УТВЕРЖДЕНО

RU.86432418.00001-01 91 04-1 - ЛУ

Программное обеспечение «Deckhouse Platform Certified Security Edition»

Руководство пользователя

RU.86432418.00001-01 91 04-1

330 Листов

2025

Содержание

Список используемых обозначений и сокращений

CPU	Central Processing Unit, центральное процессорное устройство
CRI	Container Runtime Interface, определяет API между Kubernetes и Container Runtime (средой выполнения контейнеров)
CVE	Common Vulnerabilities and Exposures, общедоступная база данных, которая предоставляет уникальные идентификаторы для общеизвестных уязвимостей информационной безопасности
DNS	Domain Name System, система доменных имен
DVCR	Deckhouse Virtualization Container Registry, репозиторий для хранения и кэширования образов виртуальных машин.
JSON	JavaScript Object Notation, стандартный текстовый формат для хранения структурированных данных и обмена ими
HTTP	Hypertext Transfer Protocol, прикладной протокол, который определяет правила обмена данными в интернете
HTTPS	Hyper Text Transfer Protocol Secure, расширение протокола HTTP для поддержки шифрования в целях повышения безопасности
Kubeconfig	Файл конфигурации клиента
OCI	Open Container Initiative, проект, содержащий открытые отраслевые стандарты для форматов контейнеров и сред выполнения
OCI-репозиторий	места для хранения образов контейнеров, совместимых со стандартами OCI, которые определяют формат образов и способы их запуска
QEMU	Quick Emulator, программа с открытым исходным кодом, которая может использоваться как эмулятор и как виртуализатор
RPS	Requests Per Second, количество запросов, обрабатываемых системой за одну секунду
SSH	Secure Shell, безопасный сетевой протокол, который позволяет удаленно управлять компьютером или сервером,

	обмениваться данными и выполнять команды через Интернет
TTY	Teletypewriter, виртуальный телетайп, эмулируемый аппаратным обеспечением
URL	Uniform Resource Locator, адрес веб-ресурса в интернете, который позволяет браузеру найти и открыть страницу, файл или другой контент
VNC	Virtual Network Computing, система удалённого доступа к рабочему столу компьютера
ВМ	Виртуальная машина
КТС	Комплекс технических средств
ОС	Операционная система
ПО	Программное обеспечение
ПО «Deckhouse Platform»	Программное обеспечение «Deckhouse Platform Certified Security Edition»
ТУ	Технические условия
УПД	Управление доступом
ФО	Формуляр
ФСТЭК России	Федеральная служба по техническому и экспортