned
  commonName: example.com    # Основной домен сертификата.
  dnsNames:                  # Опциональные дополнительные домены сертификата (как
                              # минимум одно DNS-имя или IP-адрес).
    - www.example.com
    - admin.example.com
```

Создание самоподписанного сертификата

При самостоятельной генерации сертификатов важно корректно заполнить все поля запроса на сертификат, чтобы итоговый сертификат был правильно издан и гарантированно проходил валидацию в различных сервисах.

Важно придерживаться следующих правил:

1. Указывать доменные имена в поле `SAN` (Subject Alternative Name).

Поле `SAN` является более современным и распространенным методом указания доменных имен, на которые распространяется сертификат. Некоторые сервисы на данный момент уже не рассматривают поле `CN` (Common Name) как источник для доменных имен.

2. Корректно заполнять поля `keyUsage`, `basicConstraints`, `extendedKeyUsage`, а именно:

- `basicConstraints = CA:FALSE`

Данное поле определяет, относится ли сертификат к конечному пользователю (end-entity certificate) или к центру сертификации (CA certificate). CA-сертификат не может использоваться в качестве сертификата сервиса.

- `keyUsage = digitalSignature, keyEncipherment`

Поле `keyUsage` ограничивает допустимые сценарии использования данного ключа:

- `digitalSignature` — позволяет использовать ключ для подписи цифровых сообщений и обеспечения целостности соединения.
- `keyEncipherment` — позволяет использовать ключ для шифрования других ключей, что необходимо для безопасного обмена данными с помощью TLS (Transport Layer Security).

- `extendedKeyUsage = serverAuth`

Поле `extendedKeyUsage` уточняет дополнительные сценарии использования ключа, которые могут требоваться конкретными протоколами или приложениями:

- `serverAuth` — указывает, что сертификат предназначен для использования на сервере для аутентификации сервера перед клиентом в процессе установления защищенного соединения.

Также рекомендуется:

1. Издать сертификат на срок не более 1 года (365 дней).

Срок действия сертификата влияет на его безопасность. Срок в 1 год позволяет обеспечить актуальность криптографических методов и своевременно обновлять сертификаты в случае возникновения угроз. Также некоторые современные браузеры на текущий момент отвергают сертификаты со сроком действия более 1 года.

2. Использовать стойкие криптографические алгоритмы, например, алгоритмы на основе эллиптических кривых (в т.ч. `prime256v1`).

Алгоритмы на основе эллиптических кривых (ECC) предоставляют высокий уровень безопасности при меньшем размере ключа по сравнению с традиционными методами, такими как RSA. Это делает сертификаты более эффективными по производительности и безопасными в долгосрочной перспективе.

3. Не указывать домены в поле `CN` (Common Name).

Исторически поле `CN` использовалось для указания основного доменного имени, для которого выдается сертификат. Однако современные стандарты, такие как RFC 2818, рекомендуют использовать поле `SAN` (Subject Alternative Name) для этой цели. Если сертификат распространяется на несколько доменных имен, указанных в поле `SAN`, то при дополнительном указании одного из доменов в `CN` в некоторых сервисах может возникнуть ошибка валидации при обращении к домену, не указанному в `CN`. Если указывать в `CN` информацию, не относящуюся напрямую к доменным именам (например, идентификатор или имя сервиса), то сертификат также будет распространяться на эти имена, что может быть использовано для вредоносных целей.

Пример создания сертификата

Для генерации сертификата воспользуемся утилитой `openssl`.

1. Заполните конфигурационный файл `cert.cnf`:

```
[ req ]
default_bits      = 2048
default_md        = sha256
prompt           = no
distinguished_name = dn
req_extensions    = req_ext

[ dn ]
C = RU
ST = Moscow
L = Moscow
O = Example Company
OU = IT Department
# CN = Не указывайте поле CN.

[ req_ext ]
subjectAltName = @alt_names

[ alt_names ]
# Укажите все доменные имена.
DNS.1 = example.com
DNS.2 = www.example.com
DNS.3 = api.example.com
# Укажите IP-адреса (если требуется).
IP.1 = 192.0.2.1
IP.2 = 192.0.4.1

[ v3_ca ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

# Параметры эллиптических кривых.
[ ec_params ]
name = prime256v1
```

2. Сгенерируйте ключ на основе эллиптических кривых:

```
openssl ecparam -genkey -name prime256v1 -noout -out ec_private_key.pem
```

3. Создайте запрос на сертификат:

```
openssl req -new -key ec_private_key.pem -out example.csr -config cert.cnf
```

4. Сгенерируйте самоподписанный сертификат:

```
openssl x509 -req -in example.csr -signkey ec_private_key.pem -out example.crt -days 365 -
extensions v3_req -extfile cert.cnf
```

Защита учётных данных

Если вы не хотите хранить учётные данные в конфигурации DKP, можно создать отдельный Secret и ссылаться на него в ресурсе ClusterIssuer.

Для этого выполните следующее:

1. Создайте Secret с ключом доступа:

```
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
```

2. Создайте ресурс 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
```

3. 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
```

Поддержка аннотации tls-acme

DKP поддерживает аннотацию `kubernetes.io/tls-acme: "true"` в ресурсах Ingress. Компонент `cert-manager-ingress-shim` следит за появлением аннотации и автоматически создаёт ресурсы Certificate в тех же пространствах имён, что и Ingress-ресурсы.



При использовании аннотации ресурс Certificate создается в связке с существующим Ingress-ресурсом. Для подтверждения владения доменом (challenge) не создаётся отдельный Ingress, а вносятся дополнительные записи в существующий. Следовательно, если на основном ресурсе Ingress настроена аутентификация или whitelist, попытка подтверждения окончится неудачей. По этой причине вместо аннотации рекомендуется использовать ресурс Certificate напрямую.

При переходе с аннотации на Certificate удалите ресурс Certificate, который был автоматически создан с аннотацией. Иначе по обоим ресурсам Certificate будет обновляться один Secret, что может привести превышению лимита запросов Let's Encrypt.

Пример конфигурации ресурса Ingress с аннотацией:

```
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                # Ещё 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                # Ещё один дополнительный домен.
      secretName: example-com-tls          # Имя для Certificate и Secret.
```

Дополнительные ресурсы

- Параметры модуля `cert-manager`

Параметры безопасности подов и контейнеров

В манифестах подов можно задавать ряд параметров, влияющих на безопасность контейнеров.

Большинство таких параметров настраивается в блоках `securityContext` и `containers[].securityContext`.

На данной странице рассмотрены основные параметры безопасности подов и контейнеров:

- `runAsUser`;
- `runAsNonRoot`;
- `runAsGroup`;
- `readOnlyRootFilesystem`;
- `fsGroup`;
- `fsGroupChangePolicy`;
- `appArmorProfile`;
- `seccompProfile`;
- `capabilities (add/drop)`;
- `allowPrivilegeEscalation`;
- `privileged`;
- `procMount`;
- `sysctls`;
- `hostNetwork`;
- `hostPID`;
- `hostIPC`;
- `hostPath`;
- `automountServiceAccountToken`.



В Deckhouse Kubernetes Platform (DKP) допустимые значения указанных параметров безопасности контролируются модулем `admission-policy-engine`.

runAsUser

Параметр `runAsUser` задаёт цифровой идентификатор пользователя (UID), от имени которого выполняются все процессы внутри контейнера.

Он напрямую управляет правами доступа процессов в Linux. С помощью этого параметра можно принудительно лишить приложение прав суперпользователя (`root`) на уровне ядра, даже если сам Docker-образ запускается с UID `0`.

В ядре Linux безопасность файловой системы и процессов строится на цифровых идентификаторах (UID), а не на текстовых именах. Суперпользователь `root` всегда имеет UID `0`. По умолчанию, если в `Dockerfile` не указана инструкция `USER`, контейнерный рантайм запускает приложение с UID `0`.

Например, если в контейнере работает веб-сервер от имени `root` и злоумышленник находит уязвимость (например, `Remote Code Execution`), он получает полный контроль над изолированной средой с UID `0`. Это значительно облегчает проведение атак на хостовую операционную систему.

При использовании `runAsUser` Kubernetes передаёт указанный UID контейнерному рантайму при создании процесса. При этом:

- **Игнорируются настройки образа.** Текстовые или цифровые пользователи, прописанные разработчиком в `Dockerfile` (через инструкцию `USER`), полностью переопределяются значением из манифеста.
- **Меняются права процессов.** Все создаваемые контейнером процессы получают указанный UID (например, `10001`). С этого момента они подчиняются стандартным правилам разграничения доступа Linux для обычных пользователей.

Запуск процессов с некорневым UID (любое число, отличное от `0`) является фундаментальным правилом безопасности (принцип наименьших привилегий). Если приложение скомпрометировано, злоумышленник с UID `10001` не сможет изменить системные файлы внутри контейнера, установить вредоносные пакеты через `apt/apk` или выполнить чувствительные системные вызовы к ядру хоста.



Если вы указываете случайный UID (например, `2000`), у этого пользователя внутри контейнера может не быть прав на чтение файлов самого приложения, если они были скопированы в образ с правами `root:root`. Разработчикам необходимо заранее подготавливать Docker-образы (выполнять `CHOWN` для рабочих директорий), чтобы приложение могло успешно запуститься под некорневым пользователем.

Расположение параметра

Параметр может задаваться как для всего пода (настройки унаследуют все контейнеры), так и индивидуально для конкретного контейнера. Значение, заданное для контейнера, имеет более высокий приоритет.

Параметр можно указать в одном из следующих полей:

- `spec.securityContext.runAsUser` — для всего пода;
- `spec.containers[].securityContext.runAsUser` — для конкретного контейнера.

Доступные значения параметра

Параметр принимает целое положительное число (`integer`), представляющее собой UID в Linux. Доступны следующие варианты:

- `0` — запуск от имени `root` (категорически не рекомендуется);
- `1000` и выше (до `65535` или `4294967295` в зависимости от архитектуры) — запуск от имени обычного пользователя (рекомендуется выбирать случайные высокие ID, например `10001`, чтобы исключить совпадения с системными UID на хосте).

Пример конфигурации

Принудительный запуск контейнера от имени некорневого пользователя с UID `10001`:

```
spec:
  containers:
  - name: secure-web-app
    image: my-app:latest
    securityContext:
      runAsUser: 10001
```

runAsNonRoot

Параметр `runAsNonRoot` определяет, обязан ли контейнер запускаться исключительно от имени некорневого пользователя (UID отличный от `0`).

Он включает встроенный механизм валидации на стороне агента `kubelet`. С помощью этого параметра Kubernetes проверяет манифест и сам Docker-образ перед стартом, полностью блокируя запуск контейнера, если итоговый UID процесса окажется `0`.

Обычно Kubernetes доверяет метаданным Docker-образа. Если разработчик забыл указать некорневого пользователя в `Dockerfile`, контейнер запустится с правами `root` (UID `0`). Параметр `runAsNonRoot` заставляет `kubelet` провести строгую инспекцию перед запуском контейнера.

Например, если в манифесте пода выставлен флаг `runAsNonRoot: true`, `kubelet` запрашивает у контейнерного рантайма информацию о пользователе из образа. Если там настроен `USER root` или вовсе нет этой инструкции (что означает UID `0`), Kubernetes прерывает запуск и переводит под в состояние ошибки.

При использовании `runAsNonRoot: true` агент `kubelet` анализирует итоговый UID, с которым должен запуститься процесс. При этом:

- **Проверяется связка параметров.** Если в манифесте указан `runAsUser: 0`, Kubernetes сразу заблокирует под;
- **Анализируется манифест образа.** Если в `securityContext` не указан конкретный `runAsUser`, `kubelet` проверяет UID внутри самого Docker-образа. Если там обнаруживается `0` (или текстовое имя `root`), под не запустится.
- **Выводится ошибка запуска.** Вместо работающего небезопасного контейнера вы получите статус `CreateContainerConfigError` или `ContainerCannotRun` со следующим описанием: `Container has runAsNonRoot and image will run as root`.

Этот параметр служит «подушкой безопасности» и главным рубежом защиты кластера от человеческого фактора. Он гарантирует, что в production-окружение физически не сможет попасть контейнер, работающий с правами суперпользователя. Это критически важно, так как компрометация `root`-контейнера открывает злоумышленнику прямой путь к эксплуатации уязвимостей ядра хоста и потенциальному «побегу из контейнера» (container escape).



Некоторые Docker-образы используют текстовые имена пользователей вместо цифровых UID (например, `USER nginx`). Если в образе не настроена таблица соответствия имён и ID (`/etc/passwd`), kubelet не сможет определить числовой UID на этапе подготовки и заблокирует контейнер, даже если этот пользователь не является `root`. Чтобы избежать этой проблемы, используйте `runAsNonRoot: true` совместно с явным указанием цифрового ID через `runAsUser`.

Расположение параметра

Параметр может задаваться как для всего пода (настройки унаследуют все контейнеры), так и индивидуально для конкретного контейнера:

- `spec.securityContext.runAsNonRoot` — для всего пода;
- `spec.containers[].securityContext.runAsNonRoot` — для конкретного контейнера.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — проверка на некорневого пользователя отключена (значение по умолчанию);
- `true` — запуск от имени `root` запрещён (рекомендуется для всех приложений).

Пример конфигурации

Пример конфигурации, которая блокирует запуск от имени `root` и принудительно назначает безопасный UID:

```
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 10001
  containers:
  - name: protected-app
    image: my-app:latest
```

runAsGroup

Параметр `runAsGroup` определяет цифровой идентификатор основной группы (GID), под которым будут запускаться все процессы внутри контейнера.

Он дополняет параметр `runAsUser` и управляет групповыми правами доступа процессов к файлам и устройствам Linux. С его помощью можно гибко разграничить права общего доступа для нескольких контейнеров или настроить безопасное взаимодействие с локальными дисками.

В Linux доступ к файлам распределяется на трёх уровнях: для владельца (`User`), для членов его основной группы (`Group`) и для всех остальных (`Others`). Группы идентифицируются цифровыми GID, где `0` — это группа суперпользователя `root`. По умолчанию, если параметр не задан, контейнерный рантайм присваивает процессам GID `0` или ту группу, которая жёстко прописана для пользователя в файле `/etc/passwd` внутри Docker-образа.

Например, если приложение создаёт логи или временные файлы, они по умолчанию записываются с группой `root` (GID `0`). Если к этой же директории нужно дать доступ соседнему контейнеру (например, агенту сбора логов), инженерам приходится давать файлам избыточные права `chmod 666` (чтение и запись для всех в системе), что нарушает правила безопасности.

При использовании `runAsGroup` Kubernetes передаёт указанный GID контейнерному рантайму, переопределяя стандартное поведение операционной системы. При этом:

- **Назначается основная группа.** Первичному процессу контейнера и всем его дочерним процессам принудительно присваивается указанный GID (например, `20002`).
- **Маркируются новые файлы.** Все новые файлы, директории или сокеты, которые контейнер создаст во время своей работы, автоматически получают этот GID в качестве

группы-владельца.

Фиксация некорневого GID (отличного от `0`) предотвращает случайный или намеренный доступ процессов контейнера к защищённым системным файлам группы `root`. Кроме того, это позволяет безопасно организовать совместную работу нескольких контейнеров с общими данными (через `emptyDir` или `PersistentVolume`) без выдачи опасных глобальных прав на чтение и запись для посторонних процессов.



Так же, как и в случае с `runAsUser`, указание случайного GID может привести к ошибкам вида `Permission denied` при старте, если бинарные файлы приложения внутри Docker-образа принадлежат исключительно группе `root` и закрыты для чтения остальным. Образ должен проектироваться с учётом того, что приложение будет работать под нестандартной группой.

Расположение параметра

Параметр может задаваться как для всего пода (настройки унаследуют все контейнеры), так и индивидуально для конкретного контейнера:

- `spec.securityContext.runAsGroup` — для всего пода;
- `spec.containers[].securityContext.runAsGroup` — для конкретного контейнера.

Доступные значения параметра

Параметр принимает целое положительное число (`integer`), представляющее собой GID в Linux. Доступны следующие варианты:

- `0` — назначение основной группы `root` (не рекомендуется);
- `1000` и выше (до `4294967295`) — назначение некорневой группы (рекомендуется выбирать высокие значения, например `10002`, согласованные с вашей политикой разграничения прав).

Пример конфигурации

Пример конфигурации пода, при которой процессы работают под некорневым пользователем и входят в выделенную безопасную группу:

```
spec:
  securityContext:
    runAsUser: 10001
    runAsGroup: 10002
  containers:
  - name: shared-data-app
    image: my-app:latest
```

readOnlyRootFilesystem

Параметр `readOnlyRootFilesystem` определяет, разрешено ли процессам внутри контейнера изменять или создавать файлы на его собственном базовом диске.

Он переводит неизменяемость контейнеров на аппаратный уровень ядра Linux. С помощью этого параметра можно заблокировать любые попытки записи в файловую систему контейнера, превращая его образ в статичную и защищенную среду.

Контейнеры создаются на основе слоев Docker-образа, поверх которых рантайм накладывает один верхний тонкий слой с правами на запись (`Read-Write layer`). По умолчанию приложение может свободно создавать файлы в `/tmp`, устанавливать утилиты или перезаписывать свои же конфигурационные файлы.

Например, если злоумышленник взламывает веб-приложение, его стандартный первый шаг — скачать вредоносный скрипт или вредоносную программу во временную директорию (например, `/tmp/malware.sh`), выдать ему права на исполнение и запустить. Если файловая система открыта на запись, ядро Linux позволит это сделать.

При использовании `readOnlyRootFilesystem: true` Kubernetes даёт команду контейнерному рантайму смонтировать корневой слой контейнера (`/`) в режиме `ro` (`read-only`). При этом:

- **Блокируется запись в корень.** Любая системная попытка создать, изменить или удалить

файл в любой директории контейнера (если это не специально смонтированный внешний диск) будет пресечена ядром.

- **Вызывается системная ошибка.** Приложение, попытавшееся записать лог или временный файл на базовый диск, мгновенно получит отказ операционной системы с ошибкой `Read-only file system`.

Этот параметр реализует концепцию неизменяемой инфраструктуры (Immutable Infrastructure) и сводит к нулю риски долгосрочного закрепления злоумышленника в контейнере. Даже если в приложении найдена критическая уязвимость (RCE), взломщик физически не сможет сохранить вредоносные файлы, подменить исполняемые файлы или модифицировать код. Это одна из самых эффективных практик защиты от современных автоматизированных атак.



Большинство современных приложений (а также системные библиотеки внутри образа) не могут запуститься, если им полностью запрещено писать временные данные (например, PID-файлы или логи). Чтобы приложение не падало с ошибкой при старте, все необходимые для записи пути (такие как `/tmp`, `/var/run`, `/cache`) необходимо точно смонтировать как временные RAM-диски с помощью механизмов `emptyDir` прямо в манифесте Kubernetes.

Расположение параметра

Параметр задаётся исключительно на уровне конкретного контейнера в поле `spec.containers[].securityContext.readOnlyRootFilesystem`.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — файловая система контейнера доступна на запись (значение по умолчанию);
- `true` — корневая файловая система заблокирована на запись (рекомендуется для максимальной безопасности).

Пример конфигурации

Пример конфигурации с переводом контейнера в режим `read-only` и безопасным выделением временной папки `/tmp` для легитимной записи:

```
spec:
  containers:
    - name: immutable-api
      image: my-app:latest
      securityContext:
        readOnlyRootFilesystem: true
      volumeMounts:
        - mountPath: /tmp
          name: temp-storage
      volumes:
        - name: temp-storage
          emptyDir: {}
```

fsGroup

Параметр `fsGroup` определяет цифровой идентификатор группы (GID) Linux, которой будут принудительно принадлежать все смонтированные к поду постоянные тома (`PersistentVolumes`).

Он управляет правами доступа к внешним дискам на уровне файловой системы хоста и контейнера. С его помощью Kubernetes автоматически решает проблему совместимости прав, позволяя некорневым процессам беспрепятственно читать и писать данные на смонтированные хранилища без использования избыточных прав администратора.

Когда к поду монтируется постоянный диск (`PersistentVolume`), файлы на нём сохраняют те UID и GID, с которыми они были изначально записаны (часто это `0:0`, то есть `root`). Если сам контейнер запущен от имени безопасного некорневого пользователя (например, `runAsUser: 10001`), ядро Linux заблокирует доступ к диску.

Например, база данных PostgreSQL запускается в контейнере под пользователем `postgres` (UID `999`). При монтировании сетевого диска процесс пытается инициализировать файлы в каталоге

данных, но получает ошибку `Permission denied`, так как каталог на диске принадлежит `root`. Инженерам приходится вручную менять права на диске, что неудобно и небезопасно.

При использовании `fsGroup` при старте пода Kubernetes выполняет специальную процедуру подготовки томов перед тем, как запустить основные контейнеры. При этом:

- **Меняется владелец на диске.** Kubernetes автоматически выполняет операцию, аналогичную `chown`, делая указанный `fsGroup` (например, `30003`) группой-владельцем всех файлов и директорий на смонтированном диске.
- **Устанавливается SGID-флаг.** На корневой каталог тома накладывается специальный флаг `set-group-ID`. Благодаря этому все новые файлы, которые контейнер создаст на этом диске в процессе работы, автоматически унаследуют `GID` из `fsGroup`, а не основную группу контейнера.
- **Добавляются права процессам.** Идентификатор `fsGroup` добавляется в список дополнительных (supplemental) групп для каждого контейнера в поде, предоставляя процессам легитимный доступ к файлам тома.

Параметр устраняет необходимость запускать контейнеры от имени `root` только ради того, чтобы они имели доступ к своим дискам. Он также избавляет от опасной практики выставления прав `chmod 777` на сетевых хранилищах, изолируя данные конкретного приложения на уровне выделенной группы Linux.



По умолчанию при каждом перезапуске пода Kubernetes рекурсивно обходит все файлы на смонтированном диске, чтобы проверить и изменить их `GID`. Если на диске базы данных хранится множество мелких файлов, этот процесс может занять долгое время, из-за чего под будет зависать в статусе `ContainerCreating`. Чтобы исправить это поведение, используйте `fsGroup` в паре с параметром `fsGroupChangePolicy: OnRootMismatch`.

Расположение параметра

Настройка задаётся в поле `spec.securityContext.fsGroup` исключительно на уровне всего пода, так как тома монтируются к поду целиком.

Доступные значения параметра

Параметр принимает целое положительное число (`integer`), представляющее собой `GID` в Linux. Доступны следующие варианты:

- `0` — использование группы `root` для томов (не рекомендуется);
- `1000` и выше (до `4294967295`) — выделенный идентификатор группы для совместного доступа к дискам (рекомендуется).

Пример конфигурации

Пример конфигурации пода, при которой некорневой контейнер получает автоматический безопасный доступ к смонтированному постоянному тому:

```
spec:
  securityContext:
    runAsUser: 10001
    fsGroup: 30003
  containers:
  - name: db-app
    image: my-db:latest
    volumeMounts:
    - mountPath: /var/lib/data
      name: storage
  volumes:
  - name: storage
    persistentVolumeClaim:
      claimName: db-pvc
```

supplementalGroups

Параметр `supplementalGroups` определяет список дополнительных цифровых идентификаторов групп (`GID`) Linux, которые будут принудительно добавлены к процессу контейнера в

дополнение к его основной группе.

В Linux каждый процесс всегда имеет одну основную группу (задаваемую через `runAsGroup`) и список дополнительных (supplemental) групп. Механизм дополнительных групп используется ядром для предоставления пользователю прав доступа к различным разделяемым системным ресурсам, общим каталогам или физическим устройствам хоста.

Список дополнительных групп формируется путем перечисления числовых идентификаторов. При запуске пода ядро Linux объединяет основную группу процесса и этот список, расширяя права доступа контейнера.

Пример конфигурации процесса внутри контейнера:

Параметр	Значение
Основной пользователь, UID (<code>runAsUser</code>)	10001
Основная группа, GID (<code>runAsGroup</code>)	10002
Дополнительные группы, GID (<code>supplementalGroups</code>)	[40001, 40002]

Если на смонтированном диске или внутри контейнера есть файлы, принадлежащие группе 40001, процесс с такой конфигурацией сможет беспрепятственно прочитать или изменить их (в зависимости от стандартных UNIX-прав файлов), поскольку он легитимно входит в состав этой группы.

Расположение параметра

Настройка задаётся исключительно на уровне всего пода в поле `spec.securityContext.supplementalGroups`, поскольку список дополнительных групп применяется ко всем контейнерам (включая `init`-контейнеры) внутри пода.

Доступные значения параметра

Параметр принимает список целых положительных чисел (`array` из `integer`), представляющих собой системные GID в Linux. Доступны следующие варианты:

- 0 — добавление группы `root` в качестве дополнительной (категорически не рекомендуется);
- 1–999 — системные группы Linux (например, группы для доступа к аудиоустройствам или логированию). Используйте с осторожностью, чтобы случайно не выдать лишние права на узле;
- 1000 и выше (до 4294967295) — пользовательские группы безопасности для совместного разграничения прав доступа к файловым хранилищам (рекомендуемый вариант).

Параметр `supplementalGroups` позволяет реализовать точечное разделение прав доступа к общим данным (например, к файлам в `emptyDir` или `PersistentVolume`) между разными подами без повышения привилегий до уровня `root`. Вместо выдачи избыточных прав на чтение и запись для всех пользователей системы (`chmod 777`), инженеры могут объединить несколько независимых сервисов в одну общую безопасную группу.



Обратите внимание:

- Взаимосвязь с `fsGroup`. Параметр `fsGroup` также добавляет указанный GID в список дополнительных групп процесса. Разница в том, что `fsGroup` дополнительно меняет владельца файлов на монтируемом диске и устанавливает SGID-флаг, а `supplementalGroups` просто дополняет права самого процесса, никак не модифицируя файловую систему томов при старте.
- Игнорирование текстовых имен. Kubernetes принимает только цифровые форматы ID групп. Настройки текстовых групп из файла `/etc/group` внутри Docker-образа полностью игнорируются.

Пример конфигурации

Пример конфигурации пода, процесс которого запускается под некорневым пользователем и получает доступ к двум разным разделяемым хранилищам через дополнительные группы 40001 и 40002:

```

apiVersion: v1
kind: Pod
metadata:
  name: shared-groups-pod
spec:
  securityContext:
    runAsUser: 10001
    runAsGroup: 10002
    supplementalGroups:
      - 40001 # Группа для доступа к общему хранилищу логов.
      - 40002 # Группа для работы с медиа-файлами.
  containers:
    - name: app
      image: nginx
      volumeMounts:
        - mountPath: /var/log/shared
          name: log-volume
  volumes:
    - name: log-volume
      persistentVolumeClaim:
        claimName: shared-logs-pvc

```

fsGroupChangePolicy

Параметр `fsGroupChangePolicy` определяет, каким образом Kubernetes будет проверять и изменять права владения (GID) для файлов на смонтированных дисках при запуске пода.

Он напрямую управляет поведением агента `kubelet` на этапе инициализации хранилищ. С помощью этого параметра можно оптимизировать время старта приложений, работающих с большими объемами данных, предотвращая длительные простои кластера.

Когда для пода настроен параметр `fsGroup`, Kubernetes должен гарантировать, что все файлы на подключенном томе принадлежат этой группе. По умолчанию `kubelet` при каждом запуске или перезапуске пода выполняет полную рекурсивную проверку и смену владельца (аналог команды `chown -R`) для каждого файла на диске.

Например, если вы запускаете базу данных (такую как PostgreSQL или Elasticsearch) с диском объемом в несколько терабайт, на котором лежит множество мелких файлов, системный обход диска агентом `kubelet` может занять 15–30 минут. Всё это время под будет находиться в состоянии `ContainerCreating`, блокируя работу приложения и нарушая метрики доступности (uptime).

При использовании `fsGroupChangePolicy` Kubernetes меняет алгоритм подготовки тома перед стартом контейнеров. При этом:

- **включается стандартный обход** (`Always`) — `kubelet` всегда рекурсивно проходит по всем файлам и меняет GID на ходу, независимо от того, правильные там права или нет;
- **включается умная проверка** (`OnRootMismatch`) — `kubelet` проверяет права только у самого корневого каталога смонтированного диска. Если его GID уже совпадает с указанным в `fsGroup`, рекурсивный обход терабайтов данных полностью пропускается, и контейнеры запускаются мгновенно. Если права не совпадают (например, диск подключен впервые), то права обновляются для всех вложенных файлов.

Параметр устраняет критическую проблему «зависания» инфраструктуры при масштабировании, обновлении подов или аварийном переключении узлов (failover). При этом сохраняется строгая изоляция данных: права доступа гарантированно будут скорректированы, если диск переехал из другого окружения или был создан со стандартными правами `root`.



Политика `OnRootMismatch` поддерживается не всеми типами хранилищ (хотя отлично работает со стандартными `PersistentVolumeClaims` на базе блочных и сетевых устройств, таких как AWS EBS, Ceph RBD или локальные диски). Кроме того, если файлы внутри тома были созданы сторонним процессом с разными GID вручную, умная проверка по корневой папке может не заметить скрытых расхождений в правах более глубоких подкаталогов.

Расположение параметра

Настройка задаётся в поле `spec.securityContext.fsGroupChangePolicy` исключительно на уровне всего пода, так как тома монтируются к поду целиком.

Доступные значения параметра

Параметр имеет строковый тип (`string`). Доступны следующие значения:

- `Always` — всегда рекурсивно изменять права на все файлы при каждом старте (значение по умолчанию);
- `OnRootMismatch` — изменять права рекурсивно только в том случае, если права корневой директории тома не совпадают с параметром `fsGroup` (рекомендуется для баз данных и больших хранилищ).

Пример конфигурации

Пример оптимизированной конфигурации для быстрого запуска пода с базой данных и терабайтным диском:

```
spec:
  securityContext:
    runAsUser: 10001
    fsGroup: 30003
    fsGroupChangePolicy: OnRootMismatch # Пропускать проверку, если корень диска уже настроен.
  containers:
  - name: heavy-db
    image: my-db:latest
    volumeMounts:
    - mountPath: /data
      name: big-volume
  volumes:
  - name: big-volume
    persistentVolumeClaim:
      claimName: heavy-db-pvc
```

appArmorProfile

Параметр `appArmorProfile` определяет, какой профиль принудительного контроля доступа (Mandatory Access Control, MAC) будет наложен на контейнер на уровне ядра Linux.

Он управляет поведением системного модуля безопасности AppArmor, работающего на хост-узлах кластера. С помощью этого параметра можно заблокировать конкретные действия внутри контейнера (например, чтение конфигурационных файлов хоста, выполнение бинарных файлов или запись в системные папки), даже если процесс запущен под пользователем `root`.

AppArmor — это подсистема безопасности Linux, которая позволяет связать программу с профилем безопасности, определяющим ее полномочия. В отличие от стандартных прав Linux, ограничения AppArmor накладываются сверху, и их не может обойти даже суперпользователь `root`.

Список разрешенных действий формируется с помощью создания специального файла — профиля AppArmor на хост-узле в директории `/etc/apparmor.d`.

Профиль представляет собой текстовую структуру, в которой задаются детальные правила доступа к файловой системе, сети и системным вызовам. Пример:

```
profile k8s-apparmor-example-deny-write flags=(attach_disconnected) {
  include <abstractions/base>

  # Разрешение на чтение всего.
  /** r,

  # Разрешение на запись везде, кроме /etc.
  /** w,
  audit deny /etc/** w,
}
```

Профили бывают трёх видов:

- встроенный профиль рантайма (`RuntimeDefault`) — содержит базовый набор ограничений контейнеризации, оптимальный для большинства задач;
- профиль без ограничений (`Unconfined`) — полностью отключает защиту AppArmor для контейнера;
- пользовательские профили (`Localhost`) — кастомные профили, которые администратор должен заранее загрузить в ядро Linux на каждом узле в директорию `/etc/apparmor.d`.

AppArmor обеспечивает мощный дополнительный уровень изоляции и защиты от уязвимостей нулевого дня. Если злоумышленник получил выполнение кода в контейнере и даже повысил свои права до `root`, строго настроенный профиль AppArmor не позволит ему изменить конфигурационные файлы приложения, прочитать секреты хоста или выполнить опасные системные утилиты.

Расположение параметра

Параметр задаётся в следующих полях:

- `spec.securityContext.appArmorProfile` — на уровне пода;
- `spec.containers[].securityContext.appArmorProfile` — на уровне контейнера.

Доступные значения параметра

Доступны следующие значения:

- `RuntimeDefault` — профиль контейнерного рантайма по умолчанию (рекомендуемый вариант);
- `Unconfined` — AppArmor без ограничений (менее безопасно);
- `Localhost` — пользовательский профиль, который должен быть предварительно загружен в ядро операционной системы на каждом узле кластера.

Для данного параметра также указывается строковый параметр `localhostProfile` — точное имя профиля, зарегистрированное в операционной системе хоста.



До появления полноценного поля в API профили AppArmor настраивались через текстовые аннотации в метаданных пода. Начиная с версии Kubernetes 1.30, этот синтаксис объявлен устаревшим, а в версии 1.36 поддержка аннотаций была полностью удалена из кода Kubernetes.

Синтаксис требовал конструирования составного ключа, содержащего точное имя целевого контейнера:

- расположение — `metadata.annotations["container.apparmor.security.beta.kubernetes.io/<имя_контейнера>"]`;
- значения — `runtime/default` (в настоящий момент `RuntimeDefault`), `localhost/<имя_профиля>` (в настоящий момент `Localhost`) или `unconfined` (в настоящий момент `Unconfined`).

Критический недостаток устаревшего формата заключался в том, что API-сервер не валидировал опечатки в имени контейнера внутри аннотации. Если в названии допускалась ошибка, Kubernetes запускал контейнер без какой-либо защиты AppArmor. При обновлении кластеров до версии 1.30+ данный формат необходимо переписать на новый `securityContext.appArmorProfile`.

Пример конфигурации

Пример конфигурации пода, использующей кастомный профиль AppArmor с именем `k8s-apparmor-example-deny-write`, который был предварительно загружен в ОС на узлах кластера:

```

apiVersion: v1
kind: Pod
metadata:
  name: secure-apparmor-pod
spec:
  containers:
    - name: app
      image: nginx
      securityContext:
        appArmorProfile:
          type: Localhost
          localhostProfile: k8s-apparmor-example-deny-write # Точное имя профиля в ядре ОС узла.

```

seccompProfile

Параметр `seccompProfile` ограничивает список системных вызовов, доступных процессам в контейнере с помощью специального профиля.

Системный вызов (`syscall`) — это обращение программы из пользовательского пространства к ядру Linux за «привилегированным» действием. Процесс не может напрямую управлять устройствами, памятью, сетью и другими системными ресурсами. Он просит ядро сделать это через `syscall`.

Список разрешенных системных вызовов формируется с помощью создания специального файла с профилем `seccomp`. Профиль представляет собой JSON-структуру, в которой задаются разрешения. Пример:

```

{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "names": [
        "read",
        "write",
        "exit",
        "exit_group"
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}

```

Профили бывают трёх видов:

- встроенный профиль рантайма (`RuntimeDefault`) — содержит базовый набор разрешённых системных вызовов, оптимальный для большинства контейнеров;
- профиль без ограничений (`Unconfined`) — позволяет выполнить любое действие в контейнере;
- пользовательские профили.

Чем меньше разрешённых системных вызовов, тем меньше «поверхность атаки». Даже если злоумышленник смог выполнить код в контейнере, запрет критичных вызовов может ограничить развитие атаки.

Расположение параметра

Параметр задаётся в следующих полях:

- `spec.securityContext.seccompProfile` — на уровне пода;

- `spec.containers[].securityContext.seccompProfile` — на уровне контейнера.

Доступные значения параметра

Доступны следующие значения:

- `RuntimeDefault` — профиль контейнерного рантайма по умолчанию (рекомендуемый вариант);
- `Unconfined` — без `seccomp`-ограничений (менее безопасно);
- `Localhost` — пользовательский профиль, размещённый на каждом узле в директории `/var/lib/kubelet/seccomp`.

Для данного параметра также указывается параметр `localhostProfile` — относительный путь к профилю внутри директории `/var/lib/kubelet/seccomp`.

Пример конфигурации

Пример конфигурации пода, в которой используется пользовательский профиль из файла `/var/lib/kubelet/seccomp/my-profiles/secure.json`:

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: my-profiles/secure.json # Путь относительно
/var/lib/kubelet/seccomp/.
  containers:
    - name: app
      image: nginx
```

seLinuxOptions

Параметр `seLinuxOptions` ограничивает возможности процессов в контейнере с помощью меток принудительного контроля доступа (Mandatory Access Control, MAC) на уровне ядра Linux.

SELinux (Security-Enhanced Linux) — это подсистема безопасности Linux, которая управляет доступом на основе политик и контекстов безопасности (лейблов), присваиваемых процессам, файлам, портам и устройствам. В отличие от стандартных прав Linux, ограничения SELinux накладываются сверху, благодаря чему процессы изолируются друг от друга, даже если они запущены от имени суперпользователя `root`. Этот механизм по умолчанию активен и является стандартом в дистрибутивах Red Hat, CentOS, Fedora, Rocky Linux и AlmaLinux.

Контекст безопасности SELinux представляет собой строку, разделённую двоеточиями, которая состоит из четырёх основных компонентов: `user:role:type:level`.

В Kubernetes эти компоненты передаются контейнерному рантайму через специальную структуру параметров. Пример структуры компонентов:

```
User (Пользователь):  system_u      # Системный пользователь SELinux.
Role (Роль):          system_r      # Системная роль для процессов.
Type (Тип/Домен):     container_t   # Домен, определяющий правила доступа контейнера.
Level (Уровень/MCS):  s0:c123,c456 # Категории для многоуровневой изоляции (Multi-Category Security).
```

В большинстве случаев контейнерные рантаймы (`containerd`, `CRI-O`) автоматически генерируют случайные уникальные категории (`level`) для каждого пода. Благодаря этому два контейнера с одинаковым типом `container t` физически не могут получить доступ к файлам друг друга на хост-узле. Параметр `seLinuxOptions` используется тогда, когда этот стандартный механизм нужно переопределить.

SELinux обеспечивает строгую изоляцию на уровне файловой системы хоста и межпроцессного взаимодействия. Если злоумышленник скомпрометирует приложение внутри контейнера,

SELinux заблокирует любые попытки прочитать чужие файлы на узле (например, в директории `/var/lib/kubelet/pods/`), получить доступ к сокетам хоста или устройствам в `/dev`, предотвращая «побег из контейнера» (container escape).

Расположение параметра

Параметр задаётся в следующих полях:

- `spec.securityContext.seLinuxOptions` — на уровне пода;
- `spec.containers[].securityContext.seLinuxOptions` — на уровне контейнера.

Доступные значения параметра

Параметр принимает объект, состоящий из четырех опциональных строковых полей, которые соответствуют компонентам контекста SELinux:

- `user` — пользователь SELinux (например, `system_u`);
 - `role` — роль SELinux (например, `system_r`);
 - `type` — тип или домен безопасности SELinux (например, `container_t` или специализированный домен вида `spc_t` для супер-привилегированных контейнеров);
 - `level` — уровень чувствительности и категории MCS (например, `s0:c100,c200`).
- Применяется для явного разделения доступа к общим дискам.



По умолчанию Kubernetes автоматически проставляет лейблы (выполняет `relabeling`) на все смонтированные постоянные тома (`PersistentVolumes`), присваивая файлам на диске тот же MCS-уровень (`level`), с которым запускается под. Если вам нужно отключить этот автоматический процесс (например, диск монтируется одновременно в несколько подов в режиме `ReadWriteMany` и автоматическая смена лейблов ломает доступ), в настройках тома (`spec.volumes[].persistentVolumeClaim`) используется специальная точка монтирования с флагом `mountOptions: ["context=xyz"]`.

Пример конфигурации

Пример конфигурации пода, запускаемого со строго зафиксированным контекстом SELinux для работы со специализированным типом данных и разделяемой директорией:

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-selinux-pod
spec:
  securityContext:
    seLinuxOptions:
      user: "system_u"
      role: "system_r"
      type: "container_t"
      level: "s0:c123,c456" # Явная фиксация MCS-категорий для доступа к общим файлам.
  containers:
    - name: app
      image: nginx
```

capabilities

Параметр `capabilities` отвечает за управление привилегиями ядра Linux (Linux Capabilities) для процессов внутри контейнера.

Linux Capabilities — это механизм ядра Linux, который разбивает полномочия суперпользователя (`root`) на десятки независимых привилегий. Каждая привилегия имеет собственный текстовый идентификатор. Всего ядро Linux имеет около 40 различных `capabilities`. Вот самые частые из них:

- `NET_BIND_SERVICE` — позволяет процессу (даже не от имени `root`) слушать системные порты (например, `80` или `443`);
- `SYS_TIME` — разрешает изменять системное время на узле;
- `NET_ADMIN` — позволяет настраивать сетевые интерфейсы, правила маршрутизации и фаервол внутри сети контейнера (часто используется в `service mesh` — например, Istio);

- `CHOWN` — разрешает произвольно менять владельца файлов.

С помощью `capabilities` реализуется принцип наименьших привилегий. С его помощью можно дать пользователю одну конкретную привилегию ядра без выдачи полного `root`-доступа, либо отобрать некоторые привилегии у пользователя `root`.

Расположение параметра

Параметр состоит из двух частей:

- `Drop` — список привилегий, которые будут запрещены контейнеру;
- `Add` — список привилегий, которые будут разрешены контейнеру.

Параметр задаётся на уровне контейнера в полях `spec.containers[].securityContext.capabilities.add` и `spec.containers[].securityContext.capabilities.drop`.

Итоговый расчёт разрешенных привилегий производится по формуле `(Default + Add) - Drop`, где `Default` — это стандартный набор контейнерного рантайма (`containerd/CRI0`).

Таким образом, приоритет имеют операции запрета привилегий. Если разрешить и запретить одну и ту же привилегию, то в итоге привилегия будет запрещена.



Исключение — обработка при `drop: ALL`. В этом случае все привилегии сначала удаляются, а затем добавляются только те, что перечислены в поле `add`.

Доступные значения параметра

Параметры `Add` и `Drop` представлены в виде массива строк и не валидируются на уровне Kubernetes. Допустимо указанию любых значений. Полный перечень поддерживаемых привилегий ядра можно узнать в документации Linux.

Рекомендуемый практический подход:

1. Начинайте с `drop: ["ALL"]`.
2. Добавляйте в `add` только минимально необходимые `capabilities`.

Примеры конфигурации

1. Одинаковые привилегии в `add` и `drop`:

```
securityContext:
  capabilities:
    add: ["NET_ADMIN"]
    drop: ["NET_ADMIN"]
```

В результате привилегия `NET_ADMIN` будет запрещена (запрет имеет приоритет).

2. Добавление одной привилегии без `drop`:

```
securityContext:
  capabilities:
    add: ["NET_BIND_SERVICE"]
```

Итоговый набор `capabilities`:

- все `capabilities` из `Default` (стандартный набор контейнерного рантайма);
- а также `NET_BIND_SERVICE` (если её не было в `Default`).

3. Удаление части привилегий:

```
securityContext:
  capabilities:
    drop: ["CHOWN", "FOWNER"]
```

В итоге будут разрешены все `capabilities` из `Default` (стандартный набор контейнерного рантайма), кроме `CHOWN` и `FOWNER`.

4. Жёсткое ограничение через `drop: ["ALL"]`:

```
securityContext:
  capabilities:
    drop: ["ALL"]
    add: ["NET_BIND_SERVICE"]
```

В итоге будет разрешена только capability `NET_BIND_SERVICE`.

allowPrivilegeEscalation

Параметр `allowPrivilegeEscalation` определяет, может ли процесс внутри контейнера получить больше прав, чем его родительский процесс.

Он напрямую управляет системным флагом ядра Linux `no new privs` для запускаемого контейнера. Если вы устанавливаете `allowPrivilegeEscalation: false`, Kubernetes включает этот флаг, блокируя любые механизмы повышения прав.

Главный инструмент повышения прав в Linux — это файлы с флагами `SUID` (Set User ID) или `SGID` (Set Group ID). Когда обычный пользователь запускает SUID-файл, процесс временно получает права владельца этого файла (часто это `root`).

Например, утилиты `passwd` (для смены пароля) или `sudo` имеют флаг `SUID`. Обычный пользователь запускает `sudo`, и ядро Linux временно повышает его привилегии до уровня суперпользователя, чтобы выполнить системную задачу.

При использовании `allowPrivilegeEscalation: false` ядро Linux активирует режим `no_new_privs`. При этом:

- **блокируются SUID/SGID-файлы** — при попытке запустить `sudo` или `passwd` ядро проигнорирует `SUID`-флаг. Утилита выполнится с правами текущего пользователя контейнера и выдаст ошибку `Permission denied`;
- **запрещается активация capabilities** — процесс не сможет приобрести новые Linux Capabilities в ходе своей работы (например, через выполнение файлов с предустановленными файловыми мандатами).

Даже если контейнер запущен от имени пользователя `root` (UID 0), Docker и Kubernetes по умолчанию урезают часть системных прав (capabilities). Однако если в контейнере доступно повышение привилегий, злоумышленник, получивший доступ к приложению, может найти уязвимость в ядре Linux или системной утилите внутри контейнера и использовать SUID-вызовы для «побега из контейнера» (container escape) на хостовый узел с полными правами `root`. Настройка `allowPrivilegeEscalation: false` закрывает этот вектор атаки.



Если вы делаете файловую систему контейнера доступной только для чтения (`readOnlyRootFilesystem: true`), это косвенно мешает злоумышленнику создать свой SUID-файл, но не защищает от использования уже существующих в образе утилит.

Расположение параметра

Параметр задаётся на уровне контейнера в поле `spec.containers[].securityContext.allowPrivilegeEscalation`.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — повышение привилегий запрещено (рекомендуется для большинства приложений);
- `true` — повышение привилегий разрешено.

Пример конфигурации

Полный запрет на повышение прав:


```

spec:
  containers:
  - name: secure-app
    image: my-app:latest
    securityContext:
      allowPrivilegeEscalation: false

```

privileged

Параметр `privileged` определяет, запускается ли контейнер с полным доступом к хостовой операционной системе на узле на уровне ядра.

При установке параметра `privileged: true` Kubernetes полностью отключает все стандартные механизмы изоляции контейнера. С точки зрения безопасности, этот контейнер становится обычным процессом, запущенным от имени `root` прямо на хосте, хотя и запертым в своем сетевом и cgroup-пространстве.

При выставлении параметра `privileged: true` происходят следующие изменения:

- **выдача доступа ко всем capabilities** — контейнер получает все Linux Capabilities (около 40 штук);
- конфигурации `add` и `drop` в `securityContext` для этого контейнера начинают игнорироваться.
- **доступ к устройствам хоста** — внутри контейнера в директории `/dev` появляются все физические устройства узла (жёсткие диски, NVMe-накопители, графические карты, USB-порты). Контейнер может напрямую читать и писать в них;
- **отключение AppArmor и SELinux** — системы принудительного контроля доступа (MAC) на хосте перестают накладывать ограничения на процессы внутри этого контейнера;
- обход `sysfs` и `procfs` — контейнер может свободно изменять параметры ядра хоста через `/sys` и `/proc`.

Флаг `privileged: true` — это самая большая брешь в безопасности, если он выдан недоверенному приложению. Злоумышленник, получивший доступ в такой контейнер, может гарантированно скомпрометировать весь узел за одну команду. Например, он может выполнить команду `mount` для корневого диска хоста и получить полный контроль над файлами операционной системы узла, совершив «побег из контейнера» (container escape).



Обычным бизнес-приложениям этот флаг категорически противопоказан. Он необходим исключительно для системных компонентов кластера: сетевых плагинов (CNI), драйверов хранилищ (CSI) или низкоуровневых агентов мониторинга.

Расположение параметра

Параметр задаётся на уровне контейнера в поле `spec.containers[].securityContext.privileged`.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — привилегированный режим отключен (значение по умолчанию, рекомендуется для безопасности);
- `true` — привилегированный режим включен.

Пример конфигурации

Пример конфигурации, включающей максимальные права на хосте для системной утилиты:

```

spec:
  containers:
  - name: admin-tool
    image: ubuntu
    securityContext:
      privileged: true

```


procMount

Параметр `procMount` определяет, как подкаталоги системной файловой системы `/proc` будут монтироваться внутри контейнера.

Он напрямую управляет уровнем изоляции системной информации ядра Linux. С помощью этого параметра можно отключить стандартные защитные маски Kubernetes, которые скрывают от процессов контейнера критически важные и потенциально опасные системные пути хоста.

Файловая система `/proc` в Linux является окном в ядро. Через неё можно не только читать системные метрики, но и изменять параметры конфигурации ОС на лету. По умолчанию контейнерные рантаймы (`containerd`, `CRI-O`) используют защитные маски (`MaskedPaths`), которые скрывают или делают доступными только для чтения критические пути (например, `/proc/sys`, `/proc/sysrq-trigger`, `/proc/scsi`).

Например, если обычному контейнеру нужно изменить параметры сетевого стека через `sysctl`, защитная маска ядра заблокирует запись в `/proc/sys/net`, выдавая ошибку `Read-only file system`, чтобы защитить хостовую операционную систему от несанкционированных изменений.

При использовании `procMount` Kubernetes передаёт инструкции рантайму изменить тип монтирования. При этом:

- **включается стандартная защита** (`Default`) — применяются все стандартные маски контейнеризации. Опасные системные пути хоста скрываются или блокируются на запись;
- **снимается системная изоляция** (`Unmasked`) — отключаются все защитные маски для файловой системы `/proc`. Контейнер видит структуру ядра хоста «как есть», без ограничений на чтение и запись.

Установка значения `Unmasked` открывает прямой доступ к конфигурации ядра хоста. Если злоумышленник скомпрометирует контейнер с отключенным маскированием, он сможет использовать уязвимости ядра или напрямую изменить системные параметры узла (например, через `/proc/sysrq-trigger` вызвать мгновенную панику ядра или перезагрузку хоста). Этот параметр является критическим вектором для «побега из контейнера» (`container escape`).



Использование типа `Unmasked` требуется крайне редко. Основной сценарий его применения — запуск специализированных инструментов внутри контейнеров, таких как инструменты сборки образов (`Kaniko`, `Buildah` и др.) или вложенные контейнеры (`nested containerization`), которым необходим полноценный, немодифицированный доступ к подсистемам `/proc` для эмуляции процессов.

Расположение параметра

Параметр задаётся на уровне контейнера в поле `spec.containers[].securityContext.procMount`.

Доступные значения параметра

Параметр имеет строковый тип (`string`). Доступны следующие значения:

- `Default` — стандартное маскирование опасных путей ядра (рекомендуется по умолчанию);
- `Unmasked` — отключение защитных масок и предоставление полного доступа к `/proc`.

Пример конфигурации

Пример конфигурации с отключением защитных масок `/proc` для специализированного контейнера сборщика:

```
spec:
  containers:
    - name: image-builder
      image: kaniko-project/executor:latest
      securityContext:
        procMount: Unmasked
```

sysctls

Параметр `sysctls` определяет список изменяемых параметров ядра Linux, которые можно безопасно или небезопасно переопределить для неймспейса конкретного пода.

Он напрямую управляет поведением сетевого стека, памяти и виртуальной файловой системы на уровне контейнеров. С помощью этого параметра можно оптимизировать производительность высоконагруженных приложений (например, баз данных или веб-серверов) без необходимости менять глобальные настройки операционной системы на всей хост-узле.

В операционной системе Linux утилита `sysctl` позволяет изменять конфигурацию ядра во время работы системы через интерфейс `/proc/sys/`. Параметры ядра делятся на изолированные (на уровне неймспейсов контейнера) и глобальные (влияющие на всю физический узел).

Например, по умолчанию максимальное количество незавершенных соединений в очереди (`somaxconn`) ограничено небольшим системным значением. Высоконагруженному балансировщику трафика NGINX этого может не хватать, из-за чего он начнет отбрасывать пакеты. С помощью `sysctl` контейнеру можно индивидуально выделить увеличенный размер очереди.

i В кластере DKP ряд параметров `sysctl` настраивается автоматически при установке. Ознакомиться с полным списком таких параметров можно в разделе «Параметры `sysctl`, настраиваемые платформой».

При настройке `sysctl` Kubernetes делит все параметры на две категории, требующие разного уровня доверия. При этом:

- **разрешаются безопасные параметры** (`Safe sysctl`) — параметры, которые полностью изолированы внутри контейнера и их изменение не может навредить соседним подам или стабильности узла. Kubernetes применяет их сразу (например, `net.ipv4.ip_local_port_range`);
- **блокируются небезопасные параметры** (`Unsafe sysctl`) — параметры, которые могут повлиять на весь узел (вызвать перегрузку памяти, нарушить общую маршрутизацию). По умолчанию Kubernetes заблокирует поды с такими параметрами. Для их работы администратор кластера должен предварительно разрешить их в конфигурации `kubelet` через флаг `--allowed-unsafe-sysctl`.

Актуальный список безопасных `sysctl`-параметров доступен в документации Kubernetes.

Бесконтрольное использование небезопасных `sysctl` может привести к отказу в обслуживании (DoS) всего сервера. Если злоумышленник получит доступ к контейнеру с правами на изменение глобальных сетевых параметров, он сможет нарушить связность других подов на этом узле или забить системную память, спровоцировав аварийное завершение критических процессов хоста (`Out of Memory`).

⚠ В Kubernetes существует лишь несколько безопасных параметров ядра (в основном это `kernel.shm_rmid_forced` и часть сетевых параметров `net.ipv4.*` для локальной сети контейнера). Если вам требуется применить параметр из категории `Unsafe`, рекомендуется вынести такие приложения на изолированные группы узлов (`dedicated node pools`) с помощью `taints` и `tolerations`, чтобы минимизировать риски для остального кластера.

Расположение параметра

Параметр задаётся исключительно на уровне всего пода в поле `spec.securityContext.sysctl`.

Доступные значения параметра

Параметр принимает список объектов (`array`), где каждый элемент состоит из пары ключ-значение (`name` и `value`), имеющих строковый тип (`string`).

Пример конфигурации

Пример настройки параметров локального сетевого стека для высоконагруженного пода:

```

spec:
  securityContext:
    sysctls:
      - name: net.core.somaxconn
        value: "8192"
      - name: net.ipv4.ip_local_port_range
        value: "1024 65535"
  containers:
    - name: high-load-nginx
      image: nginx:latest

```

hostNetwork

Параметр `hostNetwork` определяет, будет ли под использовать сетевой неймспейс узла, на котором он запущен, вместо отдельного изолированного сетевого неймспейса.

По умолчанию каждый под получает собственный сетевой неймспейс (`netns`) с виртуальным сетевым интерфейсом, отдельным IP-адресом и собственными правилами маршрутизации. Трафик изолирован от хоста и соседних подов сетевым плагином (CNI). Параметр `hostNetwork: true` отключает эту изоляцию.

При использовании `hostNetwork: true` Kubernetes даёт команду контейнерному рантайму не создавать отдельный сетевой неймспейс. При этом:

- **разделяется сетевой стек хоста** — контейнер использует сетевые интерфейсы узла (`eth0`, `loopback`), его IP-адрес и таблицы маршрутизации напрямую;
- **открываются порты на хосте** — любой порт, который слушает приложение в контейнере, автоматически открывается на интерфейсах узла и доступен извне кластера;
- **игнорируется CNI** — сетевой плагин кластера не назначает поду отдельный IP-адрес и не применяет сетевые политики (`NetworkPolicy`) к его трафику.

Включение `hostNetwork: true` разрушает сетевую изоляцию пода. Приложение получает прямой доступ ко всем сетевым интерфейсам узла, может перехватывать чужой трафик (`sniffing`), вмешиваться в маршрутизацию кластера или конфликтовать портами с системными службами хоста. Сетевые политики Kubernetes перестают действовать, так как трафик пода неотличим от трафика самого узла. Злоумышленник, скомпрометировавший такой под, получает возможность атаковать соседние поды и узлы напрямую, минуя механизмы микросегментации.



`hostNetwork` часто используется вместе с параметром `dnsPolicy: ClusterFirstWithHostNet`. Без явного указания этой политики под с `hostNetwork: true` не сможет корректно разрешать имена внутренних сервисов кластера через CoreDNS, так как по умолчанию будет использовать резолверы хоста.

Расположение параметра

Параметр задаётся в поле `spec.hostNetwork` исключительно на уровне всего пода, так как сетевой неймспейс создаётся для пода целиком.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — под использует собственный изолированный сетевой неймспейс (значение по умолчанию, рекомендуется);
- `true` — под использует сетевой неймспейс узла.

Пример конфигурации

Пример конфигурации пода, использующего сеть узла (например, системный сетевой агент):

```
apiVersion: v1
kind: Pod
metadata:
  name: host-network-pod
spec:
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  containers:
    - name: network-agent
      image: my-agent:latest
```

hostPID

Параметр `hostPID` определяет, будет ли под использовать PID-неймспейс узла вместо собственного изолированного.

В Linux идентификаторы процессов (PID) уникальны только в рамках одного неймспейса. По умолчанию каждый под получает собственный PID-неймспейс: процессы внутри контейнера видят только сами себя (PID 1 и его дочерние процессы), а процессы хоста и соседних подов для них невидимы. Параметр `hostPID: true` отключает эту изоляцию.

При использовании `hostPID: true` контейнерный рантайм не создаёт отдельный PID-неймспейс для пода. При этом:

- **видны все процессы узла** — процессы внутри контейнера видят полный список процессов, запущенных на узле (аналог команды `ps aux` на хосте);
- **доступно взаимодействие с процессами хоста** — при наличии достаточных прав процессы контейнера могут посылать сигналы (`kill`, `SIGTERM`) процессам узла или инспектировать их память через `/proc`.

Параметр `hostPID: true` открывает процессам контейнера окно в операционную систему узла. Злоумышленник, скомпрометировавший такой под, может анализировать запущенные на узле системные компоненты (включая `kubelet`, `containerd`, другие поды), собирать информацию о командных строках и аргументах процессов, а при наличии привилегий — завершать чужие процессы, вызывая отказ в обслуживании. Кроме того, доступ к `/proc` чужих процессов позволяет читать их переменные окружения, в которых часто находятся секреты и токены.



Возможность отправлять сигналы процессам хоста зависит от привилегий пользователя внутри контейнера. Даже с `hostPID: true` некорневой контейнер не сможет напрямую управлять процессами `root` на хосте, однако чтение метаданных процессов через `/proc` останется доступным.

Расположение параметра

Параметр задаётся исключительно на уровне всего пода в поле `spec.hostPID`.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — под использует собственный изолированный PID-неймспейс (значение по умолчанию, рекомендуется);
- `true` — под использует PID-неймспейс узла.

Пример конфигурации

Пример конфигурации пода с доступом к процессам узла (например, для системного мониторинга):

```
apiVersion: v1
kind: Pod
metadata:
  name: host-pid-pod
spec:
  hostPID: true
  containers:
    - name: process-inspector
      image: my-tool:latest
```

hostIPC

Параметр `hostIPC` определяет, будет ли под использовать IPC-неймспейс узла вместо собственного изолированного.

В Linux IPC-механизмы (очереди сообщений, разделяемая память, семафоры) разделяются по неймспейсам. По умолчанию каждый под получает собственный IPC-неймспейс, изолированный от хоста и соседних подов. Это означает, что процессы контейнера не могут использовать IPC-ресурсы, созданные процессами узла или другими подами. Параметр `hostIPC: true` отключает эту изоляцию.

При использовании `hostIPC: true` контейнерный рантайм использует IPC-неймспейс узла. При этом:

- **разделяются IPC-объекты хоста** — очереди сообщений (message queues), сегменты разделяемой памяти (shared memory) и семафоры узла становятся доступны процессам контейнера;
- **доступно межпроцессное взаимодействие с хостом** — контейнер может читать и модифицировать IPC-объекты, созданные системными процессами узла.

Параметр `hostIPC: true` создаёт канал обмена данными между контейнером и процессами узла в обход стандартных сетевых и файловых интерфейсов. Злоумышленник, скомпрометировавший такой под, может перехватывать или подменять данные, которыми обмениваются системные службы хоста через IPC, а также внедрять вредоносные payload в разделяемую память, эксплуатируя уязвимости процессов, читающих эти сегменты. На практике параметр применяется крайне редко — в основном для устаревших-приложений, использующих System V IPC для синхронизации с процессами, запущенными непосредственно на узле.



Большинство современных приложений не используют System V IPC, предпочитая сетевые сокеты или стандартные файловые дескрипторы. Поэтому `hostIPC` почти всегда можно безопасно оставить в значении `false` без потери функциональности.

Расположение параметра

Параметр задаётся в поле `spec.hostIPC` исключительно на уровне всего пода.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `false` — под использует собственный изолированный IPC-неймспейс (значение по умолчанию, рекомендуется);
- `true` — под использует IPC-неймспейс узла.

Пример конфигурации

Пример конфигурации пода с доступом к IPC-ресурсам узла:

```
apiVersion: v1
kind: Pod
metadata:
  name: host-ipc-pod
spec:
  hostIPC: true
  containers:
    - name: ipc-client
      image: my-app:latest
```

hostPath

Параметр `hostPath` определяет возможность монтирования произвольных файлов и директорий файловой системы узла напрямую в контейнер.

Обычно контейнеры работают с данными через изолированные тома Kubernetes (`emptyDir`, `PersistentVolume`, `configMap`), которые не дают прямого доступа к диску узла. Тип тома `hostPath` обходит эту абстракцию и пробрасывает указанный путь файловой системы хоста внутрь контейнера.

При настройке `hostPath` Kubernetes монтирует указанный путь с узла в контейнер. При этом:

- **разделяется файловая система хоста** — контейнер получает прямой доступ к файлам и директориям узла по указанному пути (`path`);
- **игнорируется изоляция хранилища** — доступ не ограничен `PersistentVolume` или квотами. Контейнер работает с реальными файлами узла;
- **тип монтирования управляется полем `type`** — доступны значения `"` (проверка отключена), `Directory`, `File`, `Socket`, `CharDevice`, `BlockDevice`. Они определяют, какой тип объекта должен существовать по пути до монтирования.

Параметр `hostPath` — один из самых опасных типов томов, так как он открывает контейнеру прямой доступ к файловой системе узла. Злоумышленник, получивший доступ в такой контейнер, может прочитать конфиденциальные файлы хоста (например, `/etc/shadow`, закрытые ключи, токены `kubelet`), модифицировать системные исполняемые файлы или подменить конфигурацию критических служб узла. Монтирование корневых или системных путей (`/`, `/var/lib/kubelet`, `/etc`, `/proc`, `/sys`) фактически эквивалентно выдаче полного контроля над узлом и является прямым путём к «побегу из контейнера» (`container escape`).



При использовании `hostPath` настоятельно рекомендуется монтировать том в режиме только для чтения (`readOnly: true`), если приложению не требуется запись. Поле `type` также следует указывать явно, чтобы Kubernetes проверял существование и тип объекта перед монтированием. Это предотвращает непредвиденное создание файлов на хосте.

Расположение параметра

Том `hostPath` описывается в массиве томов пода и затем монтируется в конкретный контейнер:

- `spec.volumes[].hostPath` — описание тома;
- `spec.containers[].volumeMounts` — точка монтирования внутри контейнера.

Доступные значения параметра

Объект `hostPath` содержит следующие поля:

- `path` — абсолютный путь в файловой системе узла, который будет смонтирован (строка, обязательное поле);
- `type` — тип объекта по указанному пути (строка, опционально). Доступны значения:
 - `"` — проверка типа отключена (значение по умолчанию);
 - `Directory` — директория должна существовать;
 - `DirectoryOrCreate` — директория будет создана, если отсутствует;
 - `File` — файл должен существовать;
 - `FileOrCreate` — файл будет создан, если отсутствует;
 - `Socket` — UNIX-сокеты должны существовать;
 - `CharDevice` — символьное устройство должно существовать;
 - `BlockDevice` — блочное устройство должно существовать.

Пример конфигурации

Пример конфигурации с монтированием конфигурационного файла узла в режиме только для чтения:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-pod
spec:
  containers:
    - name: app
      image: my-app:latest
      volumeMounts:
        - mountPath: /host/etc/app.conf
          name: host-config
          readOnly: true
  volumes:
    - name: host-config
      hostPath:
        path: /etc/app/app.conf
        type: File
```

automountServiceAccountToken

Параметр `automountServiceAccountToken` определяет, будет ли Kubernetes автоматически монтировать токен `ServiceAccount` в контейнер по стандартному пути.

В Kubernetes каждый под по умолчанию ассоциируется с `ServiceAccount` (если не указан явно — используется `default`). В целях аутентификации в API-сервере Kubernetes автоматически генерирует для этого `ServiceAccount`-токен (`ServiceAccountToken`) и монтирует его в каждый контейнер пода по пути `/var/run/secrets/kubernetes.io/serviceaccount`. Этот токен позволяет процессам внутри контейнера обращаться к API Kubernetes от имени своего `ServiceAccount`.

При использовании `automountServiceAccountToken: false` Kubernetes не монтирует токен `ServiceAccount` в контейнер. При этом:

- **отсутствует файл токена** — по пути `/var/run/secrets/kubernetes.io/serviceaccount` не создаётся файл `token`;
- **нет доступа к API от имени ServiceAccount** — процессы контейнера не могут автоматически аутентифицироваться в API-сервере Kubernetes с правами своего `ServiceAccount`;
- **сохраняется возможность ручного монтирования** — при необходимости токен можно смонтировать явно через тип тома `serviceAccountToken`, либо использовать современные ограниченные по времени токены (`BoundServiceAccountTokenVolume`), появившиеся в Kubernetes 1.21+.

Автоматически смонтированный токен `ServiceAccount` — это готовый `credential`, который злоумышленник может использовать для горизонтального перемещения по кластеру. Если приложение не работает с API Kubernetes (например, обычный веб-сервер или база данных), монтирование токена создаёт избыточный риск: при компрометации контейнера злоумышленник получает возможность обращаться к API-серверу с правами `ServiceAccount` пода, в том числе читать секреты, перечислять ресурсы и, при избыточных правах (RBAC), атаковать другие компоненты кластера. Отключение автоматического монтирования реализует принцип наименьших привилегий и уменьшает поверхность атаки.



Параметр `automountServiceAccountToken` можно задавать как на уровне `ServiceAccount`, так и на уровне самого пода (`spec.automountServiceAccountToken`). Значение, указанное в поде, имеет приоритет над значением в `ServiceAccount`. Также важно убедиться, что приложение действительно не использует API Kubernetes, прежде чем отключать монтирование токена.

Расположение параметра

Параметр задаётся:

- в поле `spec.automountServiceAccountToken` — на уровне всего пода;
- в поле `serviceAccount.automountServiceAccountToken` — на уровне ресурса

ServiceAccount.

Доступные значения параметра

Параметр имеет булевый тип (`boolean`). Доступны следующие значения:

- `true` — токен ServiceAccount монтируется автоматически (значение по умолчанию, если не указано иное);
- `false` — автоматическое монтирование токена отключено (рекомендуется для подов, не взаимодействующих с API Kubernetes).

Пример конфигурации

Пример конфигурации пода с отключённым автоматическим монтированием токена ServiceAccount:

```
apiVersion: v1
kind: Pod
metadata:
  name: no-sa-token-pod
spec:
  automountServiceAccountToken: false
  containers:
    - name: app
      image: my-app:latest
```

Дополнительные ресурсы

- Документация модуля `admission-policy-engine`

Веб-интерфейс Deckhouse Kubernetes Platform

Веб-интерфейс Deckhouse Kubernetes Platform (DKP) предоставляет доступ к управлению кластером, включая мониторинг, настройку параметров, управление узлами и модулями, а также настройку безопасности и сети. Большинство операций, доступных в командной строке через утилиты `d8` (Deckhouse CLI) или `kubectl`, также можно выполнять через веб-интерфейс.

Доступ к веб-интерфейсу

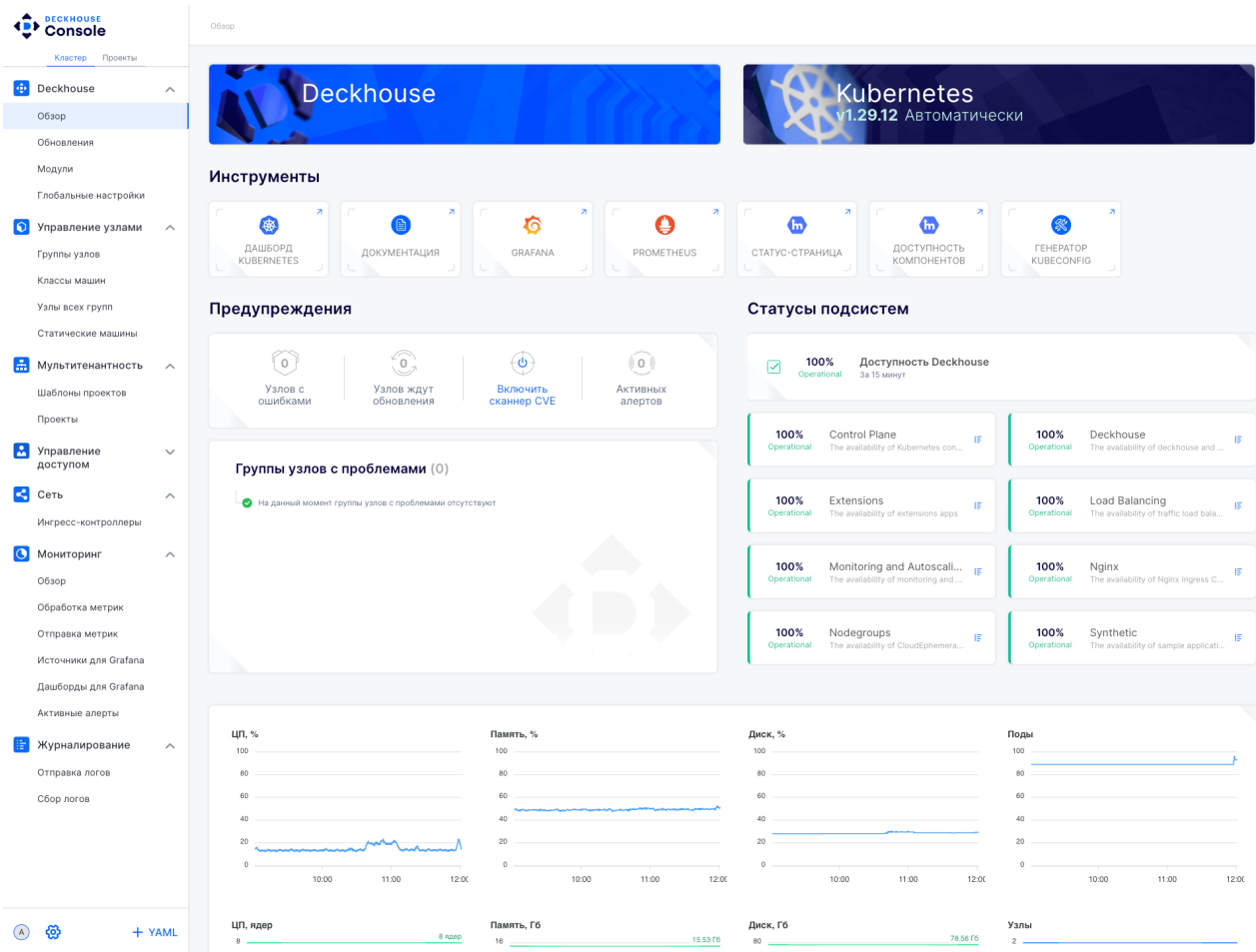
Чтобы открыть веб-интерфейс, выполните следующее:

1. Введите в адресной строке браузера `console.<ШАБЛОН_ИМЕН_КЛАСТЕРА>`, где `<ШАБЛОН_ИМЕН_КЛАСТЕРА>` — строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре `modules.publicDomainTemplate`. Формат адреса может отличаться в зависимости от конфигурации системы. Уточните URL-адрес у администратора.
2. При первом входе введите учетные данные пользователя, полученные от администратора безопасности. После успешной аутентификации откроется главная страница веб-интерфейса. На странице отображены следующие разделы:
 - «Deckhouse»;
 - «Управление узлами»;
 - «Мультиотенантность»;
 - «Доступ»;
 - «Сеть»;
 - «Хранилище»;
 - «Безопасность»;
 - «Мониторинг»;
 - «Журналирование».

Deckhouse

Подраздел «Обзор»

В подразделе «Обзор» представлена ключевая информация о кластере Deckhouse Kubernetes Platform (DKP) и его компонентах.

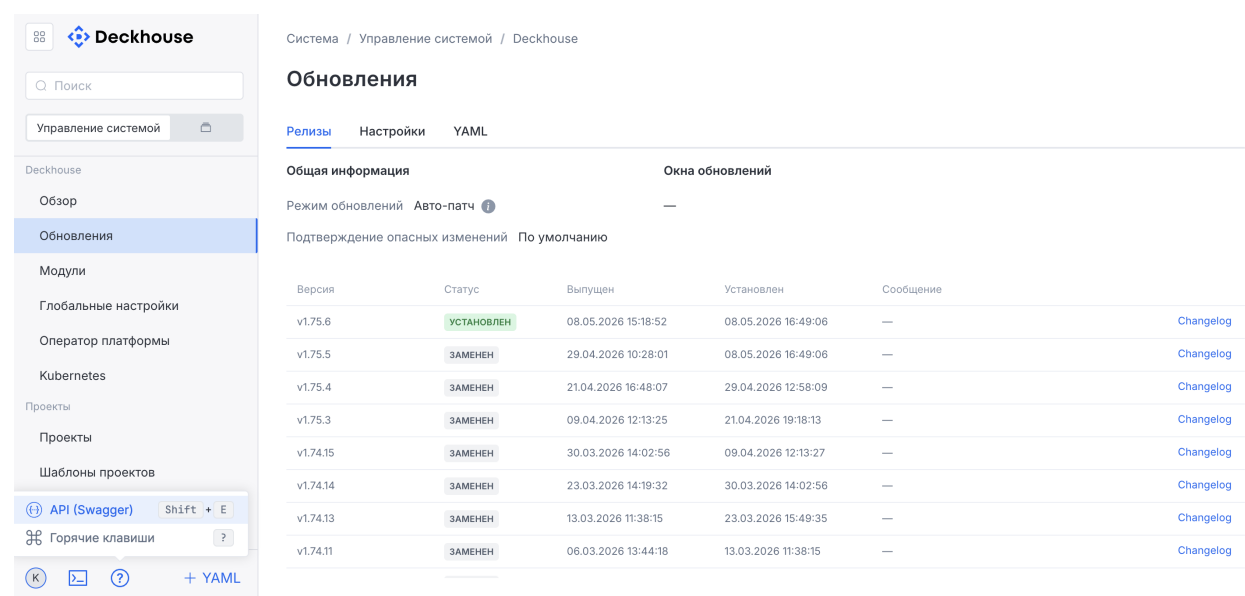


Основные элементы веб-интерфейса:

- **Панели «Deckhouse» и «Kubernetes»** — отображают текущую версию Kubernetes и основные сведения о платформе.
- **Панель «Инструменты»** — содержит кнопки для быстрого доступа к разделам:
 - «Дашборд Kubernetes»;
 - «Документация»;
 - «Grafana» (мониторинг метрик);
 - «Prometheus» (сбор и хранение метрик);
 - «Статус-страница»;
 - «Доступность компонентов»;
 - «Генератор kubernetesconfig».
- **Панель «Предупреждения»** — отображает ошибки, ожидаемые обновления и активные алерты.
- **Панель «Группы узлов с проблемами»** — анализирует работоспособность узлов и показывает проблемные группы.
- **Панель «Статусы подсистем»** — отображает состояние сервисов кластера.
- **Мониторинг ресурсов** — включает графики и показатели изменения нагрузки.
- **Боковое меню** — навигация по основным разделам (возможность отображения зависит от уровня доступа пользователя).
- **Меню пользователя (внизу слева):**
 - информация о пользователе;
 - настройки (изменение системных параметров, отображение версии веб-интерфейса);
 - редактор YAML для управления конфигурациями.

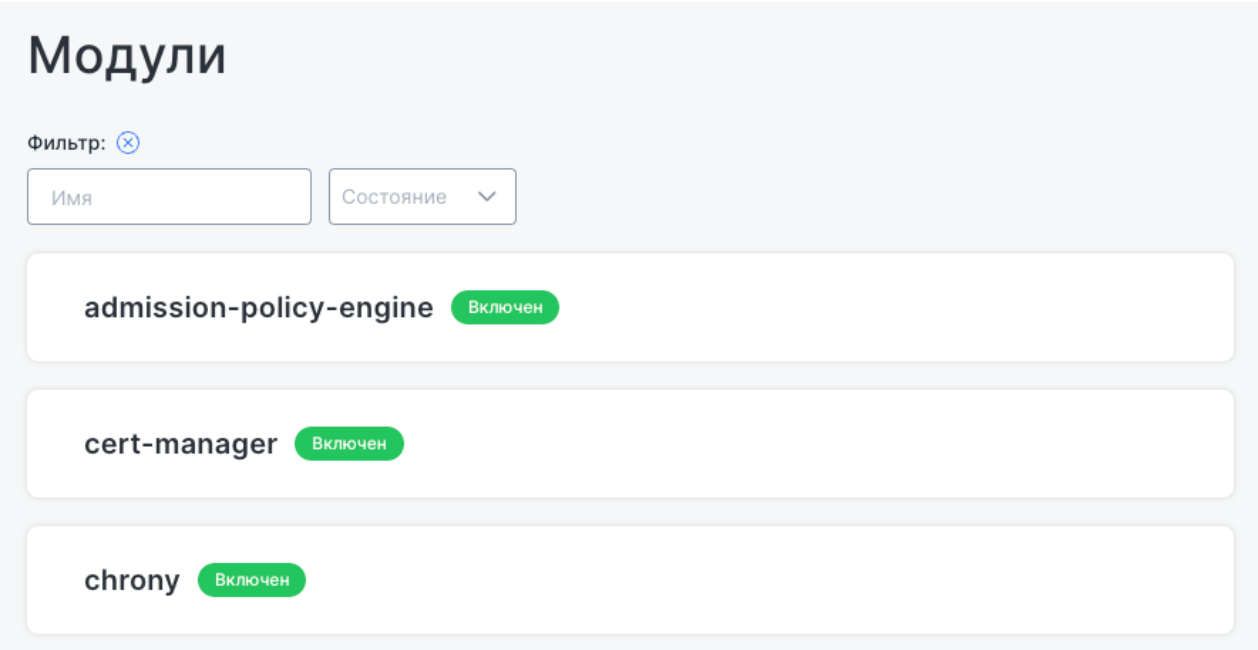
Подраздел «Обновления»

В подразделе «Обновления» содержится информация о релизах DKP.



Подраздел «Модули»

В подразделе «Модули» перечислены запущенные и отключенные модули. Для поиска необходимого модуля предусмотрен фильтр.



Подраздел «Глобальные настройки»

- Подраздел предоставляет доступ к критически важным настройкам кластера DKP:
- **«Глобальные настройки кластера»** — настройка шаблона DNS-имён и списка tolerations.
 - **«Глобальные настройки модулей»** — настройка режима отказоустойчивости, поля для указания IngressClass и StorageClass.
 - **«Режим работы HTTPS»** — настройка сертификатов безопасности.
 - **«Ресурсы управляющих компонентов Kubernetes (control plane)»** — настройка количества выделяемых ядер CPU и объёма памяти.

⚠ Эти настройки влияют на стабильность, безопасность и отказоустойчивость кластера, поэтому изменяйте их с осторожностью.

Глобальные настройки кластера

Шаблон DNS-имен

Пустое значение приведет к недоступности веб-интерфейса

%s.158.160.157.119.sslip.io

Используйте ключ %s в качестве динамической части строки. Пример: %s.kube.company.my

Список ключей пользовательских toleration

Необходимо указывать, чтобы позволить планировщику размещать критически важные компоненты Deckhouse на выделенных узлах, например компоненты CNI и CSI

+ ДОБАВИТЬ

Глобальные настройки модулей

Режим отказоустойчивости

АвтоДаНет

Режим отказоустойчивости включается автоматически для кластеров с более чем одним master-узлом. В остальных случаях значение автоматически определяется как false

Класс Ingress-контроллера (Ingress class), используемый для модулей Deckhouse

nginx

Имя StorageClass для всех компонентов Deckhouse

По умолчанию

Если значение не указано, то используется автоматически определяемый global.discovery.defaultStorageClass. Если он не определен, то используется emptyDir.

Заданный StorageClass применяется в процессе включения модуля. Этот параметр имеет смысл использовать только в исключительных ситуациях.

Режим работы HTTPS

По умолчаниюНе используетсяCert ManagerСвой сертификатТолько в URI

Ресурсы управляющих компонентов Kubernetes (control plane)

ⓘ

Ресурсы выделяются на каждом мастер-узле. Не работает для not-managed-облаков (например, GKE)

⚠

Изменение параметров перезапустит управляющие компоненты кластера. Если ресурсов ЦП и памяти окажется слишком мало, то API кластера станет недоступным

Ядра ЦП

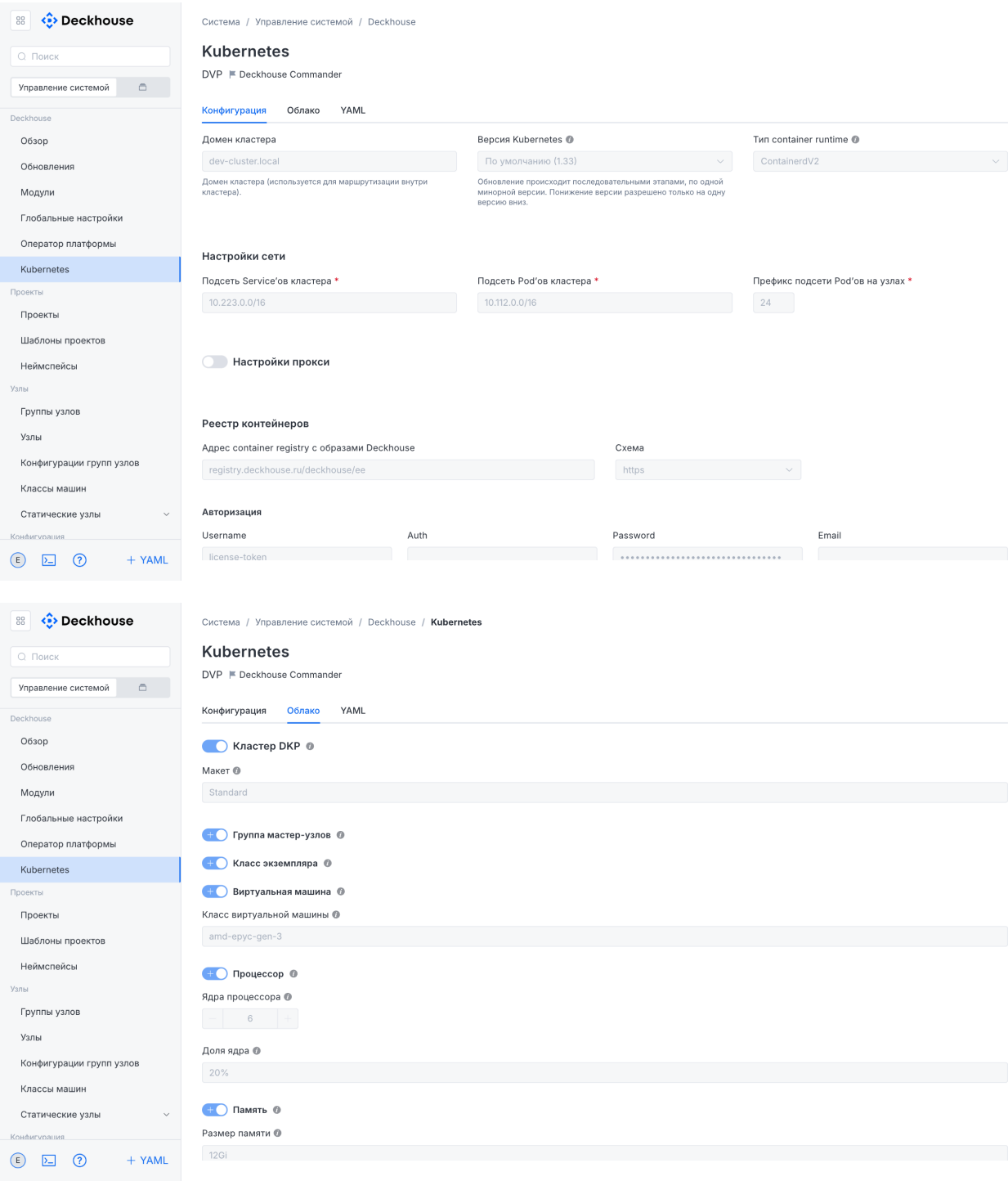
По умолчанию выделяется 40% ЦП. Задается в долях одного ядра, например 350m или 1

Память

По умолчанию выделяется 40% памяти. Объем памяти указывается с единицами изменения, например «1000Mi» или «1.5G»

Подраздел «Kubernetes»

На вкладках подраздела «Kubernetes» собрана базовая информация о кластере Kubernetes.



Управление узлами

Подраздел «Группы узлов»

Настройки в этом подразделе позволяют управлять группами узлов Kubernetes, просматривать их состояние и добавлять новые.

Форма добавления узлов содержит следующие поля:

- «Имя»;
- «Количество узлов»;
- «Селектор статических машин» (нельзя изменить после создания);
- «Выражения для лейблов».

В нижней части формы находятся дополнительные настройки, включая параметры обновления узлов, шаблон узла, системные параметры, а также параметры Chaos Monkey, которые также можно раскрыть для детальной конфигурации.

88

Deckhouse

Поиск

Управление системой

Deckhouse

Обзор

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Е

Д

?

+ YAML

Система / Управление системой / Узлы

Группы узлов

Типы узлов

worker

Тип узлов

Версия k8s

Состояние

По имени

Создать

worker

Создать на основании Удалить

Типу узлов Приоритет группы Версия Kubernetes Класс машин
CloudEphemeral - 1.33 worker

Состояние узлов

Параметры автомасштабирования

Зоны

Всего узлов

Готовые

Актуальные

Узлов на зону

Необходимо

Заказано

Резерв

agent.commander.deckhouse.io/group-id: 7be4dfef-fe7f-4fc7-86d5-d09f2813e1a9

agent.commander.deckhouse.io/is-orphan:

heritage: deckhouse-commander

Лейблы: 1



88

Deckhouse

Поиск

Управление системой

Deckhouse

Обзор

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Е

Д

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

worker

Удалить

Конфигурация Узлы События Мониторинг YAML

Статус группы

Типу узлов Приоритет группы Версия Kubernetes Класс машин
CloudEphemeral - 1.33 worker

Состояние узлов

Параметры автомасштабирования

Зоны

Всего узлов

Готовые

Актуальные

Узлов на зону

Необходимо

Заказано

Резерв

Минимум узлов на зону*
spec.cloudInstances.minPerZone

6

Класс машин*
spec.cloudInstances.classReference.kind

worker

Максимум узлов на зону*
spec.cloudInstances.maxPerZone

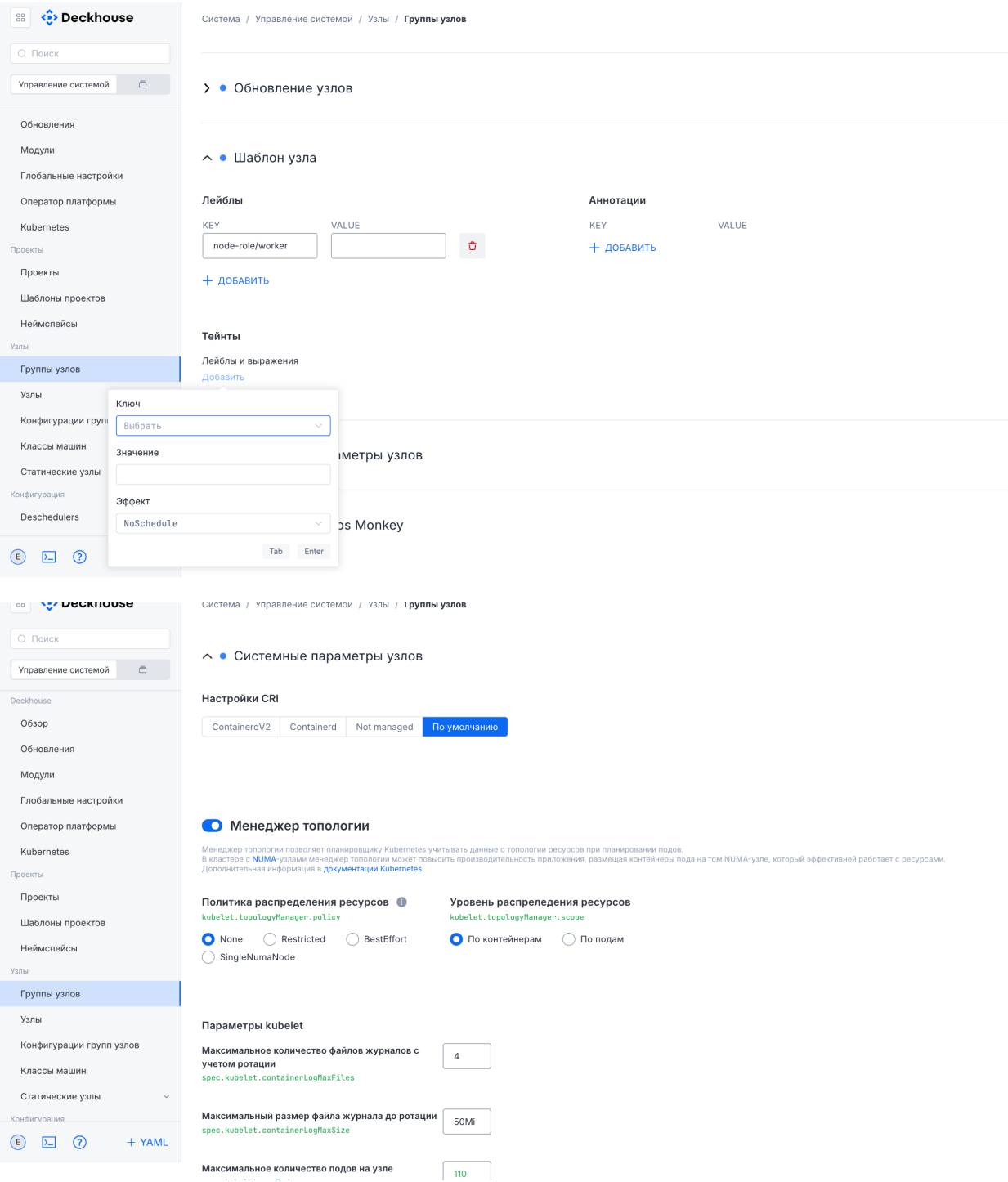
8

Количество реплик в MachineDeployment и нижняя граница в cluster autoscaler

Верхняя граница в cluster-autoscaler

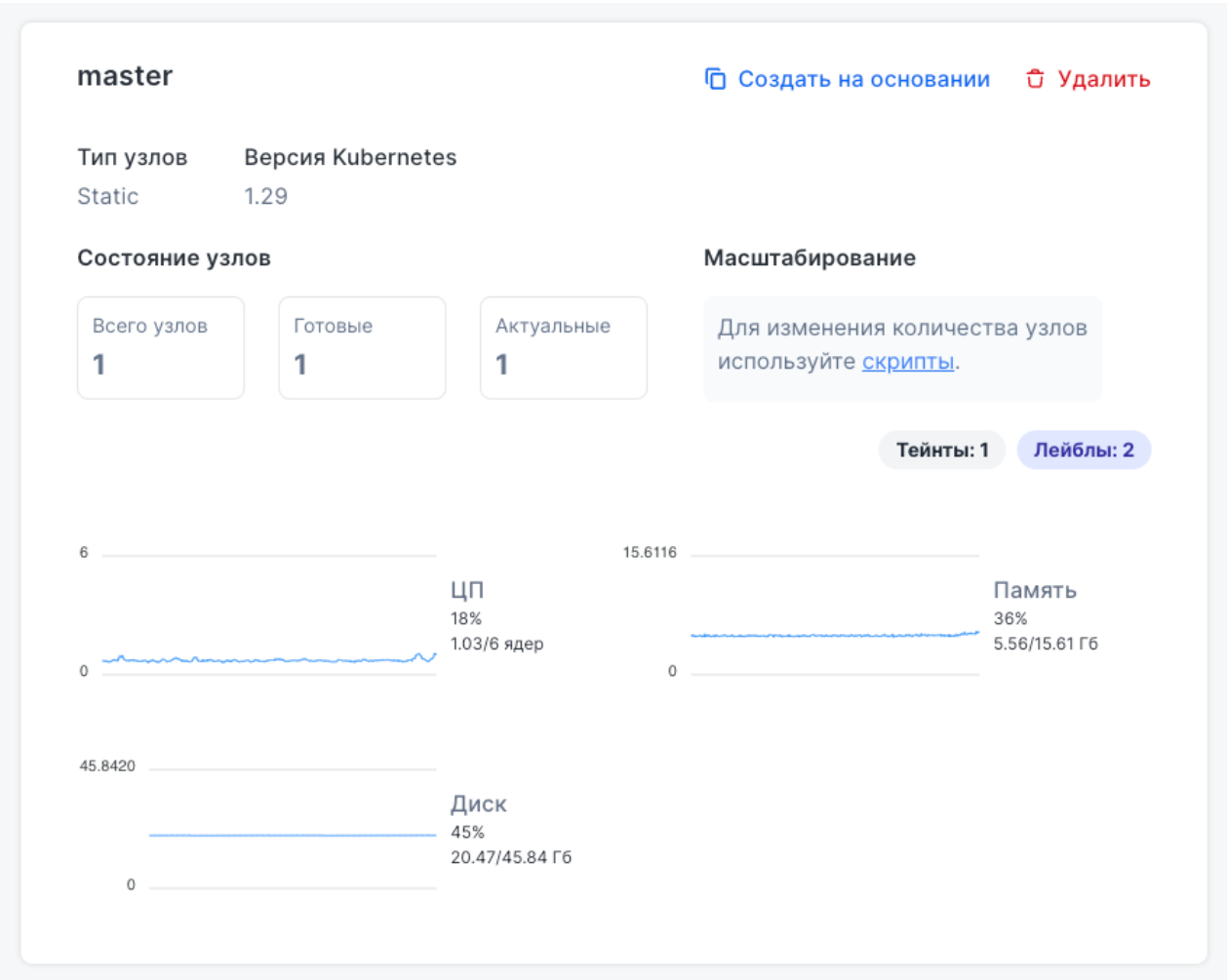
Параметры автомасштабирования

Обновление узлов

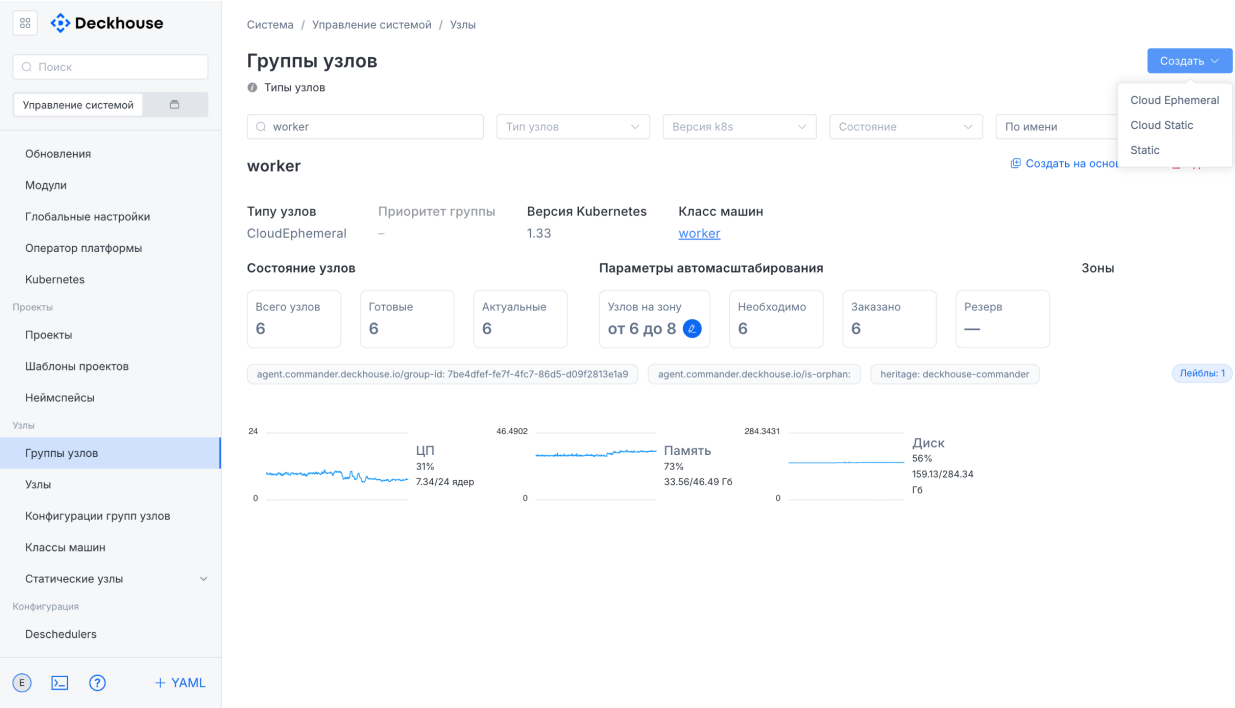


В карточке группы узлов отображаются:

- тип узлов и версия Kubernetes;
- общее количество узлов, их готовность и актуальность;
- метрики нагрузки (CPU, память, диск);
- тейнты и лейблы.



Кнопка «Создать» в правом верхнем углу открывает форму создания новой группы узлов.



В форме можно задать необходимые параметры создаваемой группы узлов.

88

Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

Добавление группы типа CloudEphemeral

Имя*

workrker

Обязательное поле

Минимум узлов на зону*

spec.cloudInstances.minPerZone

2

Класс машин*

spec.cloudInstances.classReference.kind

worker

Количество реплик в MachineDeployment и нижняя граница в cluster autoscaler

Максимум узлов на зону*

spec.cloudInstances.maxPerZone

5

Верхняя граница в cluster-autoscaler

> • Параметры автомасштабирования

> • Обновление узлов

> • Шаблон узла

Создать

88

Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

Добавление группы типа CloudEphemeral

spec.cloudInstances.maxPerZone

Верхняя граница в cluster-autoscaler

< • Параметры автомасштабирования

< • Обновление узлов

Зоны ⓘ

spec.cloudInstances.zones

Выбрать

Переопределение зон, в которых создаются машины

Приоритет масштабирования

spec.cloudInstances.priority

Если существует несколько групп узлов с одинаковым приоритетом, то группа добавления узла будет выбрана из них случайным образом

Число резервных узлов от общего количества узлов ⓘ

spec.cloudInstances.standby

Укажите число от общего количества узлов (например, 2-6) или проценты (15%). Это общее число узлов для всех зон сразу

Ресурсы, занимаемые на резервных узлах, %

spec.cloudInstances.standbyHolder.overprovisioningRate

50

Допустимые значения: 1 <= X <= 80

Quick Shutdown

spec.cloudInstances.quickShutdown

Снижает ожидание drain'a машины до 5 минут

> • Обновление узлов

Создать

88

Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

Д

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

Снижает ожидание drain'a машины до 5 минут

Обновление узлов

Максимальное количество одновременно обновляемых узлов

1

`spec.update.maxConcurrent`

Можно указать число узлов или процент от общего количества узлов в данной группе

Режим обновлений с перезагрузкой

Авто

Ручной

По умолчанию

Шаблон узла

Системные параметры узлов

Параметры Chaos Monkey

Создать

88

Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

Д

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

Режим обновлений с перезагрузкой

Авто

Ручной

По умолчанию

Шаблон узла

Лейблы

KEY

VALUE

+ ДОБАВИТЬ

Аннотации

KEY

VALUE

+ ДОБАВИТЬ

Тейнты

Лейблы и выражения

Добавить

Системные параметры узлов

Параметры Chaos Monkey

Создать

88Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

?

+ YAML

Система / Управление системой / Узлы / Группы узлов

Системные параметры узлов

Настройки CRI

ContainerdV2ContainerdNot managedПо умолчанию

Менеджер 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 подов на том NUMA-узле, который эффективней работает с ресурсами. Дополнительная информация в документации Kubernetes.

Параметры kubelet

Максимальное количество файлов журналов с учетом ротации4spec.kubelet.containerLogMaxFiles

Максимальный размер файла журнала до ротации50Mispec.kubelet.containerLogMaxSize

Максимальное количество подов на узле110spec.kubelet.maxPods

Путь к каталогу для файлов kubelet

Создать

Подраздел «Классы машин»

Этот подраздел предназначен для управления классами машин, используемых в кластере. Список классов можно сортировать.

88Deckhouse

Поиск

Управление системой

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Deschedulers

Е

?

+ YAML

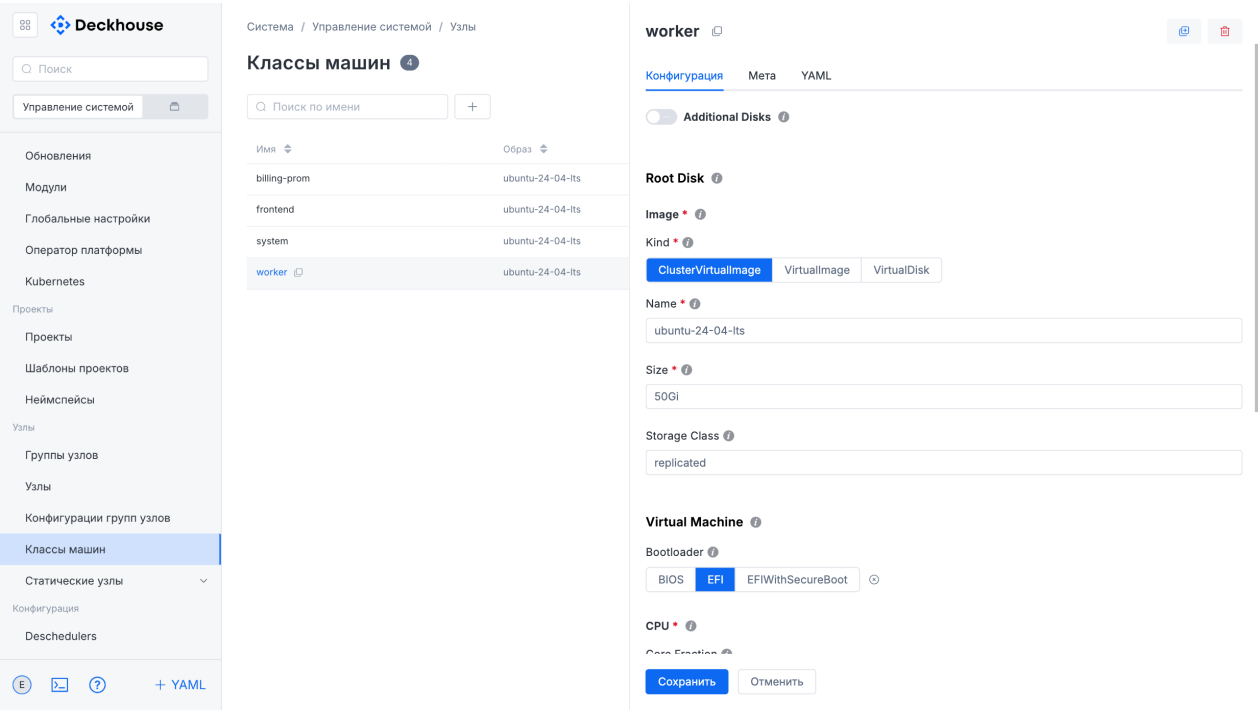
Система / Управление системой / Узлы

Классы машин 4

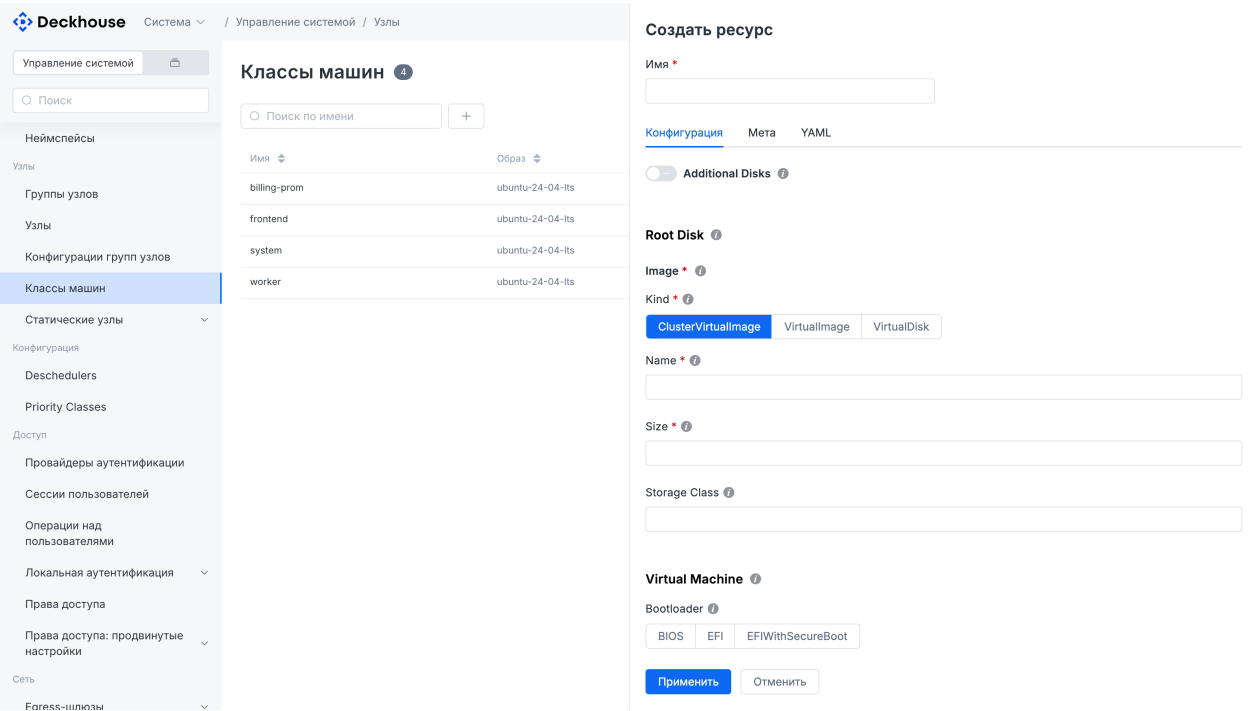
Создать

Поиск по имени

Имя	Образ	Класс хранилища	Класс ВМ	Часть ядра	Память		
billing-prom	ubuntu-24-04-lts	replicated	amd-epyc-gen-3	100%	12Gi		
frontend	ubuntu-24-04-lts	replicated	amd-epyc-gen-3	20%	4Gi		
system	ubuntu-24-04-lts	replicated	amd-epyc-gen-3	20%	8Gi		
worker	ubuntu-24-04-lts	replicated	amd-epyc-gen-3	20%	8Gi		



В форме «Создать ресурс» можно задать параметры новой конфигурации машин для кластера.



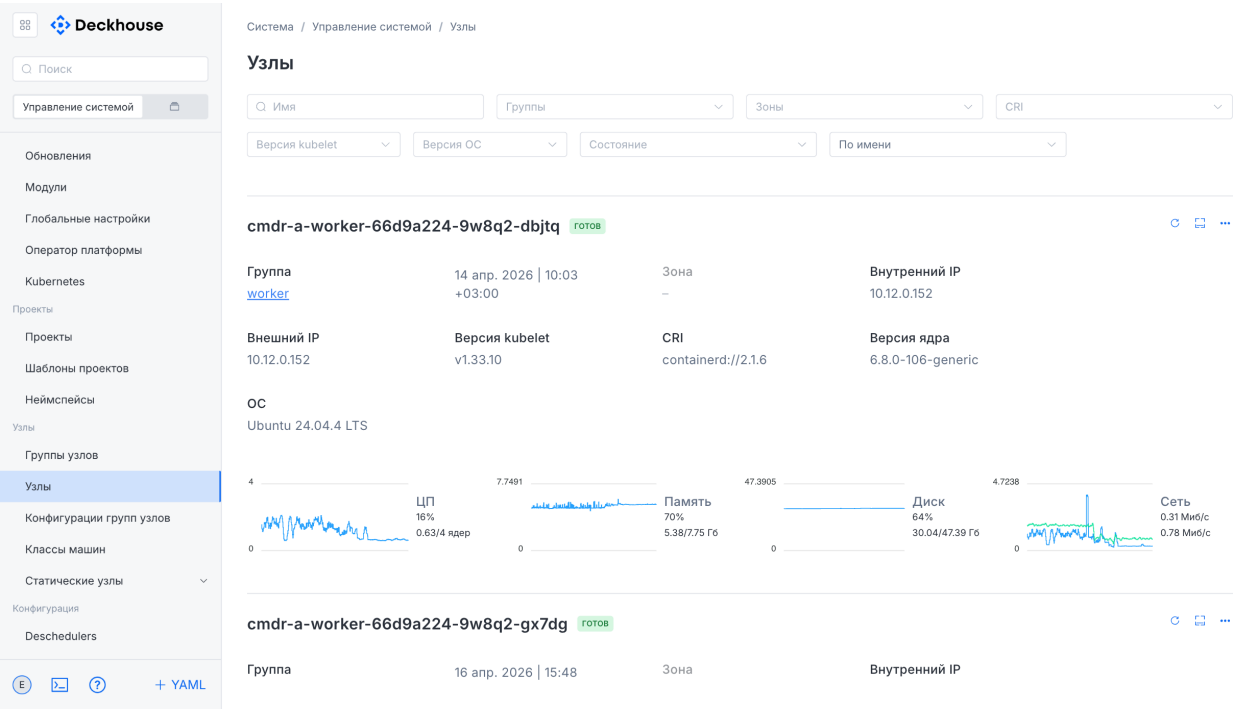
Подраздел «Узлы»

Подраздел предоставляет сводную информацию обо всех узлах кластера DKP с возможностью фильтрации и сортировки.

В карточке узла отображается:

- текущее состояние узла;
- группа;
- дата и время;
- зона размещения;
- внутренний и внешний IP-адреса;
- используемый container runtime;
- версия ядра;
- версия kubelet;
- операционная система.

Также здесь представлены графики загрузки CPU, памяти, диска и сетевого трафика, что позволяет отслеживать производительность узла. Для управления доступностью узла в кластере предусмотрены кнопки «Cordon» и «Cordon+Drain».



Подраздел «Статические машины»

Подраздел предназначен для управления статическими машинами в кластере и включает две вкладки:

- «Машины» — для работы со статическими машинами;
- «SSH-доступы» — для настройки SSH-доступа.

Интерфейс позволяет быстро находить статическими машины и управлять ими.

Кнопка «Добавить машину» на вкладке «Машины» открывает форму добавления новой машины в кластер. В форме необходимо указать:

- имя машины;
- адрес машины;
- способ SSH-доступа, выбранный из выпадающего списка.

Дополнительно можно задать лейблы, указав ключи и значения для дальнейшей идентификации и управления машиной.

Новая машина

Конфигурация

Имя машины*

Адрес*

SSH-доступ*

Лейблы

KEY

VALUE

[+ ДОБАВИТЬ](#)

Кнопка «Добавить SSH-доступ» на вкладке «SSH-доступы» открывает форму настройки подключения к узлам через SSH. В форме необходимо указать:

- имя SSH-доступа;
- имя пользователя;
- приватный SSH-ключ;
- (опционально) пароль `sudo` для выполнения привилегированных команд.

Дополнительно доступны поля для изменения SSH-порта и добавления дополнительных SSH-аргументов.

Новый SSH-доступ

Конфигурация

Имя SSH-доступа*

Имя пользователя*

Пароль sudo

Приватный SSH-ключ*

[Показать ключ](#)

SSH-порт

22

Допустимые значения 1 <= X <= 65535

Дополнительные аргументы SSH

Мультиотенантность

Подраздел «Проекты»

Подраздел предназначен для создания проектов на основе подготовленных шаблонов, определяющих набор создаваемых ресурсов и их параметры.

В процессе создания проекта происходит:

- валидация параметров по OpenAPI-схеме;
- рендеринг шаблона через Helm;
- развёртывание всех описанных ресурсов в автоматически создаваемом неймспейсе.

Для управления доступом, ограничения ресурсов и настройки сетевой изоляции используются стандартные механизмы Kubernetes. Это позволяет управлять безопасностью и нагрузкой внутри неймспейса.

Проекты

Сортировать по:

имени

[+ Создать проект](#)

Фильтр: [×](#)

Имя

Тип шаблона

deckhouse

[Deckhouse](#)

[Создать на основании](#)

[Удалить](#)

Шаблон проекта

[virtual](#)

This is a virtual project

Кнопка «Создать проект» открывает форму добавления нового проекта на основе выбранного шаблона. В форме необходимо задать имя проекта, а также можно опционально добавить лейблы и аннотации.

В центральной части формы выбирается шаблон проекта, на основе которого будут созданы необходимые ресурсы. При необходимости оставить комментарий.

В нижней части формы предусмотрены поля для ввода параметров шаблона, а также для отображения его структуры.

Система / Управление системой / Проекты / **Проекты**

Новый проект

Имя *

Конфигурация Мета

Комментарий к проекту

Шаблон проекта *

default

Для проекта будут созданы ресурсы, определенные в шаблоне проекта.

Конфигурация YAML

Administrators * ⓘ

Name *	Subject *
	User

Добавить

Cluster Log Destination Name ⓘ

Создать

Подраздел «Шаблоны проектов»

Подраздел предназначен для создания шаблонов проектов.

Шаблоны проектов по умолчанию включают базовые сценарии использования и демонстрируют возможности DKP. Для добавления нового шаблона используется кнопка «Создать» в правом верхнем углу.

Система / Управление системой / Проекты

Шаблоны проектов ⓘ

Создать

Поиск по имени +

Имя	Статус	Описание
12	готов	—
commander-basic	готов	A template that covers basic project use cases.
default	готов	A template that covers basic project use cases.
secure	готов	Includes all the capabilities of the default template and additional features.
secure-with-dedicated-nodes	готов	Includes all the capabilities of the secure template and additional features.

Форма «Новый шаблон проекта» позволяет задать имя шаблона, а также добавить лейблы и аннотации для его идентификации.

В форме представлены две вкладки:

- «Схема OpenAPI» — для описания спецификации значений в формате JSON;
- «Шаблон ресурсов проекта» — для определения ресурсов, совместимых с Helm, и управления окружением проекта.

Deckhouse

Поиск

Управление системой

Deckhouse

Обзор

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Е

Д

?

+ YAML

Система / Управление системой / Проекты / Шаблоны проектов

default

Deckhouse

Конфигурация

Мета

YAML

Схема openAPI

Шаблоны ресурсов проекта

```
1 properties:
2   administrators:
3     description: |
4       Users and groups that will have admin access to the project.
5       Administrators are eligible to manage roles and access to the project.
6     items:
7       properties:
8         name:
9           description: |
10            The name of the target resource to apply access
11            to the environment.
12            minLength: 1
13            type: string
14          subject:
15            description: |
16              Kind of the target resource to apply access to the
17              environment ('Group' or 'User').
18            enum:
19              - User
20              - Group
21            type: string
22          required:
23            - subject
24            - name
25          type: object
26          type: array
27        clusterLogDestinationName:
28          description: >
29            If specified, the project will be monitored by the Deckhouse log shipper
30            and send logs to the specified cluster log destination.
```

Опишите спецификацию OpenAPI для значений (values), которые используются в описанном шаблоне. Данные сохраняются в API в формате JSON, поэтому YAML-комментарии будут проигнорированы.

Deckhouse

Поиск

Управление системой

Deckhouse

Обзор

Обновления

Модули

Глобальные настройки

Оператор платформы

Kubernetes

Проекты

Проекты

Шаблоны проектов

Неймспейсы

Узлы

Группы узлов

Узлы

Конфигурации групп узлов

Классы машин

Статические узлы

Конфигурация

Е

Д

?

+ YAML

Система / Управление системой / Проекты / Шаблоны проектов

default

Deckhouse

Конфигурация

Мета

YAML

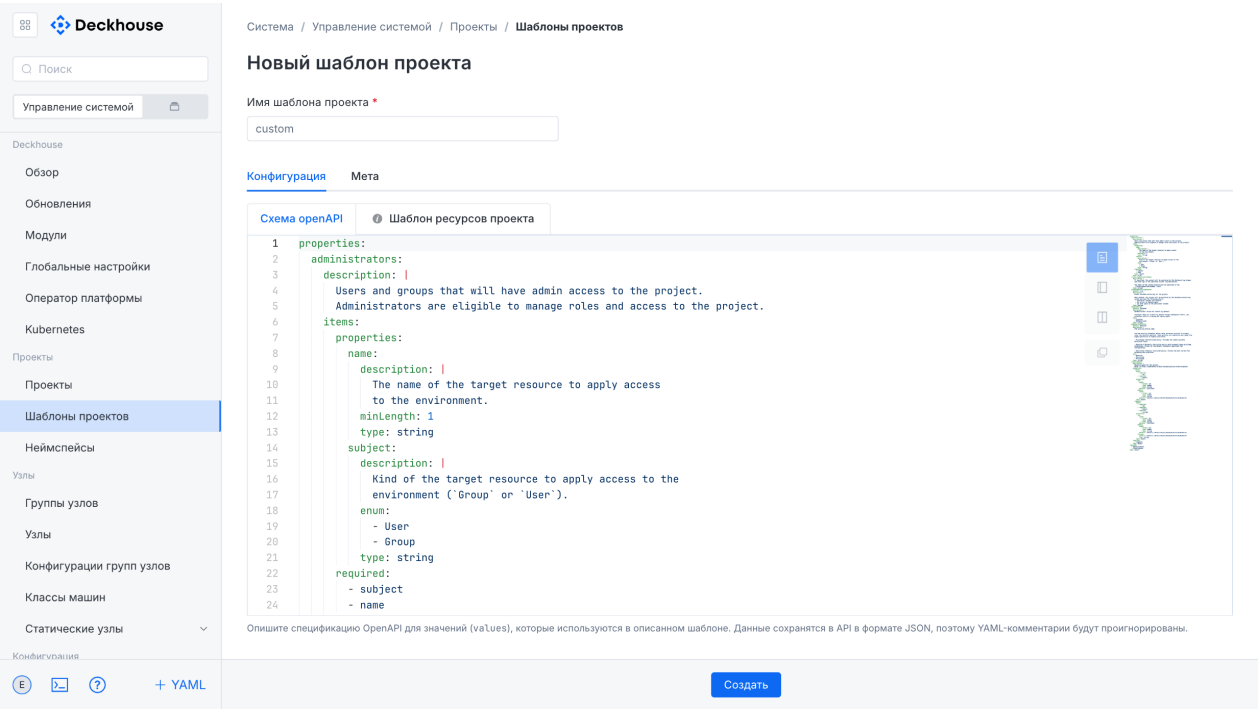
Схема openAPI

Шаблоны ресурсов проекта

```
1 ---
2 apiVersion: v1
3 kind: Namespace
4 metadata:
5   name: {{ .projectName }}
6   labels:
7     {{ with .parameters.podSecurityProfile }}security.deckhouse.io/pod-policy: "{{ lower . }}"{{ end }}
8     {{ if .parameters.extendedMonitoringEnabled }}extended-monitoring.deckhouse.io/enabled: ""{{ end }}
9
10 {{- range $administrator := .parameters.administrators }}
11 ---
12 apiVersion: deckhouse.io/v1alpha1
13 kind: AuthorizationRule
14 metadata:
15   name: {{ $administrator.name | include "normalize" }}
16 spec:
17   accessLevel: Admin
18   subjects:
19     - kind: {{ $administrator.subject }}
20       name: {{ $administrator.name }}
21 {{- end }}
22
23 ---
24 # Max requests and limits for resource and storage consumption for all pods in a namespace.
25 # Refer to https://kubernetes.io/docs/concepts/policy/resource-quotas/
26 apiVersion: v1
27 kind: ResourceQuota
28 metadata:
29   name: all-pods
30 spec:
```

Шаблоны совместимы со всеми функциями helm. Читайте подробнее про создание изолированных окружений

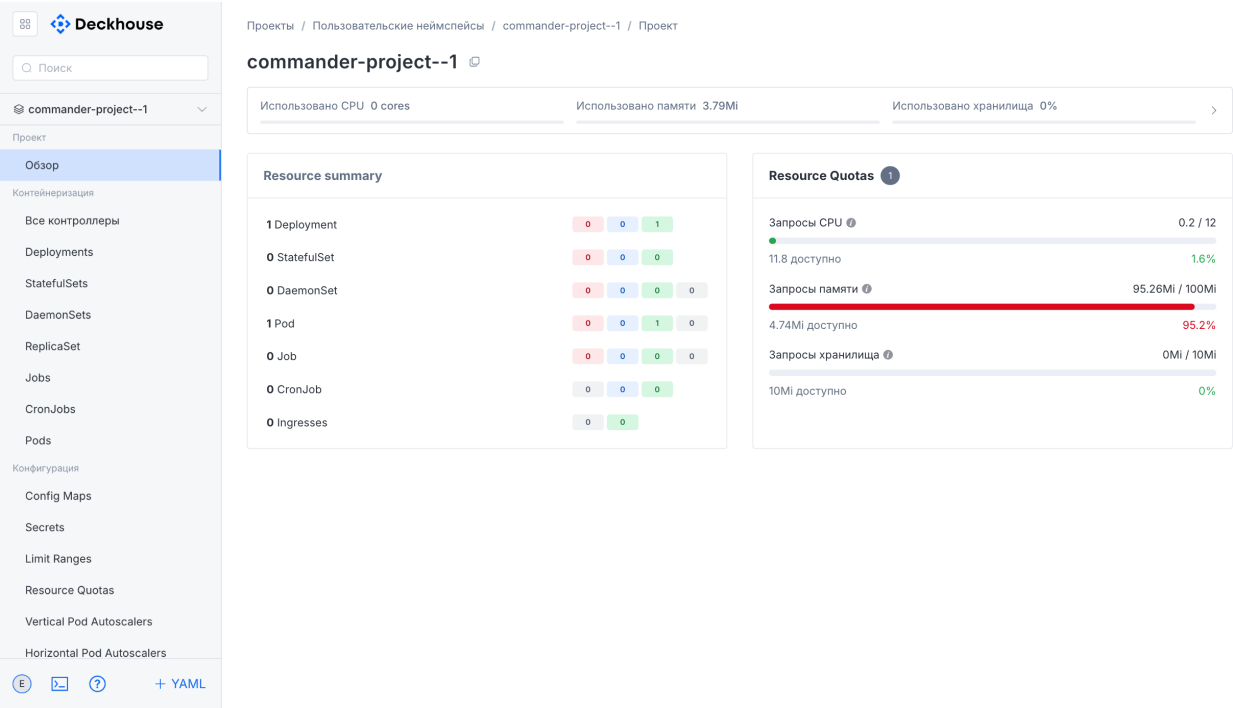
Новый шаблон можно создать на основе уже существующего.



Подраздел «Пользовательские неймспейсы»

Подраздел предназначен для управления пользовательскими неймспейсами проектов.

В секции «Обзор» предоставлена информация об общем состоянии неймспейса.



Здесь можно управлять параметрами Deployment и других сущностей кластера.

8888

Deckhouse

Поиск

commander-project--1

Проект

Обзор

Контейнеризация

Все контроллеры

Deployments

StatefulSets

DaemonSets

ReplicaSet

Jobs

CronJobs

Pods

Конфигурация

Config Maps

Secrets

Limit Ranges

Resource Quotas

Vertical Pod Autoscalers

Horizontal Pod Autoscalers

Е

📄

?

+ YAML

Проекты / Пользовательские неймспейсы / commander-project--1 / Контейнеризация / Deployments

nginx

Конфигурация

Поды

Мониторинг

События

Мета

YAML

Контроллер

Отдельно от подов

Совместно с подами

На узлах

По доменам

Под:Размещение

Обязательные условия размещать отдельно

ДОБАВИТЬ

Под:Контейнеры

Под:Томы

Под:Безопасность

Под:Сеть

Под:Прочее

Под:Мета

Предпочтительные условия размещать отдельно

ДОБАВИТЬ

Кроме того, можно настраивать лимиты и квоты ресурсов.

8888

Deckhouse

Поиск

commander-project--1

Проект

Обзор

Контейнеризация

Все контроллеры

Deployments

StatefulSets

DaemonSets

ReplicaSet

Jobs

CronJobs

Pods

Конфигурация

Config Maps

Secrets

Limit Ranges

Resource Quotas

Vertical Pod Autoscalers

Horizontal Pod Autoscalers

Е

📄

?

+ YAML

Проекты / Пользовательские неймспейсы / commander-project--1

Limit Ranges

Поиск по имени

+

Имя

Container

Создать ресурс

Имя *

Конфигурация

Мета

YAML

Определения ограничений *

Тип объекта *

Container

Pod

PersistentVolumeClaim

Минимально обязательно

Мин. CPU

100m

Мин. память

10Mi

Мин. временное хранилище

10Mi

Максимально разрешено

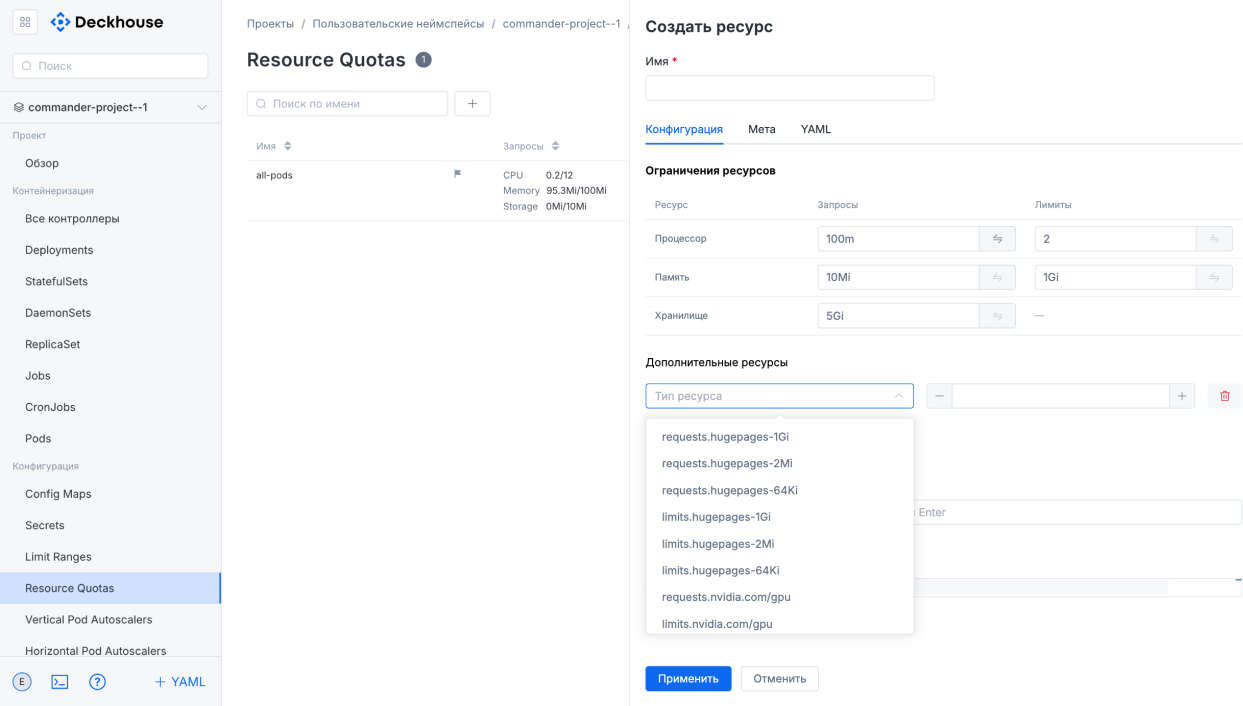
Лимиты по умолчанию

Запросы по умолчанию

Максимальное соотношение Лимит/Запрос

Применить

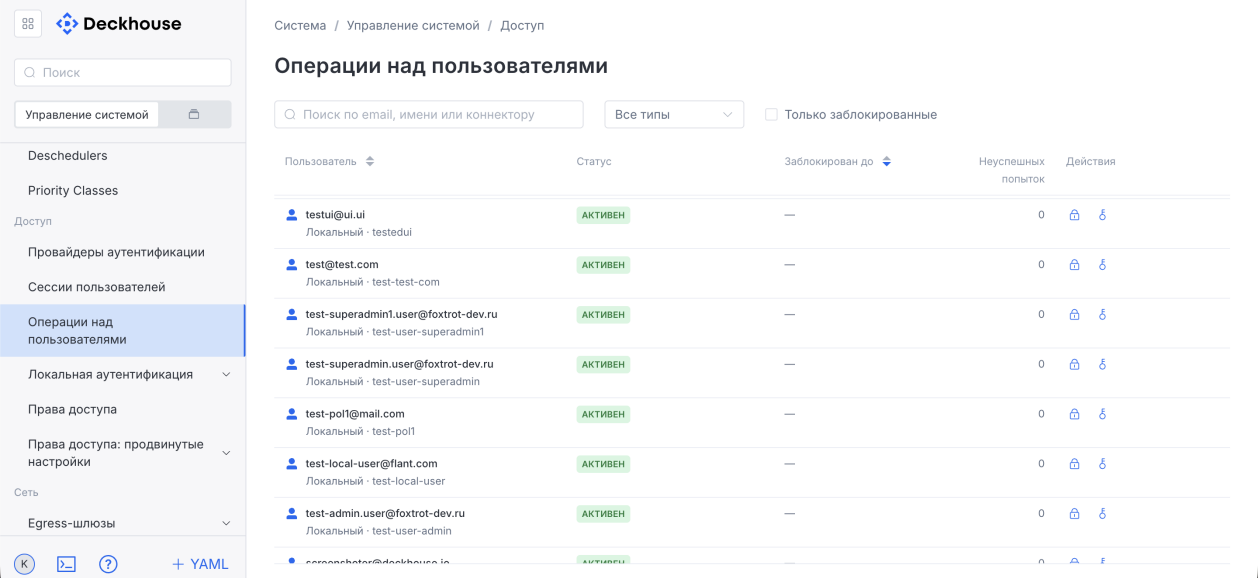
Отменить



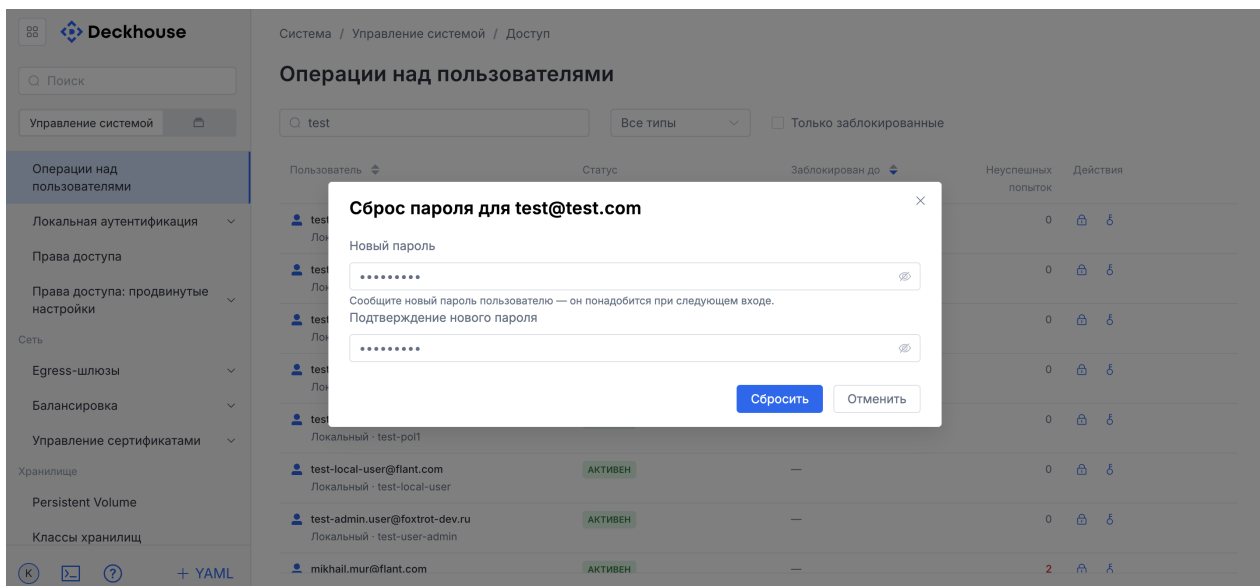
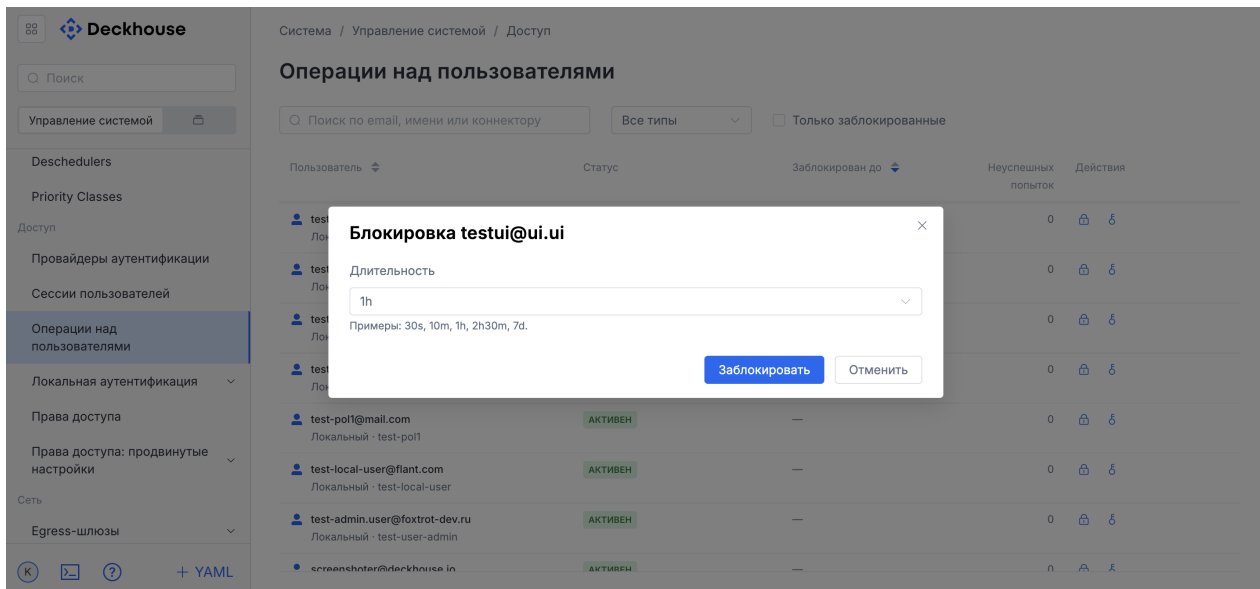
Доступ

Подраздел «Операции над пользователями»

Подраздел предоставляет интерфейс управления пользователями.



При необходимости вы можете заблокировать пользователя или сбросить пароль.



Сеть

Подраздел «Ингресс-контроллеры»

В подразделе отображается информация о текущих Ingress-контроллерах, обеспечивающих маршрутизацию трафика внутри кластера DKP. При необходимости вы можете отсортировать список контроллеров и добавить новые.

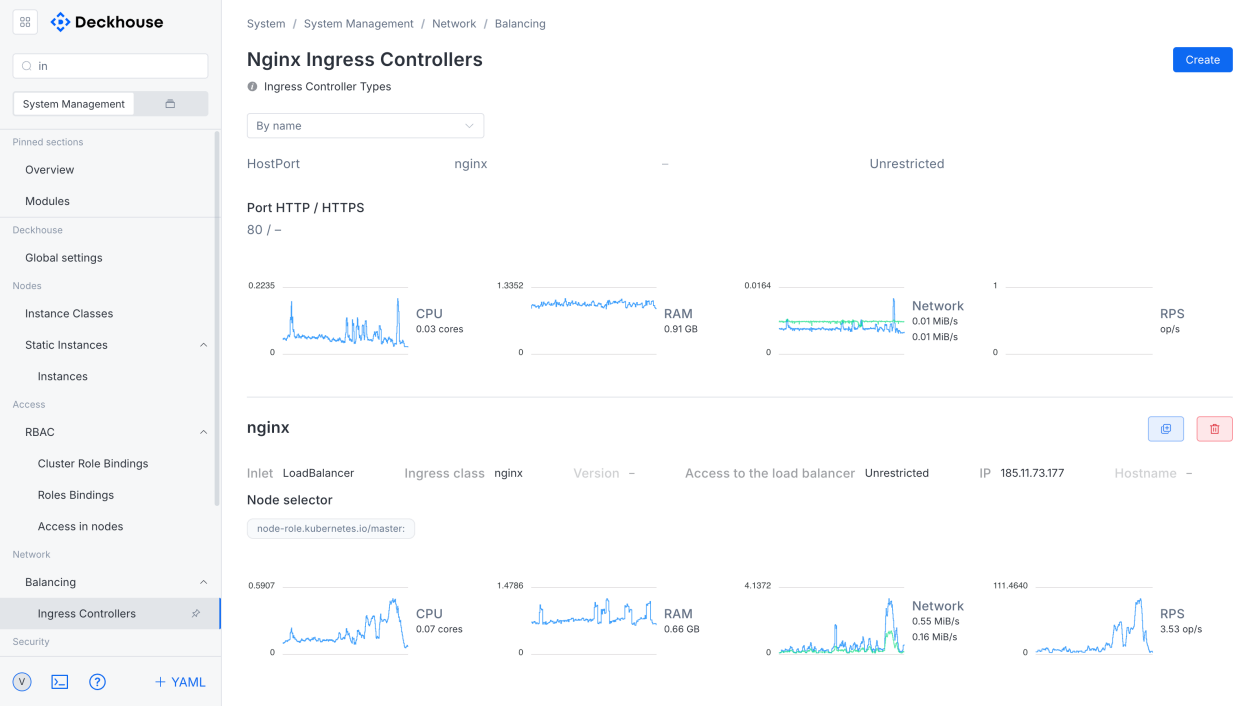
В карточке «Ингресс-контроллеры Nginx» отражены его основные параметры:

- тип входящего подключения (LoadBalancer);
- IP-адрес, Ingress-класс (nginx);
- уровень доступа к балансировщику;
- селектор узлов, который определяет, на каких узлах работает контроллер.

В нижней части представлены графики мониторинга загрузки CPU, памяти, сетевого трафика и количества запросов в секунду (RPS), что позволяет отслеживать производительность контроллера. Доступны опции «Создать на основании» для клонирования конфигурации и «Удалить» для удаления Ingress-контроллера.

Кнопка «Добавить» в правом верхнем углу позволяет выбрать тип нового входящего подключения (инлета) для Ingress-контроллера. Доступные варианты:

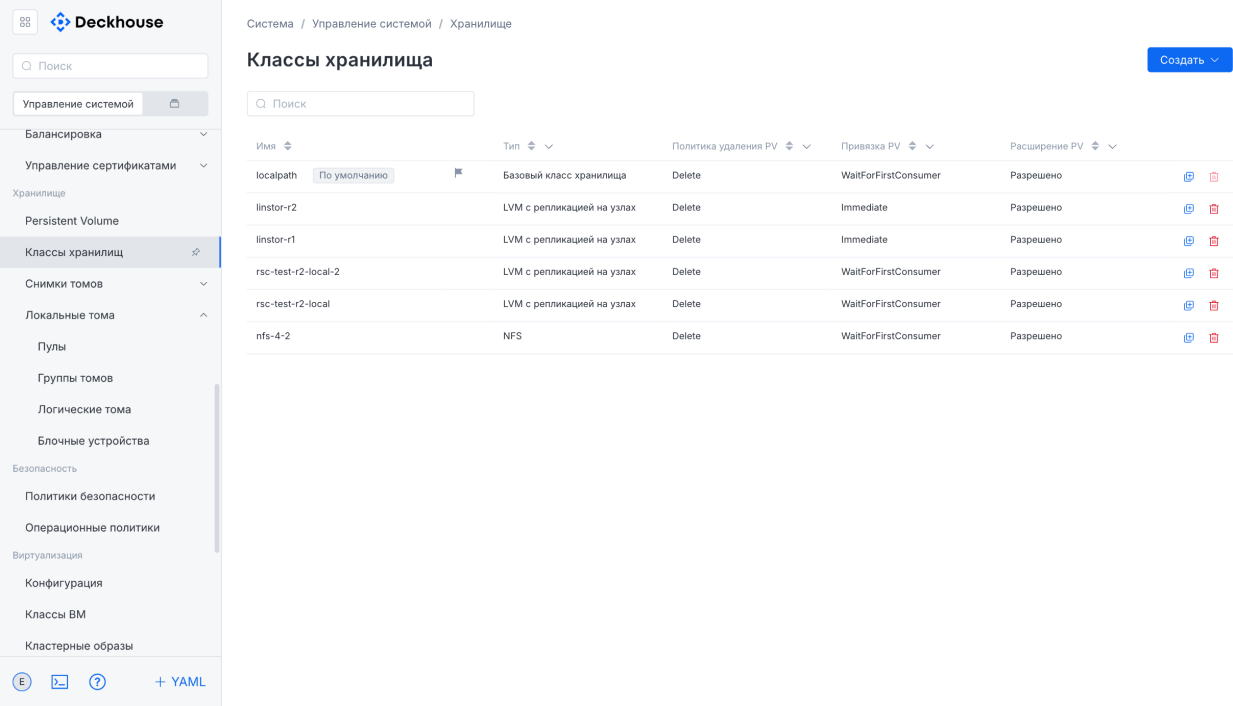
- HostPort;
- HostPort с Proxy Protocol;
- HostPort с резервным контроллером;
- LoadBalancer;
- LoadBalancer с Proxy Protocol.



Хранилище

Подраздел «Классы хранилищ»

Подраздел предназначен для управления классами хранилищ.



Подраздел «Persistent volume»

Подраздел предназначен для управления постоянными томами.

88

Deckhouse

Поиск

Управление системой

Балансировка

Управление сертификатами

Хранилище

Persistent Volume

Классы хранилищ

Снимки томов

Локальные тома

Пулы

Группы томов

Логические тома

Блочное устройства

Безопасность

Политики безопасности

Операционные политики

Виртуализация

Конфигурация

Классы VM

Кластерные образы

Е

Д

?

+ YAML

Система / Управление системой / Хранилище

Persistent Volume13

Создать

Хранилище0.9%

Поиск по имени

Имя	Статус	Вместимость	Режим доступа	Класс хранилища	PVC	
pvc-0988e7ac-b860-47b9-9c84-d e20315602b0	ПРИВЯЗАН	40Gi	ReadWriteOnce	localpath	prometheus-longterm-db- prometheus-longterm-0	 
pvc-0dff77359-687d-4f66-840c-39 be7ae1f563	ПРИВЯЗАН	10Gi	ReadWriteMany	linstor-r1	d8v-vd-cfbc31a5-5c62- 4733-98c3-d0b1edef3bcf- erhwq	 
pvc-1a4e46c0-002d-4e7b-92c2-8f b4ae7f9b93	ПРИВЯЗАН	50Gi	ReadWriteOnce	linstor-r2	nginx	 
pvc-2d1ad0eb-a648-47e6-a651-68 9729e66014	ПРИВЯЗАН	200Mi	ReadWriteOnce	linstor-r1	dataexport-experiment-pvc	 
pvc-3b65dcfa-7736-44ea-8d17-27 70d563a5b0	ПРИВЯЗАН	2Gi	ReadWriteOnce	localpath	data-upmeter-0	 
pvc-534b3252-3f30-4c97-bb09-27 c4247ab161	ПРИВЯЗАН	20Gi	ReadWriteMany	linstor-r1	d8v-vd-lac9e1d6-489c- 4b91-af9a-4b132cd794b- aktyy	 
pvc-8093821a-8c61-4335-ba0f-be eac40f4773	ПРИВЯЗАН	2Gi	ReadWriteMany	nfs-4-2	d8v-vd-ba1e0d0d-d7a1- 4a4f-8bcd-d07348fc5aa1- bknbl	 
pvc-9661549f-aa83-4851-ab57-3d1 fe6b08793	ПРИВЯЗАН	40Gi	ReadWriteOnce	localpath	prometheus-main-db- prometheus-main-0	 
pvc-b84469d6-812f-4317-aded-76 7cdf32de4e	ПРИВЯЗАН	70G	ReadWriteOnce	localpath	dvcr	 

Подраздел «Локальные тома»

Подраздел предназначен для управления локальными томами. Содержит четыре секции для управления пулами, группами томов, логическими томами и блочными устройствами.

88

Deckhouse

Поиск

Управление системой

Балансировка

Управление сертификатами

Хранилище

Persistent Volume

Классы хранилищ

Снимки томов

Локальные тома

Пулы

Группы томов

Логические тома

Блочное устройства

Безопасность

Политики безопасности

Операционные политики

Виртуализация

Конфигурация

Классы VM

Кластерные образы

Е

Д

?

+ YAML

Система / Управление системой / Хранилище / Локальные тома

Пулы

Создать

Поиск

Имя	Статус	Тип	
rsp-e2e	ДОСТУПЕН	LVM	 
thindata	ДОСТУПЕН	LVMThin	 

88

Deckhouse

Поиск

Управление системой

Балансировка

Управление сертификатами

Хранилище

Persistent Volume

Классы хранилищ

Снимки томов

Локальные тома

Пулы

Группы томов

Логические тома

Блочные устройства

Безопасность

Политики безопасности

Операционные политики

Виртуализация

Конфигурация

Классы VM

Кластерные образы

Е

📄

?

+ YAML

Система / Управление системой / Хранилище / Локальные тома

Группы томов

Создать

Поиск

Имя	Статус	Узел	VG на узле	Thin pools	Размер	Выделенный размер	Возраст	
vg-021b789a-4c28-4127-8e15-a12370352026	готов	virtlab-storage-e2e-3	vg0	1/1	44713Gi	44713Gi	год	
vg-0d27d6f8-32fe-438a-91e7-560d16a08cfe	готов	virtlab-storage-e2e-1	vg0	1/1	44713Gi	44713Gi	год	
vg-76911da0-3668-4c03-96ec-4402cc53e5af	готов	virtlab-storage-e2e-2	vg0	1/1	44713Gi	44713Gi	год	
vg-a5c45bfc-b0b2-4ade-a821-e0dfcff91758	готов	virtlab-storage-e2e-0	vg0	1/1	44713Gi	44713Gi	год	
vg-e351ec7f-7587-408c-8cdc-5a63725a8930	готов	virtlab-storage-e2e-4	vg0	1/1	44713Gi	44713Gi	год	

88

Deckhouse

Поиск

Управление системой

Балансировка

Управление сертификатами

Хранилище

Persistent Volume

Классы хранилищ

Снимки томов

Локальные тома

Пулы

Группы томов

Логические тома

Блочные устройства

Безопасность

Политики безопасности

Операционные политики

Виртуализация

Конфигурация

Классы VM

Кластерные образы

Е

📄

?

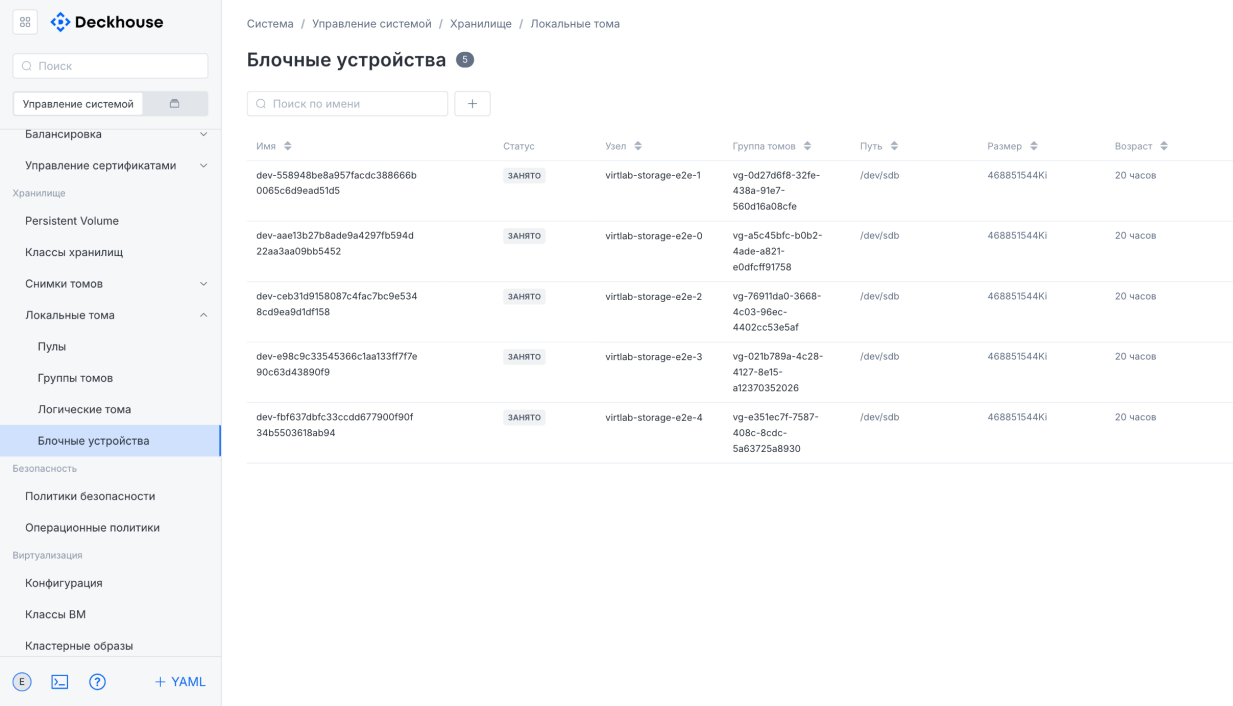
+ YAML

Система / Управление системой / Хранилище / Локальные тома

Логические тома

Поиск

Имя	Статус	Группа томов	Тип	Размер
ceph-osd-0	СОЗДАН	vg-a5c45bfc-b0b2-4ade-a821-e0dfcff91758	Thick	0
ceph-osd-1	СОЗДАН	vg-0d27d6f8-32fe-438a-91e7-560d16a08cfe	Thick	0
ceph-osd-2	СОЗДАН	vg-76911da0-3668-4c03-96ec-4402cc53e5af	Thick	0
ceph-osd-3	СОЗДАН	vg-021b789a-4c28-4127-8e15-a12370352026	Thick	0
ceph-osd-4	СОЗДАН	vg-e351ec7f-7587-408c-8cdc-5a63725a8930	Thick	0



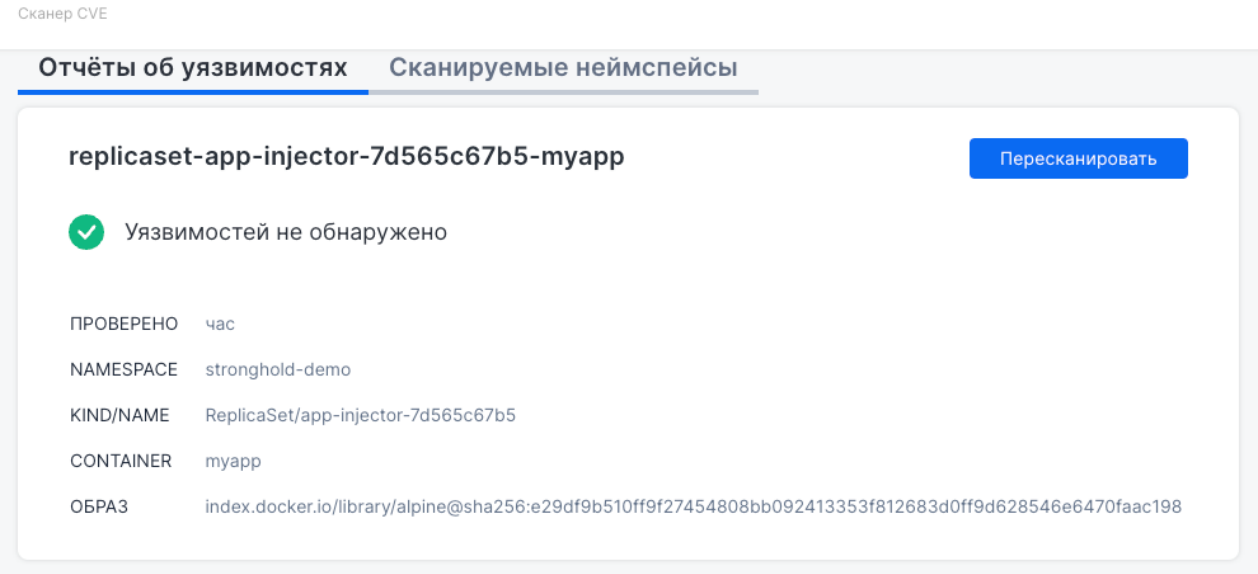
Безопасность

Подраздел «Сканер CVE»

Подраздел предназначен для проверки контейнерных образов на наличие уязвимостей (CVE) в кластере DKP.

В подразделе представлены вкладки «Отчёты об уязвимостях» и «Сканируемые неймспейсы»:

- Вкладка «Отчеты об уязвимостях» отображает результаты последних сканирований. Здесь представлена информация о проверенном объекте, включая его имя, неймспейс, тип и имя ресурса, контейнер, а также используемый образ. Если уязвимости не обнаружены, отображается зеленый индикатор. Кнопка «Пересканировать» позволяет выполнить повторное сканирование.
- Вкладка «Сканируемые неймспейсы» предназначена для управления неймспейсами, которые подлежат сканированию. Интерфейс поддерживает сортировку по имени и параметрам сканирования. Опционально можно скрыть системные неймспейсы. Доступен выбор нужных неймспейсов для сканирования и запуск процессов проверки кнопкой «Пересканировать», а также просмотр отчётов по каждому объекту.



Мониторинг

Подраздел «Обзор»

В подразделе отображается состояние экземпляров Prometheus, метрики нагрузки и секция с настройками.

Подраздел «Обзор» включает вкладки «Состояние» и «Конфигурация», предназначенные для мониторинга и настройки экземпляров Prometheus.

На вкладке «Состояние» отображается список работающих подов с указанием имени, узла размещения, статуса, IP-адреса, возраста, а также загрузки CPU и памяти. Для каждого пода указаны компоненты, такие как `init-config-reloader`, `prometheus`, `config-reloader` и `kube-rbac-proxy`, обеспечивающие его работу. Кнопка справа позволяет удалить выбранный под.

Обзор

Состояние

Конфигурация

У всех подов общий неймспейс d8-monitoring

Имя	Узел	Статус	Возраст, IP	ЦП	Память
prometheus-longterm-0	sandbox1-worker-26ff06f6-85899-zjrvx	Running init: init-config-reloader prometheus config-reloader kube-rbac-proxy	7 часов 10.111.1.217	0.0287	0.5665
Лейблы и аннотации					
prometheus-main-0	sandbox1-worker-26ff06f6-85899-zjrvx	Running init: init-config-reloader prometheus config-reloader kube-rbac-proxy	7 часов 10.111.1.145	0.1185	1.0270
Лейблы и аннотации					

На вкладке «Конфигурация» представлены раскрывающиеся секции для настройки различных аспектов работы Prometheus, включая оперативные и ретроспективные метрики, аутентификацию и подключение к Grafana, а также управление ресурсами. Этот интерфейс предназначен для отслеживания состояния метрик в реальном времени и гибкой настройки интеграции с другими сервисами.

Обзор

Состояние Конфигурация

Темная тема интерфейса по умолчанию



Дополнительные плагины

Имена плагинов из официального репозитория

[+ ДОБАВИТЬ](#)

- > ● Prometheus для оперативных метрик (main)
- > ● Prometheus для ретроспективных метрик (longterm)
- > ● Аутентификация в Grafana и Prometheus
- > ● Подключение к Grafana и Prometheus
- > ● Управление ресурсами Prometheus

Подраздел «Обработка метрик»

Подраздел позволяет создавать и управлять правилами обработки метрик. При добавлении нового правила задайте его название и укажите группу обработки. Это даст возможность организовывать и модифицировать поступающие метрики перед их дальнейшей передачей.

Новое правило обработки метрик

Название*

[+ Добавить группу](#)

Подраздел «Отправка метрик»

Подраздел предназначен для настройки экспорта данных в локальный или внешний сервер Prometheus. В процессе добавления нового ресурса указываются URL-адреса для отправки метрик, а также настраиваются параметры TLS, аутентификация и возможность предварительной обработки метрик перед отправкой.

Ресурс для включения данных из локального Prometheus

Название*

Конфигурация

URL для отправки метрик*

> ● Параметры TLS ☐

> ● Аутентификация

Обработка метрик перед отправкой

Дает возможность удалить метрики или произвести замену лейблов.
Подробнее про конфигурацию — [в документации Prometheus](#)

Подраздел «Источники для Grafana»

В подразделе можно настроить интеграцию Grafana с различными источниками данных, используемыми в дашбордах, и визуализировать данные.

Чтобы создать новый источник данных, задайте его название, тип, URL-адрес, параметры доступа и настройки аутентификации.

Добавить новый источник данных для Grafana

[Создать на основании](#)
[Удалить](#)

Название источника *

Тип datasource *

Подберите тип в официальной документации или в каталоге плагинов Grafana



Параметры ресурса подробно описаны в документации к Grafana

URL

Доступ к данным *

Параметры для jsonData

Базовая авторизация



Защищенные параметры secureJsonData

Использовать withCredentials при запросах



Подробнее о [withCredentials](#)

> • База Данных

Подраздел «Дашборды для Grafana»

Этот подраздел предназначен для управления дашбордами, используемыми для визуализации метрик.

В основном интерфейсе отображается список доступных дашбордов. Предусмотрена сортировка по времени создания, а также фильтрация по имени или каталогу.

В строке дашборда отображается его имя, принадлежность к папке, а также опции «Создать на основании» для добавления нового экземпляра на его основе и «Удалить».

Чтобы добавить новый дашборд:

1. Нажмите кнопку «Добавить».
2. Задайте имя нового дашборда и папку, в которой он будет храниться (если папка не существует, она будет автоматически создана).
3. Введите JSON-манифест, содержащий описание конфигурации дашборда. Важно, чтобы в манифесте не было локального ID (кроме UID), иначе дашборд может некорректно отображаться в Grafana.

Дашборды для Grafana

Сортировать по:

времени создания (сначала новые) ▾

+ Добавить

Фильтр: ✕

Поиск по имени

Поиск по каталогу ▾

d8-console-main-backend

Deckhouse

Создать на основании

Удалить

Папка

Main

Подраздел «Активные алерты»

В этом подразделе отображается список текущих предупреждений системы мониторинга. Алерты можно сортировать по имени и статусу.

В строке алерта отображается его имя, уровень критичности, информацию о времени создания и последнего обновления. Дополнительно указываются связанные компоненты и модули, которые помогают определить источник проблемы.

Чтобы получить подробную информацию об алерте, нажмите «Читать описание». В самом низу отображается пояснение о причине срабатывания алерта.

Активные алерты

Сортировать по:

имени ▾

Фильтр: ✕

Поиск по имени

Статус: все ▾

D8DeckhouseIsNotOnReleaseChannel

Критичность: 9

Читать описание

СОЗДАНО 7 часов

ОБНОВЛЕНО несколько секунд

d8_component: deckhouse

d8_module: deckhouse

prometheus: deckhouse



Deckhouse in the cluster is not subscribed to one of the regular release channels.

Список всех доступных алертов системы мониторинга DKP приведён на отдельной странице документации.

Журналирование

Раздел предназначен для настройки источников логов и управления их отправкой.

Подраздел «Отправка логов»

Подраздел предназначен для управления логированием и настройкой отправки логов в

различные хранилища.

Чтобы настроить отправку логов:

1. Нажмите кнопку «Добавить».
2. Выберите одно из целевых хранилищ:
 - Loki;
 - ElasticSearch;
 - Logstash;
 - Vector;
 - Kafka;
 - Splunk.
3. Задайте название хранилища и адрес подключения.
4. При необходимости задайте дополнительные настройки:
 - TLS-параметры для безопасного соединения;
 - аутентификация («Basic» или «Bearer-токен»);
 - дополнительные лейблы для фильтрации и организации логов;
 - параметры буфера, определяющие способ хранения логов перед отправкой (на диске или в памяти);
 - ограничения отправки, позволяющие задать частоту отправки записей;
 - исключения, позволяющие фильтровать определенные логи.

Тип: Loki

Название *

Параметры подключения

Подключение

Адрес Loki *

Агент автоматически добавляет /loki/api/v1/push к URL при отправке данных

> ● Параметры TLS ☒

Аутентификация

Нет Basic Beamer-токен

Дополнительные лейблы ?

Лейблы будут дописаны в логи. Вы можете использовать простые шаблоны, используя [синтаксис шаблонов Vector](#), например {{ app }}

KEY

VALUE

[+ ДОБАВИТЬ](#)

Параметры буфера

Тип буфера

Не использовать На диске В памяти

Ограничения отправки ☐

Количество записей в минуту *

Выражение, определяющее бакет для раздельного ограничения частоты отправки

Поддерживается [синтаксис шаблонов Vector](#)

Исключения ?

Логи, попавшие под правила, не будут ограничены в частоте отправки

+ Добавить

Подраздел «Сбор логов»

Подраздел предназначен для настройки источников логов, которые затем могут быть отправлены в целевые хранилища. Вы можете отсортировать и отфильтровать существующие правила сбора логов, а также добавить новые источники.

Чтобы настроить источники сбора:

1. Нажмите кнопку «Добавить».
2. Выберите один из источников:
 - File (сбор логов из файловой системы);
 - KubernetesPods (сбор логов из подов Kubernetes).
3. Задайте название и настройте параметры:
 - «Хранилище» — укажите, куда будут отправляться собранные логи;
 - «Фильтр файлов» — задайте пути к файлам логов, которые необходимо или, наоборот, не нужно считывать. Поддерживаются подстановки (wildcards).
 - «Разделитель строк» (опционально) — задайте символ, разделяющий записи в файле;

- «Фильтрация логов» (опционально) — добавьте правила фильтрации по лейблам и фильтры, чтобы сохранять только нужные записи.

Тип: File

Название*

Хранилище

Отправка логов*

Хранилища определены в разделе «Доставка логов» [ClusterLogDestination](#), с которыми будет работать этот источник логов. Поля с числовыми и булевыми типами будут преобразованы в строки.

[+ ДОБАВИТЬ](#)

Фильтр файлов

Пути файлов для чтения

Поддерживаются символы подстановки (wildcards), например `/var/log/*.log`

[+ ДОБАВИТЬ](#)

Пути файлов, которые читать не требуется

Поддерживаются символы подстановки (wildcards), например `/var/log/*.log`

[+ ДОБАВИТЬ](#)

Разделитель между строками

Пример: `\r\n`

Фильтрация логов

Список правил для фильтрации логов по их лейблам ⓘ

[+ Добавить](#)

Список фильтров для логов ⓘ

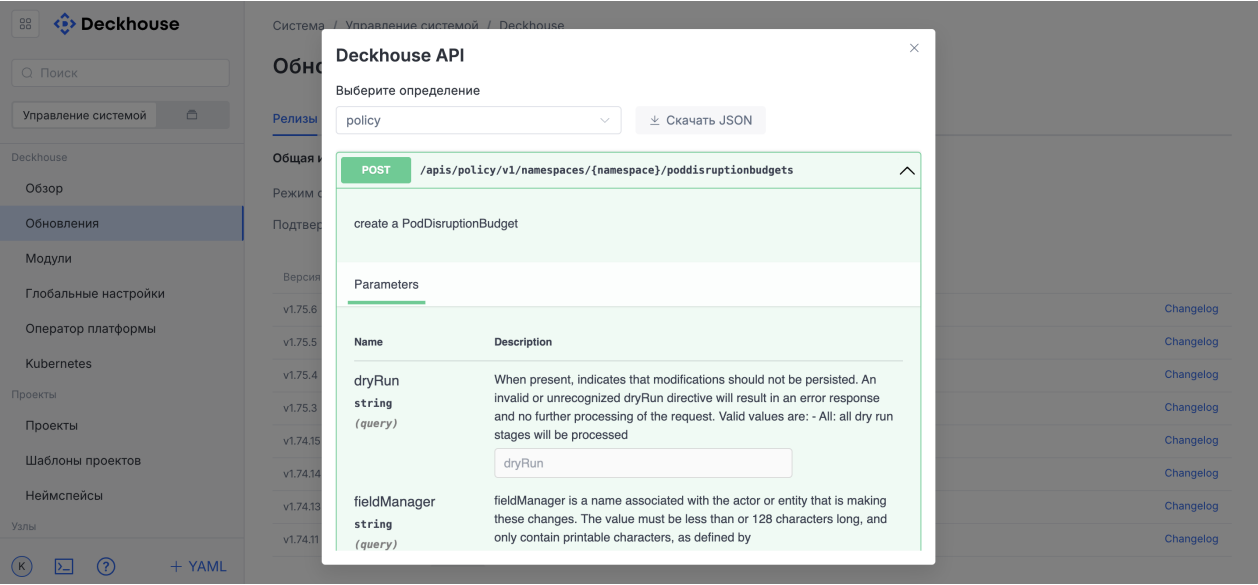
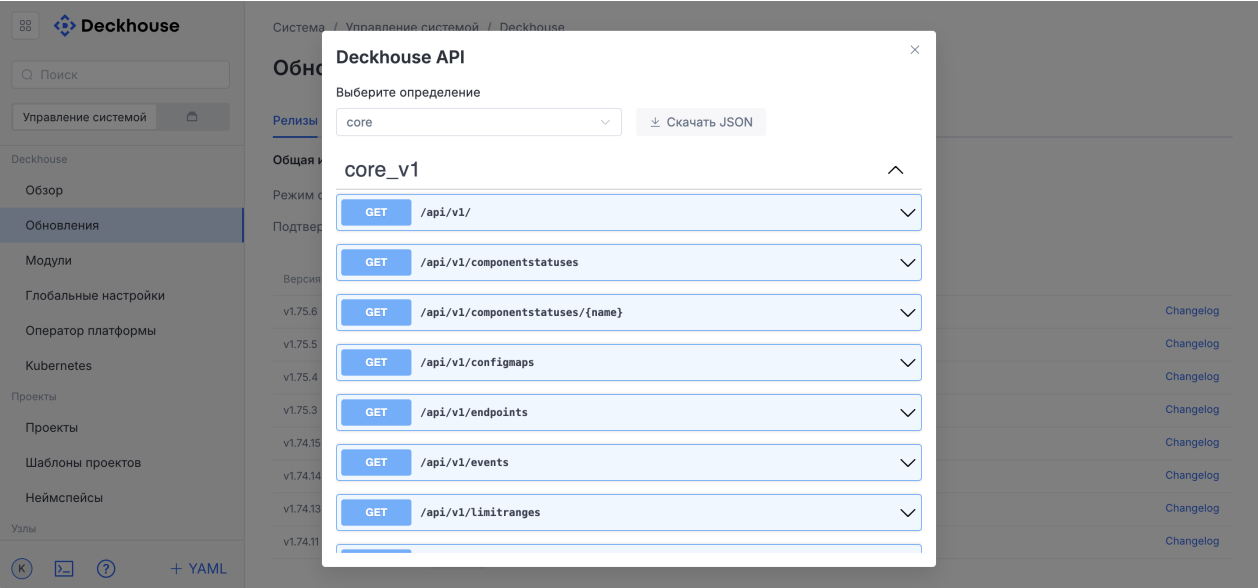
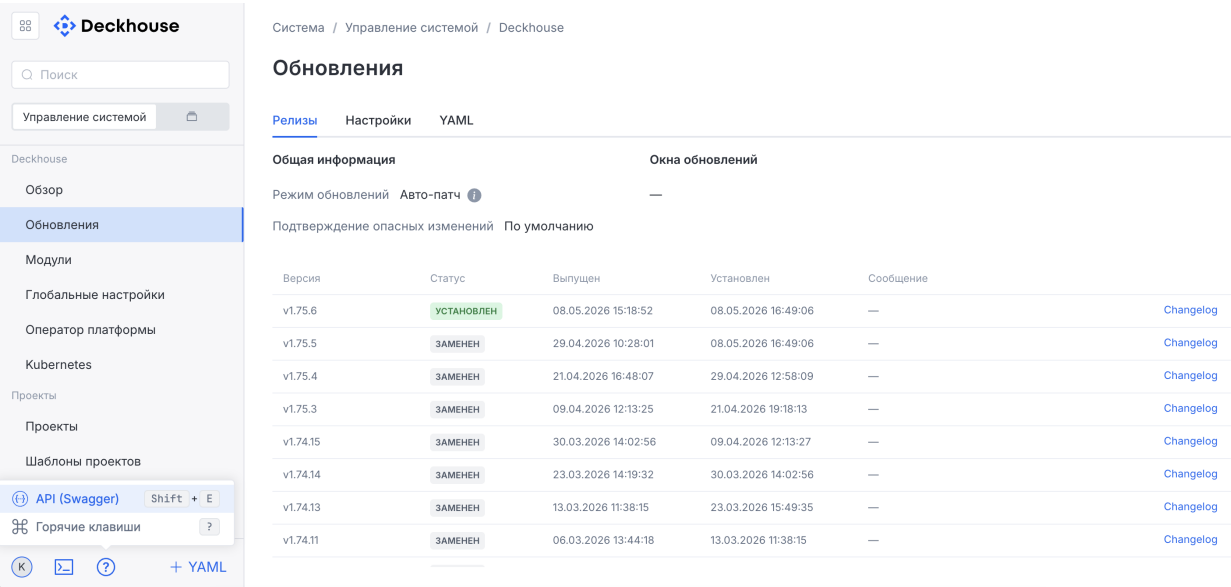
Только логи, подпадающие под правила, будут сохранены в хранилище

[+ Добавить](#)

> • Парсер многострочных логов

Просмотр информации об API

Чтобы просмотреть информацию об API в формате Swagger, нажмите кнопку со знаком вопроса в левом нижнем углу интерфейса в выпадающем пользовательском меню.



Дополнительные ресурсы

- Глобальные параметры
- Документация модуля prometheus

Веб-интерфейс Grafana

Веб-интерфейс системы мониторинга

В качестве веб-интерфейса системы мониторинга используется Grafana.

Главный экран

На главном экране Grafana расположена основная информация о кластере и его основных компонентах.

В левой верхней части экрана указаны характеристики основных компонентов кластера: версия containerd, дистрибутив Linux, на базе которого работает кластер, а также версии Grafana, Prometheus и т.д.

В правой верхней части экрана расположены удобные графические обозначения для основных параметров — количества узлов кластере, количество запущенных в ней подов и других сущностей кластера.

Для получения более подробной информации можно навести на любой элемент курсор мыши, нужная информация отображается во всплывающей подсказке.

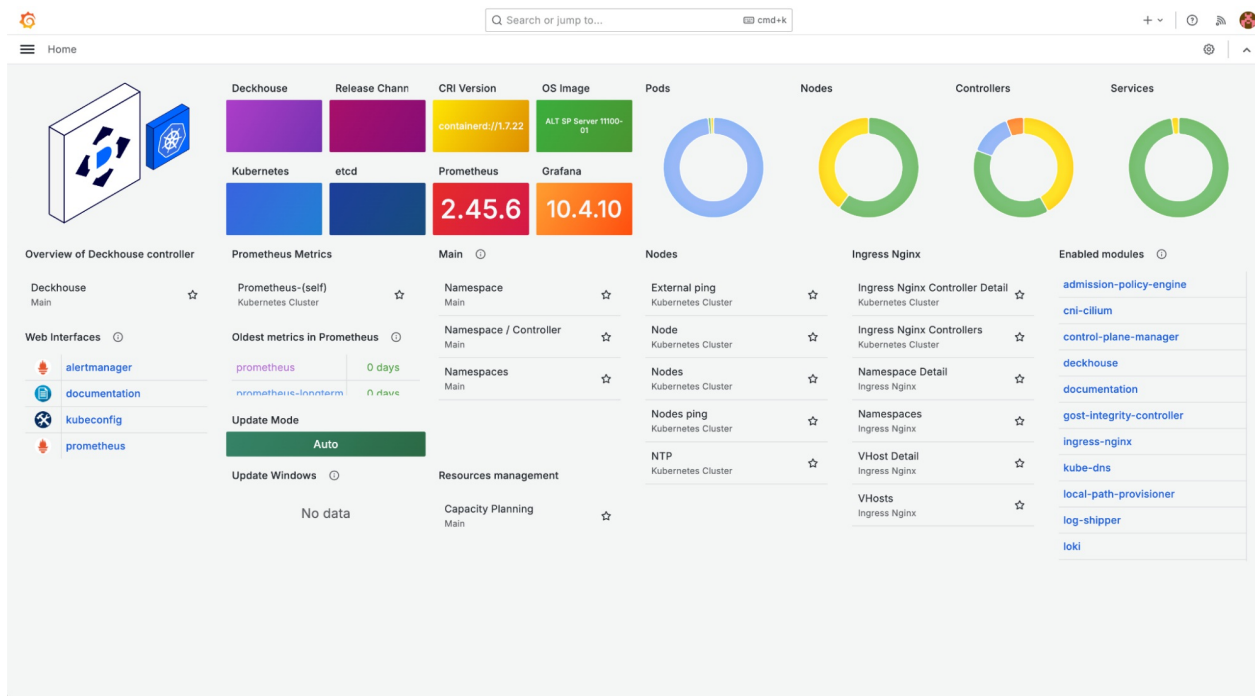
Ниже блоков с характеристиками расположены быстрые ссылки на дашборды некоторых компонентов кластера. Например, на мониторинг узлов кластера, потребления системных ресурсов его компонентами и статистику сетевого взаимодействия.

Левее расположен блок со ссылками на веб-интерфейсы кластера, доступные для пользователя, а также блок с информацией о способе обновления кластера и временных окнах, в которые это обновление должно произойти (если они настроены).

В левом верхнем углу главного экрана расположена кнопка открытия бокового меню, в котором расположены ссылки на основные элементы Grafana.

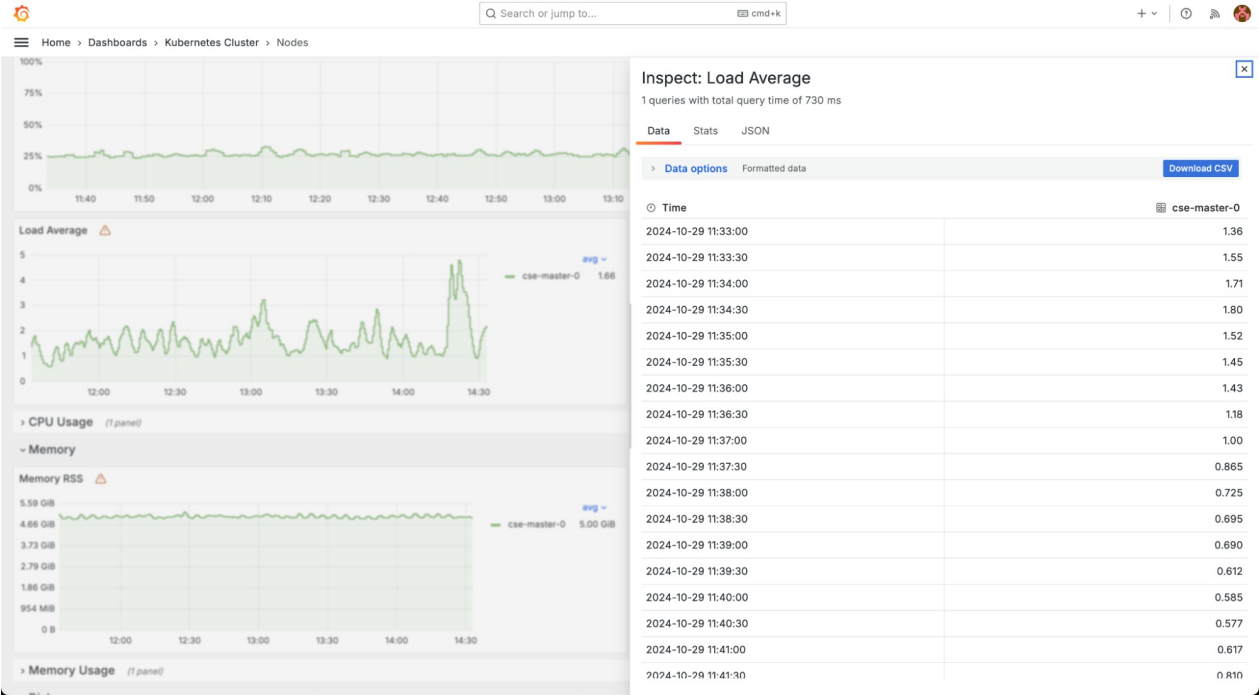
При переходе на вкладку «Dashboards» откроется список всех доступных дашбордов Deckhouse, сгруппированных по категориям.

Они имеют вложенную структуру и сгруппированы по назначению — приложения в кластере (Applications), сетевое взаимодействие (Ingress Nginx), параметры кластера (Kubernetes Cluster), основные (Main) и безопасность (Security).



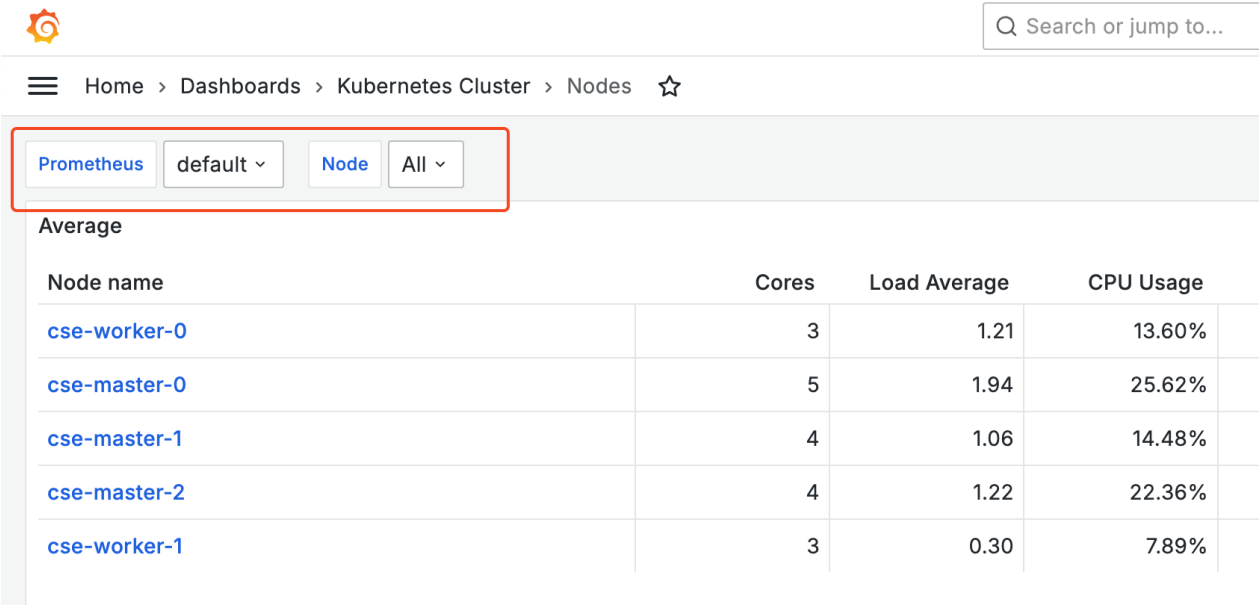
Работа с дашбордами

Дашборд представляет собой экран с расположенными на нем таблицами и графиками, содержащими информацию о выбранном компоненте кластера.

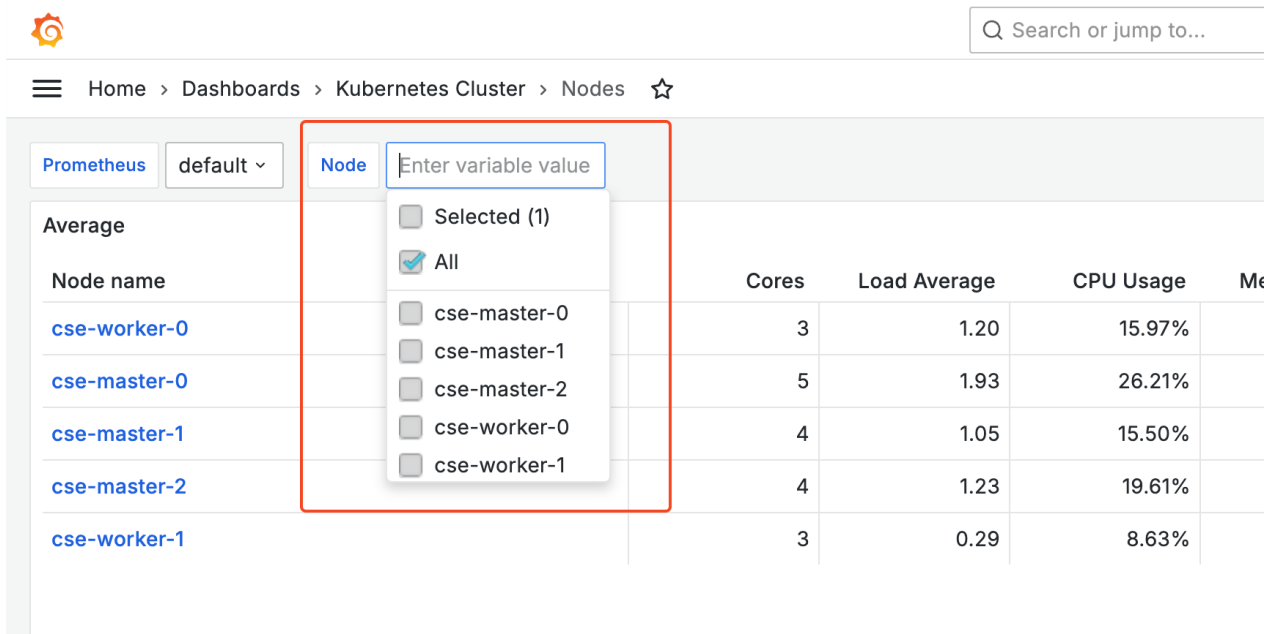


Фильтрация информации

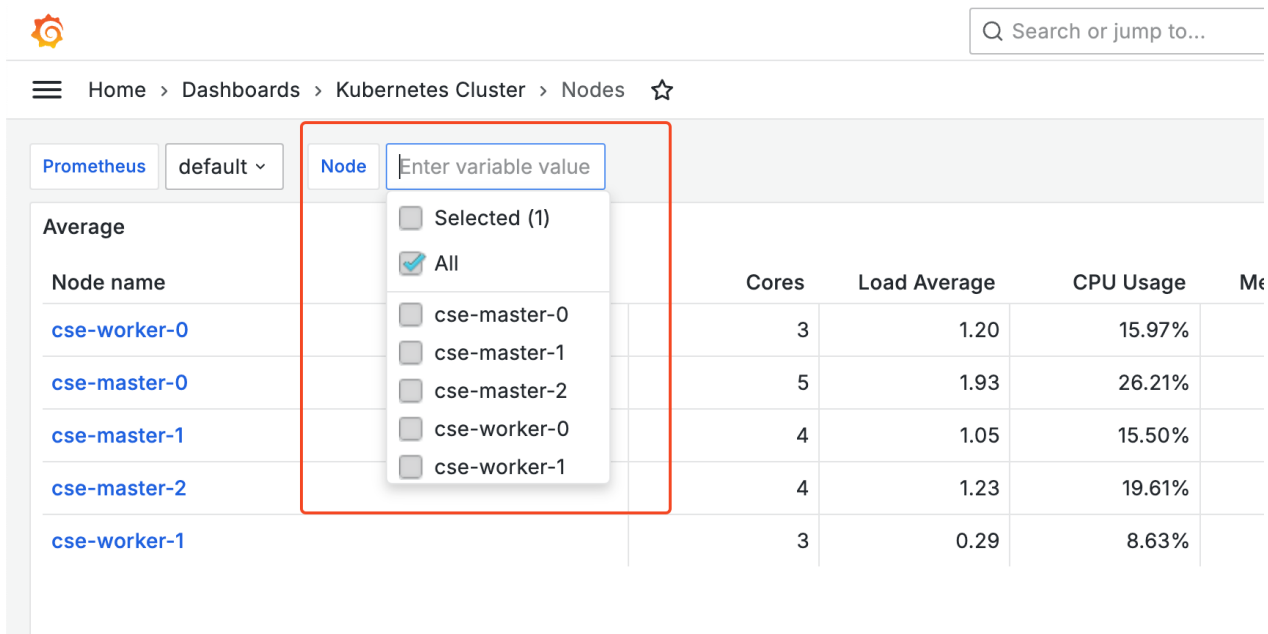
В верхней части под названием и быстрой ссылкой на родительскую категорию располагается блок фильтров, позволяющий настроить отображение, исключив из выдачи несущественную информацию или сконцентрировав выбор на одном конкретном компоненте.



Например, на приведенном выше примере с дашбордом узлов кластера можно задать в фильтре отображение только одного из трех узлов, исключив из выдачи информацию об остальных двух узлах.

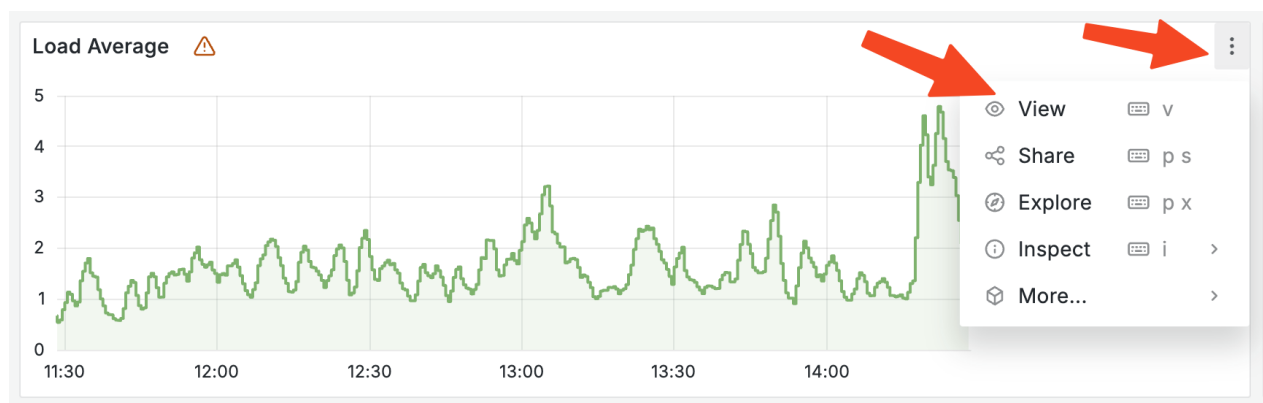


После выбора в фильтре параметров дашборд сразу же изменится, и содержимое будет заменено на соответствующее заданным параметрам фильтрации.

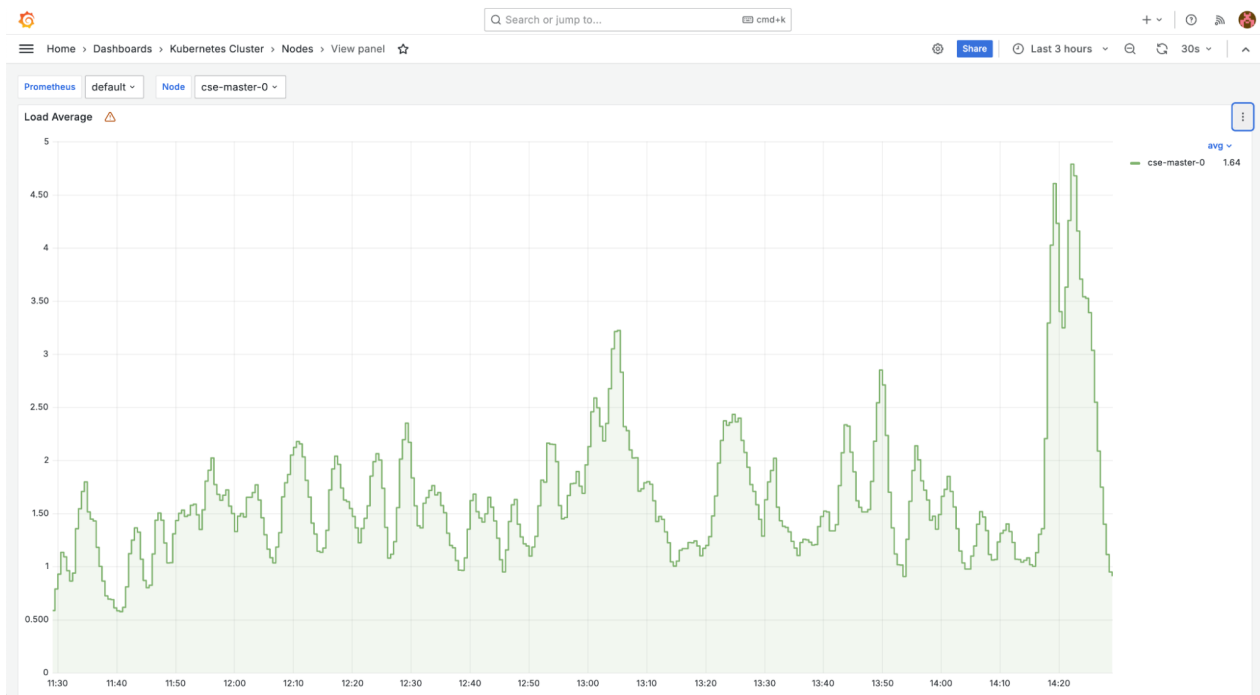


Работа с данными

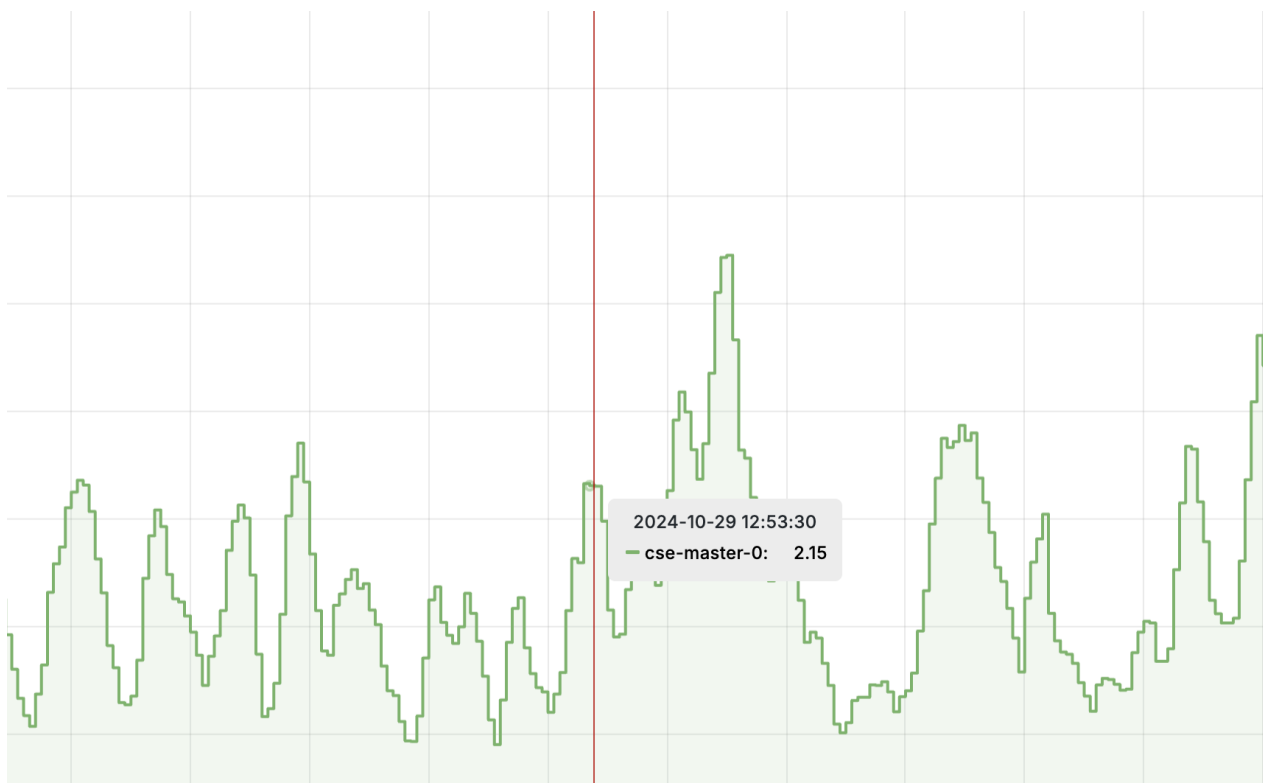
Каждый из представленных на дашборде графиков можно открыть в более подробном виде. Для этого необходимо навести курсор на правый верхний угол блока с графиком, нажать на появившуюся кнопку с тремя точками и выбрать пункт «View».



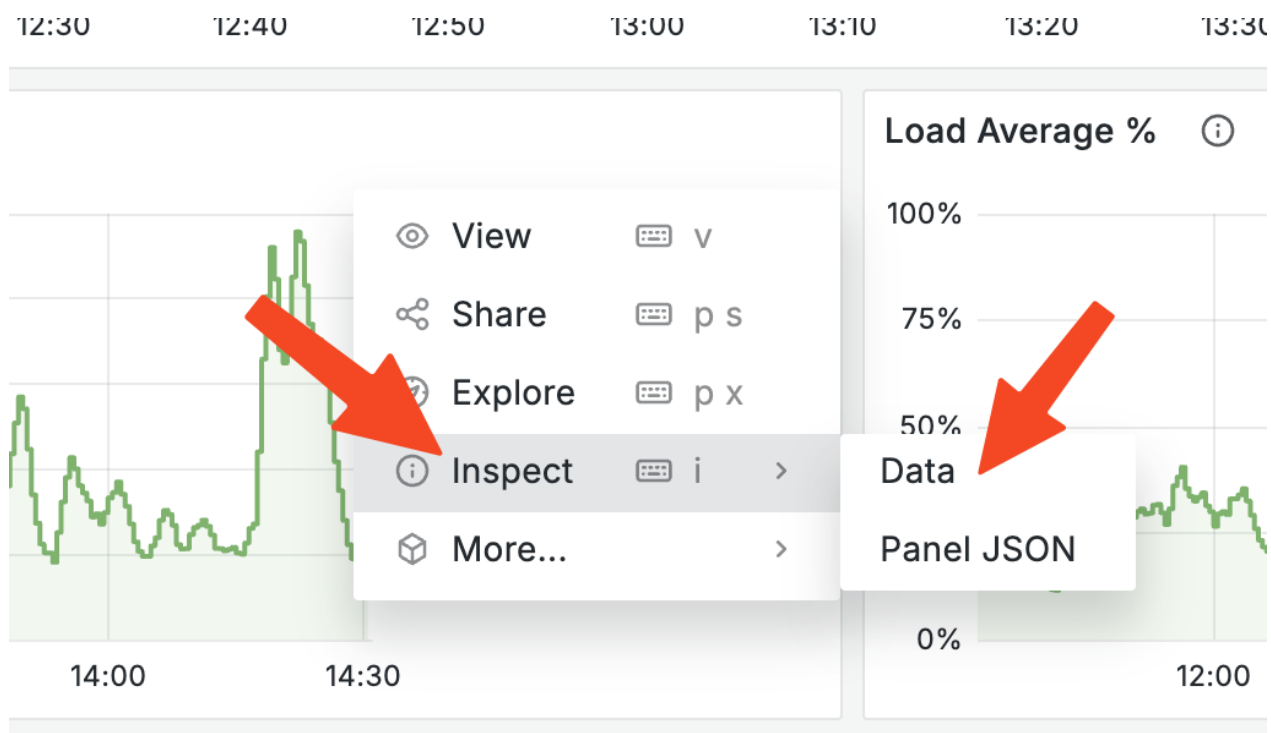
Выбранный график откроется на весь экран.



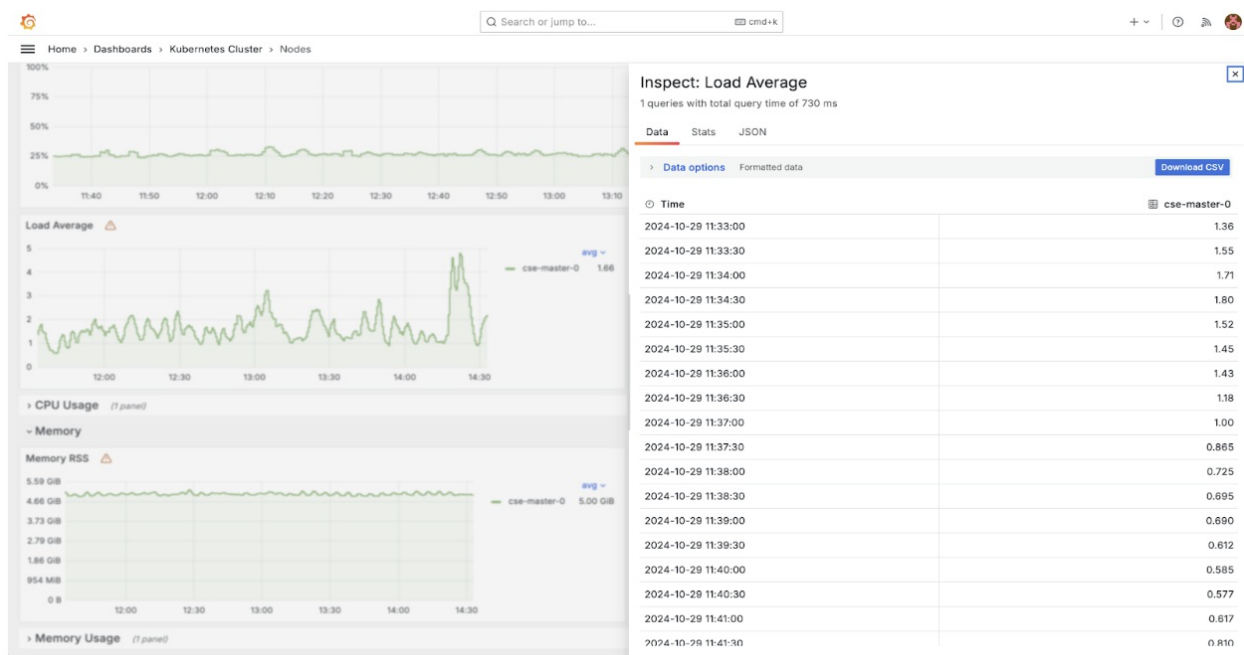
Здесь можно просмотреть более подробно информацию на графике за определенный момент времени. Для этого нужно навести курсор мыши на график — он примет вид красной горизонтальной черты, а рядом с ним отобразится всплывающее окошко с временным лейблом и значением графика на этот момент:



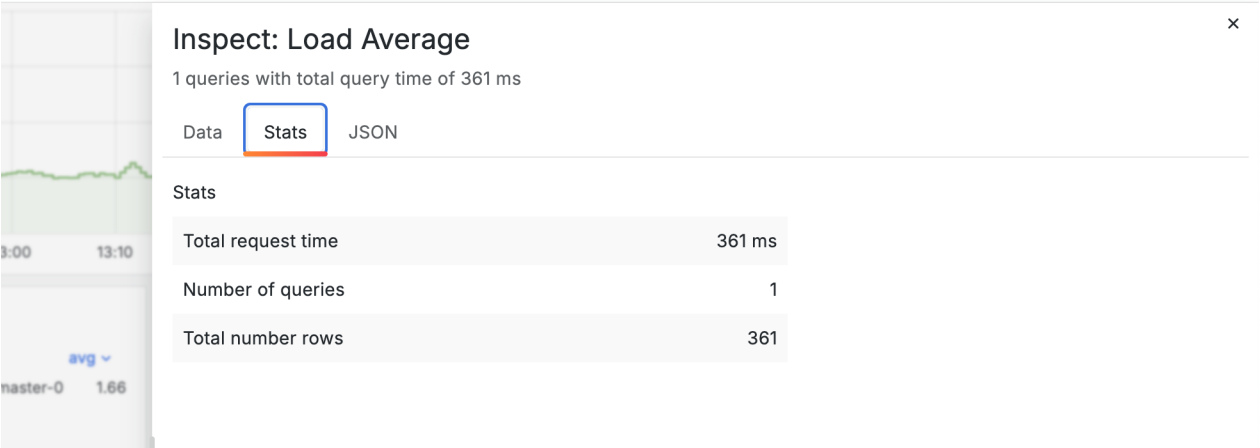
Для перехода обратно на предыдущий экран достаточно нажать «Esc». Просмотреть подробно список записей из которых строится график, можно, выбрав в меню блока (по кнопке с тремя точками в правом верхнем углу блока) пункт «Inspect» и соответствующий запросу подпункт «Data».



В правой части экрана откроется окно с подробным содержанием записей.

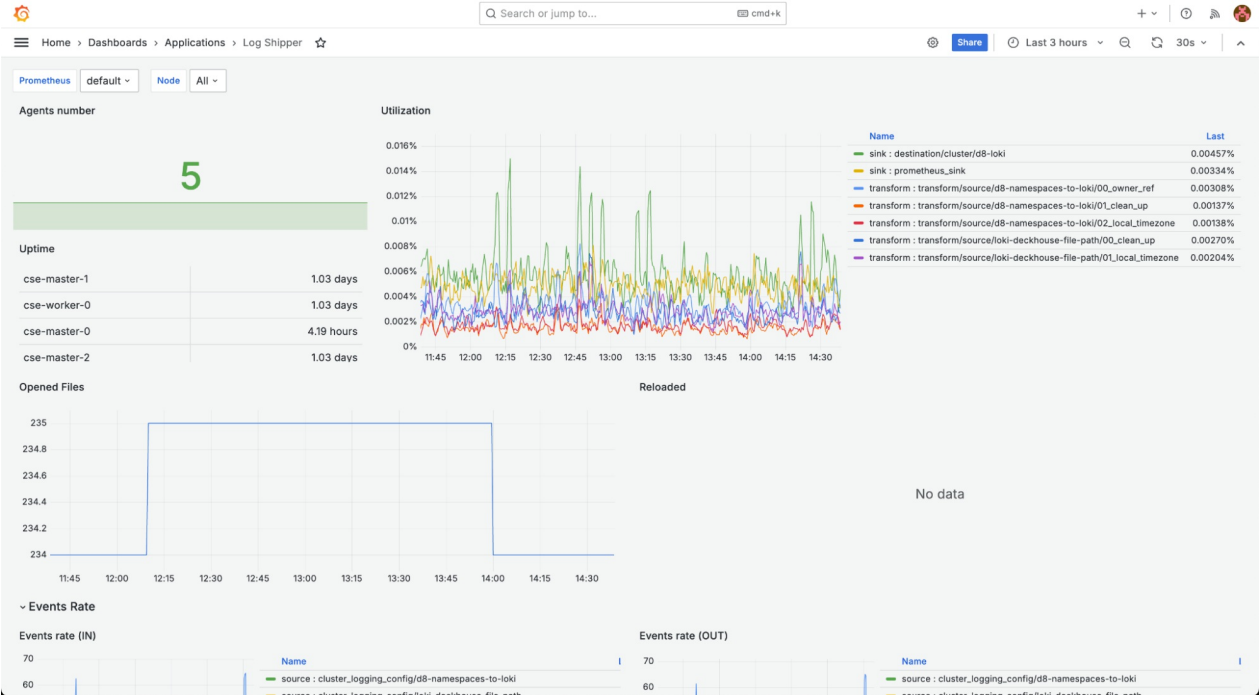


В открывшемся окне отобразятся все данные, из которых построен график. Здесь их также можно скачать в формате CSV и просмотреть общую статистику (например, общее количество записей). Для этого необходимо перейти на вкладку «Stats» окна со с данными.

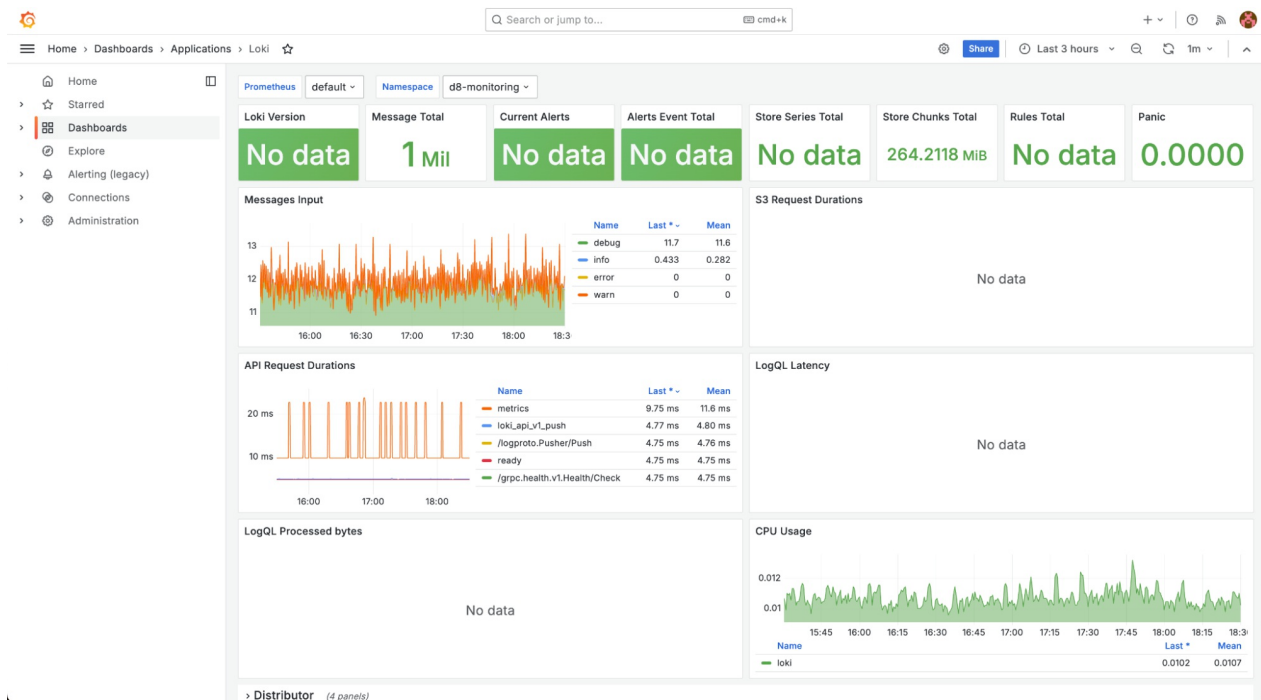


Описание дашбордов

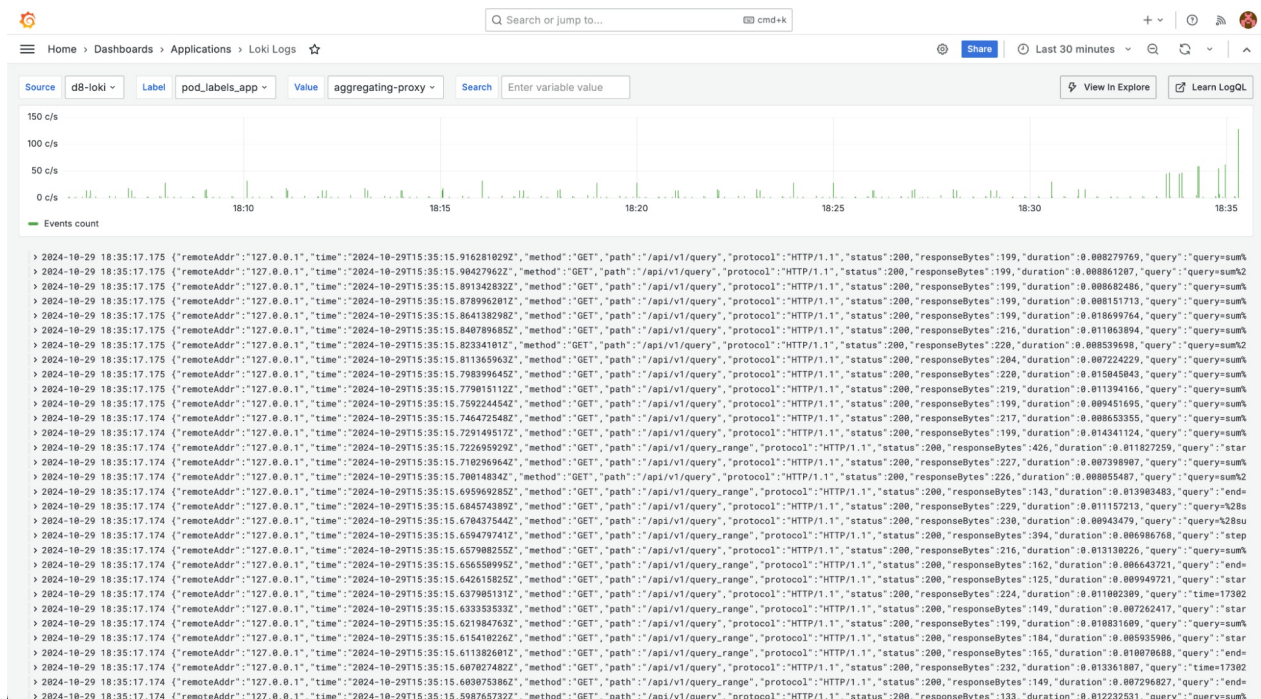
Дашборд «Applications - Log Shipper» — состояние модуля log-shipper. Здесь представлено количество агентов модуля на узлах и их нагрузка.



Дашборд «Applications - Loki» — состояние модуля loki.

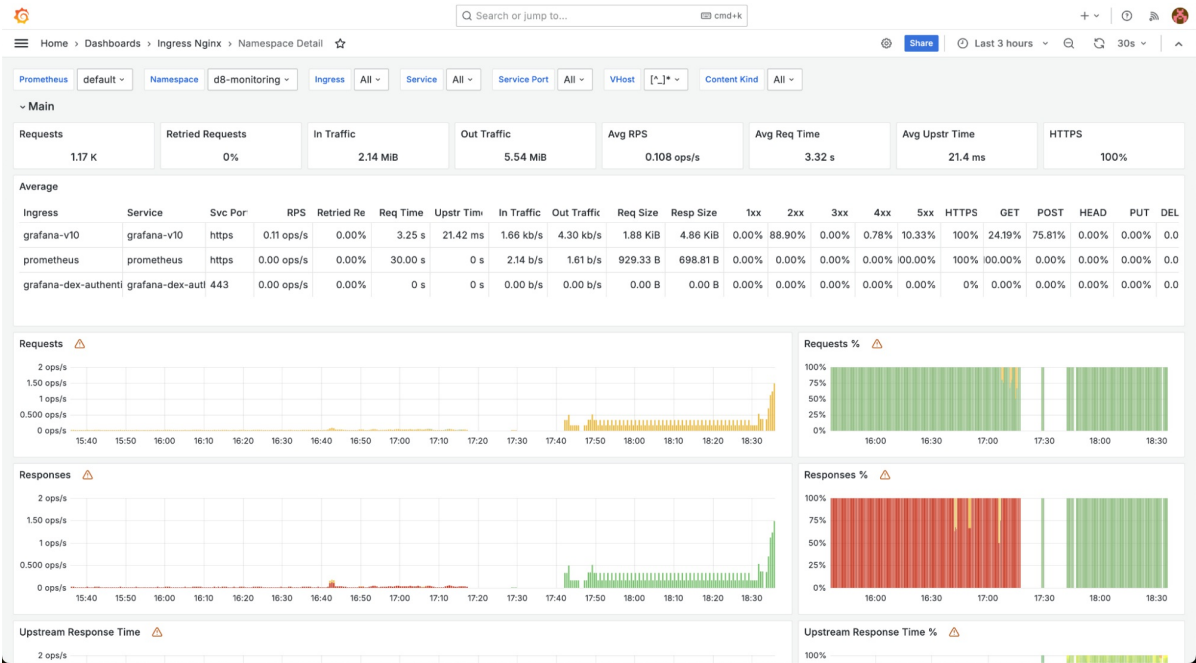


Дашборд «Applications - Loki Logs» — логи модуля loki.

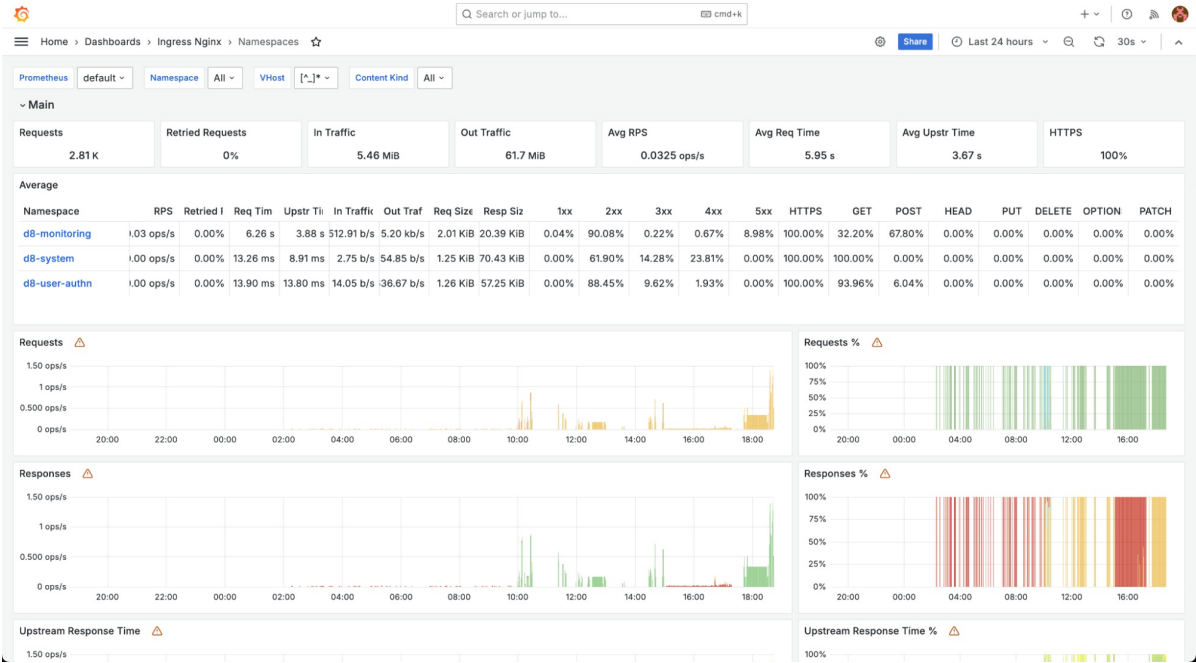


Дашборды группы Ingress Nginx — дашборды, связанные с Ingress-контроллерами:

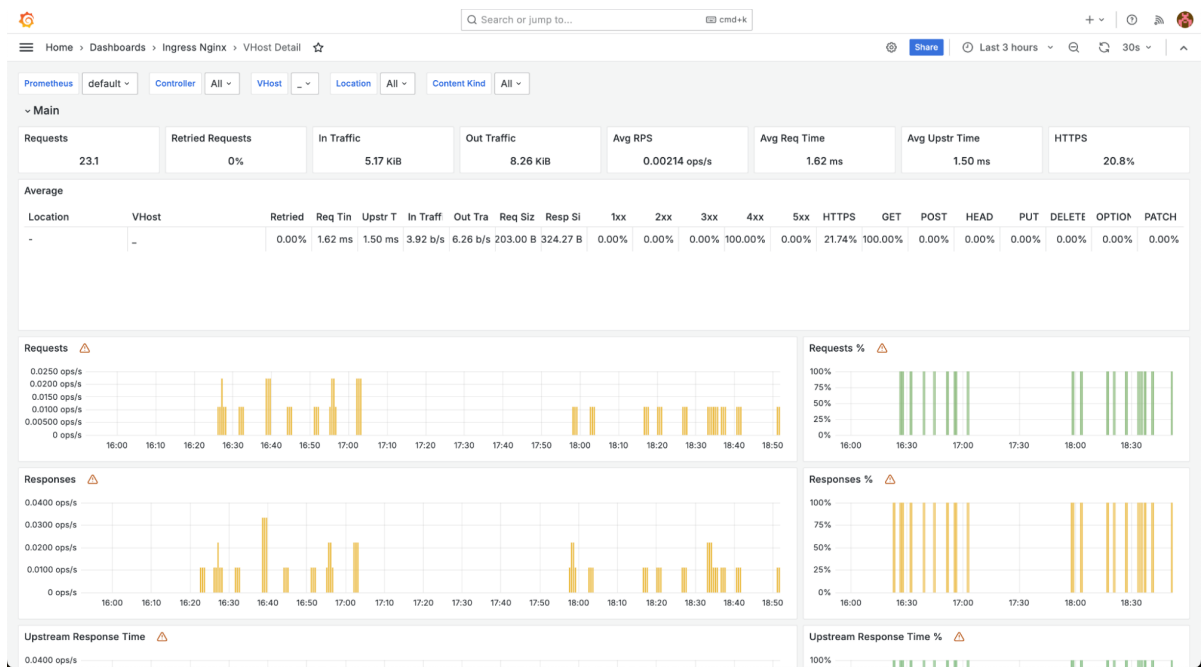
- Дашборд «Namespace Detail» — на этом дашборде отображается детализация компонентов в пространстве имен. В фильтрах возможно выбрать конкретное пространство имен, Ingress, Service и другие параметры для отображения.



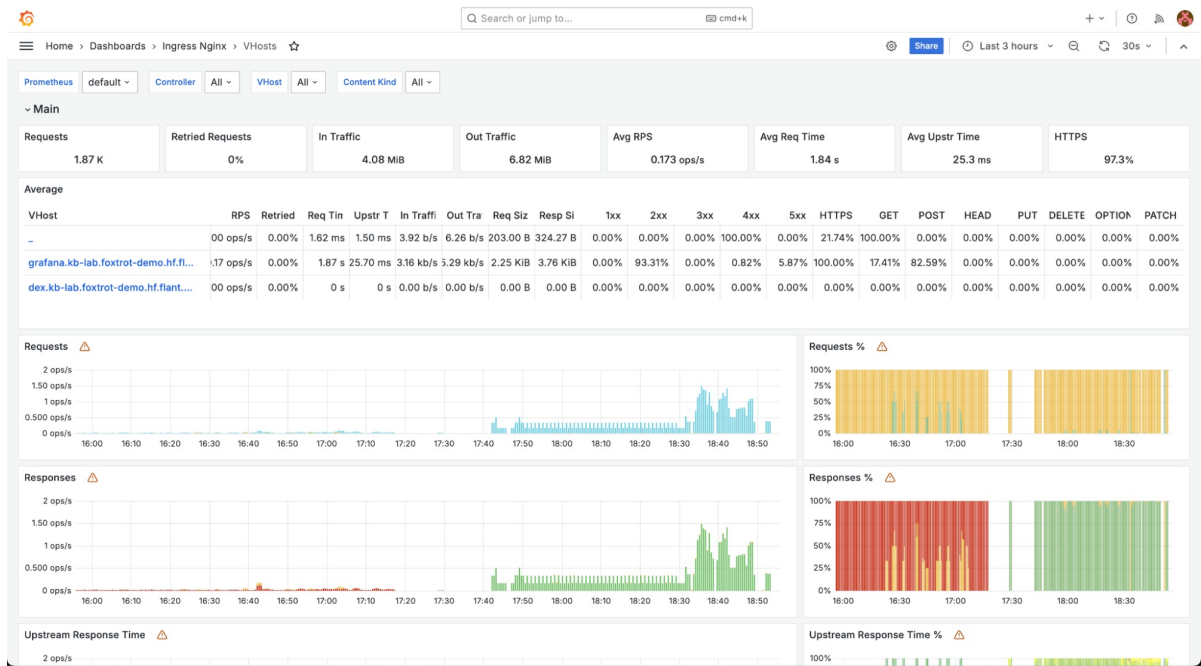
- Дашборд «Namespaces» — данные по Ingress-контроллеру в разрезе пространств имен кластера. В фильтрах можно выбрать конкретное пространство имен, виртуальные хосты и тип контента.



- Дашборд «VHost Detail» — подробные данные по Ingress-контроллеру в разрезе виртуальных хостов.

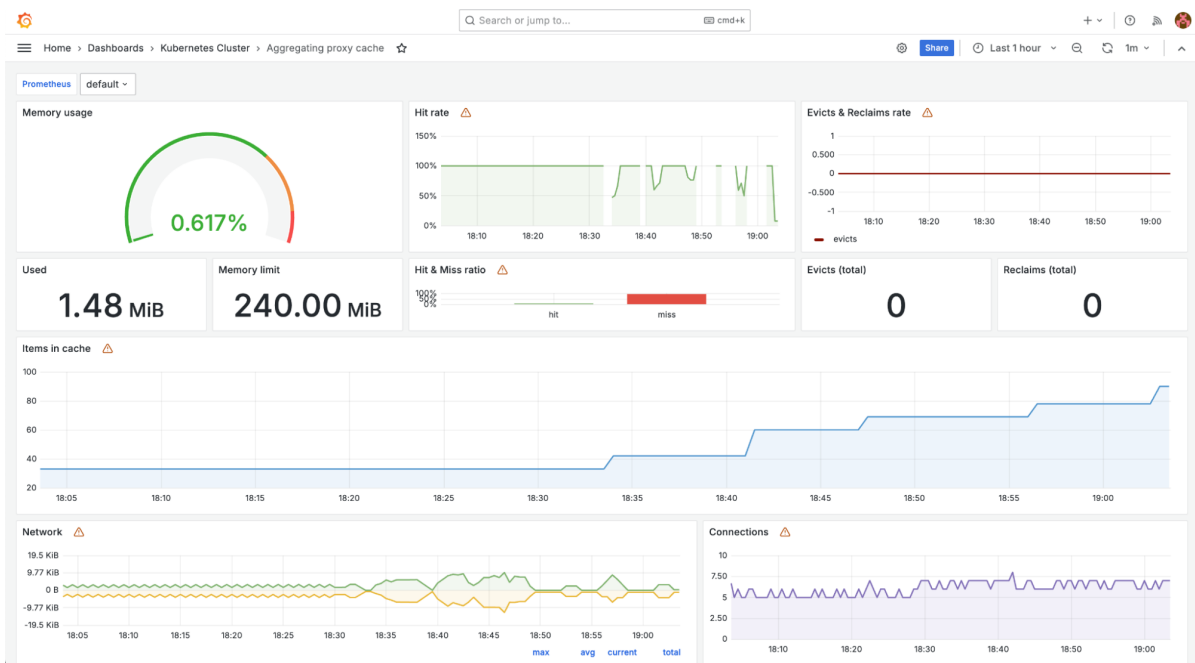


- Дашборд «VHost» — сводные данные по Ingress-контроллеру в разрезе виртуальных хостов кластера. В фильтрации можно выбрать конкретный виртуальный хост.

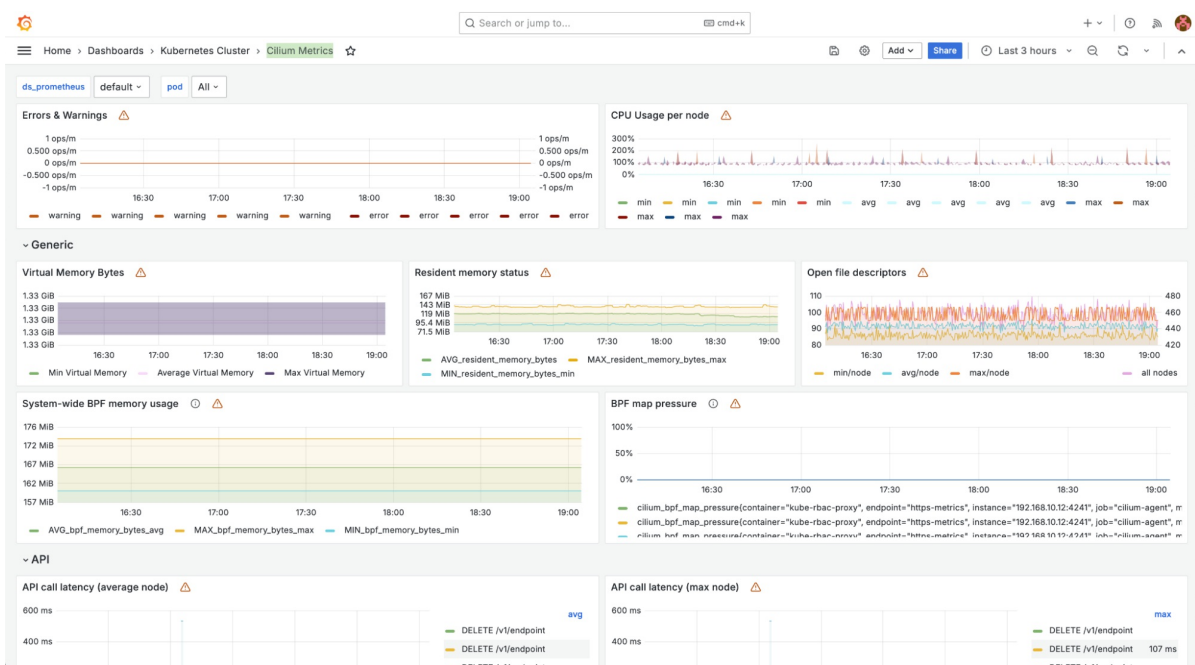


Дашборды группы «Kubernetes Cluster» — дашборды, связанные с кластером Kubernetes:

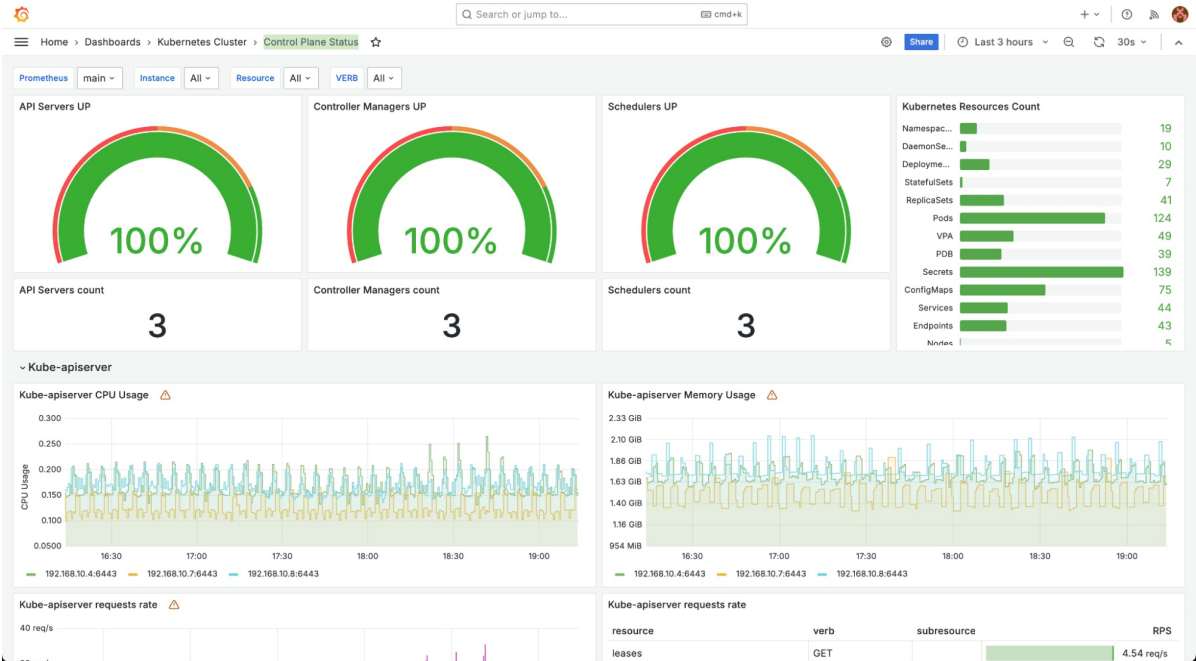
- Дашборд «Aggregating Proxy Cache» — сводная информация по потребляемым прокси-сервером ресурсам.



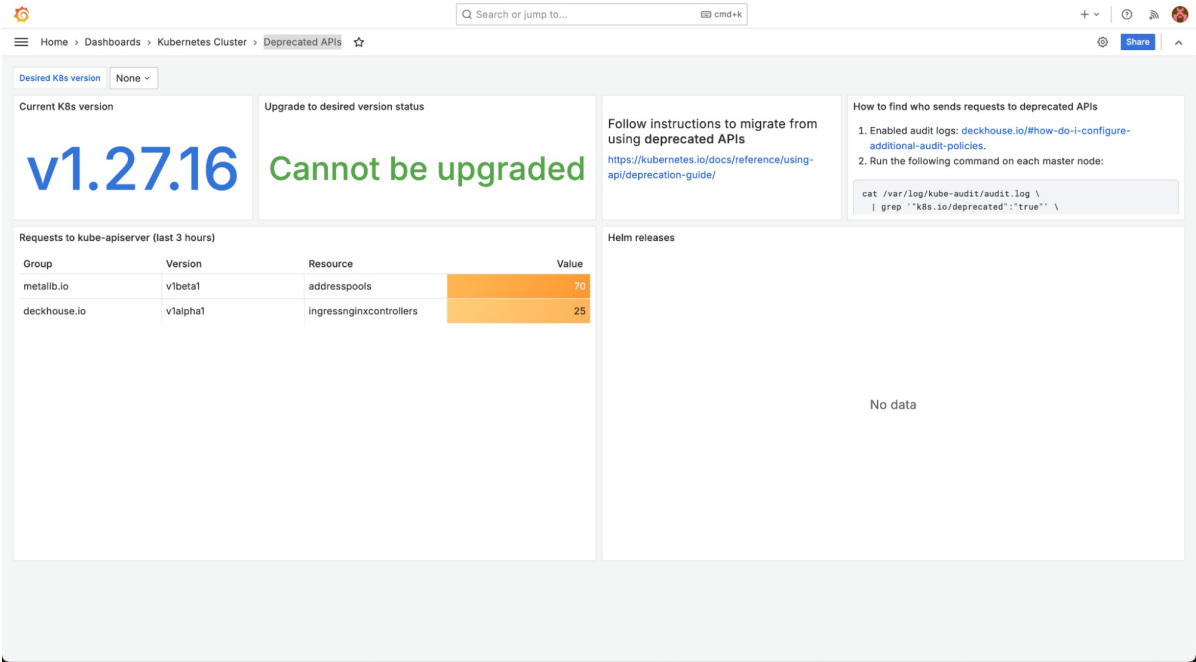
- Дашборд «Cilium Metrics» — метрики модуля cni-cilium.



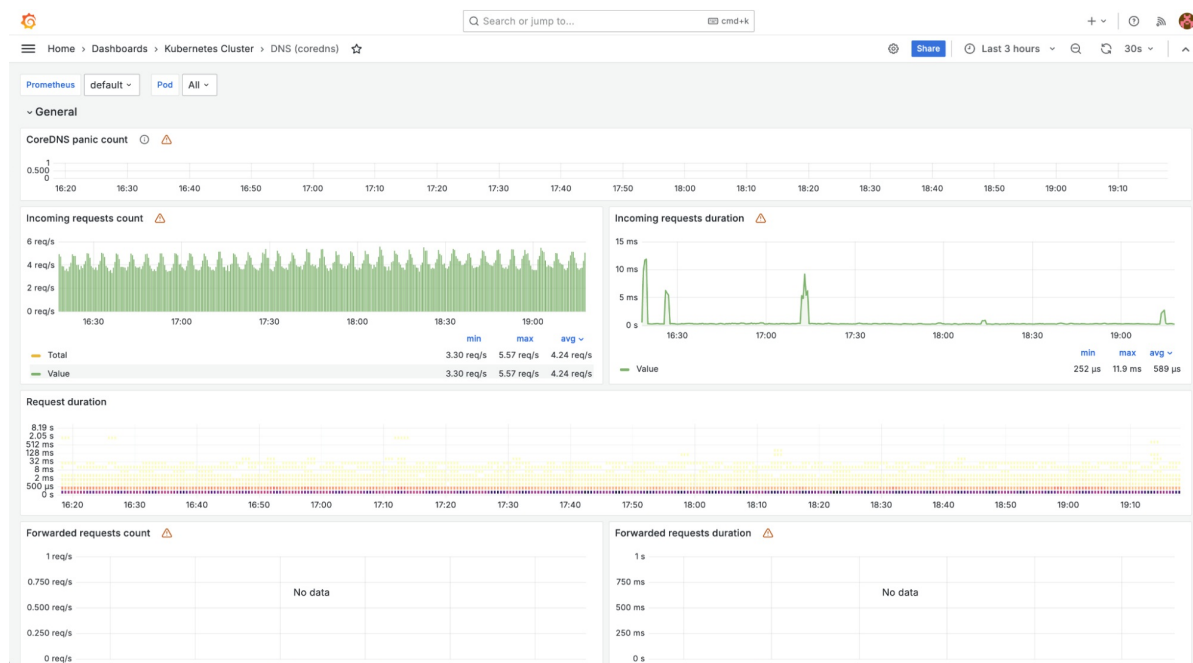
- Дашборд «Control Plane Status» — состояние управляющего слоя кластера.



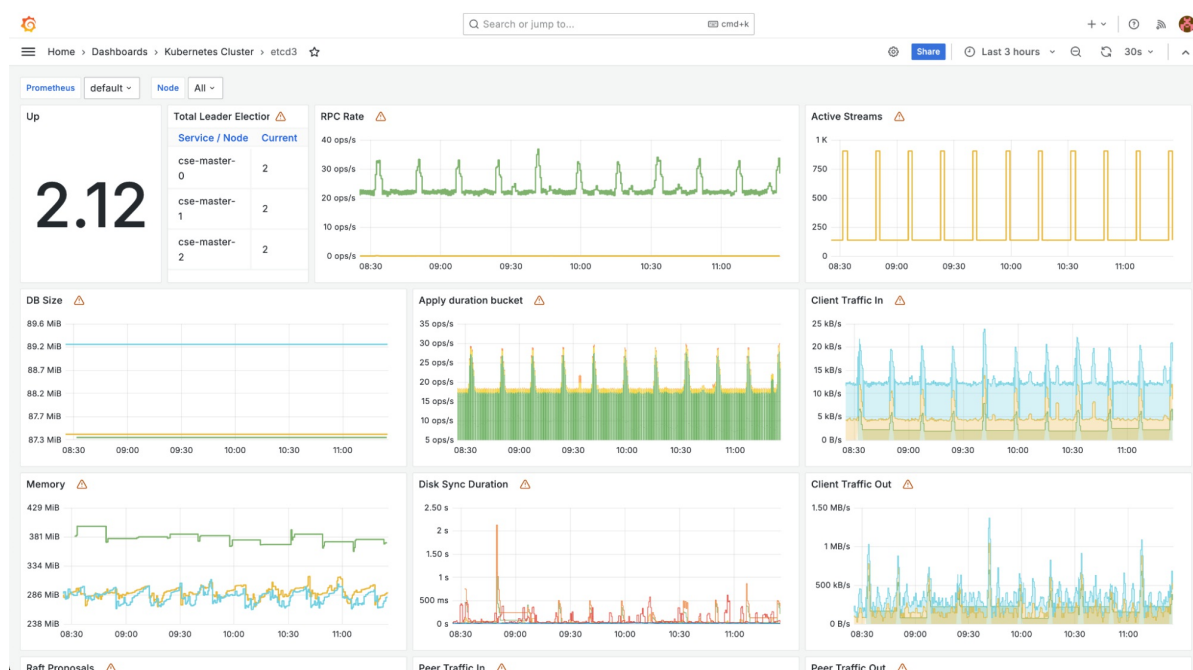
- Дашборд «Deprecated APIs» — отображает состояния Kubernetes API, которое на текущий момент находится в состоянии прекращения поддержки. Также на нем расположены инструкции по миграции на актуальные версии и запросы к эндпоинтам этого API.



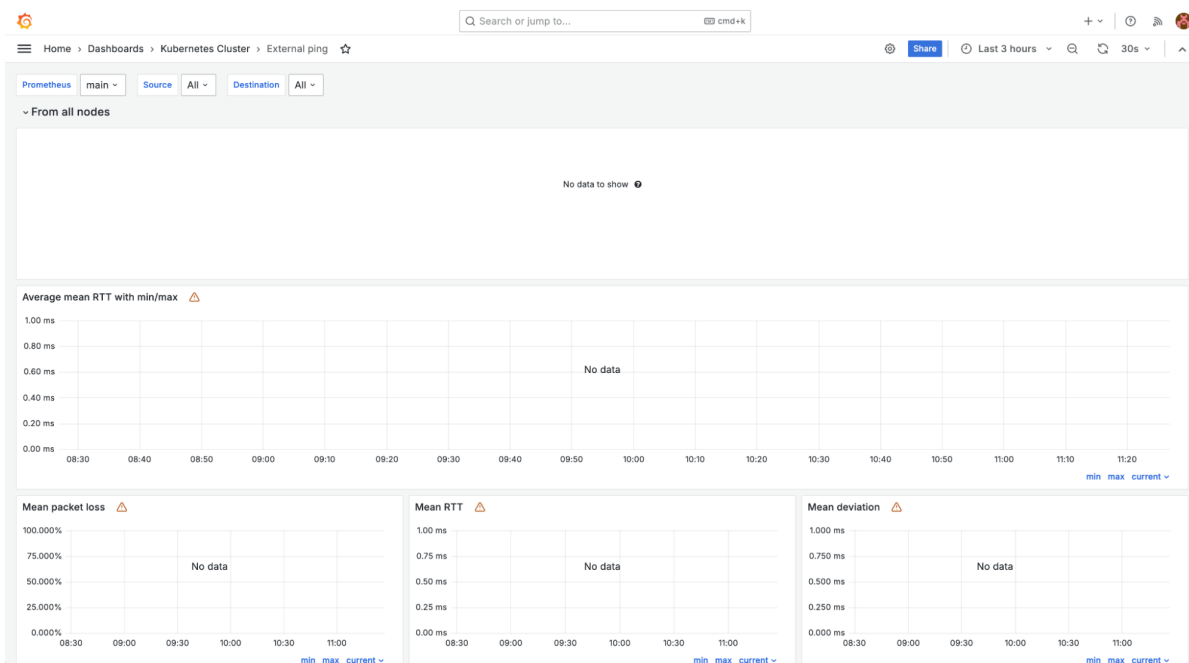
- Дашборд «DNS (coredns)» — данные о работе CoreDNS.



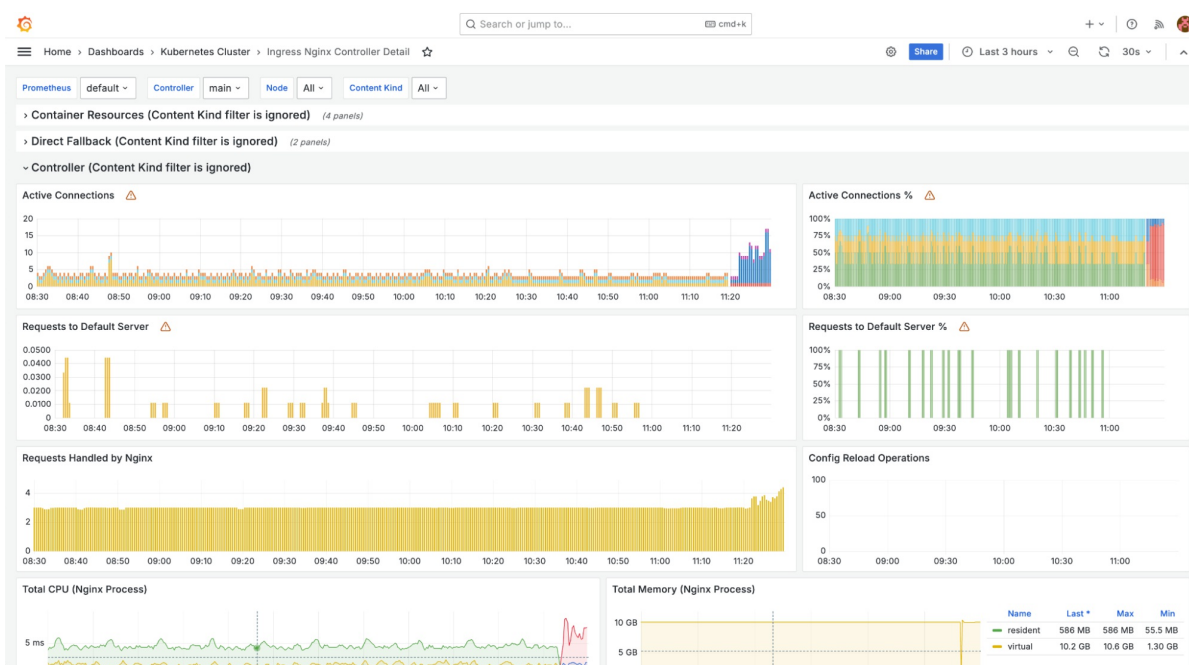
- Дашборд «etcd3» — состояние базы данных etcd.



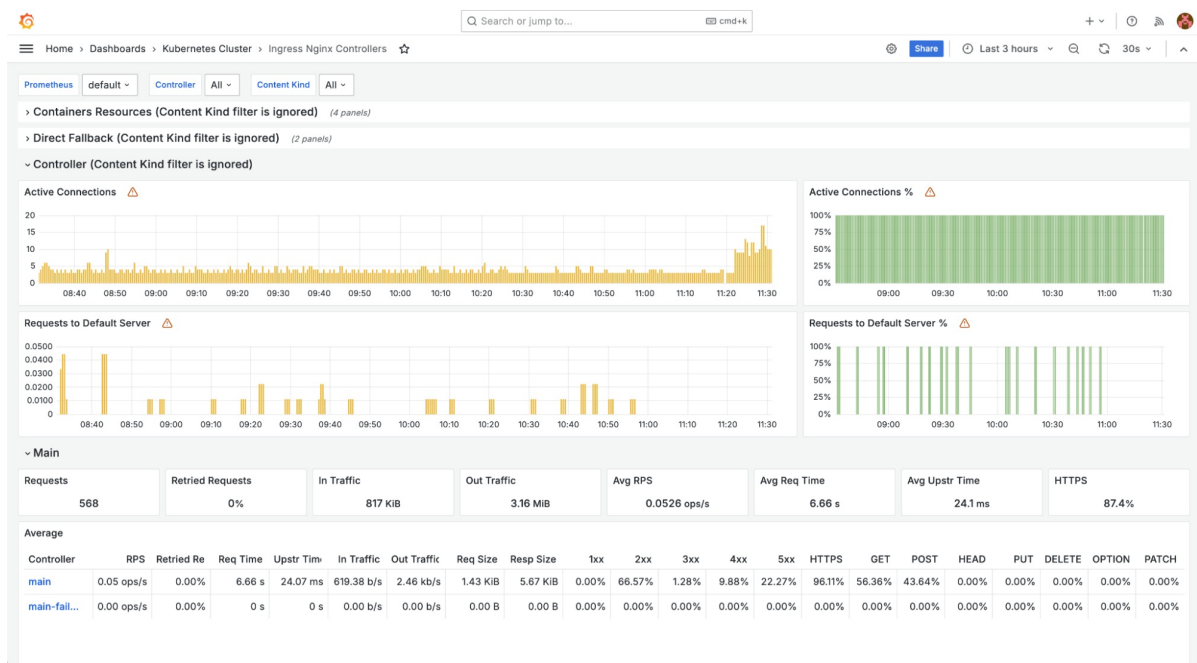
- Дашборд «External ping» — статистика внешних запросов.



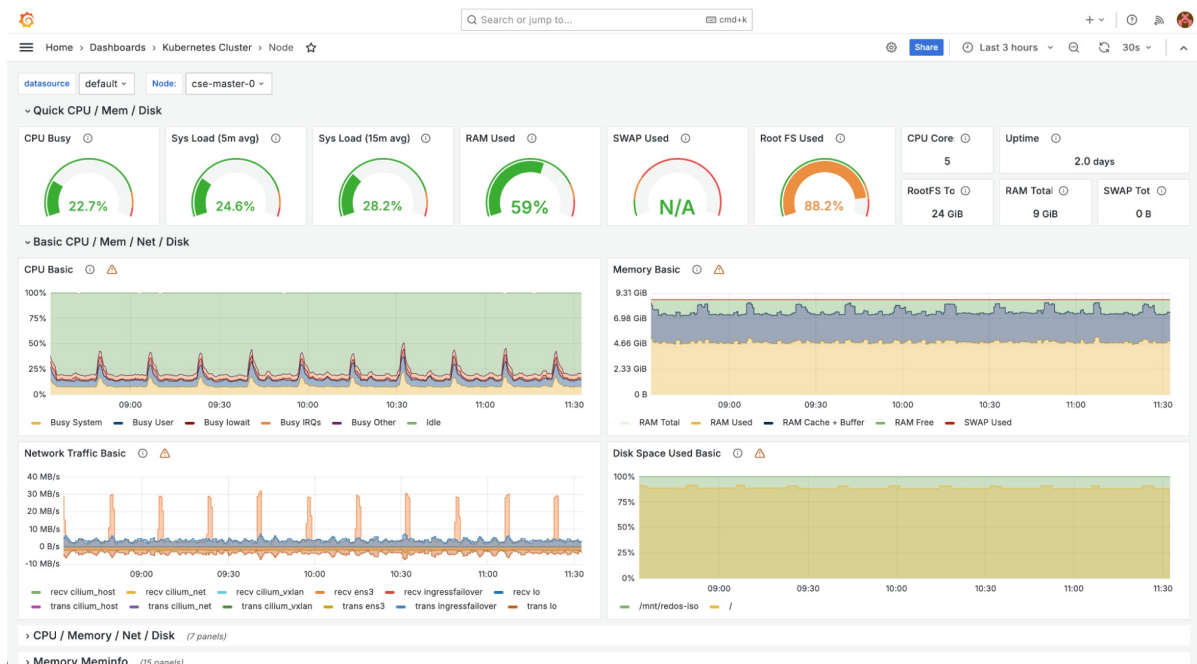
- Дашборд «Ingress Nginx Controller Detail» — параметры Ingress Nginx-контроллера.



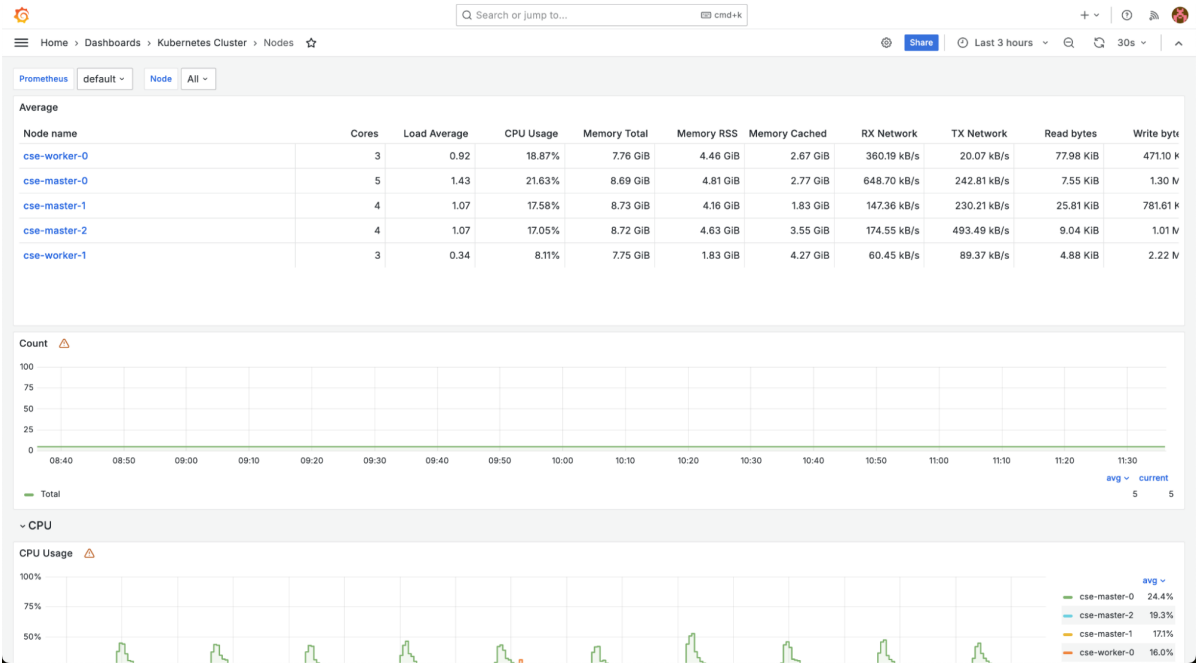
- Дашборд «Ingress Nginx Controllers» — подробные данные об Ingress Nginx-контроллерах кластера.



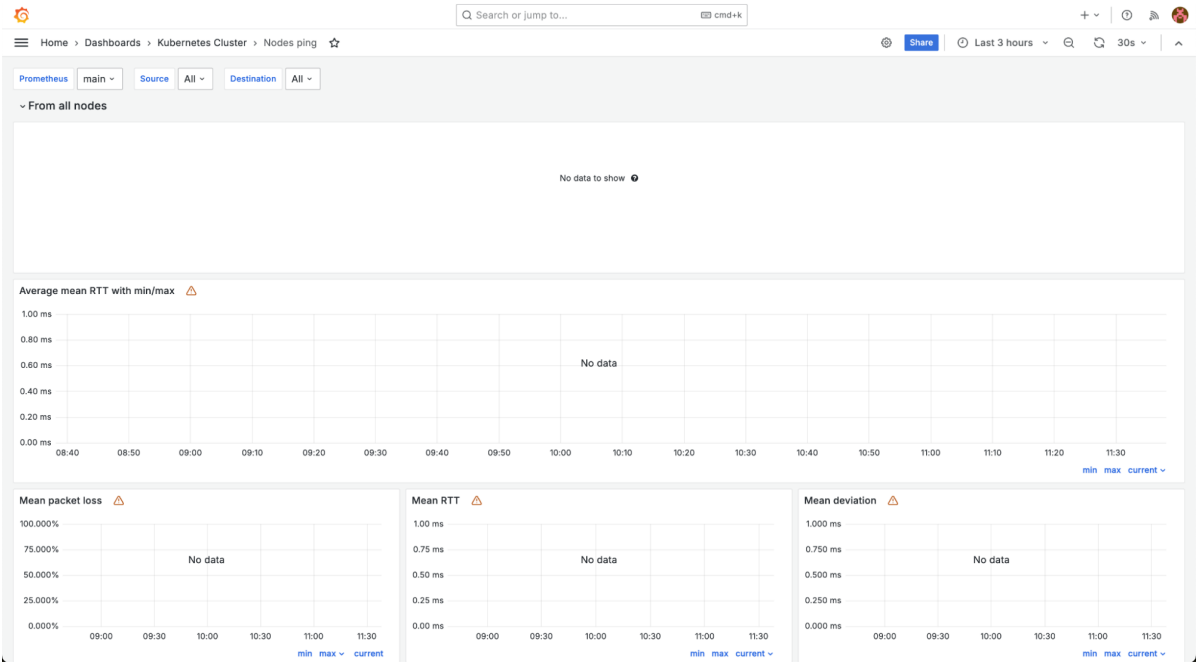
- Дашборд «Node» — данные о работе узлов. В фильтрах можно выбрать целевой узел для отображения статистики.



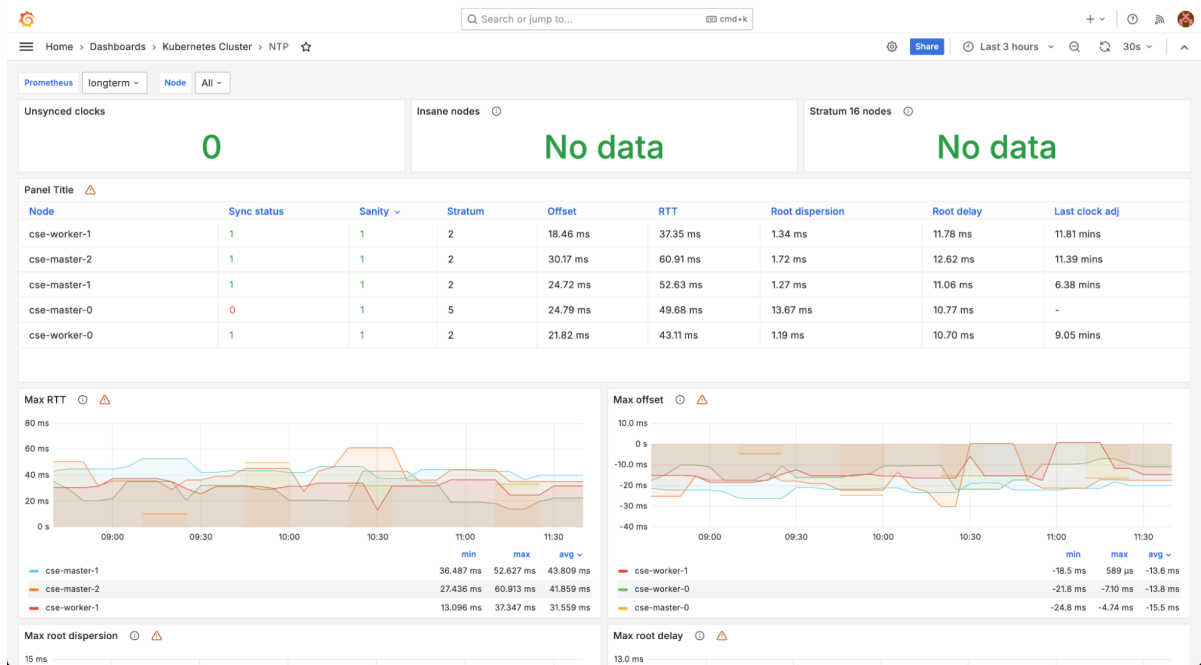
- Дашборд «Nodes» — сводные данные о работе узлов кластера. В фильтрах можно выбрать конкретный узел.



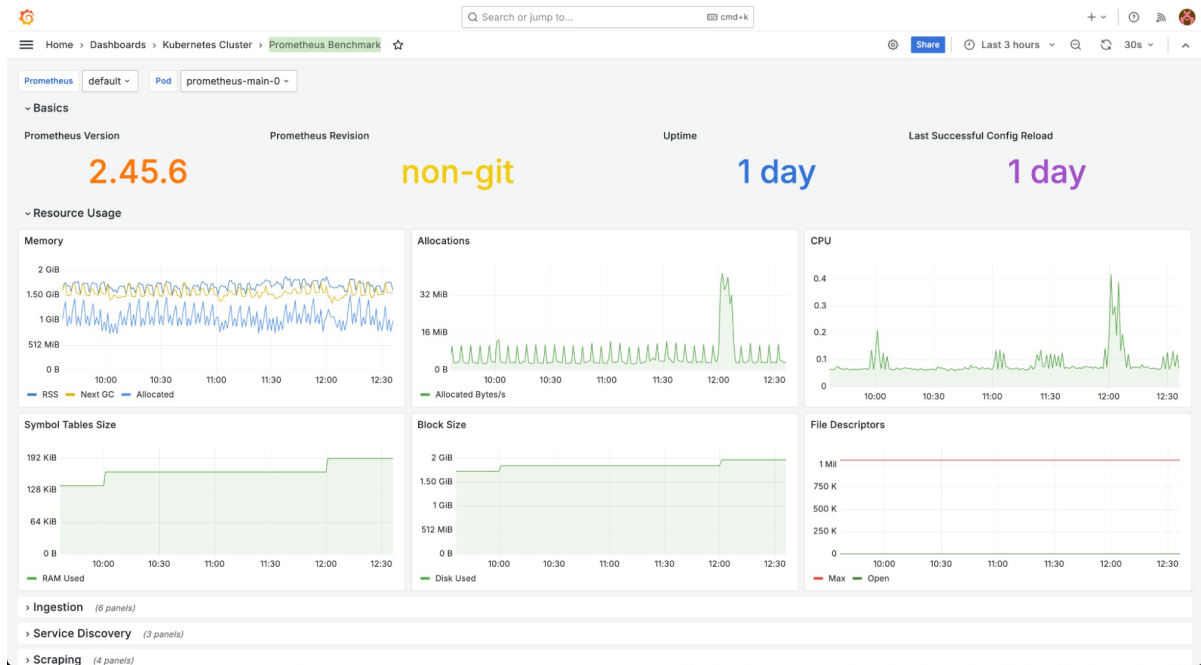
- Дашборд «Nodes ping» — пинг до узлов кластера.



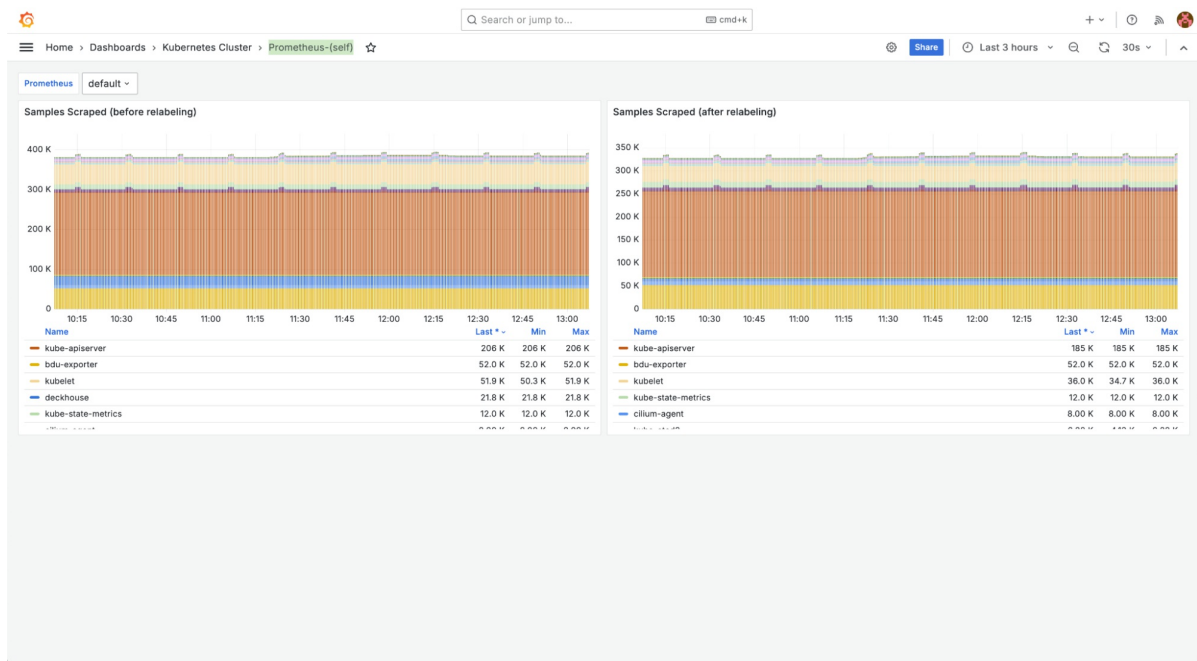
- Дашборд «NTP» — состояние сервера времени.



- Дашборд «Prometheus Benchmark» — статус модуля prometheus.

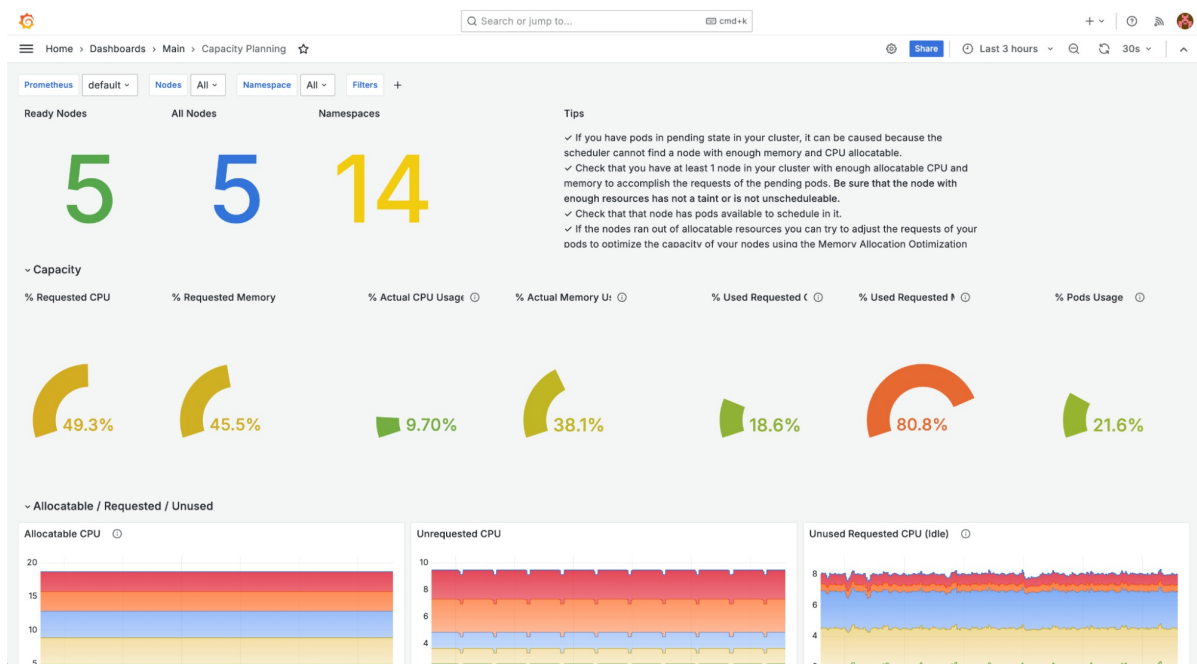


- Дашборд «Prometheus-(self)» — сводная информация о состоянии Prometheus.

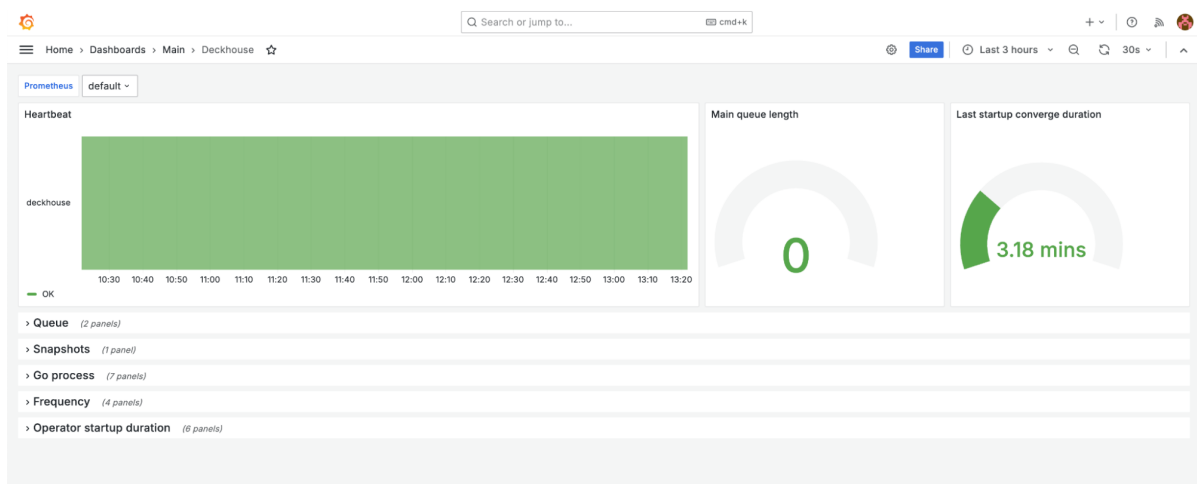


Дашборды группы «Main» — дашборды с общими данными о состоянии кластера:

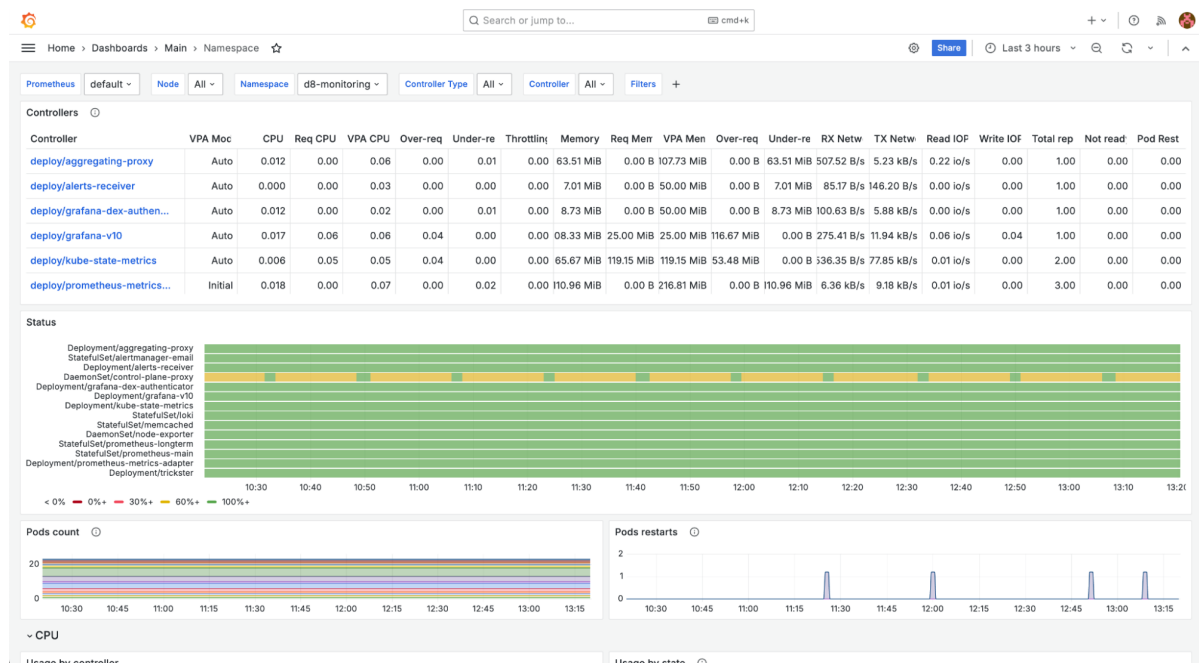
- Дашборд «Capacity Planning» — сводные данные о производительности кластера.



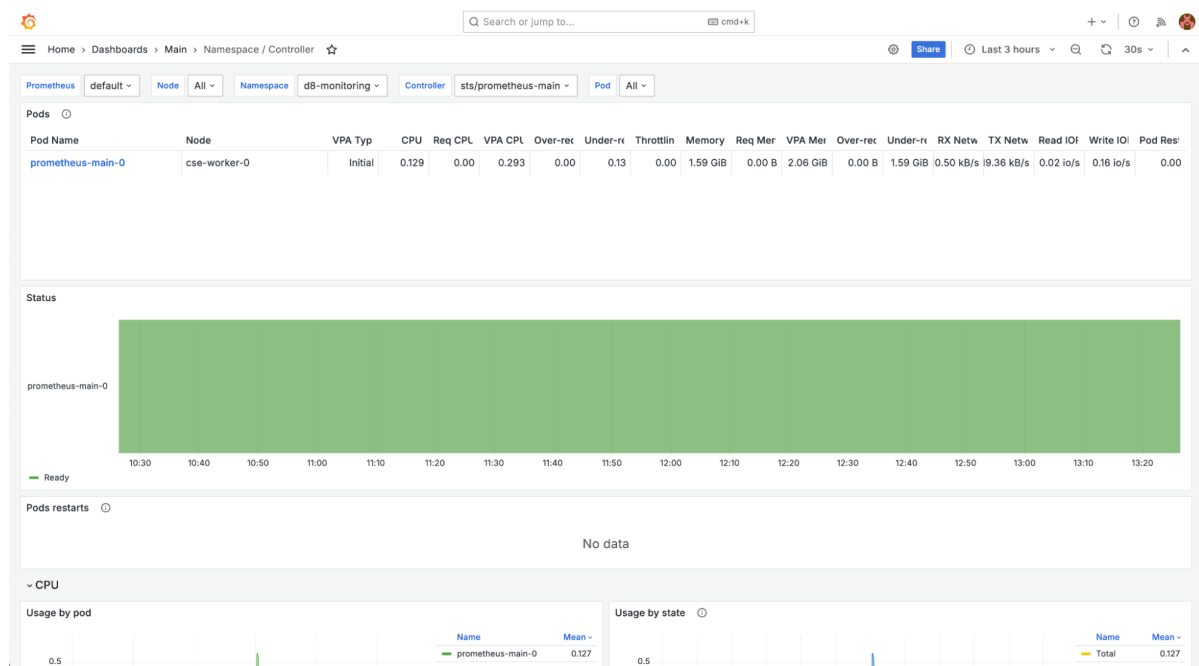
- Дашборд «Deckhouse» — сводная информация о состоянии ядра платформы.



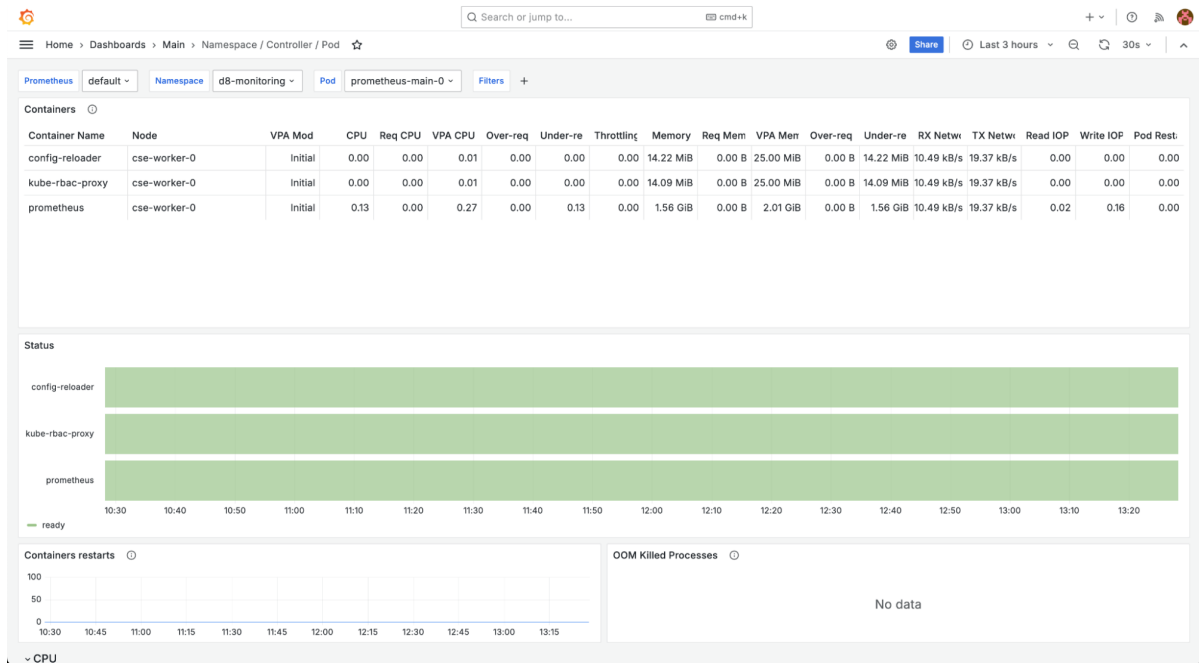
- Дашборд «Namespace» — данные по конкретному пространству имен кластера.



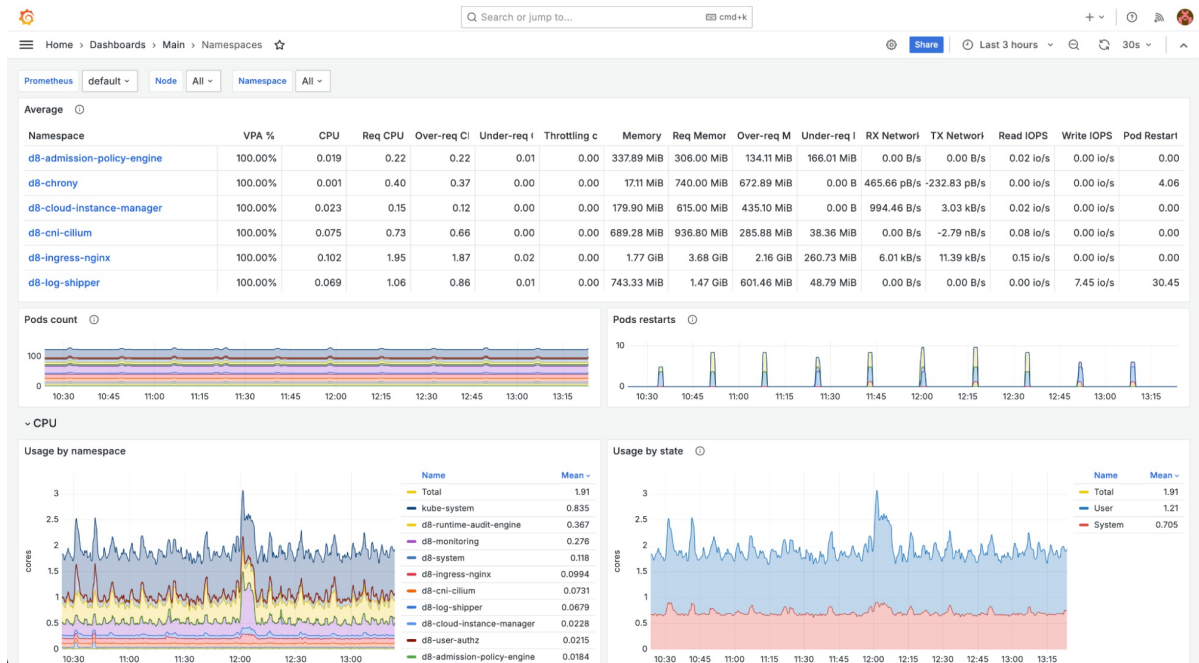
- Дашборд «Namespace / Controller» — данные по контроллерам в пространствах имен. В фильтрах можно выбрать конкретные пространства имен и контроллеры.



- Дашборд «Namespace / Controller / Pod» — данные по подам в пространствах имен. В фильтрах можно выбрать определенные пространства имен и поды в них.



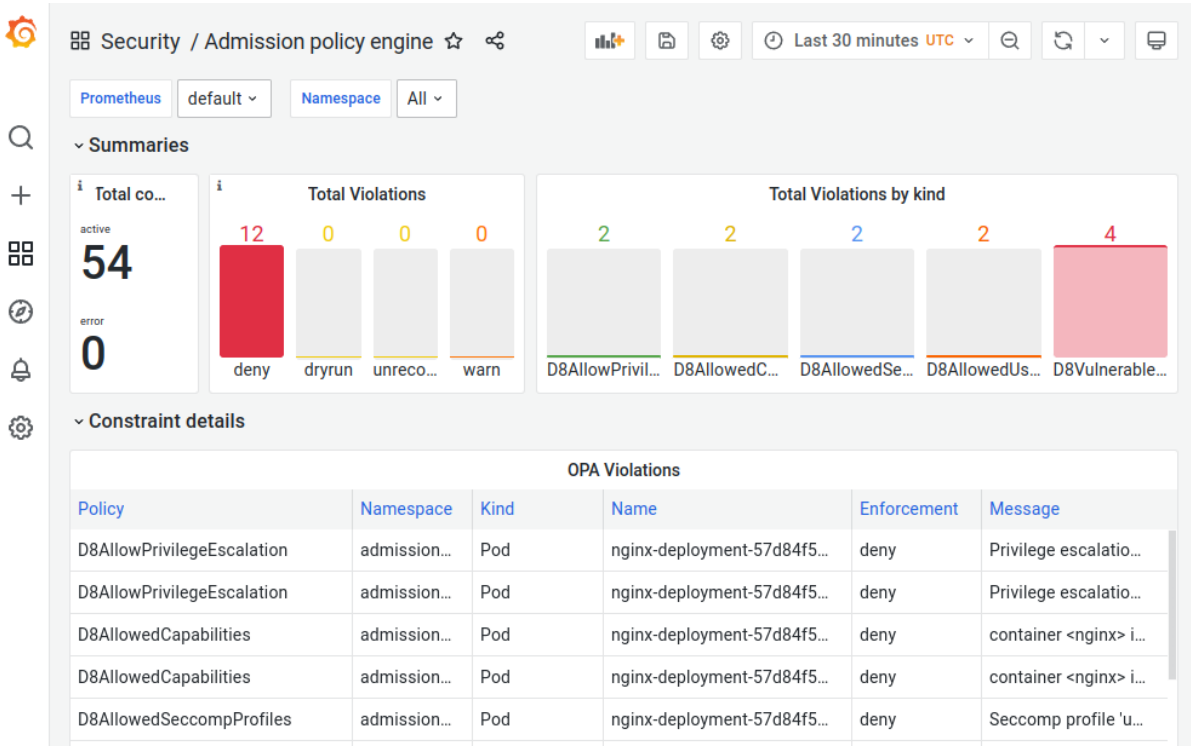
- Дашборд «Namespaces» — сводные данные в разрезе пространств имен в кластере.



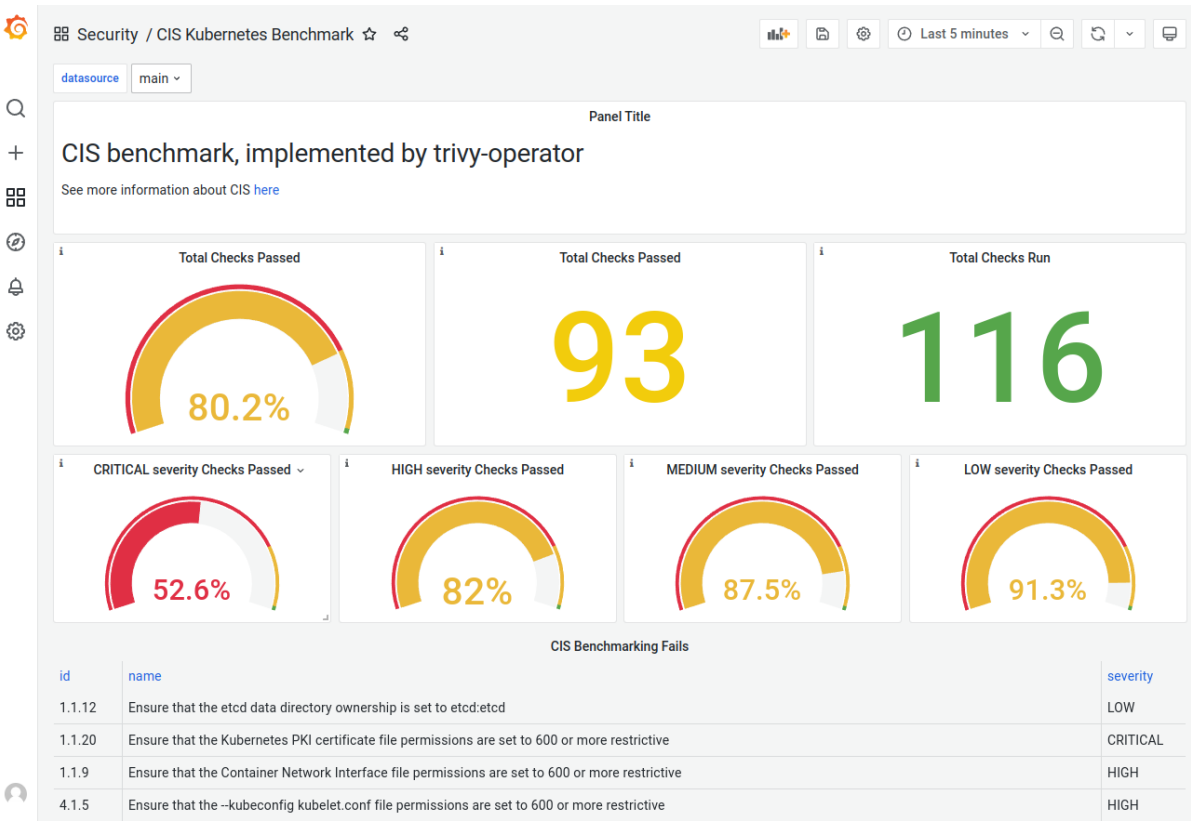
Просмотр журналов событий безопасности

Просмотр журналов событий безопасности осуществляется в веб-интерфейсе Grafana. Необходимые дашборды сгруппированы в папке «Security»:

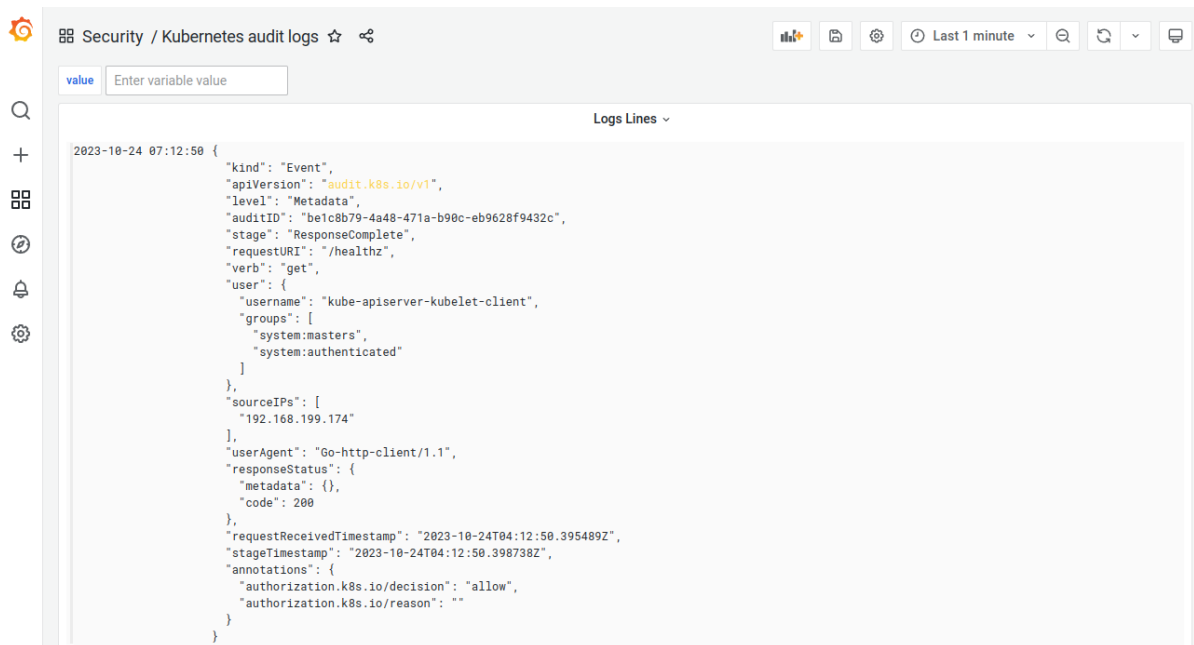
- Admission policy engine. Содержит информацию, связанную с работой политик безопасности. В том числе: количество событий запрета выполнения действий из-за нарушения политики безопасности; разбивку запретов выполнения действий по типу запрета; журнал событий. Журнал событий безопасности, связанных с политиками безопасности, находится в окне OPA Violations.



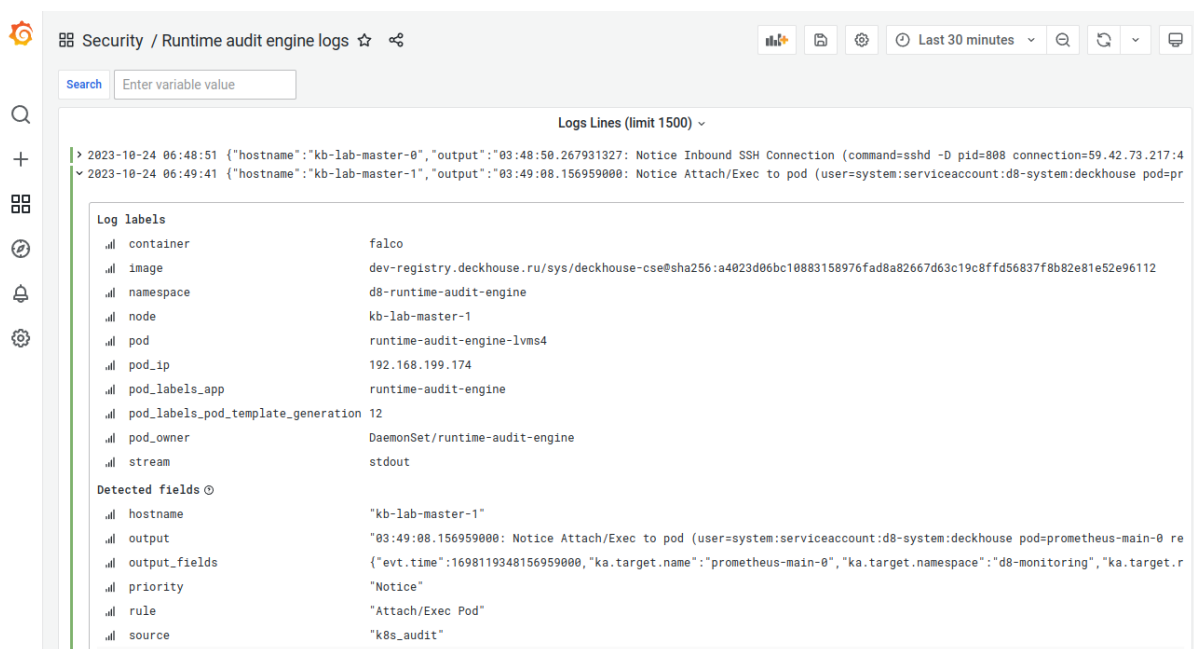
- CIS Kubernetes Benchmark. Дашборд с информацией о результатах работы сканера проверок конфигурации кластера на соответствие принятым подходам (лучшим практикам). Содержит сводную информацию о результатах проверки, без возможности детализации. Дашборд доступен при включенном модуле operator-trivy.



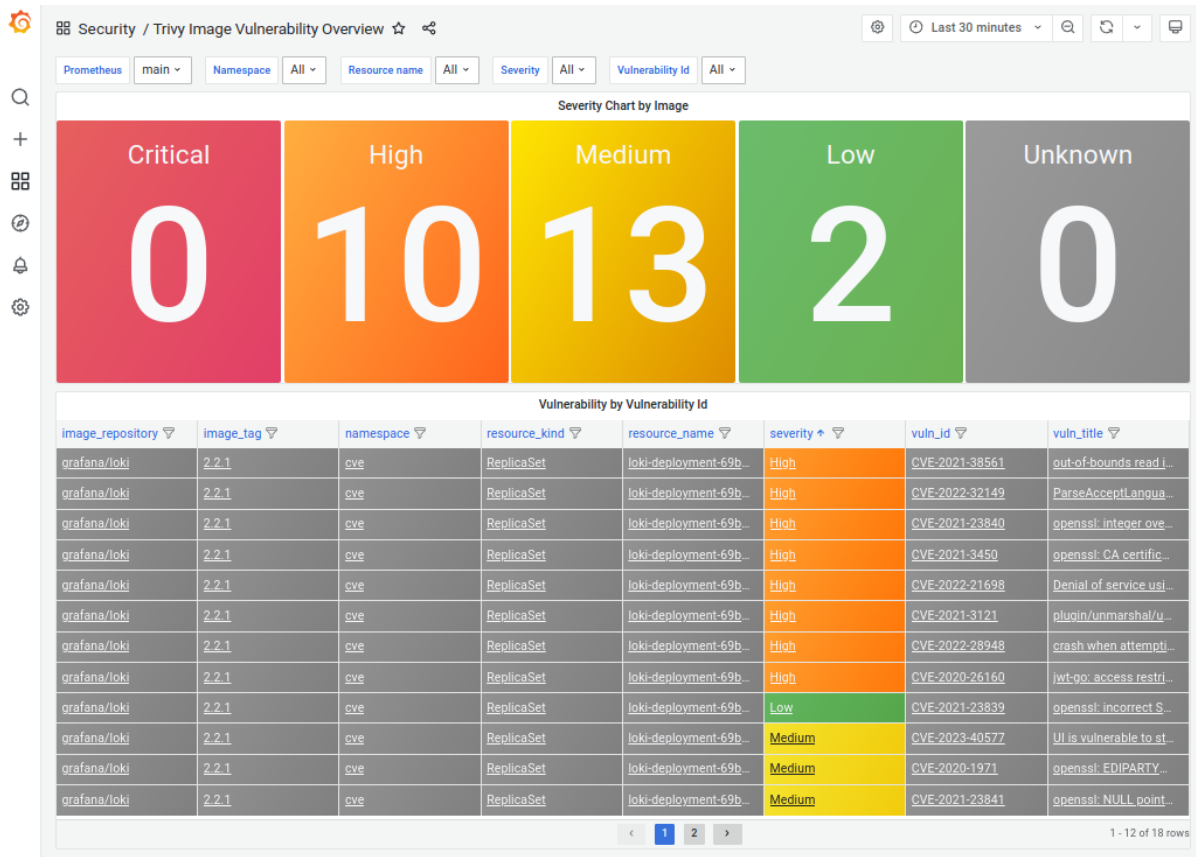
- Kubernetes audit logs. Журнал регистрации обращений к API-серверу. Содержит записи о всех обращениях к API-серверу кластера в JSON-формате.



- Runtime audit engine logs. Журнал регистрации событий безопасности аудита работы ядра Linux и API-сервера кластера.



- Trivy Image Vulnerability Overview. Дашборд со сводной и детализированной информацией о сканировании образов контейнеров подов в пространствах имен, отмеченных аннотацией `security-scanning.deckhouse.io/enabled`.



Дополнительные ресурсы

- Документация модуля `cni-cilium`
- Документация модуля `log-shipper`
- Документация модуля `loki`
- Документация модуля `operator-trivy`
- Документация модуля `prometheus`

Веб-интерфейс управления алертами

С помощью веб-интерфейса управления алертами можно получить подробную информацию об алерте, включая источник, уровень критичности и время срабатывания, а также при необходимости временно отключить оповещения об алерте (silence).

Список всех доступных алертов системы мониторинга Deckhouse Kubernetes Platform приведён на отдельной странице документации.

Доступ к веб-интерфейсу

Для открытия веб-интерфейса управления алертами перейдите по ссылке, расположенной в левой части главного экрана Grafana.

The screenshot shows the Deckhouse Kubernetes Platform v1.76 dashboard. The 'Web Interfaces' section on the left has a red arrow pointing to the 'alertmanager' link. The 'Prometheus Metrics' section shows 'prometheus' and 'prometheus-longterm' metrics. The 'Main' section on the right shows 'Namespace' and 'Namespaces' information.

При первом входе введите учетные данные пользователя. После успешной аутентификации откроется главный экран интерфейса со сводной информацией во всем алертам, возникшим в кластере.

The screenshot shows the Alertmanager interface. The top navigation bar includes 'Alertmanager', 'Alerts', 'Silences', 'Status', 'Settings', and 'Help'. The main area shows a 'Filter' tab and a 'Group' tab. A search bar is present with a '+ Silence' button. Below the search bar, there are expandable groups: 'Not grouped' with 4 alerts and 'job="deckhouse"' with 1 alert.

Алерты сгруппированы по категориям. Чтобы раскрыть список алертов в категории, нажмите + слева от названия группы.

The screenshot shows the Alertmanager interface with the 'Group' tab selected. A red arrow points to the '+' icon next to the 'Not grouped' group, indicating how to expand the list of alerts.

После раскрытия группы отобразится список всех алертов, входящих в неё.

+ Expand all groups

Not grouped 4 alerts

2024-10-28T10:56:37.543Z + Info Source Silence Link

alertname="DeadMansSwitch" + prometheus="deckhouse" + severity_level="4" +

2024-10-28T10:56:50.558Z + Info Source Silence Link

alertname="D8DeckhouseIsNotOnReleaseChannel" + d8_component="deckhouse" + d8_module="deckhouse" + prometheus="deckhouse" + severity_level="9" +

2024-10-28T11:01:58.385Z + Info Source Silence Link

alertname="D8GrafanaDeploymentReplicasUnavailable" + d8_component="grafana" + d8_module="prometheus" + prometheus="deckhouse" + severity_level="6" + tier="cluster" +

2024-10-28T11:01:58.385Z + Info Source Silence Link

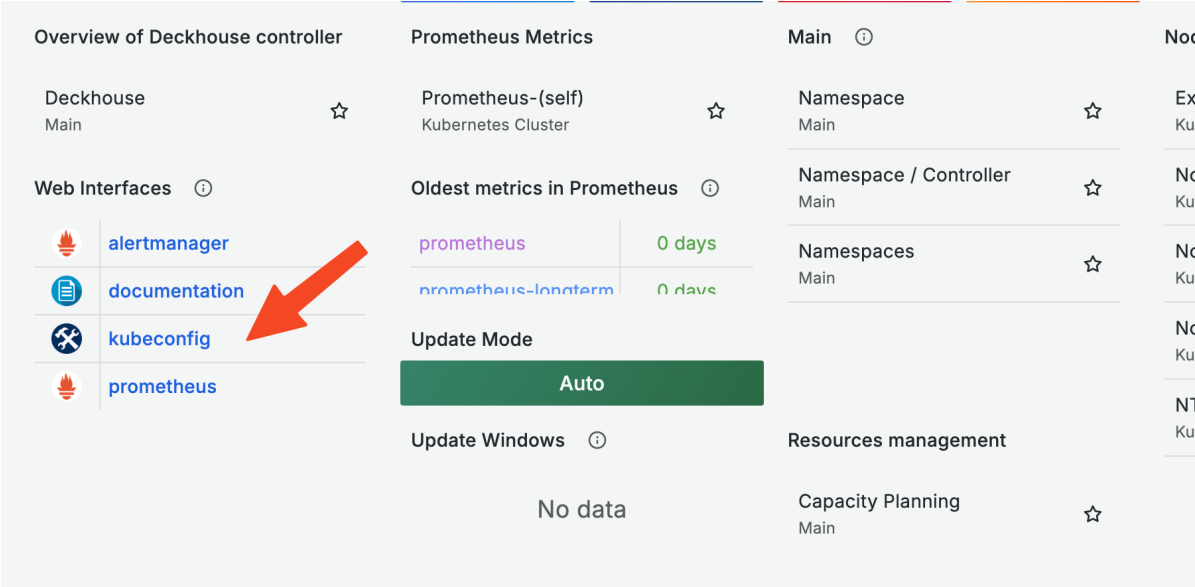
alertname="D8GrafanaTargetAbsent" + d8_component="grafana" + d8_module="prometheus" + prometheus="deckhouse" + severity_level="6" + tier="cluster" +

Веб-интерфейс генерации kubeconfig

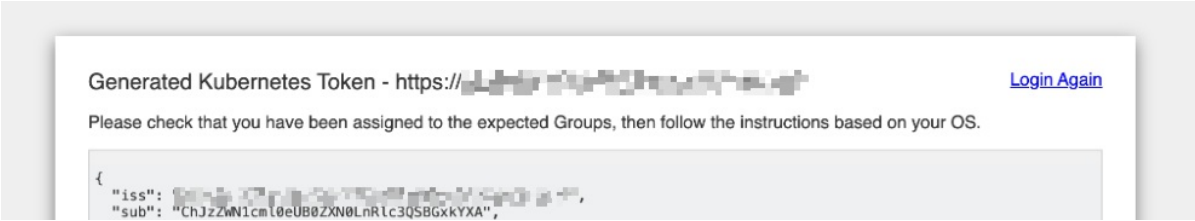
Веб-интерфейс генератора kubeconfig предназначен для создания конфигурационных файлов, необходимых для подключения к Kubernetes-кластеру с помощью утилиты `kubectl`.

Доступ к веб-интерфейсу

1. Для открытия веб-интерфейса генератора kubeconfig выполните переход по ссылке, расположенной в левой части главного экрана Grafana.



2. При первом входе введите учетные данные пользователя.
3. После успешной аутентификации откроется главный экран документации. На этом экране расположены конфигурационные настройки для доступа к кластеру.
4. Выберите ОС (Linux, macOS, Windows) во вкладках интерфейса, расположенных в средней части экрана. В зависимости от выбранной ОС будут предложены команды, после выполнения которых в системе будет создан контекст для подключения к кластеру. Предусмотрена возможность выбора «сырого» конфигурационного файла, который можно вручную расположить в каталоге с настройками `kubectl`.



```

"aud": [
  "kubernetes",
  "kubeconfig-generator"
],
"exp": 1730287093,
"iat": 1730286493,
"azp": "kubeconfig-generator",
"at_hash": "HNxwF5-6pY-CgC0JtdZ5_Q",
"c_hash": "hBFFUICs8uUtdTFm_EtQ",
"email": "security@test.test",
"email_verified": true,
"groups": [
  "security"
],
"name": "security@test.test"
}

```

Kubelogin	Linux	MacOS	Windows	Raw Config	ID Token
-----------	-------	-------	---------	------------	----------

Kubectl + Kubelogin

Kubelogin plugin is designed to run as a client-go credential plugin. When you run kubectl, kubelogin opens the browser and you can log in to the provider. It also prevents losing refresh tokens and receive errors on concurrent requests.

Install the latest release from [Homebrew](#), [Krew](#), [Chocolatey](#) or [GitHub Releases](#).

```

# Github releases (macOS, Linux, Windows and ARM)
github.com/int128/kubelogin/releases

# Homebrew (macOS and Linux)
brew install int128/kubelogin/kubelogin

# Krew (macOS, Linux, Windows and ARM)
kubectl krew install oidc-login

# Chocolatey (Windows)
choco install kubelogin

```

Kubectl configuration file that you can copy to `$(HOME)/.kube/config`

```

apiVersion: v1
kind: Config
preferences: {}
users:
- name: security@test.test
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1beta1
      args:
      - oidc-login
      - get-token
      - --oidc-issuer-url=https://kubernetes.default.svc.cluster.local
      - --oidc-client-id=kubeconfig-generator
      - --oidc-client-secret=8SYA411jn76b3XF55Au"
      - --oidc-extra-scope=email"
      - --oidc-extra-scope=profile"
      - --oidc-extra-scope=groups"
      - --oidc-extra-scope=offline_access"
      - --oidc-extra-scope=audience:server:client_id:kubernetes"
      - --certificate-authority=
data=LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUUIekNDQXMrZ0F3SUJBZ010RUFBd0RRRUUpLb1pJaHJzTFRFRUxUUF3RkRFRU01CQUD
BMVVFQ3M5EYsXQYkdGaUxTmNMQjRFRFRJME1EUXdNVEV6TVRZek1sb1hEVEkxTURRd01URXpNVF16TWxvd0dERVdNQ1FHQTFVRQpBd3d0YTJ
JdGJHRm1MWE55Y2S2bGQ0NBU013RFFZSKtVWkldmN0QVFFQkRURURnZ0V0URD00FRb0NnZ0VCCkFpNDkRjEUSX0V0k1WMX0Y0Y0VIR0cWd2V
MNHQ3Rk4vdMfQURUNZQV01BMjQ0NWM5U0Y0FI4WjBrZVM3Q0UKKz291bzdQY9aUggrZVdkbmXhZ210NHgyVUx1Ky8yUfN00WxLeEMANDLaLF
YwWkxc1LX5XVabTK0ZEHkdGpKQZFrWlCxdKxtaG1MbVpWVcIMExtTnZ1WUlpS2k1clpM0NzV01Wm0SNGRISnZkZqFwLrCxdKxtaG1MbVpZC1l
XNTBMBU52Y1RBTkJna3Foa2lHOXcwQkFRc0ZBQ0U0dWdFQXg1MCtBM0c1RVFQK0x4MGZHMjFnZ2VtN0ZVMcSkT9Yc1L5vzBrdlp2WEZSjNvY2d
GmWcW0G9xSmE0FhncKfUdE1nakVpMkLyREk1T1LFjUytBbVZyRFZ2bJBGUOpEYU1jCENU0dLaLjPnVFNDFMV320SE16VT2N19Vvm44VVFobmh
UqLU0Q3155ZVYKzVcNvU106TzNn53RKC1REU1pNkM4C2Td1N1VFM3N0Y0JJeXVWZmN1bFBG0KfGUE5xa0xxdWzS05tadE1tUzCd0VpZFBUTm1
PbG020UgK0F1wM1J1TNE1GNS1Zb3Bkb3FtScTjcu0y0HdSTEDVUTNvU1yMTNGEfpD18vVnZFQKx4SDJ0NDIVU5NMVBCUApHJZjZXT0NCSK5IVTN
3G9STZNPedNXTVhQa1lEdUz5U0Vtmbhzb1VHL290Q1Jw0ERWmVolUNXSTVkbEJx0GZxckLUNTE1RTYxMm5Dsm90M1VS0DBPYjQyNVEYmQ2N3Z
2TKExaGvDDEt4bJjJIM0xRUWLYczhDyKlRSGJIRjC3RVKkaG5qem5VQzRwdFVLskF5dZNEyLY5V3BoaGtnHspzRE5Hc29mY00waBEZT1rRXR1YU1
M1N1BPL2pJUHVVV1500p3Ly83Y3Aya21nTKxZYVjUmhaUfHTSXNXZVWwZcTaUJDUK2UyL05sRTYvK0F6RFLSRWZXM1Bma2F4a1k1Ud0Cm9NaTe
3TzRwWkaMa3NYT2Z1C2dpSU01d3ZSRWvU1ZYW8z5FVKVnBGd3FwdVb1YyUak8WnNUM1E0TGc1VLQKMTdhdE1CTHv5a0xyREDFRERpaVV
kbWFGMTBY011dFFOWEJCW9aQ3hNQzLEaG5SM2Vzb1pCFFwXR3aAoxdG1KVDJ0cU400EMRcnc9Ci0tLS0tS0tUEIENFU1RJRkLDQVRFLS0tLS0
K"
  command: kubectl
clusters:
- name: security@test.test
  cluster:
    server: https://kubernetes.default.svc.cluster.local
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB0tLS0tCk1JSUUIRENDQXMrZ0F3SUJBZ010RVVFsMDhmXpIUE0xV01ZanF0N11TS610T2tFd0RRRUUpLb1pJa
HJzTFRFRUxUUF3RkRFRU01CQUDBMVVFQ3M5EYsXQYkdGaUxTmNMQjRFRFRJME1EUXdNVEV6TVRRek9G1hEVE0wTURNegpNREV6TVRRK9G
b3dGREVTTUjBR0ExVUVBd3d0YTJdGJHRm1MWE55U0tUJ00lQ0U5C2ZtaGtR13ME1JBUUUVGCKFBT0NBZzhBTU1J0Nn50NBZ0VBM5tSS5GyxWgdI
aFh1NmVrekpMdnA2Rz2tRWseGdPV1Z4cz1TdnFU21vd1YKZV4cnJR5mXpQVB0WmR05n0yZDUYzZVYakRNd1E2MGJkTDRYmMnA2M0ZBRUdVn2Xl
NHZwc1kxWbX08VkyT3ZTbgpVMEpJNHpRNDNFYUQeI9aa0Y151hNXL0emc5YmE2d043a1FjYtK0VU5YUUYrdUdTW1Iwa1dY3ZJVGNS4TRWLVk
UVRubXf0EgVmy0t3ZDyz3j0eDjLVNjB2FhcEhSdzhhSW1ITGpVYzZ1aGd0Nnm5MRG1tnNnY4Wk4RXJqMCSKwWZ0Uk5Ym1ZYFN6R19S0kxzdG5Z
TgtLVnIycjBzClyZSUY3TjJ0Y1pa5KdybXdlZmtadEN6WFc2Vm1Lb0V0Tgp0Zk1B0XJYND1ub2Jo0G1S0EtDSUdjN3oQnhtS0d3Z3Fp3XWRB5tm
QXIZU2cyRDJiYUpqVUxT2Tm3BX0T0CK50eG1ZdFQ2MKxa0EfybHYxUk2V0xSnmMwZVJYV056cDhCRU9ybU11eGF5UHHMvNfLwnFSU2RNM2Jo
Z1h1VnckSURWZHZsYTBTm2Q4NW15v2swL1hwRs5Ty9tcU2TSKRXmYhYndCMWVPZ3NIRE1CYkhMcGfqrCtLTjFRWNg1a0pFMVb4ed1RQ0XENNVF
ZVnchVkbXg10y84UXFVmtXMHZaZVI1TlPcMvZUVz52dyZTZkRTBZEY9K5ExrY1BqCkFtY3hBVzd3UDJf0E0VFcG52adNveEpKUUE2ZkqxcU0p
mWdjbT2UBN6TknZ0RfYcE4RnhaZmFdytVSVR0ZVYKtW82WjV0Qm5QND0E01p1bUhyb2xpME93Q3R6ZGd3TmZpS1N1em8rW0V5Z0Vb1hJNHVf
d00ZwhkZ2Zm10B0WpBd0VBM0F0B1HUXdIUVE1LwT8JCWUGTnVsaGjK1JlMtwc1tLm10ZcxenBvWTFnU1A0EXVNRJd1FZCk1CYUFG
N1VsaGjK1JlMtwc1tLm10ZcxenBvWTFnU1A0EXVNRJd1FZCk1CYUFGN1VsaGjK1JlMtwc1tLm10ZcxenBvWTFnU1A0EXVNRJd1FZCk1CYUFG
MgdxM0psalprTGNRakNSNDZ4L8dCCjM0SDrukZ2TnzZ1WUlgSVRja0J0MDJ1YVNiQkLU10yZE9oc1lpTVN10DBhbT1RMUwzMHXUXhBCJ1cVzD
dhokWNTONVveWx30TFzCwQ4r1dVngt4ek0WdF0rVTEyUmd1S1U0M9MUDdWmFbMndM1hLa08yYjBP0K5xMDhPUwozTE9U0WZD1pUCX9Y1s1
cTKYN08ZwmRhZ2p50FmdFA1TmJadTExVHNW129RZEdrSww1UVF0STNCMVVKEURQC1J6QXhGvK5QekZyRnLaZw5uUuWvSzJUT0LRWTVSN3d0HBV
YnF1Yw9dZ1Vbdk36NG1YjA3YkVUULBtmUxhdVYKMDVKRwPtUS9xdZM5anR1J3A0K310Wef4Tm805I2VHRENGc1UVK3T2RYbHbEbJqbk9xUFVQ
TzhyVw3mREZQpYRj2T2A1ZjZdMHTFsQ1hJbm9HCHZL2UwkhudFdrYV1Nzwhj1BqB2b1ZKZ2p5LzFVTFBc3RqVHpb1Jw0cthmCNBMfLzWEN1
M3JjUWkg4dKvVF10EdN1bzg08Q2V2YzFVZTJYLh2FzSU15e6pMEhaYvtp0Wt5aUpFV1QRtRgKQnLYFYXfU0ZU0aUpw0TRBUZ2XUHFfaJb05KxN
dZJUE1mNnFmFmHNDM3JydA1azdkTzds0v10FpRQ59pTQpZ0WRKKZ2xV1tFcGw5kZvNY00vVTRteDZKN1dhjdjVzML1YjE2U3ZRM3ZhVVFY
QwtTdkRQZJNjQVNGVzh0UkRtCnJYekt4ZEhJRXBnbX1JSU52QUZTZ2c9PQotLS0tLV0RCBRVJUSUZJQ0FURSB0tLS0tCg==
contexts:
- context:
  cluster: security@test.test
  user: security@test.test
  name: security@test.test
current-context: security@test.test

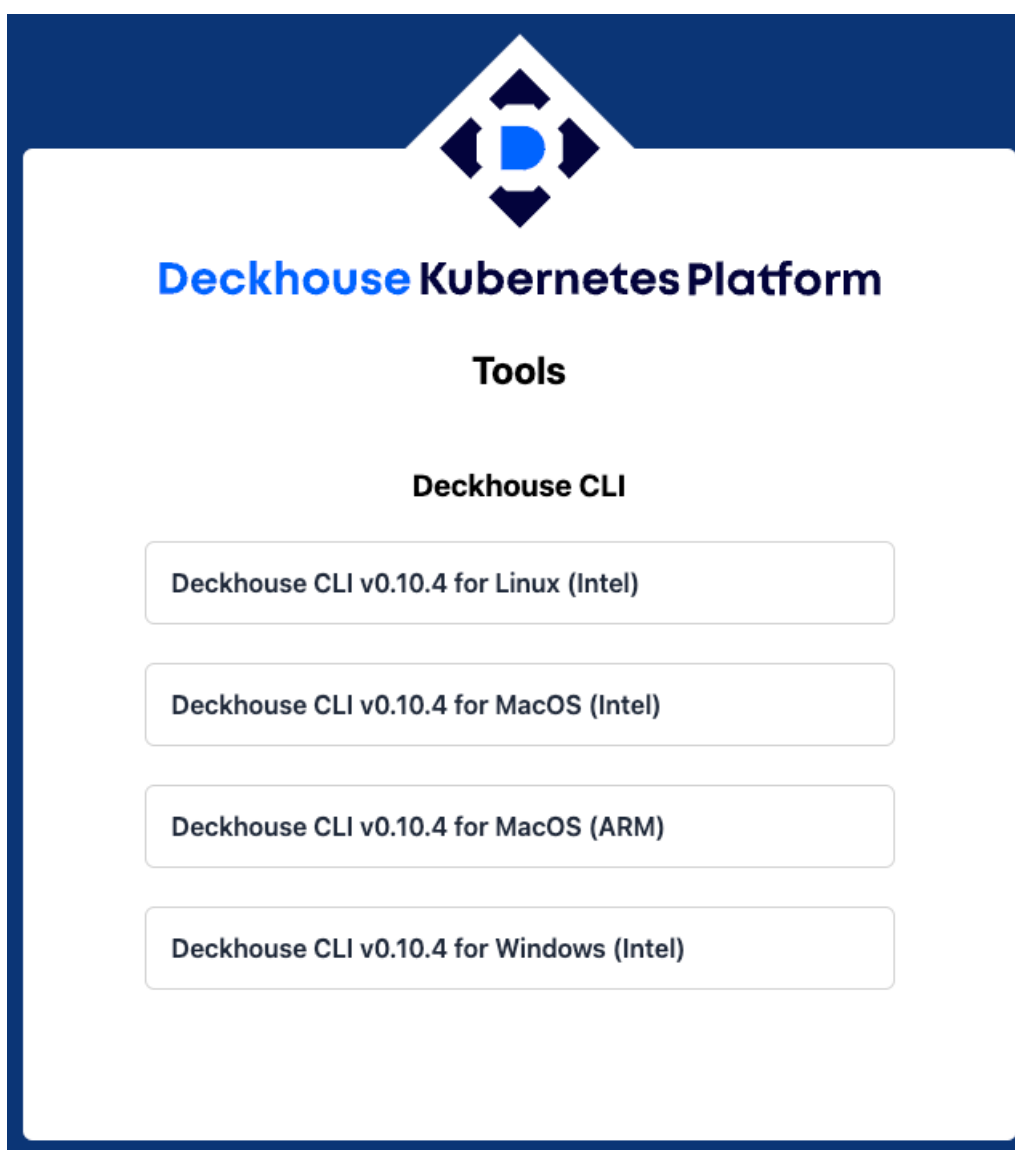
```

Веб-интерфейс загрузки CLI-утилит

Веб-интерфейс загрузки утилит Deckhouse обеспечивает централизованный доступ к CLI-утилитам для популярных операционных систем (Linux, macOS, Windows). С его помощью можно быстро найти и скачать исполняемые файлы нужной архитектуры и версии, а также получать уведомления об их обновлениях. Реализация этой функции осуществляется модулем `deckhouse-tools`.

Доступ к веб-интерфейсу

1. Для открытия веб-интерфейса введите в адресной строке браузера `tools.<ШАБЛОН_ИМЕН_КЛАСТЕРА>`, где `<ШАБЛОН_ИМЕН_КЛАСТЕРА>` — строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре `modules.publicDomainTemplate`. Формат адреса может отличаться в зависимости от конфигурации системы. Уточните точный URL у администратора.
2. При первом входе введите учетные данные пользователя. После успешной аутентификации откроется главная страница веб-интерфейса загрузки утилит Deckhouse, на которой будут доступны ссылки для загрузки утилиты Deckhouse CLI под различные операционные системы: Linux, macOS, Windows.



3. Выберите необходимую версию, скачайте и установите соответствующий исполняемый файл.

Дополнительные ресурсы

- Документация модуля `deckhouse-tools`

Веб-интерфейс управления секретами

Веб-интерфейс управления секретами (веб-интерфейс Stronghold) предназначен для управления механизмами секретов, аутентификацией и контролем доступа в кластере. Работа обеспечивается модулем stronghold.

i Подробнее инструменты безопасности, включая Stronghold, рассматриваются в курсе «Инструменты безопасности в Deckhouse Kubernetes Platform» в Deckhouse Академии.

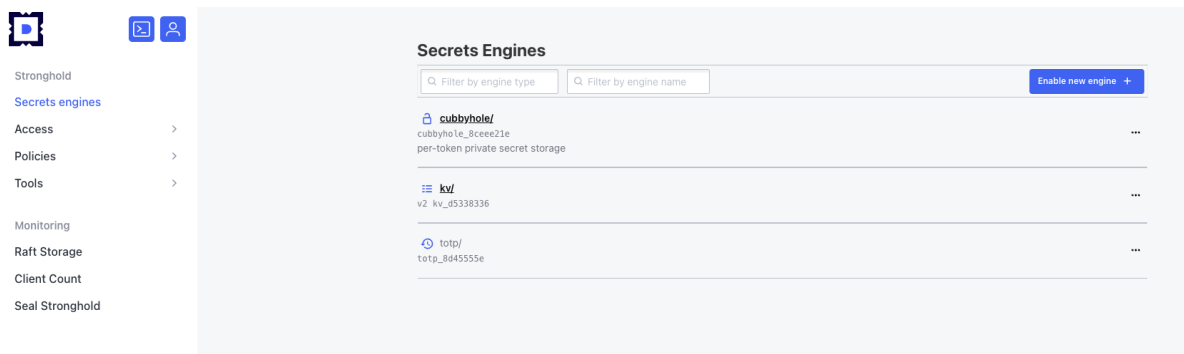
С его помощью можно:

- Просматривать и настраивать механизмы секретов, добавлять и редактировать секреты.
- Управлять методами аутентификации, группами пользователей и сущностями.
- Контролировать права доступа к данным, временные аренды (leases) и политики безопасности.

Доступ к веб-интерфейсу

Для перехода в веб-интерфейс Stronghold откройте адрес `stronghold.<ШАБЛОН_ИМЕН_КЛАСТЕРА>`, где `<ШАБЛОН_ИМЕН_КЛАСТЕРА>` — строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре `modules.publicDomainTemplate`.

1. При первом входе введите учетные данные пользователя.
2. После успешной аутентификации откроется главный экран Stronghold:



Работа с механизмами секретов

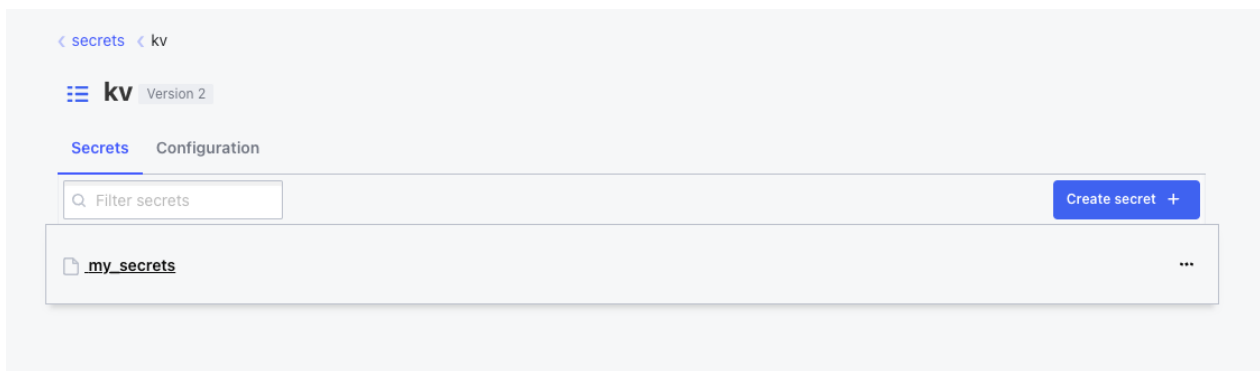
Инструкции по настройке распространённых типов механизмов — в документации Deckhouse Stronghold: механизм секретов KV и механизм секретов PKI.

i Работа с этими механизмами также подробно разобрана в курсе «Инструменты безопасности в Deckhouse Kubernetes Platform» в Deckhouse Академии.

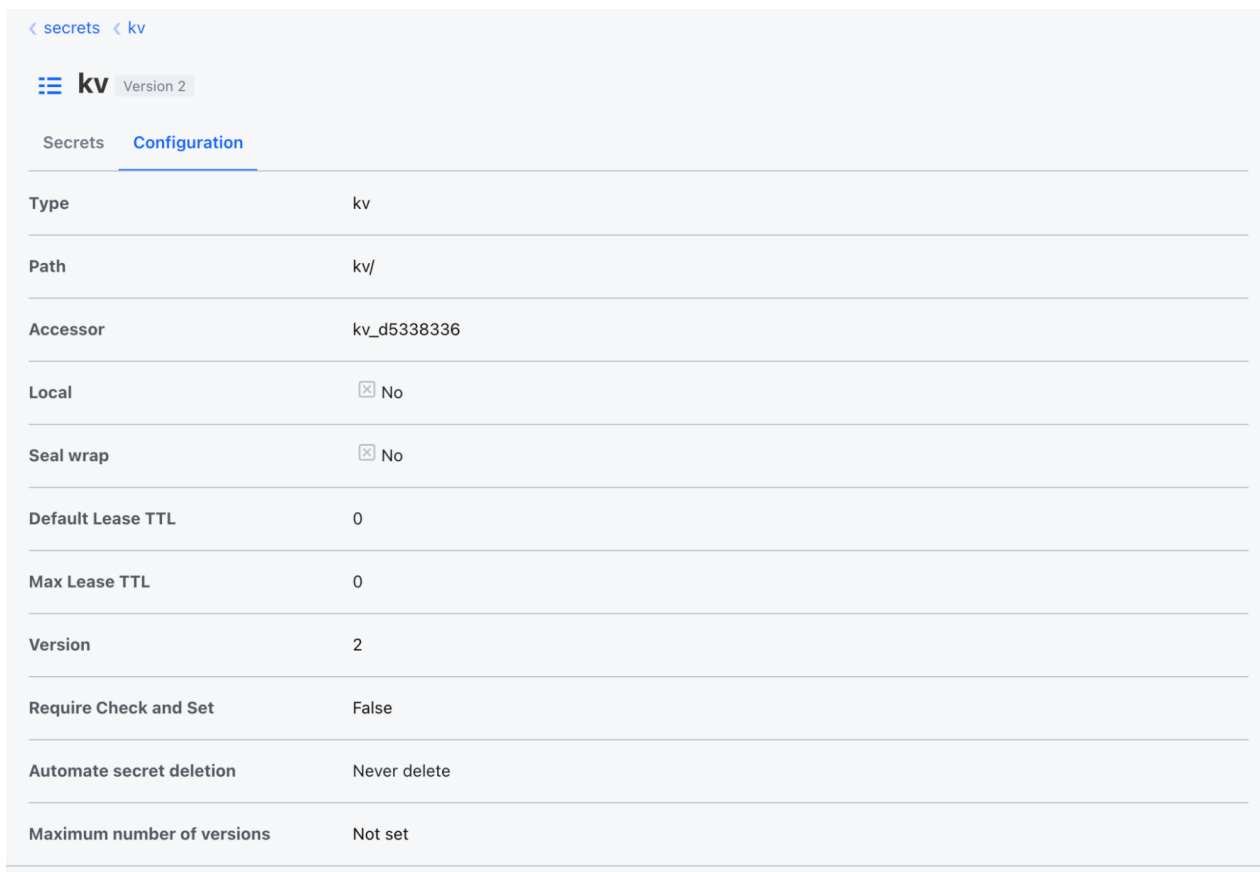
Просмотр информации о механизме секретов

Для просмотра информации о механизме секретов:

1. Кликните по его названию.
2. В открывшемся окне отобразятся вкладки:
 - «Secrets» — список секретов (ролей, ключей и т.д.).
 - «Configuration» — конфигурация механизма.
 - Кнопка для добавления нового секрета.

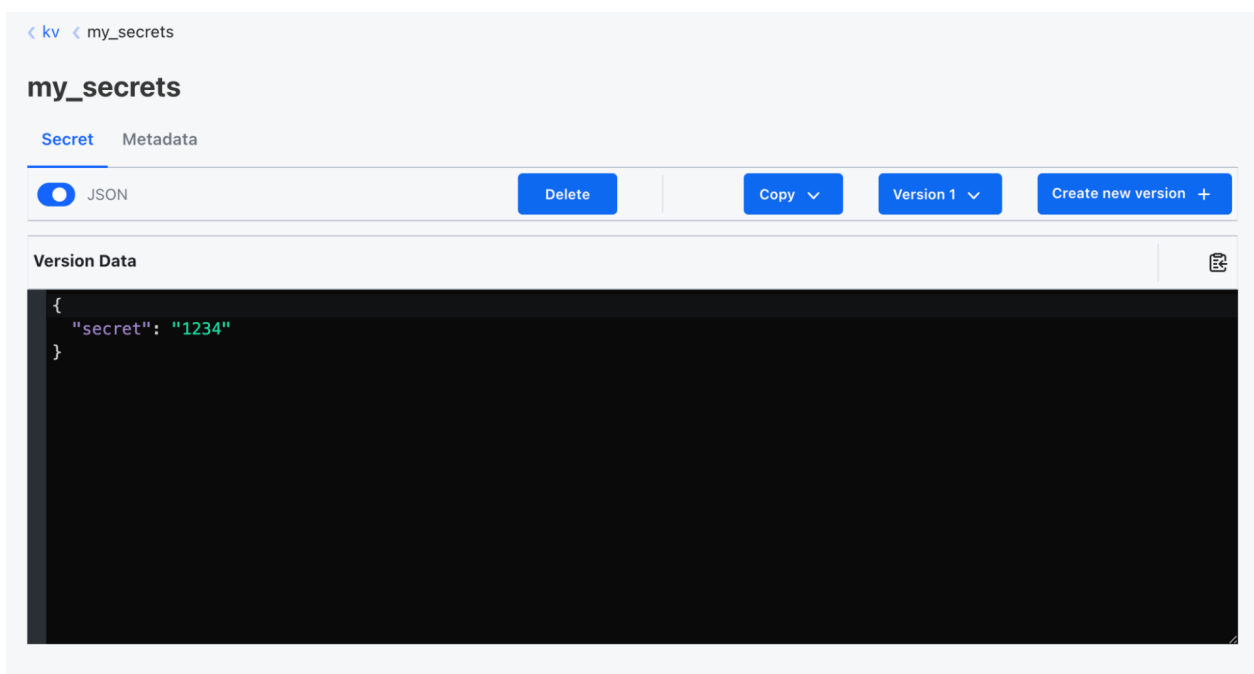


Для просмотра конфигурации механизма секретов перейдите на вкладку «Configuration». Содержимое вкладки зависит от просматриваемого механизма секретов.



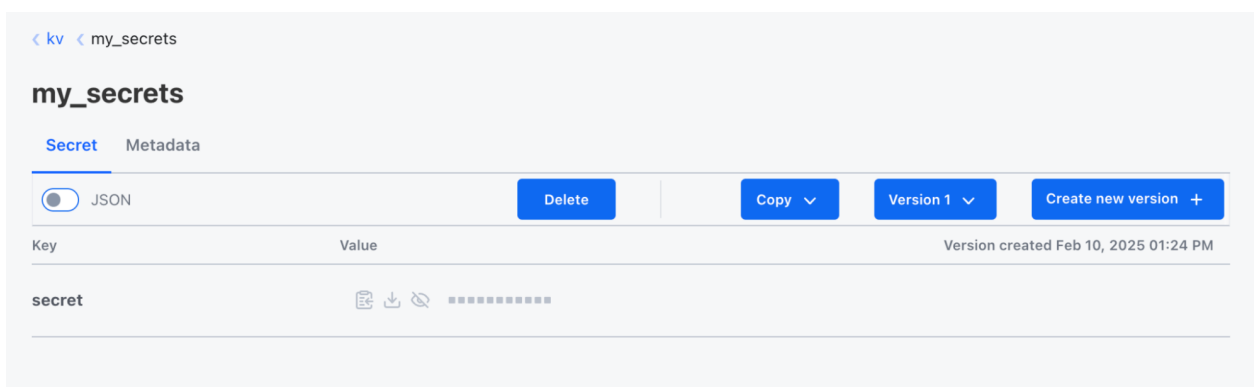
Для просмотра информации о секрете и его версиях (на примере механизма «Ключ-значение»):

1. Кликните по его названию в списке секретов.
2. В открывшемся окне будут доступны вкладки:
 - «Secret» — общая информация о секрете и его версиях.
 - «Metadata» — метаданные секрета.

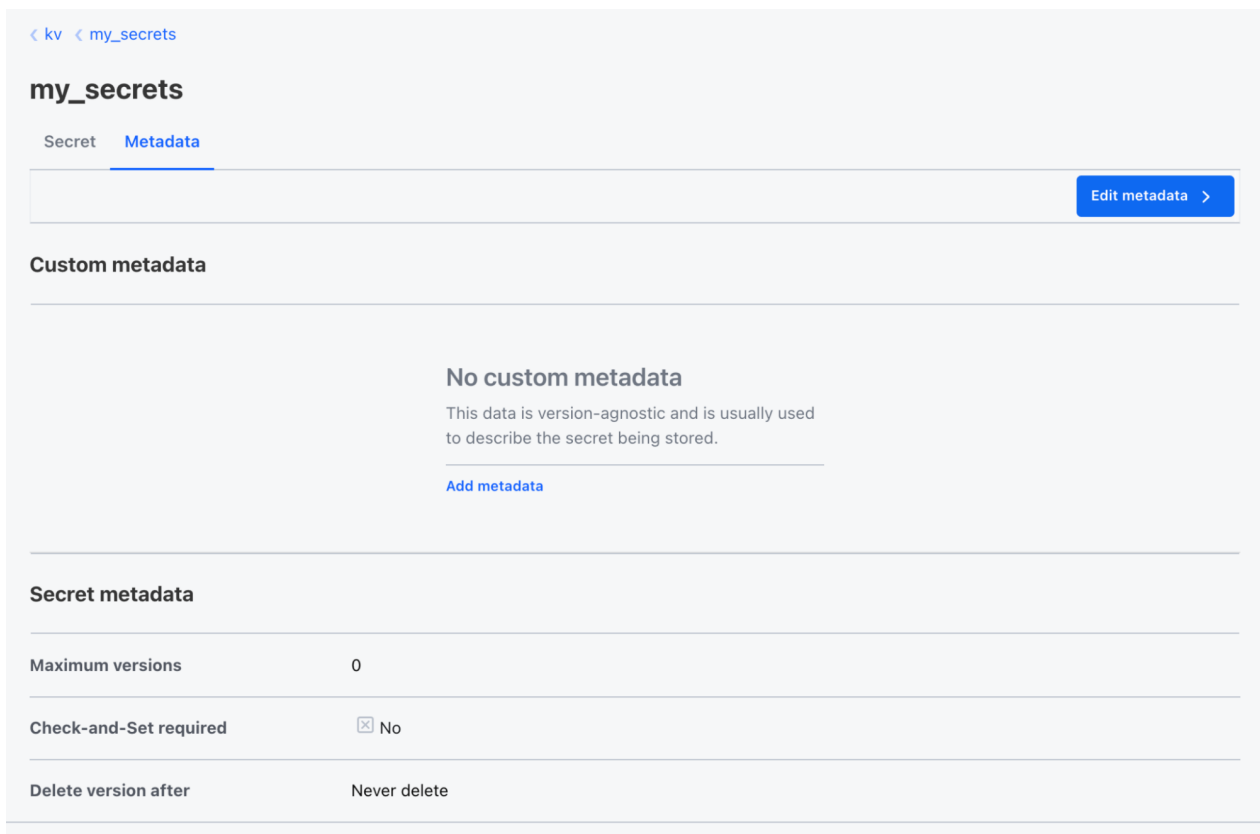


На вкладке «Secret» отображаются:

- Переключатель для просмотра сведений в формате JSON.
- Кнопки:
 - Удаление секрета;
 - Копирование секрета;
 - Выбор версии (если доступно);
 - Добавление новой версии.

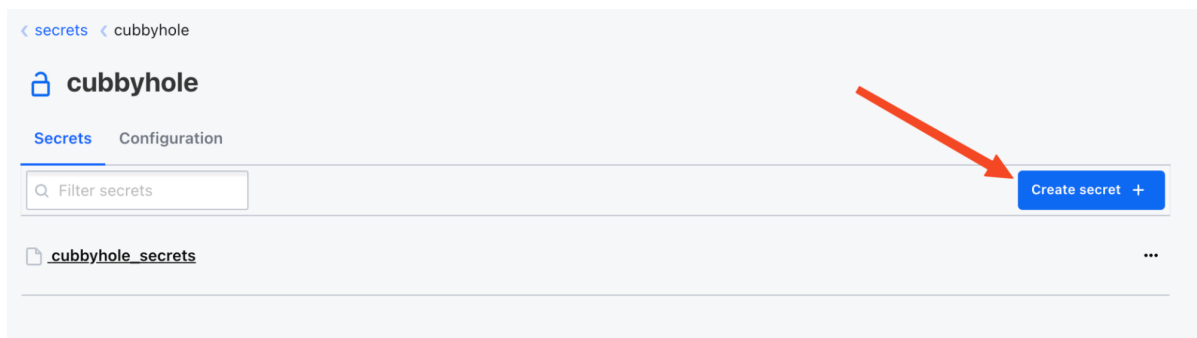


Для просмотра метаданных перейдите на вкладку «Metadata».



Чтобы добавить новый секрет:

1. Нажмите кнопку для добавления секрета (роли, ключа и т.д. — название кнопки зависит от механизма секретов) в разделе «Secrets».



2. В открывшейся форме укажите параметры (состав полей зависит от механизма секретов). Например, форма для добавления секрета «Cubbyhole» содержит:
 - переключатель для просмотра и редактирования секрета в формате JSON;
 - поле для указания пути к секрету («Path»);
 - поле для указания ключа;
 - поле для указания значения;
 - кнопку для добавления новой пары ключ-значение (если необходимо добавить несколько ключей, которые будут иметь одинаковый «Path»).
3. Для сохранения нажмите «Save».

Enable a Secrets Engine

Path

kv

Maximum number of versions

The number of versions to keep per key. Once the number of keys exceeds the maximum number set here, the oldest version will be permanently deleted. This value applies to all keys, but a key's metadata settings can overwrite this value. When 0 is used or the value is unset, Stronghold will keep 10 versions.

0

☐ **Require Check and Set**

If checked, all keys will require the cas parameter to be set on all write requests. A key's metadata settings can overwrite this value.

☒ **Automate secret deletion**

A secret's version must be manually deleted.

^ Hide Method Options

Version ⓘ

2

Description

☐ **List method when unauthenticated**

☐ **Local** ⓘ

☐ **Seal wrap** ⓘ

☒ **Default Lease TTL**

Lease will expire after

0

seconds

☐ **Max Lease TTL**

Stronghold will use the default lease duration.

Allowed managed keys

Add one item per row.

Add

Request keys excluded from HMACing in audit ⓘ

Add one item per row.

Add

Response keys excluded from HMACing in audit ⓘ

Add one item per row.

Add

Allowed passthrough request headers ⓘ

Add one item per row.

Add

Allowed response headers ⓘ

Add one item per row.

Add

Enable Engine

Back

© 2023–2025 Flant JSC. All rights reserved.

Управление доступом к данным и функциям Stronghold

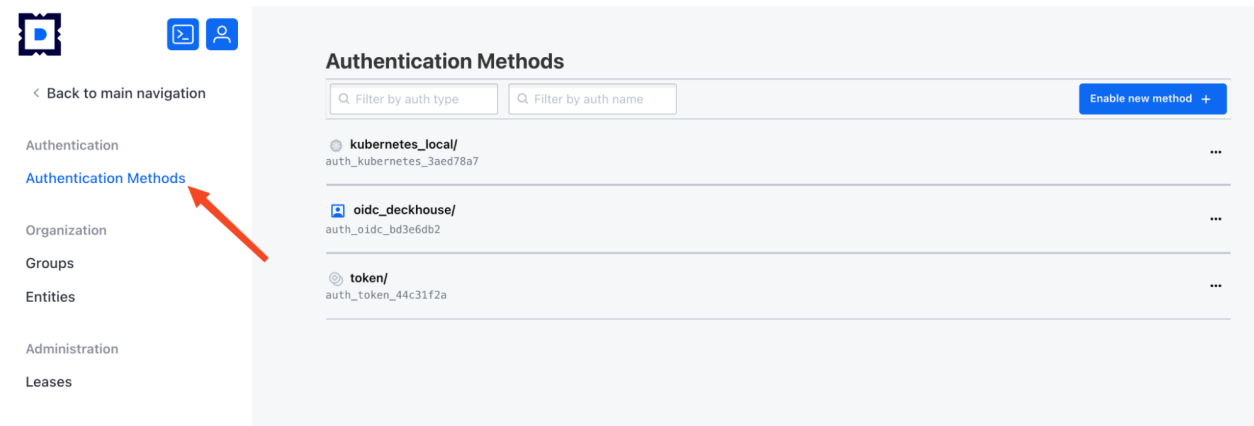
Управление доступом к данным и функциям Stronghold осуществляется в разделе **Access**.

Страница 189

Чтобы перейти в этот раздел, нажмите на «Access» на главном экране веб-интерфейса Stronghold. В левой части экрана раздела находится окно навигации по подразделам, вверху которого расположена ссылка для быстрого перехода на главный экран веб-интерфейса Stronghold. В центре — информация в зависимости от выбранного в данный момент подраздела (по умолчанию — «Методы аутентификации» («Authentication Methods»)).

Работа с методами аутентификации

Подраздел для работы с методами аутентификации открывается по умолчанию при переходе в раздел «Access» с главного экрана. Для перехода из других подразделов необходимо нажать на «Authentication Methods» в меню слева.



В этом подразделе доступны:

- список методов аутентификации, используемых в кластере;
- поля для фильтрации элементов списка;
- кнопка для добавления нового метода.

Для методов возможны следующие действия:

- просмотр конфигурации;
- изменение конфигурации;
- удаление метода.

Для просмотра информации о методе аутентификации, нажмите на его название или выберите пункт «View configuration». В окне с информацией о методе аутентификации отображаются вкладки с информацией о методе и кнопка для его конфигурирования.

Например, для метода аутентификации `oidc_deckhouse` в окне просмотра информации о нем отображается одна вкладка «Configuration» и кнопка «Configure».

Чтобы добавить метод аутентификации:

1. Нажмите кнопку добавления метода «Enable new method» в разделе «Authentication Methods».
2. Выберите необходимый метод из списка и нажмите «Next».
3. В открывшемся окне укажите параметры:
 - Поле «Path»;
 - Опции метода («Method options», по умолчанию скрыты, нажмите на заголовок блока для раскрытия).
4. Для сохранения метода нажмите «Enable Method». Для отмены и возврата к выбору метода нажмите «Back».

Работа с группами пользователей

Для работы с группами пользователей откройте раздел «**Groups**» в меню слева.

В этом подразделе доступны:

- список групп пользователей, имеющих в кластере;
- поля для фильтрации элементов списка;
- кнопка для добавления новой группы.

Для групп из списка возможны следующие действия:

- просмотр детальной информации о группе;
- изменение настроек группы;

- удаление групп.

Для просмотра информации о группе пользователей нажмите на её название или выберите пункт «Details».

Чтобы добавить группу пользователей:

1. Нажмите кнопку «Create group» в подразделе “Groups”.
2. Заполните форму создания группы.
3. Для сохранения группы нажмите «Create». Для отмены и возврата к списку групп нажмите «Back».

Работа с сущностями и псевдонимами

Сущности (Entities) в Stronghold — это логические объекты, представляющие пользователя или приложение. Они позволяют объединять различные методы аутентификации под единым идентификатором.

Для работы с сущностями и псевдонимами откройте раздел **Entities** в левом меню раздела управления доступами.

В центре экрана расположены две вкладки:

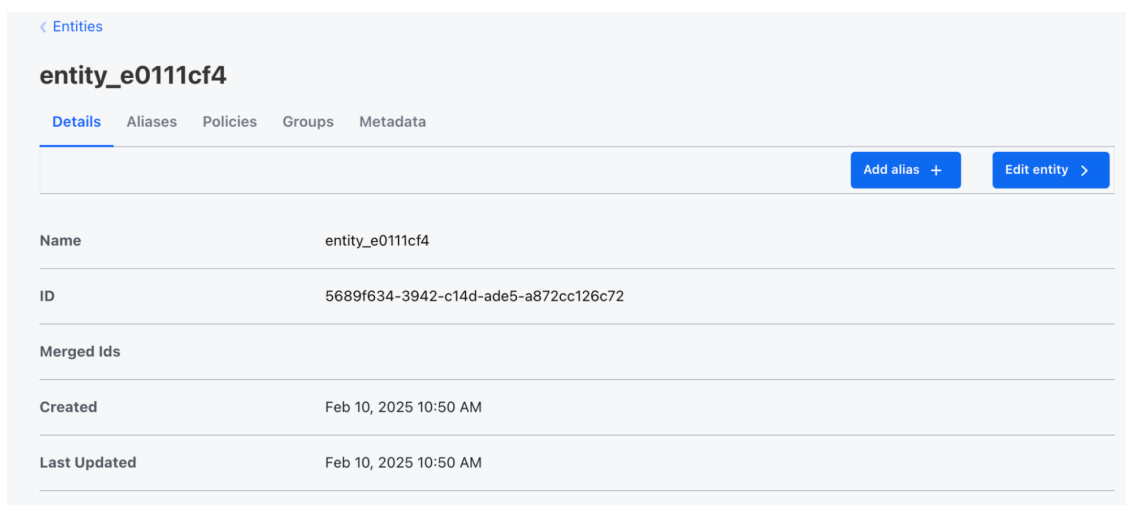
- **Entities** — список сущностей;
- **Aliases** — список псевдонимов.

Также доступны:

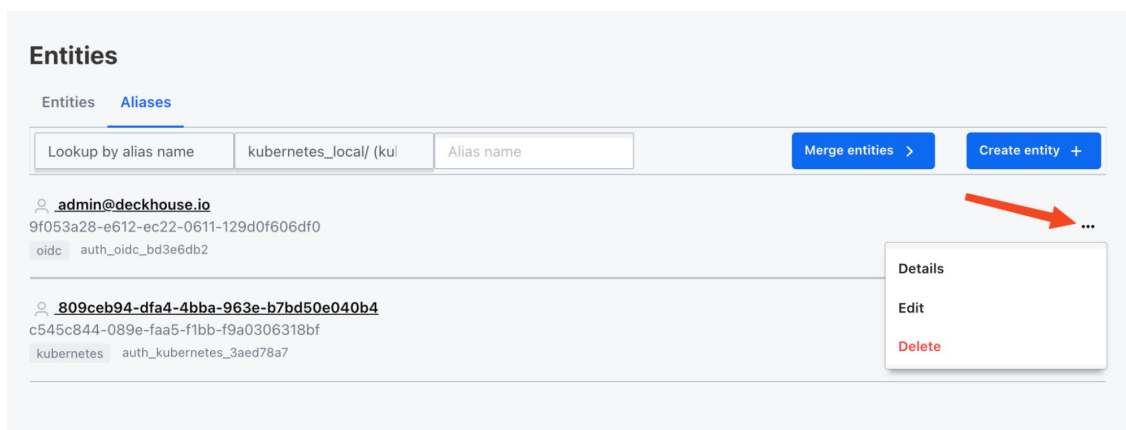
- Поля для фильтрации элементов списка;
- Кнопка **Merge entities** для объединения сущностей;
- Кнопка **Create entity** для добавления новой сущности.

Возможные действия с сущностями и псевдонимами:

- Для сущностей (вкладка «Entities»):
 - Просмотр детальной информации;



- Создание сущности.



- Удаление сущности.

Merge Entities

Warning
Metadata on merged entities is not preserved, you will need to recreate it on the entity you merge to.

Entity to merge to

Entities to merge from
Add one item per row.

Add

☐ Keep MFA secrets from the "to" entity if there are merge conflicts

Save
Cancel

- Для псевдонимов (вкладка «Aliases»):
 - Просмотр детальной информации;

[< Entity aliases](#)

admin@deckhouse.io

[Details](#) [Metadata](#)

Edit entity alias >

Name	admin@deckhouse.io
ID	9f053a28-e612-ec22-0611-129d0f606df0
Entity ID	5689f634-3942-c14d-ade5-a872cc126c72
Mount	auth/oidc_deckhouse/ oidc auth_oidc_bd3e6db2
Created	Feb 10, 2025 10:50 AM
Last Updated	Feb 10, 2025 10:50 AM

- Создание псевдонима;

Create Entity Alias for 5689f634-3942-c14d-ade5-a872cc126c72

Name

Auth Backend

Create
Cancel

- Редактирование псевдонима;
- Удаление псевдонима.

Управление временными правами доступа к секретам и ресурсам (Leases)

Чтобы управлять временными правами доступа к секретам и ресурсам (Leases), нажмите на «Leases», откроется окно для поиска информации об аренде по ее идентификатору.

Lookup a Lease

Lease ID

If you know the id of a lease, enter it above to lookup details of the lease.

Lookup

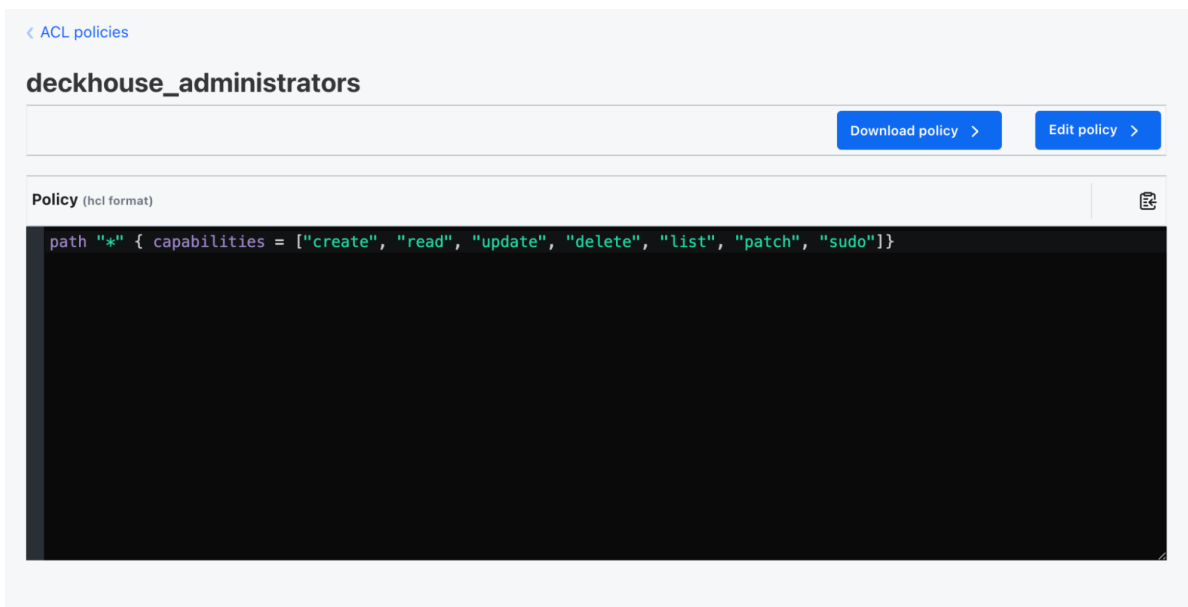
Работа с политиками контроля доступа

Работа с политиками контроля доступа в Stronghold осуществляется в разделе «Policies». Перейти в него можно, нажав на «Access». В левой части экрана раздела для работы с политиками находится окно навигации. В центре размещен список политик, фильтр для поиска нужной и меню для добавления новой политики.

Чтобы выбрать нужное действие, нажмите на меню с тремя точками, которая находится в конце строки с названием политики.

Для политик из списка доступны следующие действия:

- **просмотр детальной информации о политике.** Чтобы просмотреть информацию о политике, нажмите на её название или выберите пункт «Details». В окне с информацией о политике отображаются сведения о ней в формате HCL, а также меню для загрузки данных на компьютер и для редактирования политики.



- **редактирование политики.** Чтобы добавить политику, нажмите «Create ACL policy» на экране для работы с политиками. Откроется форма для ввода имени политики и ее описания в формате HCL.

[< ACL Policies](#)

Create ACL Policy

Name

Policy

☒
☐ Upload file

```
1
```

You can use Alt+Tab (Option+Tab on MacOS) in the code editor to skip to the next field. See [example template](#).

Create policy Cancel

- **удаление политики.**

Работа с дополнительными инструментами

Работа с дополнительными инструментами в Stronghold осуществляется в разделе «Tools». В левой части раздела находится окно навигации по инструментам. В центре отображаются поля выбранного инструмента.

Доступны инструменты:

- **Wrap** - для создания wrapping token (токена обертки) для безопасной передачи секретов, который временно «упаковывает» конфиденциальные данные/секреты. Этот токен может быть передан другому пользователю или приложению, которое затем сможет «развернуть» (unwrap) его и получить доступ к «упакованным» данным.

[< Back to main navigation](#)

Tools

Wrap

Lookup

Unwrap

Rewrap

Random

Hash

API Explorer

Wrap Data

Data to wrap (json-formatted)

```
1 {
2 }
```

☒ Wrap TTL
Stronghold will use the default (30m)

Wrap data

- **Lookup** - для просмотра информации о токенах, секретах, арендах (Lease) и иных объектах в Stronghold. С его помощью можно просматривать метаданные, сроки действия, политики доступа и другую информацию, связанную с объектами.

The screenshot shows the 'Lookup Token' page. On the left is a navigation sidebar with icons for Tools, Wrap, Lookup (highlighted), Unwrap, Rewrap, Random, Hash, and API Explorer. The main content area has a title 'Lookup Token' and a section 'Wrapped token' with the instruction 'Enter your wrapped token here to display its information.' Below this is a large text input field and a blue button labeled 'Lookup token'.

- **Unwrap** - для распаковки wrapping token (токена обертки) и получения доступа к «упакованным» данным

The screenshot shows the 'Unwrap Data' page. The navigation sidebar is identical to the previous one, with 'Unwrap' highlighted. The main content area has a title 'Unwrap Data' and a section 'Wrapped token' with the instruction 'Enter your wrapped token here to unwrap it and return its original value.' Below this is a large text input field and a blue button labeled 'Unwrap data'.

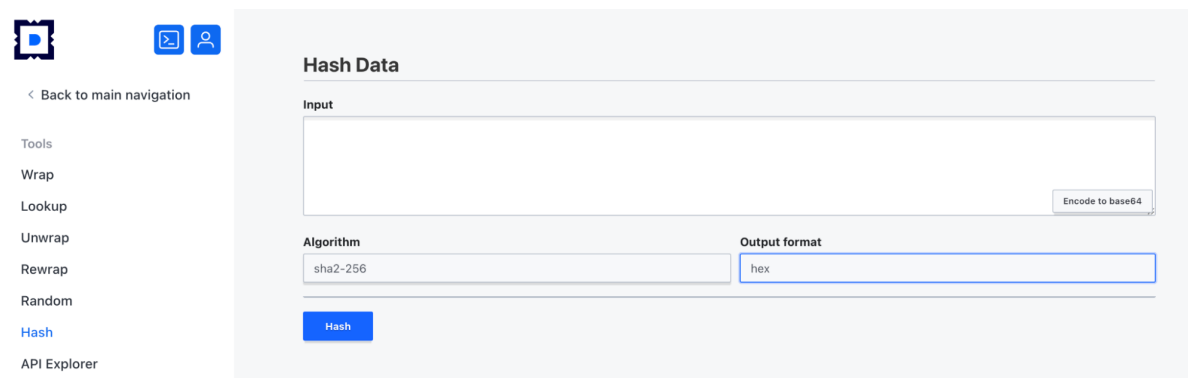
- **Rewrap** - для переупаковки — создания нового wrapping token (токена обертки) на основе существующего. Это позволяет продлить срок действия токена или изменить его параметры без необходимости раскрывать защищаемые данные.

The screenshot shows the 'Rewrap Token' page. The navigation sidebar is identical to the previous ones, with 'Rewrap' highlighted. The main content area has a title 'Rewrap Token' and a section 'Wrapped token' with the instruction 'Enter your wrapped token here to rewrap it and refresh its TTL.' Below this is a large text input field and a blue button labeled 'Rewrap token'.

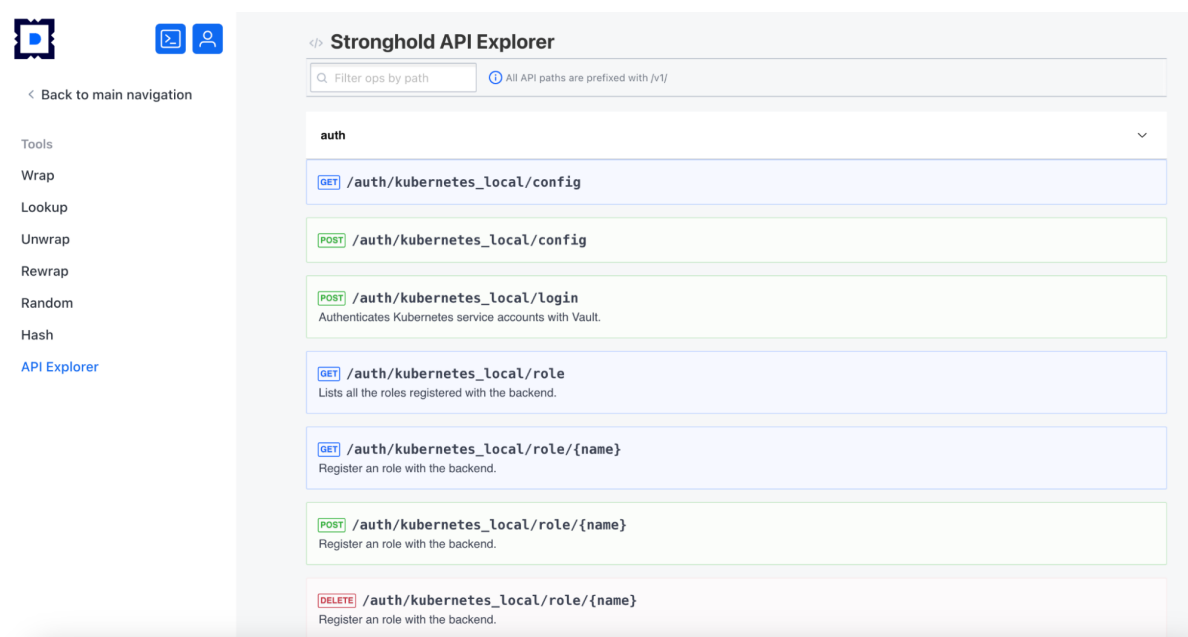
- **Random** - для генерации криптографически безопасных случайных данных для создания уникальных идентификаторов, токенов, паролей или иных данных, для которых важна высокая степень случайности и безопасности.

The screenshot shows the 'Random Bytes' page. The navigation sidebar is identical to the previous ones, with 'Random' highlighted. The main content area has a title 'Random Bytes' and two input fields: 'Number of bytes' (containing '45673') and 'Output format' (containing 'hex'). Below these fields is a blue button labeled 'Generate'.

- **Hash** - для генерации хешей для различных данных. Поддерживается несколько алгоритмов кеширования.



- **API Explorer** — для взаимодействия с API Stronghold через графический интерфейс.



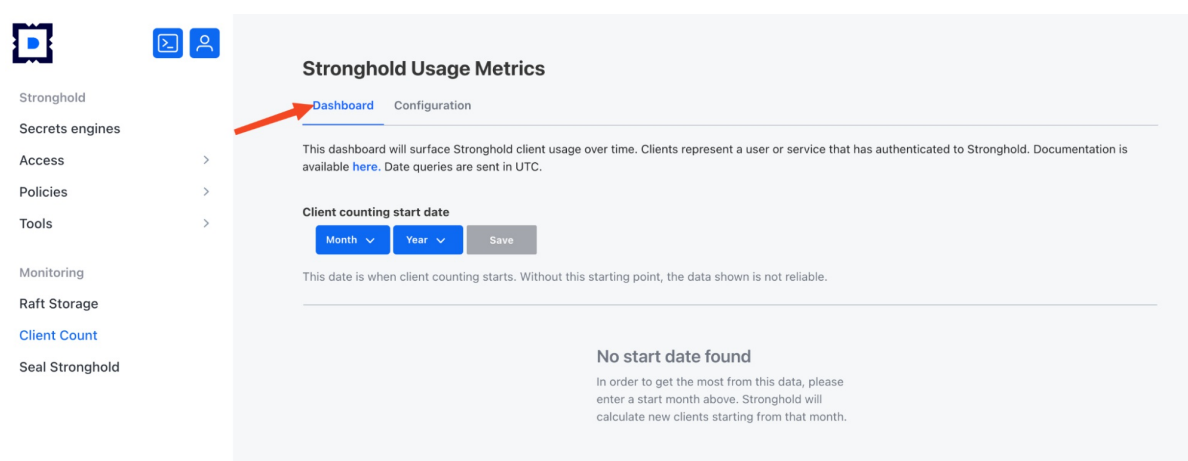
Мониторинг состояния кластера Raft Stronghold

Мониторинг состояния Raft кластера Stronghold осуществляется в разделе «Raft Storage». В левой части интерфейса находится окно навигации по разделам. В центре отображается информация о лидере и узлах кластера, а также меню «Snapshots» для создания резервной копии данных Raft кластера Stronghold и восстановления данных из резервной копии.

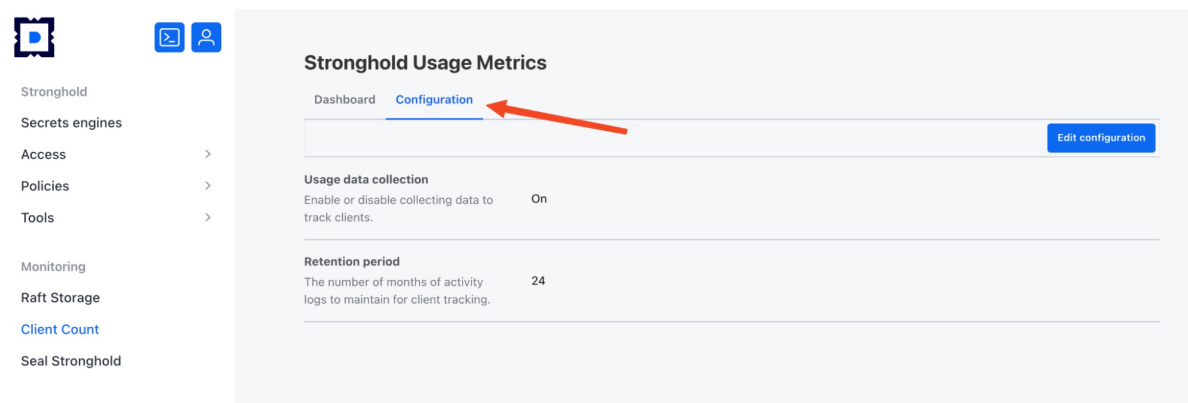
Мониторинг активности и оценка нагрузки на Stronghold

Мониторинг активности и оценка нагрузки на Stronghold осуществляется в разделе «Client Count». В левой части интерфейса находится окно навигации по разделам. В центре размещены две вкладки:

- **Dashboard** с информацией о количестве уникальных клиентов за текущий месяц и кнопками выбора другого периода.



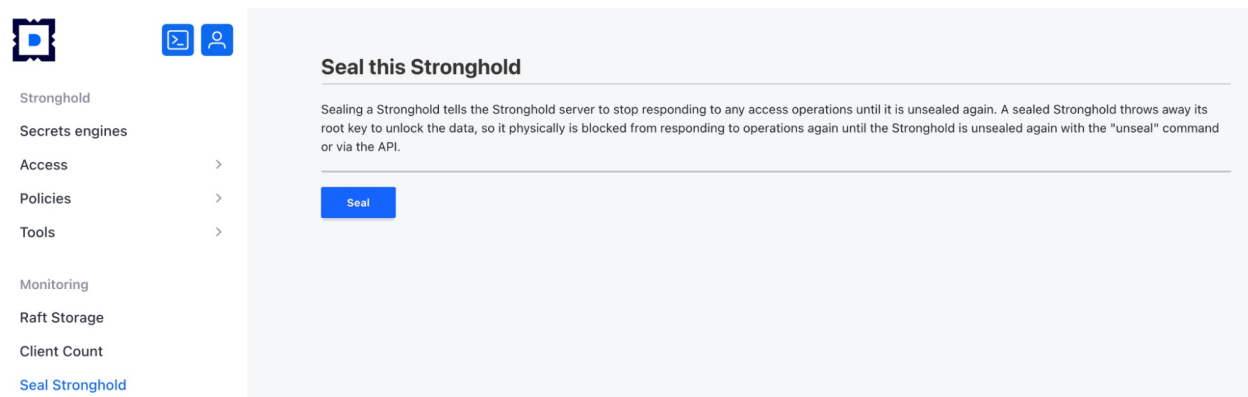
- **Configuration** с настройками сбора метрик и возможностью отредактировать их (для этого необходимо нажать «Edit configuration»).



Запечатывание и распечатывание хранилища секретов

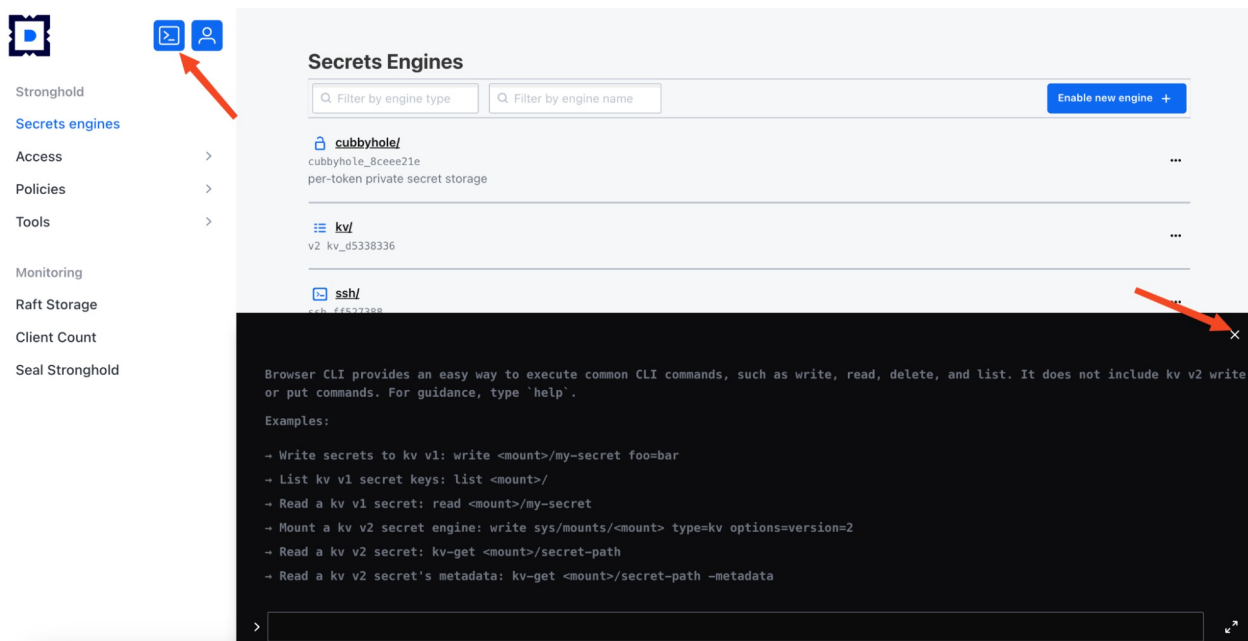
Запечатывание и распечатывание хранилища секретов осуществляется в разделе «Seal Stronghold». В левой части интерфейса находится окно навигации по разделам. В центре отображается меню для запечатывания и распечатывания хранилища секретов (в зависимости от его текущего состояния).

Когда хранилище находится в состоянии «запечатано» (sealed), он не может обрабатывать запросы на чтение или запись секретов.



Работа со Stronghold CLI

Stronghold CLI — инструмент, который применяется для различных операций по управлению секретами, настройке политик, управлению пользователями и т.д. Вызвать Stronghold CLI можно, находясь в любом из разделов интерфейса. Для его запуска нажмите кнопку в левом верхнем углу окна, для закрытия - в левом.



Дополнительные ресурсы

- Документация модуля `stronghold`

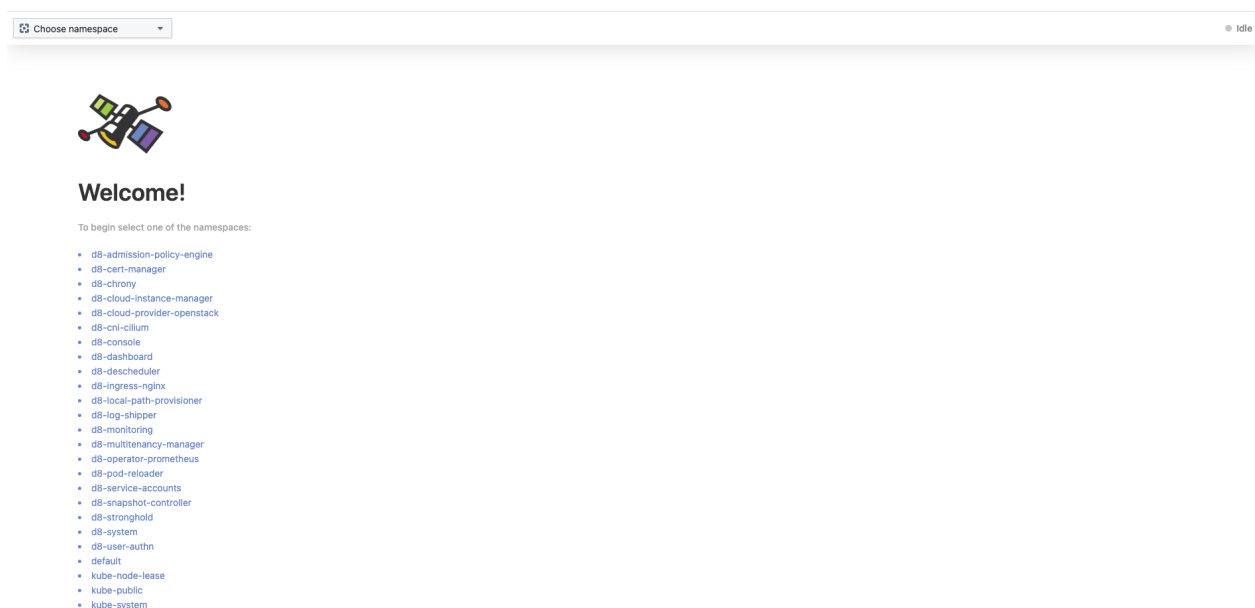
Веб-интерфейс визуализации сетевого стека кластера

Веб-интерфейс визуализации сетевого стека кластера позволяет оперативно анализировать сетевые взаимодействия между подами, сервисами и внешними ресурсами, анализировать сетевую активность и выявлять проблемы с сетью. Реализация этого интерфейса осуществляется модулем `cilium-hubble`.

Доступ к веб-интерфейсу

Для перехода в веб-интерфейс визуализации сетевого стека кластера откройте адрес `hubble.<ШАБЛОН_ИМЕН_КЛАСТЕРА>`, где `<ШАБЛОН_ИМЕН_КЛАСТЕРА>` – строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре `modules.publicDomainTemplate`.

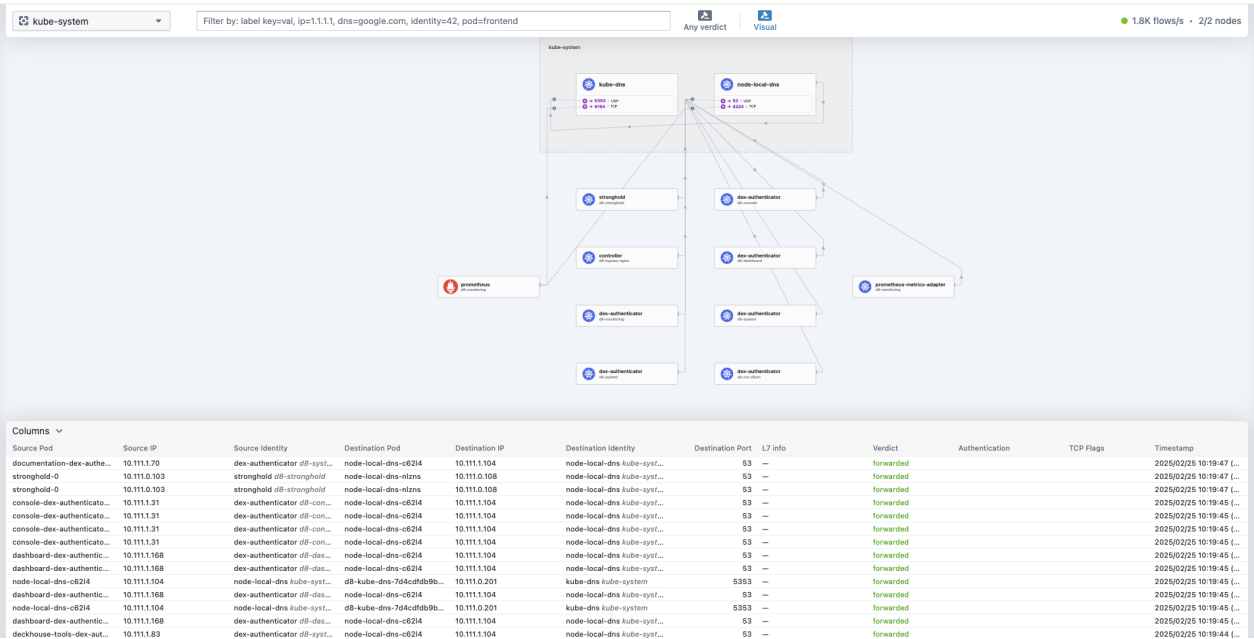
При переходе по адресу `hubble.<ШАБЛОН_ИМЕН_КЛАСТЕРА>` откроется экран выбора пространства имен, для которого будет визуализирован сетевой стек.



Выберите пространство имен с помощью выпадающего списка в левой верхней части экрана или нажав на название нужного пространства имен в списке в центре экрана.

После выбора пространства имен откроется экран с визуализацией сетевого стека и средствами анализа. Он состоит из следующих частей:

- Верхняя панель с фильтрами и краткой сводкой по кластеру (количество потоков и количество узлов). Подробнее о фильтрации данных для отображения — в разделе [Фильтрация данных для отображения](#).
- Схема сетевых потоков. Подробнее о работе с визуальной схемой сетевых потоков — в разделе [Работа со схемой сетевых потоков](#).
- Таблица сетевых потоков и событий. Подробнее о работе с таблицей — в разделе [Работа с таблицей сетевых потоков и событий](#).

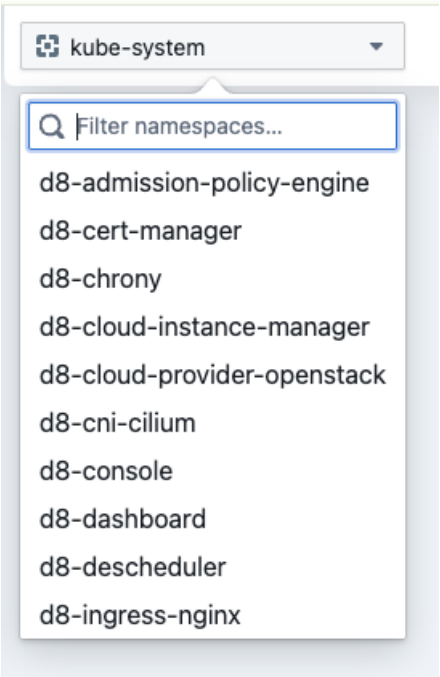


Данные на схеме и в таблице сетевых потоков отображаются в реальном времени.

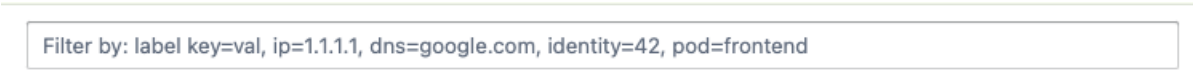
Фильтрация данных для отображения

Чтобы отфильтровать отображаемые данные о сетевом стеке и потоках, воспользуйтесь верхней панелью с фильтрами. Здесь расположены фильтры:

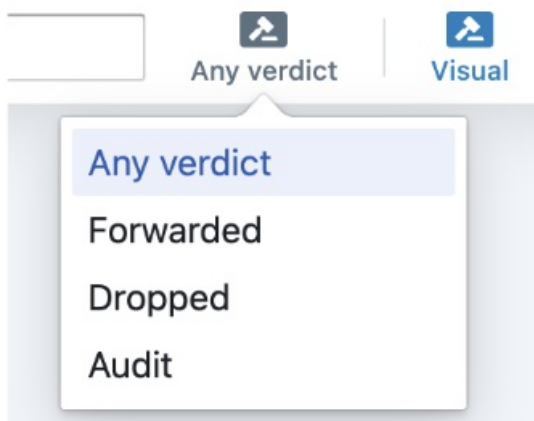
- для выбора пространства имен (выпадающий список в левой части панели);



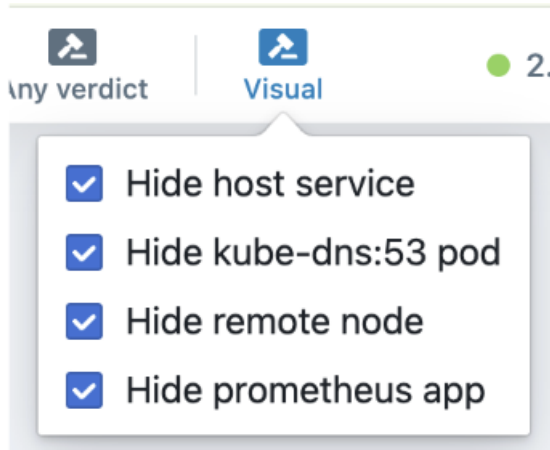
- для выбора ресурсов пространства имен, для которых нужно отобразить потоки (поле ввода в центральной части панели);



- для выбора сетевых потоков на основе решения («вердикта»), принятого по ним Cilium;



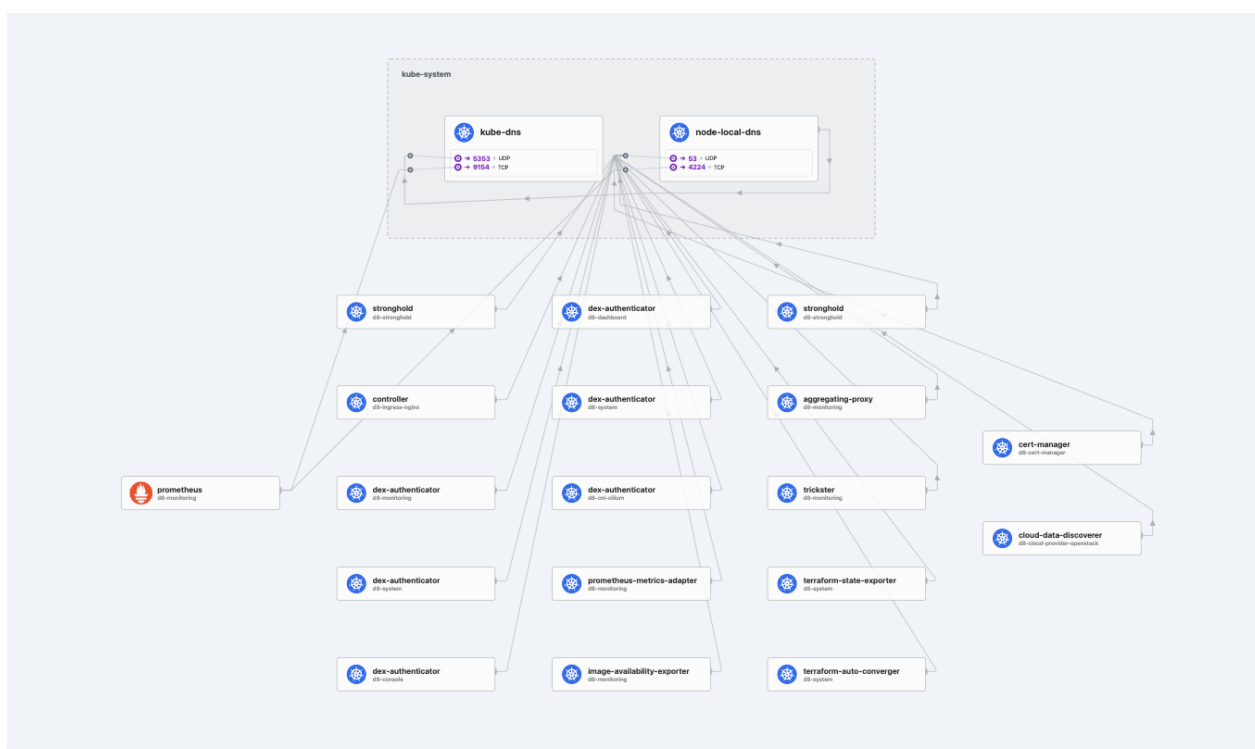
- для выбора элементов схемы анализируемого пространства имен.



Работа со схемой сетевых потоков

Схема сетевых потоков для выбранного пространства имен отображается в средней части экрана с визуализацией сетевого стека и средствами анализа. На схеме отображаются ресурсы выбранного пространства имен, расположенные в прямоугольнике с названием пространства имен, и внешние элементы, с которыми они взаимодействуют.

Чтобы посмотреть детальную информацию (список лейблов, сетевые взаимодействия и т.д.) по конкретному ресурсу на схеме, нажмите на его название.



Работа с таблицей сетевых потоков и событий

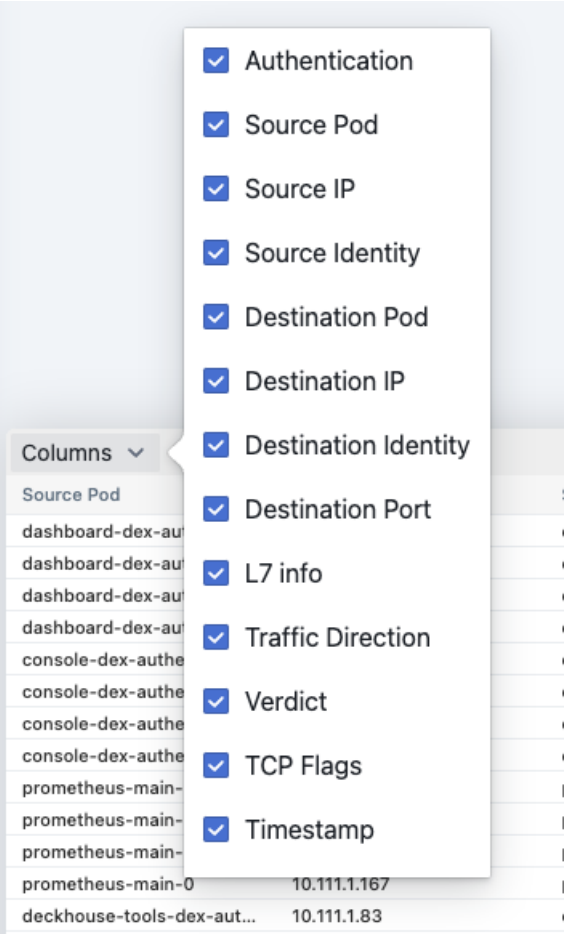
Каждая строка таблицы содержит следующую информацию о сетевом потоке:

- имя пода — источника потока (столбец «Source Pod»);
- IP-адрес пода — источника потока (столбец «Source IP»);
- идентификатор сущности — источника потока (столбец «Source Identity»);
- имя пода — получателя потока (столбец «Destination Pod»);
- IP-адрес пода — получателя потока (столбец «Destination IP»);
- идентификатор сущности-получателя (столбец «Destination Identity»);
- номер порта назначения (столбец «Destination Port»);
- информация о прикладном уровне (Layer 7), если поток использует протоколы HTTP (столбец «L7 info»);
- результат («вердикт») обработки сетевого потока Cilium (столбец «Verdict»);
- информация о результатах проверки подлинности сетевого потока, если такая проверка выполнялась (столбец «Authentication»);
- флаги TCP, связанные с потоком (столбец «TCP Flags»);
- временный лейбл потока (столбец «Timestamp»).

Columns											
Source Pod	Source IP	Source Identity	Destination Pod	Destination IP	Destination Identity	Destination Port	L7 info	Verdict	Authentication	TCP Flags	Timestamp
dashboard-dex-authentic...	10.111.1.168	dex-authenticator d8-das...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
dashboard-dex-authentic...	10.111.1.168	dex-authenticator d8-das...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	d8-kube-dns-7d4cd9b9b...	10.111.0.215	kube-dns kube-system	5353	—	forwarded			2025/02/25 12:11:40 (...)
node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	d8-kube-dns-7d4cd9b9b...	10.111.0.215	kube-dns kube-system	5353	—	forwarded			2025/02/25 12:11:40 (...)
dashboard-dex-authentic...	10.111.1.168	dex-authenticator d8-das...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
dashboard-dex-authentic...	10.111.1.168	dex-authenticator d8-das...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
console-dex-authenticat...	10.111.1.31	dex-authenticator d8-con...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
console-dex-authenticat...	10.111.1.31	dex-authenticator d8-con...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
console-dex-authenticat...	10.111.1.31	dex-authenticator d8-con...	node-local-dns-c6214	10.111.1.104	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:40 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)
prometheus-metrics-adap...	10.111.0.133	prometheus-metrics-adap...	node-local-dns-nlzm...	10.111.0.108	node-local-dns kube-syst...	53	—	forwarded			2025/02/25 12:11:39 (...)

Настройка набора столбцов таблицы

Чтобы настроить набор столбцов, отображаемых в таблице, нажмите «Columns» в ее левой верхней части и выберете нужные.



Просмотр информации о записях таблицы

Чтобы посмотреть информацию о записи таблицы в текстовом виде, нажмите в любой части

соответствующей строки. Информация отобразится в правой части таблицы. Данные здесь отображаются независимо от того, какой набор столбцов выбран для отображения в таблице.

Flow Details	
Timestamp	2025-02-25T09:18:37.673Z
Verdict	forwarded
Traffic direction	ingress
Cilium event type	to-endpoint
Source pod	stronghold-0
Source identity	15378
Source labels	instance=stronghold name=stronghold apps.kubernetes.io/pod-index=0 component=server io.cilium.k8s.namespace.labels.managed-by=Helm

Использование мультитенантности

Мультитенантность — 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 мультитенантное 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 мультитенантности.

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.

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 создался:

```
d8 k get projects
```

В списке проектов появится новый проект, соответствующий неймспейсу:

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 неймспейсе, то применить шаблон не получится.

Подробнее с шаблонами проектов и их созданием можно ознакомиться в разделе администрирования.

Дополнительные ресурсы

- Кастомные ресурсы модуля `multitenancy-manager`
- Документация модуля `secret-copier`

Обзор

Managed-сервисы позволяют пользователям создавать и использовать готовые прикладные сервисы с помощью кастомных ресурсов Kubernetes.

Как это работает

Пользователь создаёт в своём неймспейсе (проекте) кастомный ресурс managed-сервиса, например Postgres, и описывает в нём желаемую конфигурацию: ресурсы, режим работы, топологию, пользователей и другие параметры. Ресурс ссылается на класс managed-сервиса (например, `PostgresClass`), который подготовил администратор кластера, — класс определяет допустимые параметры и ограничения.

Оператор отслеживает такие ресурсы и создаёт или изменяет рабочие экземпляры managed-сервиса в соответствии с указанной конфигурацией. Результат и текущее состояние managed-сервиса отражаются в статусе ресурса.

Используйте этот раздел, чтобы:

- создать managed-сервис в своём неймспейсе (проекте);
- настроить параметры экземпляра managed-сервиса, топологию и репликацию;
- подключиться к созданному managed-сервису;
- настроить резервное копирование и восстановление;
- выполнить базовые операции сопровождения;
- проверить состояние ресурса и результат настройки;
- диагностировать частые проблемы.

Создание и использование PostgreSQL описано в разделе Managed PostgreSQL, частые проблемы и их диагностика — в разделе «Частые вопросы».

Сведения о подготовке инфраструктуры, настройке классов managed-сервисов и других административных параметрах приведены в разделе «Администрирование».

Managed PostgreSQL

Пользователь может создавать и настраивать PostgreSQL, если в кластере DKP администратором включена такая возможность.

Пользователь задаёт требуемую конфигурацию с помощью объекта Postgres, указывающего на конкретный класс сервиса (`PostgresClass`), который определяет доступные параметры и ограничения. `PostgresClass` создаёт и настраивает администратор кластера.

В руководстве используются два примера:

- `app-postgres` — основной пример для создания и эксплуатации PostgreSQL: ресурсы,

режим `Cluster`, репликация, пользователи, базы данных, параметры PostgreSQL, TLS и наблюдаемость;

- `snapshot-pg` — отдельный пример для создания и восстановления снимков, поскольку для него требуется `StorageClass` с поддержкой CSI-снимков.

i

В примерах используются два worker-узла, чтобы экземпляры PostgreSQL в режиме `Cluster` могли размещаться на разных узлах. Кластер находится в одной зоне `default`.

Проверка доступных ресурсов

Перед созданием Postgres проверьте доступные ресурсы worker-узлов. Это позволяет подобрать значения CPU и памяти для примера с учётом реальной загрузки кластера.

Сначала посмотрите список узлов:

```
d8 k get nodes -O wide
```

В примере доступны два worker-узла.

Пример вывода:

NAME	STATUS	ROLES	AGE	VERSION
worker-1	Ready	worker	25d	v1.34.9
worker-2	Ready	worker	43m	v1.34.9

Проверьте занятые ресурсы первого worker-узла:

```
d8 k describe node worker-1 | grep -A 5 "Allocated resources"
```

Пример вывода:

```
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests           Limits
-----
cpu                 1104m (28%)        500m (12%)
memory              4096854330 (53%)   390Mi (5%)
```

Проверьте второй worker-узел:

```
d8 k describe node worker-2 | grep -A 5 "Allocated resources"
```

Пример вывода:

```
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests           Limits
-----
cpu                 472m (12%)         500m (12%)
memory              1004Mi (13%)        256Mi (3%)
```

Проверка хранилища

Перед созданием Postgres проверьте доступные `StorageClass` и выберите класс хранилища, в котором будут размещаться данные PostgreSQL:

```
d8 k get storageclass
```

Пример вывода тестового стенда:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION			
local	csi.dvp.deckhouse.io	Delete	WaitForFirstConsumer
true			
replicated (default)	csi.dvp.deckhouse.io	Delete	WaitForFirstConsumer
true			



Параметр `spec.instance.persistentVolumeClaim.storageClassName` задаётся только при создании Postgres. Изменить его после создания нельзя.

Основной пример: создание Postgres

Создайте неймспейс:

```
d8 k create namespace my-postgres
```

Для создания объекта Postgres необходимо выполнить следующие шаги:

- Выбрать PostgresClass;
- Настроить ресурсы;
- Выбрать режим работы;
- Настроить топологию и режим репликации;
- Создать логическую базу данных и пользователя;
- Настроить параметры PostgreSQL;
- Настроить TLS;
- Настроить наблюдаемость.

Ниже представлен типовой пример манифеста Postgres `app-postgres`, который можно применить сразу, а затем настроить под свои задачи, последовательно проходя каждый из этих шагов.

```

apiVersion: managed-services.deckhouse.io/v1alpha1
kind: Postgres
metadata:
  name: app-postgres
  namespace: my-postgres
spec:
  postgresClassName: default

  configuration:
    maxConnections: 120

  instance:
    cpu:
      cores: 1
      coreFraction: 50
    memory:
      size: 1Gi
    persistentVolumeClaim:
      size: 10Gi
      storageClassName: replicated

  type: Cluster
  cluster:
    topology: Ignored
    replication: Consistency

  users:
    - name: app-rw
      role: rw
      storeCredsToSecret: app-postgres-rw

  databases:
    - name: app

  tls:
    mode: K8s

  observability: Enabled

```

Сохраните манифест в `postgres.yaml` и примените его:

```
d8 k apply -f postgres.yaml
```

Проверьте состояние созданного Postgres:

```
d8 k get postgres app-postgres -n my-postgres -O wide
```

После завершения развёртывания основные условия (`status.conditions`) должны перейти в `True` — что означает каждое условие, см. в разделе «Проверка состояния».

Пример вывода:

NAME	AVAILABLE	CONFIGURATIONVALID	LASTVALIDCONFIGURATIONAPPLIED
SCALED	TOLASTVALIDCONFIGURATION	DATABASESSYNCD	USERSSYNCD
app-postgres	True	True	True
True	True		True

Чтобы изменить параметр `app-postgres`, отредактируйте соответствующий фрагмент `postgres.yaml` и примените файл повторно.

Выбор PostgresClass

Параметр `spec.postgresClassName` определяет PostgresClass, который задаёт доступные

параметры и ограничения для Postgres. Посмотреть доступные в кластере PostgresClass можно командой:

```
d8 k get postgresclass
```

Пример вывода:

```
NAME      AGE
default   13d
```

В данном примере используется PostgresClass `default` со стандартными ограничениями. Если выбран другой PostgresClass, его ограничения можно посмотреть в конфигурации:

```
d8 k get postgresclass <CLASS_NAME> -o yaml
```

Где `<CLASS_NAME>` — имя выбранного PostgresClass.



При выборе PostgresClass учитывайте допустимые значения CPU, памяти и `coreFraction`, доступные топологии и параметры PostgreSQL, разрешённые для переопределения. Если конфигурация Postgres не соответствует ограничениям выбранного класса, API отклонит её при применении.

Настройки и ограничения PostgresClass описаны в разделе «Ограничение ресурсов CPU и памяти», разделе «Управление отказоустойчивостью через зоны доступности» и разделе «Автоматическая проверка настроек PostgreSQL».

Ограничения размещения

PostgresClass также может определять правила размещения экземпляров PostgreSQL с помощью `nodeSelector`, `nodeAffinity` и `tolerations`. Эти правила применяются автоматически при выборе класса и не указываются в объекте Postgres.

Настройка ресурсов

Для настройки ресурсов каждого экземпляра PostgreSQL — количества CPU, доли гарантированного CPU и объёма памяти — используется параметр `spec.instance`.

В примере за ресурсы и хранилище отвечает этот фрагмент:

```
spec:
  instance:
    cpu:
      cores: 1
      coreFraction: 50
    memory:
      size: 1Gi
```

В примере экземпляру выделяется одно ядро CPU и 1Gi памяти. Параметр `coreFraction` определяет отношение CPU request к CPU limit. Для `cores: 1` и `coreFraction: 50` модуль сформировал:

Пример вывода:

```
limits.cpu:    1
requests.cpu:  500m
```

Подробнее — в разделе «Ограничение ресурсов CPU и памяти».

Изменение ресурсов существующего Postgres

Ресурсы Postgres можно изменять повторным применением манифеста, если новые значения разрешены выбранным PostgresClass. Сначала узнайте текущие значения ресурсов командой:

```
d8 k get pods -n postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O custom-
columns='NAME:.metadata.name,CPU_REQUEST:.spec.containers[0].resources.requests.cpu,M
EMORY_REQUEST:.spec.containers[0].resources.requests.memory'
```

Пример вывода:

NAME	CPU_REQUEST	MEMORY_REQUEST
d8ms-pg-app-postgres-1	500m	1Gi
d8ms-pg-app-postgres-2	500m	1Gi

Можно сразу применить изменения как для памяти, так и для CPU, но для наглядности сначала увеличьте память с `1Gi` до `2Gi`:

```
spec:
  instance:
    memory:
      size: 2Gi
```

Примените изменённый манифест:

```
d8 k apply -f postgres.yaml
```

После завершения обновления CPU request останется `500m`, а memory request экземпляров изменится на `2Gi`.

Пример вывода:

NAME	CPU_REQUEST	MEMORY_REQUEST
d8ms-pg-app-postgres-1	500m	2Gi
d8ms-pg-app-postgres-2	500m	2Gi

Затем измените `coreFraction` с `50` на `100`, оставив `cores: 1`:

```
spec:
  instance:
    cpu:
      cores: 1
      coreFraction: 100
```

Повторно примените манифест:

```
d8 k apply -f postgres.yaml
```

После завершения обновления CPU request экземпляров изменится с `500m` на `1`, а memory request останется `2Gi`.

Пример вывода:

NAME	CPU_REQUEST	MEMORY_REQUEST
d8ms-pg-app-postgres-1	1	2Gi
d8ms-pg-app-postgres-2	1	2Gi

Таким образом, у работающего Postgres можно изменять память и `coreFraction` в пределах, разрешённых выбранным PostgresClass.

Проверка ограничения CPU и памяти через PostgresClass

Значения CPU и памяти должны соответствовать ограничениям выбранного PostgresClass. Если указанные ресурсы не соответствуют допустимым значениям или их сочетаниям, API отклонит конфигурацию.

PostgresClass `default` не подходит для наглядной проверки ограничений. Поэтому в этом примере используется отдельный PostgresClass `check`, который разрешает для `1–2` CPU память от `512Mi` до `2Gi` с шагом `512Mi`.

При `cores: 1` и `coreFraction: 50` значение памяти `700Mi` не соответствует установленному шагу, поэтому манифест будет отклонён:

```
spec:
  postgresClassName: check
  instance:
    cpu:
      cores: 1
      coreFraction: 50
    memory:
      size: 700Mi
```

Примените манифест:

```
d8 k apply -f postgres.yaml
```

API отклонит запрос. Пример вывода:

```
spec.instance.memory.size: Invalid value: 734003200: memory setting does not fit
Step 536870912 of the selected PostgresClass
```

Выбор режима работы

От выбора режима работы зависит состав экземпляров PostgreSQL: `Cluster` создаёт основной экземпляр и реплики, состав которых зависит от выбранного режима репликации. А `Standalone` — один экземпляр PostgreSQL без реплик.

Режим Cluster

Для работы с основным экземпляром и репликами используется режим `Cluster`, который задаётся параметром `spec.type`.

```
spec:
  type: Cluster
  cluster:
    topology: Ignored
    replication: Consistency
```

Режим репликации и его параметры настраиваются отдельно. Доступные режимы и примеры их использования описаны в разделе «Настройка репликации».

Режим Standalone

Режим `Standalone` используется для запуска PostgreSQL с одним экземпляром без репликации. В отличие от режима `Cluster`, для него не используются параметры топологии и репликации.

Чтобы использовать этот режим, укажите:

```
spec:
  type: Standalone
```

После создания Postgres будет запущен один экземпляр PostgreSQL. Проверьте созданные экземпляры PostgreSQL:

```
d8 k get pods -n postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

Пример вывода:

NAME	STATUS	NODE
d8ms-pg-app-postgres-1	Running	worker-1

Проверьте Service, созданные для подключения к PostgreSQL:

```
d8 k get svc -n my-postgres | grep app-postgres
```

Пример вывода:

d8ms-pg-app-postgres-r	ClusterIP	10.223.234.52	<none>	5432/TCP
d8ms-pg-app-postgres-ro	ClusterIP	10.223.70.248	<none>	5432/TCP
d8ms-pg-app-postgres-rw	ClusterIP	10.223.120.250	<none>	5432/TCP

Проверьте, на какие экземпляры направлены Service, через эндпоинты:

```
d8 k get endpoints -n my-postgres | grep app-postgres
```

Пример вывода:

d8ms-pg-app-postgres-r	10.112.2.31:5432	42h
d8ms-pg-app-postgres-ro	<none>	42h
d8ms-pg-app-postgres-rw	10.112.2.31:5432	42h

Сервисы с суффиксами `-r` и `-rw` направляют подключения на единственный экземпляр PostgreSQL. Сервис с суффиксом `-ro` также создаётся, но не имеет эндпоинта, поскольку в режиме `Standalone` отсутствуют реплики.

Настройка топологии и режима репликации

В режиме `Cluster` пользователь может управлять отказоустойчивостью экземпляров PostgreSQL, задавая их размещение по узлам и зонам доступности в параметре `spec.cluster.topology`.

Настройка топологии

Поддерживаются следующие значения:

- `Ignored` — размещение выполняется по стандартным правилам планирования Kubernetes с разнесением экземпляров по разным узлам;
- `Zonal` — экземпляры размещаются в пределах одной из разрешённых зон;
- `TransZonal` — экземпляры размещаются в разных зонах доступности.

Доступные значения топологии и зоны определяются выбранным `PostgresClass`. Для `Zonal` и `TransZonal` инфраструктура кластера должна предоставлять соответствующие зоны доступности. Подробнее — в разделе «Управление отказоустойчивостью через зоны доступности».

Размещение без выбора зоны

При `topology: Ignored` размещением экземпляров управляет планировщик Kubernetes. Режим обеспечивает разнесение экземпляров по разным узлам без дополнительных настроек со стороны пользователя. В основном примере используется этот режим:

```
spec:
  cluster:
    topology: Ignored
```

Проверьте размещение экземпляров командой:

```
d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

Экземпляры должны находиться на разных узлах.

Пример вывода:

NAME	STATUS	NODE
d8ms-pg-app-postgres-1	Running	worker-1
d8ms-pg-app-postgres-2	Running	worker-2

Размещение в одной зоне

При `topology: Zonal` для размещения Postgres выбирается одна из зон, разрешённых выбранным PostgresClass. Все экземпляры кластера размещаются в пределах этой зоны.

```
spec:
  cluster:
    topology: Zonal
```

Для использования `Zonal` узлы должны иметь лейбл `topology.kubernetes.io/zone` со значением соответствующей зоны. Зона должна быть разрешена выбранным PostgresClass.

Например, если два доступных узла относятся к зоне `default`:

Пример вывода:

NAME	ZONE
worker-1	default
worker-2	default

Экземпляры могут быть размещены следующим образом.

Пример вывода:

NAME	STATUS	NODE
d8ms-pg-app-postgres-1	Running	worker-1
d8ms-pg-app-postgres-2	Running	worker-2

В этом примере оба экземпляра размещены в зоне `default`.

Настройка репликации

В режиме `Cluster` репликация обеспечивает передачу данных с основного экземпляра PostgreSQL на реплики. Режим репликации задаётся в `spec.cluster.replication`.

Поддерживаются следующие режимы:

- `Availability` — основной экземпляр и одна асинхронная реплика;
- `Consistency` — основной экземпляр и одна синхронная реплика;
- `ConsistencyAndAvailability` — основной экземпляр, одна синхронная и одна асинхронная реплика.

Проверка режима репликации

Состояние репликации проверяется через представление `pg_stat_replication` на основном экземпляре. Эту процедуру используйте для любого режима `Cluster`, в том числе после смены `spec.cluster.replication`.

Сначала определите текущий основной экземпляр:

```
PRIMARY="$(d8 k get clusters.cnpg.internal.managed.deckhouse.io d8ms-pg-app-postgres \
  -n my-postgres \
  -o jsonpath='{.status.targetPrimary}')
```

Затем выполните запрос:


```
d8 k exec -n my-postgres "$PRIMARY" -- \
  psql -U postgres -d postgres -c \
  "SELECT application_name, state, sync_state FROM pg_stat_replication;"
```

Ожидаемые значения `sync_state` зависят от режима:

Режим	Число реплик	Ожидаемый sync_state
Availability	1	async
Consistency	1	quorum
ConsistencyAndAvailability	2	quorum и async

Значение `state: streaming` означает, что реплика получает изменения от основного экземпляра. Во время смены режима роли экземпляров могут меняться, поэтому каждый раз определяйте основной экземпляр заново через `status.targetPrimary`, а не по номеру Pod.

Режим Availability

Режим `Availability` создаёт основной экземпляр PostgreSQL и одну асинхронную реплику.

Чтобы использовать этот режим, укажите:

```
спес:
  type: Cluster
  cluster:
    topology: Zonal
    replication: Availability
```

После создания Postgres запускаются два экземпляра PostgreSQL:

```
d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

Пример вывода:

```
NAME                                READY   STATUS
d8ms-pg-app-postgres-1             1/1     Running
d8ms-pg-app-postgres-2             1/1     Running
```

Проверьте режим репликации, как описано в разделе «Проверка режима репликации». Для `Availability` ожидается одна реплика со `sync_state = async`:

```
application_name | state | sync_state
-----+-----+-----
d8ms-pg-app-postgres-2 | streaming | async
(1 row)
```

Распределение Service между основным экземпляром и репликой можно проверить через `EndpointSlice`:

```
d8 k get endpointslice -n my-postgres | grep app-postgres
```

Пример вывода:

```
d8ms-pg-app-postgres-r-v8kcv      IPv4    5432    10.112.2.249,10.112.2.155
d8ms-pg-app-postgres-ro-696bp     IPv4    5432    10.112.2.155
d8ms-pg-app-postgres-rw-8nx8s     IPv4    5432    10.112.2.249
```

Service `-rw` направляет подключения на основной экземпляр, `-ro` — на реплику, а `-r` — на оба экземпляра.

Режим Consistency

Режим `Consistency`, используемый в основном примере, создаёт основной экземпляр PostgreSQL и одну синхронную реплику:

```
спес:
  type: Cluster
  cluster:
    topology: Ignored
    replication: Consistency
```

После создания Postgres запускаются два экземпляра PostgreSQL:

```
d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

Пример вывода:

NAME	STATUS	NODE
d8ms-pg-app-postgres-1	Running	worker-1
d8ms-pg-app-postgres-2	Running	worker-2

Проверьте режим репликации, как описано в разделе «Проверка режима репликации». Для `Consistency` ожидается одна реплика со `sync_state = quorum`:

application_name	state	sync_state
d8ms-pg-app-postgres-2	streaming	quorum

(1 row)

Дополнительно работу синхронной репликации можно проверить по фактической передаче данных. Определите основной экземпляр так же, как в разделе «Проверка режима репликации»:

```
PRIMARY="$(d8 k get clusters.cnpg.internal.managed.deckhouse.io d8ms-pg-app-postgres \
  -n my-postgres \
  -o jsonpath='{.status.targetPrimary}')"
```

Создайте на основном экземпляре контрольную таблицу и добавьте запись:

```
d8 k exec -n my-postgres "$PRIMARY" -- \
  psql -U postgres -d postgres -c "
    CREATE TABLE consistency_check (
      id integer PRIMARY KEY,
      value text
    );
    INSERT INTO consistency_check VALUES (1, 'replicated');
  "
```

Определите реплику:

```
REPLICA="$(d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{end}' | \
  grep -v "^${PRIMARY}$" | head -n1)"
```

Проверьте наличие записи непосредственно на реплике:

```
d8 k exec -n my-postgres "$REPLICA" -- \
  psql -U postgres -d postgres -c \
  "SELECT pg_is_in_recovery(), * FROM consistency_check;"
```

Пример вывода:

```
pg_is_in_recovery | id | value
-----+-----+-----
t                | 1 | replicated
(1 row)
```

Значение `pg_is_in_recovery() = t` показывает, что запрос выполнен на реплике. Наличие строки `replicated` подтверждает передачу данных с основного экземпляра на синхронную реплику.

Режим ConsistencyAndAvailability

Режим `ConsistencyAndAvailability` создаёт основной экземпляр PostgreSQL, одну синхронную и одну асинхронную реплику.

Чтобы использовать этот режим, укажите:

```
spec:
  type: Cluster
  cluster:
    topology: Zonal
    replication: ConsistencyAndAvailability
```

После создания Postgres запускаются три экземпляра PostgreSQL:

```
d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

Пример вывода:

NAME	READY	STATUS
d8ms-pg-app-postgres-1	1/1	Running
d8ms-pg-app-postgres-2	1/1	Running
d8ms-pg-app-postgres-3	1/1	Running

Проверьте режим репликации, как описано в разделе «Проверка режима репликации». Для `ConsistencyAndAvailability` ожидаются две реплики — со `sync_state = quorum` и `sync_state = async`:

application_name	state	sync_state
d8ms-pg-app-postgres-2	streaming	quorum
d8ms-pg-app-postgres-3	streaming	async

(2 rows)

Изменение режима репликации существующего кластера

Режим репликации можно изменить у уже существующего Postgres в режиме `Cluster`. Для этого измените `spec.cluster.replication` в манифесте `app-postgres` и примените его повторно.

Например, чтобы перейти с `Availability` на `Consistency`, укажите:

```
spec:
  cluster:
    replication: Consistency
```

Примените изменения:

```
d8 k apply -f postgres.yaml
```

Во время обновления `ScaledToLastValidConfiguration` может временно перейти в `False`. После завершения обновления условия объекта (`status.conditions`) должны вернуться в `True`.

Проверьте новый режим, как описано в разделе «Проверка режима репликации». После перехода на `Consistency` реплика должна работать в синхронном режиме:

```
d8ms-pg-app-postgres-1 | streaming | quorum
```

При обратном переходе на `Availability` та же проверка должна показывать асинхронную репликацию:

```
d8ms-pg-app-postgres-2 | streaming | async
```

При переходе на `ConsistencyAndAvailability` число экземпляров увеличивается с двух до трёх. Проверьте запущенные экземпляры:

```
d8 k get pods -n my-postgres \
  -l managed-services.deckhouse.io/managed-service-name=app-postgres \
  -O wide
```

После завершения обновления проверка `pg_stat_replication` должна показывать синхронную и асинхронную реплики:

application_name	state	sync_state
d8ms-pg-app-postgres-3	streaming	async
d8ms-pg-app-postgres-2	streaming	quorum

(2 rows)

При обратном переходе с `ConsistencyAndAvailability` на `Consistency` число экземпляров уменьшается с трёх до двух, а оставшаяся реплика работает в режиме `streaming | quorum`.

Создание логической базы данных и пользователя

В основном примере создаются пользователь `app-rw` и логическая база данных `app`:

```
spec:
  users:
    - name: app-rw
      role: rw
      storeCredsToSecret: app-postgres-rw

  databases:
    - name: app
```

После применения манифеста дождитесь синхронизации пользователей и баз данных. Состояния `USERSSYNCED` и `DATABASESSYNCED` должны иметь значение `True`:

```
d8 k get postgres app-postgres -n my-postgres -O wide
```

Пользователь PostgreSQL

Учётные данные пользователя сохраняются в Secret, указанном в `storeCredsToSecret`.

Проверьте созданный Secret:

```
d8 k get secret app-postgres-rw -n my-postgres
```

Пример вывода:

NAME	TYPE	DATA
app-postgres-rw	kubernetes.io/basic-auth	4

Secret содержит параметры, необходимые для подключения:

```
app-dsn
host
password
username
```

Получите параметры подключения следующим образом:

```
echo "host: $(d8 k get secret app-postgres-rw -n my-postgres -o jsonpath='{.data.host}' |
base64 --decode)"
echo "username: $(d8 k get secret app-postgres-rw -n my-postgres -o
jsonpath='{.data.username}' | base64 --decode)"
echo "password: $(d8 k get secret app-postgres-rw -n my-postgres -o
jsonpath='{.data.password}' | base64 --decode)"
echo "app-dsn: $(d8 k get secret app-postgres-rw -n my-postgres -o jsonpath='{.data.app-dsn}'
| base64 --decode)"
```

Пример вывода:

```
host: d8ms-pg-app-postgres-rw
username: app-rw
password: <PASSWORD>
app-dsn: postgresql://app-rw:<PASSWORD>@d8ms-pg-app-postgres-rw:5432/app
```

Где `<PASSWORD>` — пароль пользователя из Secret.

Значения из Secret можно использовать для настройки подключения приложения или PostgreSQL-клиента.

i Для подключения приложений используйте Secret, имя которого указано в `storeCredsToSecret`. Внутренние Secret с именами вида `d8ms-pg-...` для этого использовать не следует.

Декларативное управление пользователями

Список пользователей в `spec.users` описывает требуемое состояние PostgreSQL. При изменении списка модуль синхронизирует роли пользователей и связанные с ними Secret.

Например, удалите пользователя `app-rw` из манифеста:

```
spec:
  users: []
```

Примените изменённый манифест:

```
d8 k apply -f postgres.yaml
```

После завершения синхронизации условие `USERSSYNCD` должно вернуться в `True`:

```
d8 k get postgres app-postgres -n my-postgres -O wide
```

Проверьте отсутствие роли непосредственно в PostgreSQL:

```
PRIMARY="$(d8 k get clusters.cnpg.internal.managed.deckhouse.io d8ms-pg-app-postgres \
-n my-postgres \
-o jsonpath='{.status.targetPrimary}')"

d8 k exec -n my-postgres "$PRIMARY" -- \
psql -U postgres -d postgres -Atc \
"SELECT rolname FROM pg_roles WHERE rolname = 'app-rw';"
```

Команда не должна вернуть имя роли.

При выполнении `d8 k exec` может отображаться служебное сообщение о выборе контейнера:

```
Defaulted container "postgres" out of: postgres, bootstrap-controller (init)
```

Логическая база данных `app`, оставшаяся в `spec.databases`, при удалении пользователя не удаляется. Проверьте её наличие командой:

```
d8 k exec -n my-postgres "$PRIMARY" -- \
psql -U postgres -d postgres -Atc \
"SELECT datname FROM pg_database WHERE datname = 'app';"
```

Ожидаемый вывод:

```
app
```



Удаление пользователя из `spec.users` приводит к удалению соответствующей роли PostgreSQL. Перед удалением убедитесь, что пользователь больше не используется приложениями.

Чтобы создать пользователя заново, добавьте его обратно в `spec.users`:

```
spec:
  users:
    - name: app-rw
      role: rw
      storeCredsToSecret: app-postgres-rw
```

После повторного применения манифеста модуль создаст роль PostgreSQL и Secret заново.

Логическая база данных

Логические базы данных, которые должен создать и поддерживать модуль, задаются в `spec.databases`:

```
spec:
  databases:
    - name: app
```

После создания базы условие `DATABASESYNCED` должно иметь значение `True`.



Удаление базы данных из `spec.databases` приводит к удалению соответствующей логической базы данных PostgreSQL вместе с её данными.

Подключение к PostgreSQL

После создания Postgres модуль создаёт Service `-r`, `-ro` и `-rw`, которые используются для подключения к экземплярам PostgreSQL в зависимости от их роли:

- `-rw` — основной экземпляр;
- `-ro` — реплики;
- `-r` — все доступные экземпляры.

Для наглядности в примере `app-postgres` создаются следующие Service:

NAME	TYPE	PORT(S)
<code>d8ms-pg-app-postgres-r</code>	ClusterIP	5432/TCP
<code>d8ms-pg-app-postgres-ro</code>	ClusterIP	5432/TCP
<code>d8ms-pg-app-postgres-rw</code>	ClusterIP	5432/TCP

По умолчанию эти Service имеют тип `ClusterIP` и доступны внутри кластера. Учётные данные и параметры подключения пользователя сохраняются в Secret, указанном в `storeCredsToSecret`.

Подключаться к PostgreSQL можно как из приложений внутри кластера, так и из внешней сети. Для внешнего подключения соответствующий Service необходимо дополнительно опубликовать.

Подключение из кластера

Для подключения из кластера используйте соответствующий Service и учётные данные из Secret пользователя. В основном примере приложение с правами на запись подключается к Service `d8ms-pg-app-postgres-rw` от имени пользователя `app-rw` к базе данных `app`.

Для проверки подключения не требуется устанавливать `psql` на control-plane-узел. Для этого можно использовать временный клиентский Pod:

```
d8 k run postgres-client \
  -n my-postgres \
  --rm -it \
  --restart=Never \
  --image=postgres:17 \
  --env="PGPASSWORD=$(d8 k get secret app-postgres-rw -n my-postgres -o
jsonpath='{.data.password}' | base64 --decode)" \
  -- \
  psql \
  -h d8ms-pg-app-postgres-rw \
  -U app-rw \
  -d app \
  -c 'SELECT current_database(), session_user, current_user;'
```

Пример вывода:

```
current_database | session_user | current_user
-----+-----+-----
app              | app-rw       | rw
(1 row)
```

`session_user` показывает пользователя, от имени которого выполнено подключение (`app-rw`), а `current_user` — действующую роль прав (`rw`).

Внешнее подключение к PostgreSQL

Для работы с PostgreSQL из внешней сети можно использовать графические клиенты и другие приложения, поддерживающие подключение к PostgreSQL. Для этого Service, через который публикуется PostgreSQL, должен быть доступен извне кластера, а клиенту необходимо указать адрес сервера, порт, базу данных и учётные данные пользователя.



В этом разделе внешнее подключение рассматривается на примере DBeaver. Аналогичным образом можно использовать другие PostgreSQL-клиенты и приложения.

В примере подключение выполняется к созданной ранее базе данных `app` от имени пользователя `app-rw` в Postgres `app-postgres`.

Публикация PostgreSQL для внешнего доступа

Способ публикации зависит от сетевой инфраструктуры кластера. В этом примере внешний балансировщик нагрузки принимает подключения к `<EXTERNAL_IP>:5432` и перенаправляет их на `NodePort 30001` узла кластера. Отдельный Service направляет этот трафик на основной экземпляр PostgreSQL.

Не изменяйте созданный модулем Service `d8ms-pg-app-postgres-rw`. Создайте отдельный Service для внешнего доступа:

```
apiVersion: v1
kind: Service
metadata:
  name: app-postgres-external
  namespace: my-postgres
spec:
  type: NodePort
  selector:
    cnpg.internal.managed.deckhouse.io/cluster: d8ms-pg-app-postgres
    cnpg.internal.managed.deckhouse.io/instanceRole: primary
  ports:
    - name: postgres
      protocol: TCP
      port: 5432
      targetPort: 5432
      nodePort: 30001
```

Примените манифест:

```
d8 k apply -f app-postgres-external.yaml
```

Проверьте созданный Service:

```
d8 k get svc app-postgres-external -n my-postgres -O wide
```

Пример вывода:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
app-postgres-external	NodePort	10.223.111.45	<none>	5432:30001/TCP
cnpg.internal.managed.deckhouse.io/cluster=d8ms-pg-app-postgres, cnpg.internal.managed.deckhouse.io/instanceRole=primary				

На внешнем балансировщике нагрузки настройте приём TCP-соединений на порту `5432` и перенаправление на `NodePort 30001` узла кластера. В текущем примере создана такая схема:

```
<EXTERNAL_IP>:5432
|
внешний балансировщик нагрузки
|
<NODE_IP>:30001
|
NodePort
|
primary PostgreSQL :5432
```



При публикации PostgreSQL во внешней сети убедитесь, что доступ к порту базы данных ограничен только доверенными источниками. Для этого могут использоваться файрвол, списки разрешённых IP-адресов, VPN и другие средства сетевой инфраструктуры. Не рекомендуется оставлять PostgreSQL доступным из Интернета без ограничений.

Перед проверкой внешнего подключения можно убедиться, что созданный `NodePort` направляет трафик на основной экземпляр PostgreSQL. Для этого получите пароль пользователя:


```
PGPASSWORD="$(d8 k get secret app-postgres-rw -n my-postgres \
-o jsonpath='{.data.password}' | base64 --decode)"
```

Запустите временный клиентский Pod и подключитесь через IP узла и `NodePort`:

```
d8 k run nodeport-test \
-n my-postgres \
--rm -i \
--restart=Never \
--image=postgres:17 \
--env="PGPASSWORD=$PGPASSWORD" \
-- \
psql \
-h <NODE_IP> \
-p 30001 \
-U app-rw \
-d app \
-c "SELECT current_database(), pg_is_in_recovery(), inet_server_addr();"
```

Пример успешного результата:

```
current_database | pg_is_in_recovery | inet_server_addr
-----+-----+-----
app              | f                  | <POD_IP>
(1 row)
```

Значение `pg_is_in_recovery = f` подтверждает, что соединение направлено на основной экземпляр PostgreSQL.

Подключение по IP-адресу

Подключение непосредственно по IP-адресу технически возможно и может использоваться, например, для проверки внешней доступности PostgreSQL. Для постоянного подключения рекомендуется использовать DNS-имя и TLS с проверкой сертификата сервера, как описано далее.

Получите пароль пользователя:

```
d8 k get secret app-postgres-rw -n my-postgres \
-o jsonpath='{.data.password}' | base64 --decode; echo
```

Создайте в DBeaver подключение PostgreSQL и укажите:

```
Host:      <EXTERNAL_IP>
Port:      5432
Database:  app
Username:  app-rw
Password:  <PASSWORD>
```

Где `<PASSWORD>` — пароль из Secret `app-postgres-rw`.

После подключения откройте SQL Editor и выполните:

```
SELECT
  current_database(),
  session_user,
  inet_server_addr(),
  inet_server_port(),
  pg_is_in_recovery();
```

На проверенном стенде запрос вернул:

```

current_database | session_user | inet_server_addr | inet_server_port |
pg_is_in_recovery
-----+-----+-----+-----+-----
app              | app-rw          | <POD_IP>         |              5432 | f

```

Значение `pg_is_in_recovery = f` подтверждает подключение к основному экземпляру PostgreSQL.

Подключение с проверкой TLS

Для постоянного внешнего подключения рекомендуется использовать TLS с проверкой сертификата сервера.

В примере для `app-postgres` используется режим `K8s`, поэтому TLS-сертификаты PostgreSQL выпускаются автоматически. Серверный сертификат подписан `cluster-selfsigned-ca`.

Сохраните автоматически созданный серверный сертификат в файл, чтобы определить DNS-имя из SAN:

```

d8 k get secret d8ms-pg-app-postgres-server-cert \
  -n my-postgres \
  -o jsonpath='{.data.tls\.crt}' | \
  base64 --decode > /tmp/app-postgres-server.crt

```

Посмотрите сведения о сертификате и его Subject Alternative Name (SAN):

```

openssl x509 \
  -in /tmp/app-postgres-server.crt \
  -noout \
  -subject -issuer -dates -ext subjectAltName

```

Для `app-postgres` сертификат содержит DNS-имя Service `-rw`:

```

d8ms-pg-app-postgres-postgres-rw.<EXTERNAL_IP>.sslip.io

```

При использовании режима `verify-full` клиент проверяет соответствие имени сервера сертификату, поэтому для подключения используйте DNS-имя из SAN.

Получите CA-сертификат:

```

d8 k get secret selfsigned-ca-key-pair \
  -n d8-cert-manager \
  -o jsonpath='{.data.tls\.crt}' | \
  base64 --decode > /tmp/app-postgres-ca.crt

```

Проверьте цепочку доверия:

```

openssl verify \
  -CAfile /tmp/app-postgres-ca.crt \
  /tmp/app-postgres-server.crt

```

Пример успешного результата:

```

/tmp/app-postgres-server.crt: OK

```

Перенесите CA-сертификат на компьютер, с которого выполняется подключение. Например, если к узлу кластера доступен SSH, скопируйте сертификат с помощью `scp`:

```

scp user@<NODE_IP>:/tmp/app-postgres-ca.crt ~/app-postgres-ca.crt

```

В DBeaver укажите параметры подключения:

```
Host:      d8ms-pg-app-postgres-postgres-rw.<EXTERNAL_IP>.sslip.io
Port:      5432
Database:  app
Username:  app-rw
Password:  <PASSWORD>
```

Где `<PASSWORD>` — пароль из Secret `app-postgres-rw`.

В настройках SSL укажите CA-сертификат и режим `verify-full`:

```
CA Certificate: <CA_CERT_PATH>
SSL mode:      verify-full
```

Где `<CA_CERT_PATH>` — путь к файлу `app-postgres-ca.crt`.

После подключения выполните:

```
SELECT
  current_database(),
  session_user,
  inet_server_addr(),
  inet_server_port(),
  pg_is_in_recovery();
```

Успешное выполнение запроса и значение `pg_is_in_recovery = f` подтверждают подключение к основному экземпляру PostgreSQL.

Если при использовании `verify-full` вместо DNS-имени из SAN указать IP-адрес `<EXTERNAL_IP>`, проверка имени сервера завершится ошибкой:

```
The hostname <EXTERNAL_IP> could not be verified by hostnameverifier PgjdbcHostnameVerifier.
```

Таким образом, для подключения с `verify-full` используйте DNS-имя, указанное в SAN серверного сертификата.

Настройка параметров PostgreSQL

Параметры PostgreSQL можно изменять через `spec.configuration`, если выбранный `PostgresClass` разрешает их переопределение.

Возможность изменения параметра определяется настройками `PostgresClass`:

- параметр должен быть разрешён для переопределения;
- значение параметра должно соответствовать установленным правилам проверки.

Если параметр запрещён для изменения или его значение не соответствует ограничениям, API отклонит запрос.

Изменение разрешённого параметра

В основном примере `app-postgres` использует `PostgresClass default`, который разрешает изменять параметр `maxConnections`.

Измените значение:

```
spec:
  configuration:
    maxConnections: 100
```

Примените изменения:

```
d8 k apply -f postgres.yaml
```

После завершения обновления проверьте применённое значение непосредственно в PostgreSQL:

```
PRIMARY="$(d8 k get clusters.cnpg.internal.managed.deckhouse.io d8ms-pg-app-postgres \
-n my-postgres \
-o jsonpath='{.status.targetPrimary}')"

d8 k exec -n my-postgres "$PRIMARY" -- \
psql -U postgres -d postgres -c \
"SHOW max_connections;"
```

Пример вывода:

```
max_connections
-----
100
```

Параметр был изменён, так как он разрешён для переопределения выбранным PostgresClass.

Ограничения на переопределение параметров

PostgresClass может ограничивать список параметров PostgreSQL, которые пользователь может изменять через `spec.configuration`.

Например, если PostgresClass разрешает переопределять только:

```
overridableConfiguration:
- maxConnections
- sharedBuffers
- walKeepSize
```

попытка изменить параметр, отсутствующий в этом списке, будет отклонена.

Примените, например:

```
список:
configuration:
workMem: 16Mi
```

```
d8 k apply -f postgres.yaml
```

API вернёт ошибку. Пример вывода:

```
Configuration field workmem restricted to override by administrator in selected
postgresClass
```

В этом случае конфигурация Postgres не изменится, так как параметр запрещён выбранным PostgresClass.

Проверка значений параметров

Помимо списка разрешённых параметров, PostgresClass может задавать правила проверки допустимых значений.

Например, если для `maxConnections` установлено ограничение:

```
configuration.maxConnections >= 100
```

следующее изменение будет отклонено:

```
список:
configuration:
maxConnections: 50
```

Примените манифест:

```
d8 k apply -f postgres.yaml
```

API вернёт ошибку. Пример вывода:

```
Rule: configuration.maxConnections >= 100
```

Существующий Postgres продолжит работать с последней успешно применённой конфигурацией.

Настройка TLS

Для управления TLS-сертификатами PostgreSQL используется параметр `spec.tls`. Поддерживаются режимы `CertManager`, `CustomCertificate` и `K8s`.

Для использования сертификатов, выпускаемых cert-manager, укажите режим `CertManager`:

```
spec:
  tls:
    mode: CertManager
    certManager:
      clusterIssuerName: postgres-ca
```

Соответствующий Issuer или ClusterIssuer должен быть предварительно подготовлен. Административные зависимости описаны в разделе «Зависимости для отдельных функций».

Чтобы использовать существующие сертификаты из Secret, выберите режим `CustomCertificate`:

```
spec:
  tls:
    mode: CustomCertificate
    customCertificate:
      serverCASecret: postgres-ca
      serverTLSSecret: postgres-tls
```

Режим K8s

В режиме `K8s` сертификаты для PostgreSQL выпускаются автоматически:

```
spec:
  tls:
    mode: K8s
```

После перехода объекта в готовое состояние модуль создаёт Secret с CA, сертификаты сервера и репликации.

Проверьте использование TLS на стороне PostgreSQL через представление `pg_stat_ssl`. Определите основной экземпляр так же, как в разделе «Проверка режима репликации», и выполните запрос:

```
PRIMARY="$(d8 k get clusters.cnpg.internal.managed.deckhouse.io d8ms-pg-app-postgres \
-n my-postgres \
-o jsonpath='{.status.targetPrimary}')"

d8 k exec -n my-postgres "$PRIMARY" -- \
psql -U postgres -d postgres -c "
SELECT
  a.pid,
  a.username,
  a.client_addr,
  a.client_port,
  s.ssl,
  s.version,
  s.cipher
FROM pg_stat_activity a
LEFT JOIN pg_stat_ssl s USING (pid)
WHERE a.username = 'app-rw';
"
```

Для TLS-соединения поле `ssl` имеет значение `t`, а в `version` и `cipher` отображаются используемые версия TLS и шифр.

Настройка клиентского подключения с проверкой сертификата сервера описана в разделе «Подключение с проверкой TLS».

Мониторинг и алерты

Для Postgres можно включить мониторинг с алертами, полностью отключить мониторинг или оставить мониторинг без алертов. Режим наблюдаемости задаётся параметром `spec.observability`.

В основном примере включены мониторинг и алерты:

```
spec:
  observability: Enabled
```

Чтобы полностью отключить мониторинг, используйте:

```
spec:
  observability: Disabled
```

Чтобы сохранить мониторинг, но отключить алерты, используйте:

```
spec:
  observability: EnabledWithoutAlerts
```

Проверьте применённый режим по лейблам Pod:

```
d8 k get pod -n my-postgres \
-l managed-services.deckhouse.io/managed-service-name=app-postgres \
-o json | \
jq '.items[].metadata.labels | with_entries(select(.key |
test("observability|prometheus")))'
```

Значение лейбла `observability.deckhouse.io/servicemonitoring` зависит от выбранного режима:

Enabled	→ enabled
Disabled	→ disabled
EnabledWithoutAlerts	→ no-alerts

При включённом мониторинге вывод также содержит лейбл:

```
"prometheus.deckhouse.io/custom-target": "managed-postgres"
```

Резервное копирование и восстановление

Для создания снимков используется объект `PostgresSnapshot`. `StorageClass`, в котором размещён `Postgres`, должен использовать CSI-драйвер с поддержкой `snapshots`, а в кластере должен быть доступен соответствующий `VolumeSnapshotClass`.

В основном примере используется `StorageClass replicated`, для которого провайдер в рассматриваемой конфигурации не поддерживает создание снимков. Поэтому для демонстрации используется отдельный `StorageClass snapshot-local` на `sds-local-volume` с LVM Thin.

Проверьте доступные классы снимков:

```
d8 k get volumesnapshotclass
```

Для `snapshot-local` доступен следующий класс снимков:

NAME	DRIVER	DELETIONPOLICY
sds-local-volume-snapshot-class	local.csi.storage.deckhouse.io	Delete

Создание снимка

Для проверки создайте отдельный `Postgres snapshot-pg` в `StorageClass snapshot-local`:

```
apiVersion: managed-services.deckhouse.io/v1alpha1
kind: Postgres
metadata:
  name: snapshot-pg
  namespace: my-postgres
spec:
  postgresClassName: default
  instance:
    cpu:
      cores: 1
      coreFraction: 50
    memory:
      size: 1Gi
    persistentVolumeClaim:
      size: 2Gi
      storageClassName: snapshot-local
  type: Standalone
  users:
    - name: snapshot-rw
      role: rw
      storeCredsToSecret: snapshot-pg-rw
  databases:
    - name: snapshotdb
```

Чтобы наглядно проверить восстановление данных на момент создания снимка, используйте контрольную таблицу: строку `BEFORE_SNAPSHOT` добавьте до создания снимка, а `AFTER_SNAPSHOT` — после.

Создайте контрольную таблицу и запишите первую строку:

```
PGPASSWORD="$(d8 k get secret snapshot-pg-rw -n my-postgres \
-o jsonpath='{.data.password}' | base64 --decode)"

d8 k run snapshot-client \
-n my-postgres \
--rm -i \
--restart=Never \
--image=postgres:17 \
--env="PGPASSWORD=$PGPASSWORD" \
-- \
psql \
-h d8ms-pg-snapshot-pg-rw \
-U snapshot-rw \
-d snapshotdb \
-c "
    CREATE TABLE snapshot_check (
        id integer PRIMARY KEY,
        value text NOT NULL
    );
    INSERT INTO snapshot_check VALUES (1, 'BEFORE_SNAPSHOT');
    SELECT * FROM snapshot_check;
"
```

Пример вывода:

```
id |      value
----+-----
  1 | BEFORE_SNAPSHOT
(1 row)
```

Создайте объект PostgresSnapshot:

```
apiVersion: managed-services.deckhouse.io/v1alpha1
kind: PostgresSnapshot
metadata:
  name: snapshot-pg-backup
  namespace: my-postgres
spec:
  postgresName: snapshot-pg
```

Примените манифест:

```
d8 k apply -f snapshot-pg-backup.yaml
```

Проверьте состояние снимка:

```
d8 k get postgressnapshot snapshot-pg-backup -n my-postgres \
-o jsonpath='{.status.phase}{"\n"}'
```

После успешного создания снимка команда возвращает:

```
completed
```

Проверьте созданный VolumeSnapshot:

```
d8 k get volumesnapshot -n my-postgres
```

Пример вывода:

NAME	READYTOUSE	SOURCEPVC	RESTORESIZE
SNAPSHOTCLASS			
d8ms-pg-snapshot-pg-backup	true	d8ms-pg-snapshot-pg-1	2Gi
local-volume-snapshot-class			sds-

`READYTOUSE=true` подтверждает готовность снимка к восстановлению.

После завершения создания снимка добавьте в исходную базу вторую контрольную строку:

```
PGPASSWORD="$(d8 k get secret snapshot-pg-rw -n my-postgres \
-o jsonpath='{.data.password}' | base64 --decode)"

d8 k run snapshot-client \
  -n my-postgres \
  --rm -i \
  --restart=Never \
  --image=postgres:17 \
  --env="PGPASSWORD=$PGPASSWORD" \
  -- \
  psql \
  -h d8ms-pg-snapshot-pg-rw \
  -U snapshot-rw \
  -d snapshotdb \
  -c "
    INSERT INTO snapshot_check VALUES (2, 'AFTER_SNAPSHOT');
    SELECT * FROM snapshot_check ORDER BY id;
  "
```

Пример вывода:

```
id |      value
----+-----
  1 | BEFORE_SNAPSHOT
  2 | AFTER_SNAPSHOT
(2 rows)
```

Восстановление из PostgresSnapshot

Для восстановления создайте новый объект Postgres и укажите созданный PostgresSnapshot в `spec.dataSource.objectRef`. Исходный Postgres удалять не требуется:

```
apiVersion: managed-services.deckhouse.io/v1alpha1
kind: Postgres
metadata:
  name: snapshot-pg-restored
  namespace: my-postgres
spec:
  dataSource:
    objectRef:
      kind: PostgresSnapshot
      name: snapshot-pg-backup
  type: Standalone
instance:
  cpu:
    cores: 1
    coreFraction: 50
  memory:
    size: 1Gi
  persistentVolumeClaim:
    size: 2Gi
    storageClassName: snapshot-local
```

Поля `type` и `instance` необходимо указывать явно — они не наследуются от исходного Postgres. После этого примените манифест:

```
d8 k apply -f snapshot-pg-restored.yaml
```

Дождитесь готовности восстановленного Postgres:

```
d8 k get postgres snapshot-pg-restored -n my-postgres -O wide -w
```

После запуска восстановленного PostgreSQL проверьте контрольную таблицу:

```
PGPASSWORD="$(d8 k get secret snapshot-pg-rw -n my-postgres \
-o jsonpath='{.data.password}' | base64 --decode)"

d8 k run snapshot-restore-check \
  -n my-postgres \
  --rm -i \
  --restart=Never \
  --image=postgres:17 \
  --env="PGPASSWORD=$PGPASSWORD" \
  -- \
  psql \
  -h d8ms-pg-snapshot-pg-restored-rw \
  -U snapshot-rw \
  -d snapshotdb \
  -c "SELECT * FROM snapshot_check ORDER BY id;"
```

Пример вывода:

```
id |      value
----+-----
  1 | BEFORE_SNAPSHOT
(1 row)
```

Наличие только `BEFORE_SNAPSHOT` подтверждает, что восстановлено состояние базы данных на момент создания снимка.

Проверка состояния

Текущее состояние Postgres отражается в `status.conditions` объекта.

Для краткой проверки используйте:

```
d8 k get postgres app-postgres -n my-postgres -O wide
```

Основные условия (`status.conditions`):

Условие	Что показывает
<code>ConfigurationValid</code>	Конфигурация прошла проверки связанного <code>PostgresClass</code>
<code>LastValidConfigurationApplied</code>	Последняя валидная конфигурация применена
<code>ScaledToLastValidConfiguration</code>	Экземпляры соответствуют последней валидной конфигурации
<code>Available</code>	Postgres доступен
<code>UsersSynced</code>	Пользователи синхронизированы
<code>DatabasesSynced</code>	Логические базы данных синхронизированы

Во время изменения ресурсов или параметров PostgreSQL часть условий может временно иметь значение `False`, в то время как `Available` остаётся `True`.

Для наблюдения за изменением состояния:

```
d8 k get postgres app-postgres -N my-postgres -O wide -w
```

Для просмотра подробностей:

```
d8 k get postgres app-postgres -N my-postgres -O yaml
```

Если Postgres не переходит в готовое состояние, диагностика — в разделе «Частые вопросы».

Дополнительные ресурсы

- Частые вопросы
- Кастомные ресурсы модуля `managed-postgres`

Мониторинг приложений и инфраструктуры

Данный раздел предназначен для пользователей проектов Deckhouse Kubernetes Platform (DKP).

В состав DKP включена система мониторинга, которая предоставляет пользователям удобные инструменты для наблюдения за состоянием инфраструктуры и приложений.

По умолчанию в DKP доступен готовый набор дашбордов и алертов, позволяющих отслеживать ключевые показатели состояния приложений. Доступ к ним осуществляется через раздел «Мониторинг» веб-интерфейса Deckhouse.

Кроме того, пользователи могут:

- собирать метрики со своих приложений;
- создавать собственные дашборды для отображения нужных показателей;
- настраивать собственные алерты и переопределять пороги их срабатывания.

Подробнее о расширенных возможностях мониторинга, включая управление дашбордами и метриками, можно прочитать в описании модуля `observability`.

Возможности по умолчанию

После установки DKP пользователям по умолчанию доступен базовый набор инструментов для наблюдения за кластером.

Дашборды

Дашборды содержат графики с данными по загрузке CPU, потреблению памяти, а также дисковой и сетевой активности в разрезе подов, узлов или пространств имён.

В разделе «Мониторинг» → «Дашборды» веб-интерфейса Deckhouse у пользователей есть доступ к следующим группам дашбордов:

- **Ingress Nginx** — метрики по работе Ingress-контроллера, включая информацию о состоянии виртуальных хостов, коды ответов и данные по задержке в обработке запросов.
- **Потребление ресурсов (Main)** — основные показатели кластера и приложений, включая данные о нагрузке ресурсов, состоянии подов, контроллеров и пространств имён.
- **Security** — метрики, связанные с безопасностью кластера.

Алерты

Алерты — это автоматические уведомления, информирующие о событиях, требующих внимания, например, о превышении пороговых значений метрик или проблемах с доступностью компонентов. Порог срабатывания большинства алертов можно переопределять при необходимости.

По умолчанию в кластере DKP включены уведомления о следующих видах событий:

- Истечение срока действия сертификатов, а также ошибки при их выпуске или продлении (модули `cert-manager`, `extended-monitoring`, `ingress-nginx`).
- Недоступность или ошибки при загрузке контейнерных образов, включая проблемы аутентификации, авторизации, некорректный формат имени образа, отсутствие образа в registry или недоступность самого registry (модуль `extended-monitoring`).
- Ошибки выполнения рабочих нагрузок, таких как CronJob, Deployment, DaemonSet и StatefulSet, включая невозможность создания подов, недоступность реплик и ошибки планирования (модуль `extended-monitoring`).

- Недоступность экспортеров метрик, из-за чего Prometheus не может получить данные (модуль `extended-monitoring`).
- Проблемы с дисковым пространством, включая нехватку места или inode на PVC (модуль `extended-monitoring`).
- Ошибки в работе Ingress-контроллера, включая высокий процент 5xx-ответов от бэкендов (модуль `extended-monitoring`).
- Проблемы с качеством работы сети (модуль `monitoring-ping`).

Настройки мониторинга

Пользователям доступны следующие настройки системы мониторинга DKP:

- **мониторинг пользовательских приложений** — можно настроить сбор метрик с приложения, следуя инструкции;
- **создание собственных дашбордов** — можно добавлять специализированные дашборды, используя ресурс `GrafanaDashboardDefinition`;
- **настройка собственных алертов** — можно задать новые правила уведомлений одним из двух способов:
 - **рекомендуемый способ** — с помощью ресурсов `ClusterObservabilityMetricsRulesGroup` и `ObservabilityMetricsRulesGroup` модуля `observability`;
 - **устаревший способ** — с помощью ресурса `CustomPrometheusRules` модуля `prometheus`. Пример этого способа доступен в документации модуля.

Дополнительные ресурсы

- Документация модуля `console`
- Документация модуля `observability`
- Документация модуля `prometheus`

Дашборды мониторинга

В этом разделе вы узнаете о работе с дашбордами для анализа состояния Deckhouse Kubernetes Platform (DKP) и запущенных в нем приложений.

Дашборды представляют собой наборы графиков и таблиц с данными о работе приложений. Они содержат информацию о загрузке CPU, потреблении памяти, дисковой и сетевой активности, а также о состоянии подов, контроллеров, узлов и неймспейсов.

Виды дашбордов

В DKP доступны предустановленные дашборды, а также пользовательские дашборды, которые можно создавать несколькими способами.

Вид дашбордов	Описание
Предустановленные	Готовые дашборды, поставляемые вместе с DKP. Предназначены для мониторинга состояния запущенных приложений.
Пользовательские, создаваемые с помощью модуля <code>observability</code>	Пользовательские дашборды, создаваемые с помощью ресурса <code>ObservabilityDashboard</code> на уровне неймспейсов и с возможностью разграничения прав доступа. Это рекомендуемый способ работы с дашбордами.
Пользовательские, создаваемые с помощью <code>GrafanaDashboardDefinition</code>	Пользовательские дашборды, создаваемые с помощью ресурса <code>GrafanaDashboardDefinition</code> на уровне кластера. Требуют расширенных прав доступа и не позволяют управлять разграничением прав доступа. Это устаревший способ работы с дашбордами, который перестанет поддерживаться в следующих версиях DKP.

Предустановленные дашборды

Пользователям DKP предоставляется доступ к базовому набору дашбордов для наблюдения за состоянием запущенных приложений. Дашборды доступны в веб-интерфейсе Deckhouse в

разделе «Мониторинг» → «Дашборды».



Предустановленные дашборды недоступны для редактирования.

Ingress Nginx

Дашборды для мониторинга работы Ingress-контроллера. Содержат метрики, отражающие состояние виртуальных хостов, статистику по HTTP-ответам, а также данные о задержках при обработке запросов.

Доступные дашборды:

- **Namespaces** — совокупные метрики Ingress-ресурсов по неймспейсам;
- **Namespace Detail** — детальная информация по Ingress-ресурсам в выбранном неймспейсе;
- **VHosts** — обзор состояния виртуальных хостов;
- **VHost Detail** — детальная информация по выбранному виртуальному хосту.

Потребление ресурсов (Main)

Набор дашбордов для анализа потребления ресурсов приложениями. Дашборды предназначены для оценки нагрузки, поиска проблем с ресурсами и анализа состояния рабочих нагрузок.

Доступные дашборды:

- **Namespaces** — сводная информация по всем неймспейсам;
- **Namespace** — основные показатели использования ресурсов в выбранном неймспейсе;
- **Namespace / Controller** — статистика использования ресурсов контроллерами в рамках выбранного неймспейса;
- **Namespace / Controller / Pod** — детальные метрики по отдельным подам.

Security

Дашборды, содержащие метрики, связанные с безопасностью компонентов кластера.

Доступные дашборды:

- **Admission policy engine** — метрики работы модуля `admission-policy-engine`, включая информацию о проверках и применении политик.

Пользовательские дашборды

Пользователи DKP могут создавать собственные дашборды несколькими способами, в зависимости от требований к управлению доступом и области видимости дашборда.

С помощью модуля observability

Модуль `observability` расширяет функциональность модуля `prometheus` и веб-интерфейса Deckhouse, предоставляя дополнительные возможности для гибкого управления визуализацией метрик и разграничения доступа к ним.

Модуль добавляет новые типы дашбордов, включая ресурсы, ограниченные неймспейсом. Это даёт пользователям возможность создавать и управлять собственными дашбордами без необходимости иметь права на объекты кластерного уровня. Кроме того, модуль упрощает редактирование дашбордов — настройка выполняется напрямую в веб-интерфейсе, без необходимости работы с ресурсами вручную.



Перед началом работы с этими ресурсами убедитесь, что модуль `observability` включён в кластере. При необходимости обратитесь к администратору DKP.

Для создания дашбордов предусмотрены следующие ресурсы:

- **ObservabilityDashboard** — дашборды в рамках неймспейса. Отображаются в веб-интерфейсе Deckhouse в разделе «Мониторинг» → «Проекты».

Пример:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ObservabilityDashboard
metadata:
  name: example-dashboard
  namespace: my-namespace
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

- ClusterObservabilityDashboard — дашборды для отображения компонентов кластера. Отображаются в веб-интерфейсе Deckhouse в разделе «Мониторинг» → «Система».

Пример:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityDashboard
metadata:
  name: example-dashboard
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

- ClusterObservabilityPropagatedDashboard — дашборды, расширяющие список дашбордов из двух предыдущих категорий. Такие дашборды автоматически добавляются в веб-интерфейс Deckhouse и отображаются в разделах «Мониторинг» → «Система» и «Мониторинг» → «Проекты». Они становятся доступны пользователям, обладающим правами на соответствующий неймспейс или системный раздел.

Пример:

```

apiVersion: observability.deckhouse.io/v1alpha1
kind: ClusterObservabilityPropagatedDashboard
metadata:
  name: example-dashboard
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example dashboard"
spec:
  definition: |
    {
      "title": "Example dashboard",
      ...
    }

```

Разграничение прав доступа

Доступ к дашбордам настраивается с помощью механизмов действующей ролевой модели (RBAC).

В зависимости от типа дашборда (системный или пользовательский) права назначаются на следующие ресурсы:

- `observabilitydashboards.observability.deckhouse.io` — дашборды в рамках неймспейсов;
- `clusterobservabilitydashboards.observability.deckhouse.io` — системные дашборды;
- `clusterobservabilitypropagateddashboards.observability.deckhouse.io` — дашборды, распространяемые на всех пользователей.

Для выполнения операций с дашбордами требуются следующие разрешения:

- чтение — `get`;
- создание и редактирование — `create`, `update`, `patch`, `delete`.

Доступ к метрикам в дашбордах также контролируется с помощью RBAC. В зависимости от выданных прав фильтрация метрик осуществляется автоматически.

Поддерживаются следующие сценарии доступа:

- Пользователи неймспейсов получают доступ только к метрикам своего неймспейса. Проверяется RBAC-доступ к ресурсу `metrics.observability.deckhouse.io`.
- Администраторы DKP получают доступ ко всем системным метрикам:
 - метрики Deckhouse (`d8-*`);
 - метрики Kubernetes (`kube-*`);
 - метрики без лейбла `namespace`.

Используется RBAC-доступ к ресурсу `clustermetrics.observability.deckhouse.io`.

- Метрики из пользовательских неймспейсов также могут быть доступны администраторам при наличии соответствующих прав на ресурс `metrics.observability.deckhouse.io`.

Пример настройки ресурсов `ClusterRole` и `RoleBinding` для доступа к метрикам и дашбордам на чтение и редактирование:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: observability-editor
rules:
  - apiGroups: ["observability.deckhouse.io"]
    resources: ["metrics", "observabilitydashboards"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["observability.deckhouse.io"]
    resources: ["observabilitydashboards"]
    verbs: ["create", "update", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-observability-editor
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: observability-editor
subjects:
  - kind: User
    name: user@example.com
    apiGroup: rbac.authorization.k8s.io
```

Конвертация дашбордов из `GrafanaDashboardDefinition`

Чтобы перенести дашборды, созданные с помощью устаревшего ресурса `GrafanaDashboardDefinition`, в один из форматов модуля `observability`, отредактируйте каждый манифест соответствующего дашборда вручную. Обратите внимание на важные отличия:

Формат <code>GrafanaDashboardDefinition</code>	Формат модуля <code>observability</code>
Папка в Grafana для отображения дашборда задается в поле <code>spec.folder</code> .	Папка задается с помощью аннотации <code>observability.deckhouse.io/category</code> .

Формат <code>GrafanaDashboardDefinition</code>	Формат модуля <code>observability</code>
Название дашборда задается в поле <code>title</code> JSON-манифеста.	Название задаётся с помощью аннотации <code>observability.deckhouse.io/title</code> . Если аннотация отсутствует, используется поле <code>title</code> из JSON-манифеста.

Пример конвертации:

- Старый формат:

```
apiVersion: deckhouse.io/v1
kind: GrafanaDashboardDefinition
metadata:
  name: example-dashboard
spec:
  folder: "Apps"
  json: '{
    "title": "Example Dashboard",
    ...
  }'
```

- Новый формат:

```
apiVersion: observability.deckhouse.io/v1alpha1
kind: ObservabilityDashboard
metadata:
  name: example-dashboard
  namespace: my-namespace
  annotations:
    metadata.deckhouse.io/category: "Apps"
    metadata.deckhouse.io/title: "Example Dashboard"
spec:
  definition: |
    {
      "title": "Example Dashboard",
      ...
    }
```

С помощью `GrafanaDashboardDefinition`

i Это устаревший способ, который не рекомендуется для новых дашбордов. Поддержка этого способа будет прекращена в следующих версиях ДКР.

Чтобы добавить дашборд напрямую в Grafana, используйте 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,
            ...
```

При использовании этого способа учитывайте следующие ограничения:

- Дашборды, добавленные через GrafanaDashboardDefinition, нельзя изменить через интерфейс Grafana.
- Алерты, настроенные в панели «Dashboard», не работают с шаблонами datasource — такой дашборд считается невалидным и не импортируется. Начиная с Grafana 9.0, функционал legacy alerting признан устаревшим и заменён на Grafana Alerting. В связи с этим не рекомендуется использовать legacy alerting (оповещения панели мониторинга) в дашбордах.
- Если после применения ресурса дашборд не появляется в Grafana, возможно, в JSON-файле дашборда содержится ошибка. Чтобы просмотреть логи компонента, отвечающего за применение дашбордов, используйте следующую команду:

```
d8 k logs -n d8-monitoring deployments/grafana-v10 dashboard-provisioner
```

Дополнительные ресурсы

- Документация модуля admission-policy-engine
- Документация модуля console
- Документация модуля observability
- Кастомные ресурсы модуля prometheus

Настройка мониторинга приложений

Deckhouse Kubernetes Platform (DKP) поддерживает четыре способа подключения приложения к системе мониторинга:

Способ подключения	Описание
Через лейблы и аннотации	Самый простой и быстрый способ, требующий лишь добавления метаданных к сервису или поду. Позволяет задать базовые параметры мониторинга.
С помощью PodMonitor или ServiceMonitor	Расширенный способ настройки мониторинга для случаев, когда требуется использование relabeling-правил Prometheus. Позволяет гибко управлять сбором метрик и обработкой лейблов, а также добавлять пользовательские лейблы в метрики. Данный подход подходит для сложных сценариев мониторинга, но требует более глубокого понимания принципов работы Prometheus и его механизма сбора метрик.

Способ подключения	Описание
С помощью ScrapeConfig	Способ настройки мониторинга, максимально приближенный к нативной структуре конфигурации Prometheus. Предоставляет полный контроль над scrape-настройками, включая relabeling, и позволяет собирать метрики как из Kubernetes, так и с targets, расположенных за пределами кластера.
Мониторинг доступности с помощью blackbox-exporter	Способ мониторинга доступности эндпоинтов с помощью проверок (probes). Подключается с помощью blackbox-exporter, который необходимо установить в кластере отдельно.

Настройка сбора метрик через лейблы и аннотации

i Здесь описан базовый сценарий подключения приложения. Для более гибкой настройки доступны дополнительные аннотации.

1. Убедитесь, что включен модуль `monitoring-custom`. При необходимости обратитесь к администратору DKP.
2. Убедитесь, что приложение, с которого будут собираться метрики, отдает их в формате Prometheus.
3. Установите лейбл `prometheus.deckhouse.io/custom-target` на сервис или под, которые необходимо подключить к мониторингу. Значение лейбла определит имя в списке target'ов Prometheus.

Пример:

```
labels:
  prometheus.deckhouse.io/custom-target: my-app
```

В качестве значения лейбла `prometheus.deckhouse.io/custom-target` рекомендуется использовать название приложения, которое позволяет его уникально идентифицировать в кластере.

Формат лейбла должен соответствовать требованиям Kubernetes: не более 63 символов, среди которых могут быть буквенно-цифровые символы ([a-z0-9A-Z]), а также дефисы (-), знаки подчеркивания (_), точки (.).

Если приложение ставится в кластер больше одного раза (staging, testing и т. д.) или даже ставится несколько раз в одно пространство имён, достаточно одного общего названия, так как у всех метрик в любом случае будут лейблы `namespace`, `pod` и, если доступ осуществляется через сервис, лейбл `service`. Это название, которое уникально идентифицирует приложение в кластере, а не его единичную инсталляцию.

4. Для порта, с которого нужно собирать метрики, укажите имя `http-metrics` и `https-metrics` для подключения по HTTP или HTTPS соответственно.

Если это невозможно (например, порт уже определен и назван другим именем), используйте следующие аннотации:

- `prometheus.deckhouse.io/port: номер_порта` — для указания порта;
- `prometheus.deckhouse.io/tls: "true"` — если сбор метрик будет проходить по HTTPS.

i При указании аннотации для сервиса в качестве значения порта необходимо использовать `targetPort` — порт, который открыт и слушается приложением, а не порт сервиса.

- 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.
```

5. При использовании service mesh Istio в режиме STRICT mTLS укажите для сбора метрик аннотацию `prometheus.deckhouse.io/istio-mtls: "true"` у сервиса или пода. Важно, что метрики приложения должны экспортироваться по протоколу HTTP без TLS.

Пример:

```
annotations:
  prometheus.deckhouse.io/istio-mtls: "true"
```

Пример настройки сбора метрик с Service

Ниже приведён пример настройки сбора метрик с 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 настройки сбора метрик с Deployment

Ниже приведён пример настройки сбора метрик с 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

```

Дополнительные аннотации для тонкой настройки

Для более точной настройки мониторинга приложения можно указать дополнительные аннотации для пода или сервиса, для которых настраивается мониторинг:

- `prometheus.deckhouse.io/path` — путь для сбора метрик (по умолчанию: `/metrics`);
- `prometheus.deckhouse.io/query-param-$name` — GET-параметры, которые будут преобразованы в map вида `$name=$value` (по умолчанию: `''`). Можно указать несколько таких аннотаций. Например, `prometheus.deckhouse.io/query-param-foo=bar` и `prometheus.deckhouse.io/query-param-bar=zxc` будут преобразованы в запрос вида `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 прогреваются кеши), 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 который добавлен лейбл `prometheus.deckhouse.io/custom-target`.

Настройка сбора метрик с помощью ресурса PodMonitor или ServiceMonitor

DKP поддерживает подключение приложений через два схожих по функциональности ресурса:

- **PodMonitor** (рекомендуемый вариант) — обнаруживает поды напрямую и собирает метрики с их контейнеров. В большинстве случаев это предпочтительный вариант, поскольку он работает напрямую с подами и не зависит от наличия сервисов.
- **ServiceMonitor** — обнаруживает сервисы и собирает метрики с подов, находящихся за ними. При этом сервисы используются как источник метаданных (например, лейблов), а фактический сбор метрик выполняется с адресов подов, входящих в соответствующие эндпоинты.

Оба ресурса позволяют задавать интервал опроса, пути, TLS-параметры, параметры переписывания лейблов (`relabeling`) и другие настройки.

Разница между ресурсами заключается в источнике собираемых метрик. Используйте **PodMonitor**, если требуется сбор метрик напрямую с подов, и **ServiceMonitor** — если ваше

приложение публикует метрики через Service.

Чтобы подключить приложение к системе мониторинга с помощью одного из этих ресурсов, выполните следующие шаги:

1. Установите лейбл `prometheus.deckhouse.io/monitor-watcher-enabled: "true"` в том же пространстве имен, где будет находиться PodMonitor или ServiceMonitor:

```
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/monitor-watcher-enabled: "true"
```

2. Создайте ресурс PodMonitor или ServiceMonitor, указав обязательный лейбл `prometheus: main`, а также перечислив параметры необходимых эндпоинтов.

Поле `spec.selector.matchLabels` определяет, какие объекты будут использоваться для сбора метрик. Для PodMonitor указываются лейблы целевых подов, а для ServiceMonitor — лейблы целевого сервиса.

Пример для PodMonitor:

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      # Лейбл подов, с которых необходимо собирать метрики.
      app: example-app
  podMetricsEndpoints:
    - port: metrics
      interval: 30s
      path: /metrics
```

Пример для ServiceMonitor:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      # Лейбл сервиса, через который доступны метрики приложения.
      app: example-app
  endpoints:
    - port: web
      interval: 30s
      path: /metrics
```

При необходимости задайте дополнительные настройки, используя справку по доступным параметрам ресурсов: PodMonitor, ServiceMonitor.

Добавление пользовательских лейблов в метрики

Если вам нужно добавить дополнительные лейблы непосредственно в собираемые метрики, воспользуйтесь правилами переписывания метрик (с помощью параметра `metricRelabelings`).

Ниже приведен пример ресурса PodMonitor, который добавляет в собираемые метрики лейбл `application` со значением `dotnet`:

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: example-app
  namespace: frontend
  labels:
    prometheus: main
spec:
  selector:
    matchLabels:
      app: example-app
  podMetricsEndpoints:
    - port: metrics
      interval: 30s
      path: /metrics
      metricRelabelings:
        - targetLabel: application
          replacement: dotnet
          action: Replace
```

После применения такого правила в метриках появится дополнительный лейбл:

```
http_requests_total{
  application="dotnet",
  code="404",
  container="example-app",
  instance="10.111.5.69:8080",
  namespace="frontend",
  pod="example-app-75476c469d-rqq4s"
}
```

Настройка сбора метрик через `scrape_configs` с помощью ресурса `ScrapeConfig`

ScrapeConfig — это кастомный ресурс, который позволяет настраивать раздел конфигурации Prometheus под названием `scrape_config` для полного контроля над процессом сбора метрик.

Чтобы подключить приложение к системе мониторинга, выполните следующие шаги:

1. Установите лейбл `prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"` в пространстве имен, где будет находиться ScrapeConfig:

```
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/scrape-configs-watcher-enabled: "true"
```

2. Создайте ресурс ScrapeConfig, указав 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'

```

При необходимости задайте дополнительные настройки, используя справку по доступным параметрам ресурса.

Настройка сбора метрик с помощью blackbox-exporter

DKP поддерживает сбор метрик доступности с blackbox-exporter, который не входит в состав DKP и должен быть установлен отдельно в кластере. Для этого используется кастомный ресурс Probe, который описывает проверки доступности (probes), выполняемые Prometheus.

Чтобы подключить Probe к системе мониторинга DKP, выполните следующие шаги:

1. Установите лейбл `prometheus.deckhouse.io/probe-watcher-enabled: "true"` в пространстве имен, где будет находиться Probe:

```

apiVersion: v1
kind: Namespace
metadata:
  name: frontend
  labels:
    prometheus.deckhouse.io/probe-watcher-enabled: "true"

```

2. Создайте ресурс Probe, указав обязательный лейбл `prometheus: main`:

```
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
```

При необходимости задайте дополнительные настройки, используя справку по доступным параметрам ресурса.

Дополнительные ресурсы

- Документация модуля `monitoring-custom`
- Кастомные ресурсы модуля `operator-prometheus`

Доставка приложений с помощью Argo CD

В этом разделе описано, как организовать доставку приложений с помощью Argo CD в Deckhouse Kubernetes Platform (DKP).

Argo CD позволяет описывать приложения декларативно и синхронизировать их состояние с содержимым Git-репозитория. Пользователь задаёт источник манифестов, целевой кластер, неймспейс и параметры синхронизации, после чего Argo CD развёртывает приложение и поддерживает его в целевом состоянии.

В DKP экземпляры Argo CD разворачиваются с помощью модуля `operator-argo`. Работа с Argo CD обычно включает:

- создание или использование существующего объекта `AppProject`;
- подготовку целевого неймспейса для приложения;
- создание объекта `Application`, описывающего источник приложения и правила синхронизации;
- создание приложения через веб-интерфейс Argo CD;
- создание приложения с помощью CLI-утилиты `argocd`.

Предварительные условия

Перед началом работы должны быть выполнены следующие условия:

- администратор кластера включил модуль `operator-argo`;
- администратор развернул хотя бы один экземпляр Argo CD;
- у пользователя есть доступ к нужному экземпляру Argo CD и целевым неймспейсам.

Если экземпляр Argo CD ещё не развёрнут, обратитесь к администратору или воспользуйтесь инструкцией из раздела «Запуск Argo CD».

Проекты AppProject

`AppProject` — это кастомный ресурс Argo CD, который задаёт логические границы проекта. С его помощью можно определить:

- какие Git-репозитории разрешено использовать как источник приложений;
- в какие кластеры и неймспейсы разрешено развёртывать приложения;
- какие кластерные и `namespaced`-ресурсы разрешено использовать;
- какие роли и политики доступа действуют внутри проекта.

Каждый объект `Application` должен ссылаться на проект через параметр `spec.project`.

По умолчанию в Argo CD существует проект `default`. Его можно использовать для первых экспериментов и простых сценариев. Для `production`-окружений рекомендуется создать отдельные объекты `AppProject`, чтобы ограничить доступ к репозиториям, кластерам и неймспейсам.

Манифест объекта `AppProject default` приведён ниже:

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
  namespace: argocd
spec:
  clusterResourceWhitelist:
  - group: '*'
    kind: '*'
  destinations:
  - namespace: '*'
    server: '*'
  sourceRepos:
  - '*'
```

Подготовка неймспейса

Перед развёртыванием приложения создайте целевой неймспейс и добавьте лейбл `argocd.argoproj.io/managed-by`, указывающий, какой экземпляр Argo CD управляет ресурсами в этом неймспейсе.

Пример:

```
apiVersion: v1
kind: Namespace
metadata:
  name: demo
  labels:
    argocd.argoproj.io/managed-by: argocd
```

В этом примере неймспейс `demo` будет управляться экземпляром Argo CD, развёрнутым в неймспейсе `argocd`.

Развёртывание приложения

Приложение в Argo CD можно создать несколькими способами:

- декларативно — с помощью объекта `Application`;
- интерактивно — через веб-интерфейс Argo CD;
- с помощью CLI-утилиты `argocd`.

Создание приложения с помощью объекта `Application`

Для описания приложения используется объект `Application`. В нём указываются:

- проект Argo CD (`spec.project`);
- источник манифестов или чарта (`spec.source`);
- целевой кластер и неймспейс (`spec.destination`);
- политика синхронизации (`spec.syncPolicy`).

Пример объекта `Application`:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: demo
  namespace: argocd
spec:
  destination:
    namespace: demo
    server: https://kubernetes.default.svc
  project: default
  source:
    path: helm-guestbook
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
  syncPolicy:
    # Включить автоматическую синхронизацию.
    automated:
      # Удалять устаревшие ресурсы.
      prune: true
      # Включить самовосстановление в случае сторонних изменений.
      selfHeal: true
```

После создания объекта `Application` Argo CD начнёт отслеживать состояние приложения и синхронизировать его с содержимым репозитория. В результате в неймспейсе `demo` должны появиться ресурсы, относящиеся к приложению `demo`:

```
d8 k -n demo get deployment,svc,pod
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/demo-helm-guestbook 1/1      1              1            15s

NAME                                TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
service/demo-helm-guestbook         ClusterIP      10.222.177.84  <none>         80/TCP     15s

NAME                                READY    STATUS    RESTARTS    AGE
pod/demo-helm-guestbook-66d5d69ccd-xfkb7 1/1      Running   0           15s
```

Создание приложения через веб-интерфейс Argo CD

Приложение можно создать через веб-интерфейс Argo CD. Для этого:

1. Откройте веб-интерфейс нужного экземпляра Argo CD.
2. Перейдите в раздел «Applications».
3. Нажмите «New App».
4. Укажите имя приложения, проект, репозиторий, ревизию, путь к манифестам или чарту, а также целевой кластер и неймспейс.
5. При необходимости настройте автоматическую синхронизацию и дополнительные параметры.
6. Нажмите «Create».

Поля формы в веб-интерфейсе соответствуют основным параметрам объекта Application: проекту, источнику приложения, целевому кластеру, неймспейсу и политике синхронизации.

Создание приложения с помощью CLI-утилиты argocd

CLI-утилита `argocd` позволяет создавать и сопровождать приложения из командной строки. Бинарный файл `argocd` можно загрузить из раздела «Documentation» веб-интерфейса Argo CD.

Перед созданием приложения аутентифицируйтесь с помощью команды:

```
argocd login <ARGOCD_DOMAIN>:443
```



При использовании аутентификации через SSO добавьте к команде логина флаг `--sso`.

Пример создания приложения:

```
argocd app create guestbook \
  --repo https://github.com/argoproj/argocd-example-apps.git \
  --path guestbook \
  --dest-namespace demo \
  --dest-server https://kubernetes.default.svc \
  --directory-recurse \
  --sync-policy automated \
  --self-heal \
  --auto-prune
```

Как и при создании объекта Application, в команде указываются источник приложения, целевой кластер, неймспейс и политика синхронизации.

Чтобы просмотреть статус развёрнутого приложения, используйте команду `argocd app get <APP_NAME>`, например:

```
argocd app get guestbook
Name:          argocd/guestbook
Project:       default
Server:        https://kubernetes.default.svc
Namespace:     demo
URL:           https://argocd.192.168.0.235.sslip.io/applications/guestbook
Source:
- Repo:        https://github.com/argoproj/argocd-example-apps.git
  Target:
  Path:         guestbook
SyncWindow:    Sync Allowed
Sync Policy:   Automated (Prune)
Sync Status:   Synced to (8088f4c)
Health Status: Healthy
```

GROUP	KIND	NAMESPACE	NAME	STATUS	HEALTH	HOOK	MESSAGE
apps	Deployment	demo	guestbook-ui	Synced	Healthy		
	deployment.apps/guestbook-ui		unchanged				
	Service	demo	guestbook-ui	Synced	Healthy		

Дополнительные ресурсы

- [Официальный сайт Argo CD](#)
- [Официальный сайт Argo CD Operator](#)
- [Документация модуля operator-argo](#)

Marketplace

Этот раздел описывает способы использования Marketplace в Deckhouse Kubernetes Platform (DKP).

Marketplace позволяет устанавливать готовые приложения в свой неймспейс из реестров, подключённых администратором кластера. Каждое приложение устанавливается в виде ресурса Application и может существовать в нескольких независимых экземплярах — например, отдельные Redis-инстансы для кеширования и сессий в одном неймспейсе.



Marketplace и ресурс Application доступны начиная с DKP версии 1.76.

Предварительные условия

Перед установкой приложения администратор кластера должен подключить хотя бы один реестр пакетов. Уточните у администратора, доступны ли объекты ApplicationPackageVersion:

```
d8 k get apv
```

Если вывод пуст — реестр ещё не был просканирован, обратитесь к администратору.

В разделе «Установка и управление приложениями» описан просмотр доступных версий, установка, обновление и удаление приложений.

В разделе «Диагностика» описано, как диагностировать и устранять проблемы с установленными приложениями.

Дополнительные ресурсы

- [Глобальные кастомные ресурсы](#)

Установка и управление приложениями

Просмотр доступных версий пакетов

Для получения списка всех доступных версий пакетов выполните следующую команду (можно использовать сокращённое имя — `apv`):

```
d8 k get apv
```

Пример вывода:

NAME USED BY	PACKAGE	REPOSITORY	TRANSITION TIME	METADATA LOADED
my-registry-redis-v7.2.0 1	redis	my-registry	2d	True
my-registry-redis-v7.3.0	redis	my-registry	5h	True
my-registry-pg-v15.0.0 2	postgres	my-registry	2d	True

Для фильтрации по имени пакета используйте следующую команду (в примере фильтруются версии пакета `redis`):

```
d8 k get apv -l package=redis
```

Для просмотра реестров, в которых доступен пакет, используйте следующую команду (можно использовать сокращённое имя — `ap`):

```
d8 k get ap redis \
-o jsonpath='{.status.availableRepositories}'
```

i Устанавливать можно только версии с `MetadataLoaded=True`. Это означает, что OpenAPI-схема пакета, описание и требования успешно загружены из реестра. Пакет с `MetadataLoaded=False` не может быть установлен до получения метаданных.

Установка приложения

Для установки приложения создайте объект `Application` в нужном неймспейсе.

Пример манифеста для установки Redis из пакета `redis` версии `v7.2.0` с настройкой `maxmemory`:

```
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-cache
  namespace: my-app
spec:
  packageName: redis
  packageVersion: "v7.2.0"
  # Если подключён только один репозиторий с этим пакетом, можно не указывать
  packageRepositoryName: my-registry
  settings:
    replicas: 3
    maxmemory: "256mb"
```

i `spec.settings` проверяется по OpenAPI-схеме, определённой в пакете. Если схема отклонила настройки, `Application` не будет создан. Детали схемы можно посмотреть в соответствующем `ApplicationPackageVersion`.

Ограничения на имена

Имя `Application` (`metadata.name`) должно быть **не более 24 символов**. Это обязательно, поскольку все поды получают префикс из имени экземпляра: 24 символа имени экземпляра + 24 символа имени ресурса + 15 символов суффикса `Deployment` укладываются в ограничение Kubernetes на имя пода в 63 символа.

Проверка статуса приложения

Для получения краткой информации о статусе приложения выполните (можно использовать сокращённое имя — `app`):

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME>
```

Пример вывода:

NAME	PACKAGE	VERSION	INSTALLED	READY	MESSAGE
redis-cache	redis	v7.2.0	True	True	

Для получения полного статуса, включая условия (conditions), выполните:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> -o yaml
```

Условия (Conditions)

Состояние приложения детально описывается через набор условий:

Условие	Значение
Installed	Пакет скачан, манифесты и хуки применены при первичной установке
UpdateInstalled	Новая версия скачана, манифесты и хуки применены при обновлении
ConfigurationApplied	Пользовательские настройки успешно применены
Scaled	Все реплики подов находятся в состоянии Ready
Managed	Приложение корректно управляется DKP
Ready	Приложение полностью готово к работе

Для быстрого просмотра всех условий используйте следующую команду:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> \
-o jsonpath='{range .status.conditions[*]}{.type}: {.status} ({.reason}){"\n"}{end}'
```

Пример вывода:

```
Installed: True (Installed)
UpdateInstalled: False (Pending)
ConfigurationApplied: True (ConfigurationApplied)
Managed: True (Managed)
Scaled: True (Scaled)
Ready: True (Ready)
```

Summary

Поле `status.summary` содержит краткое описание текущего состояния приложения — его удобно смотреть в первую очередь при диагностике:

```
status:
  summary:
    state: Updating
    message: "Update is waiting for dependent modules to converge; previous version is still serving"
    tip: "Waiting until DKP processes all dependent modules to start the update."
```

- **state** — текущее общее состояние приложения.
- **message** — объясняет, почему приложение находится в этом состоянии.
- **tip** — что нужно сделать для решения проблемы или чего ожидает DKP.

Несколько экземпляров

Один и тот же пакет можно установить несколько раз в одном или разных неймспейсах — каждый с отдельным именем и настройками. Например, можно создать два экземпляра Redis — один для кеширования, другой для сессий:

```
# Экземпляр для кеширования
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-cache
  namespace: team-alpha
spec:
  packageName: redis
  packageRepositoryName: my-registry
  packageVersion: "v7.2.0"
  settings:
    maxmemory: "512mb"
---
# Экземпляр для сессий
apiVersion: deckhouse.io/v1alpha1
kind: Application
metadata:
  name: redis-sessions
  namespace: team-alpha
spec:
  packageName: redis
  packageRepositoryName: my-registry
  packageVersion: "v7.2.0"
  settings:
    maxmemory: "128mb"
```

Все объекты Kubernetes, создаваемые приложением, получают префикс из имени экземпляра — например, `redis-cache-deployment` и `redis-sessions-deployment` — что исключает конфликты имён.

Обновление приложения

Обновление выполняется вручную: измените значение `spec.packageVersion` на нужную версию и примените изменение:

```
d8 k patch app -n <NAMESPACE> <APPLICATION_NAME> --type=merge -p '{"spec":
{"packageVersion":"v7.3.0"}}'
```

Пока обновление выполняется, условие `UpdateInstalled` будет иметь значение `False` с `reason: Pending`. После успешного завершения оно станет `True`. До завершения обновления продолжает работать предыдущая версия.

Если указанная версия не существует в репозитории, `UpdateInstalled` становится `False` с `reason: UpdateFailed`, а текущая версия продолжает работу.



Указание более ранней версии приложения (downgrade) допускается, но DKP не применяет никакую логику миграции при откате. Убедитесь в совместимости настроек с целевой версией перед применением изменения, при необходимости.

Удаление приложения

Для удаления приложения удалите объект Application. Например:

```
d8 k delete app -n <NAMESPACE> <APPLICATION_NAME>
```

При удалении Application, все созданные им объекты Kubernetes (если они не защищены аннотациями `helm.sh/resource-policy: keep` или `werf.io/ownership: anyone` в шаблонах пакета), будут удалены.

FAQ

Можно ли обновлять приложение автоматически?

Нет. В текущей реализации обновления требуют ручного изменения `spec.packageVersion`. Автоматические обновления через release channels запланированы на будущие версии.

Может ли Application зависеть от другого Application?

Нет. Application может объявлять зависимости только от модулей (через `requirements.modules` в `package.yaml`). Это архитектурное ограничение, обеспечивающее изоляцию экземпляров.

Можно ли установить одно приложение в разных неймспейсах?

Да. Создайте объекты Application с одинаковыми `packageName` и `packageVersion` в разных неймспейсах — каждый будет полностью независимым экземпляром.

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Диагностика

Проверка наличия CRD Marketplace

Если `d8 k get app` возвращает ошибку, возможно, CRD Marketplace не установлены. Для проверки выполните следующую команду:

```
d8 k get crd | grep -E 'application|package'
```

Ожидаемый вывод:

```
applicationpackages.deckhouse.io      2026-02-10T14:54:41Z
applicationpackageversions.deckhouse.io 2026-02-10T14:54:41Z
applications.deckhouse.io              2026-02-10T14:54:41Z
packagerepositories.deckhouse.io       2026-02-10T14:54:41Z
packagerepositoryoperations.deckhouse.io 2026-02-10T14:54:41Z
```

Если какие-либо CRD отсутствуют, обратитесь к администратору кластера. Marketplace требует DKP версии 1.76 или выше.

Чтение summary приложения

Самый быстрый способ понять, почему приложение не работает — проверить `status.summary`. Для этого выполните следующую команду (можно использовать сокращённое имя — `app`):

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> -o yaml | grep -A5 'summary:'
```

Пример вывода:

```
summary:
  state: Updating
  message: "Update is waiting for dependent modules to converge; previous version is still serving"
  tip: "Waiting until DKP processes all dependent modules to start the update."
```

- **state** — текущее общее состояние приложения.

- **message** — объясняет, почему приложение находится в этом состоянии.
- **tip** — что нужно сделать для решения проблемы или чего ожидает DKP.

Чтение отдельных условий

Для более детального просмотра состояния выполните следующую команду:

```
d8 k get app -n <NAMESPACE> <APPLICATION_NAME> \
  -o jsonpath='{range .status.conditions[*]}{.type}: {.status} ({.reason}) -
  {.message}{"\n"}{end}'
```

Пример вывода при зависшем обновлении:

```
conditions:
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Installed
  status: "True"
  type: Installed
- lastTransitionTime: "2026-02-25T17:12:25Z"
  message: "Update is waiting for dependent modules to converge"
  observedGeneration: 1
  reason: Pending
  status: "False"
  type: UpdateInstalled
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: ConfigurationApplied
  status: "True"
  type: ConfigurationApplied
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Managed
  status: "True"
  type: Managed
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Scaled
  status: "True"
  type: Scaled
- lastTransitionTime: "2026-02-25T16:39:30Z"
  message: ""
  observedGeneration: 1
  reason: Ready
  status: "True"
  type: Ready
currentVersion:
version: v0.0.20
```

В этом примере `Installed=True` (приложение запущено на версии `v0.0.20`), но `UpdateInstalled=False/Pending` означает, что обновление в очереди и ожидает завершения зависимого модуля.

Просмотр логов контроллера DKP

Если условий статуса недостаточно для диагностики, просмотрите логи контроллера:

```
d8 k logs deployments/deckhouse -n d8-system | grep <APPLICATION_NAME>
```

Просмотр логов подов приложения

Для получения списка подов, созданных приложением, выполните следующую команду:

```
d8 k get pods -n <NAMESPACE> -l app.kubernetes.io/instance=<APPLICATION_NAME>
```

Для просмотра логов конкретного пода выполните:

```
d8 k logs -n <NAMESPACE> <POD_NAME>
```

Для просмотра логов конкретного деплоя с префиксом имени экземпляра выполните:

```
d8 k logs -n <NAMESPACE> deployments/<APPLICATION_NAME>-<RESOURCE_NAME>
```

Частые условия и их значение

Условие	Reason при Status=False	Что проверить
Installed	InstallFailed	Логи контроллера DKP, проверьте настройки по OpenAPI-схеме
UpdateInstalled	Pending	Конвергенция зависимых модулей — проверьте условия модуля d8
UpdateInstalled	UpdateFailed	Указанная packageVersion не существует в репозитории — проверьте через <code>d8 k get arv -l package=<имя></code>
ConfigurationApplied	ConfigurationFailed	Ошибка валидации настроек — проверьте по схеме ApplicationPackageVersion
Scaled	NotScaled	Поды не готовы — проверьте события пода через <code>d8 k describe pod</code>
Ready	NotReady	Одно или несколько условий выше не выполнены

Дополнительные ресурсы

- Глобальные кастомные ресурсы

Настройка сборки логов приложений

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;
- хранить логи во внутреннем кратковременном хранилище на базе Grafana Loki.

Общий механизм сбора, доставки и фильтрации логов подробно описан в разделе «Архитектура».

Пользователям DKP доступна настройка параметров сбора логов из приложения с помощью ресурса PodLoggingConfig, который описывает источник логов в рамках заданного пространства имён, включая правила сбора, фильтрации и парсинга.

Настройка сбора логов из приложения

1. Уточните у администратора DKP, настроен ли сбор логов и хранилище в вашем кластере. Также попросите сообщить вам название хранилища, которое вы укажете в параметре `clusterDestinationRefs`.
2. Создайте ресурс `PodLoggingConfig` в своём пространстве имён.

В данном примере логи собираются со всех подов указанного пространства имён и отправляются в кратковременное хранилище на базе Grafana Loki:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
```

3. **(Опционально)** Ограничьте сбор логов по лейблу.

Если вам нужно собирать логи только с определённых подов, например, только от приложений с лейблом `app=backend`, добавьте параметр `labelSelector`:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
  labelSelector:
    matchLabels:
      app: backend
```

4. **(Опционально)** Настройте фильтрацию логов.

Используя фильтры `labelFilter` и `logFilter`, вы можете установить фильтрацию по метаданным или полям сообщений. Например, в данном случае в хранилище отправятся лишь те логи, в которых нет полей со строкой `.*GET /status" 200$`:

```
apiVersion: deckhouse.io/v1alpha1
kind: PodLoggingConfig
metadata:
  name: app-logs
  namespace: my-namespace
spec:
  clusterDestinationRefs:
    - loki-storage
  labelSelector:
    matchLabels:
      app: backend
  logFilter:
    - field: message
      operator: NotRegex
      values:
        - .*GET /status" 200$
```

5. Примените созданный манифест с помощью следующей команды:

```
d8 k apply -f pod-logging-config.yaml
```

Примеры

Создание source в пространстве имён и чтение логов всех подов в нём с направлением их в Loki

Следующий pipeline создает source в пространстве имён `test-whispers`, читает логи всех подов в нём и отправляет 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 подов в указанном пространстве имён с определенным лейблом

Пример настройки чтения подов с лейблом `app=booking` в пространстве имён `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
```

Дополнительные ресурсы

- Кастомные ресурсы модуля `log-shipper`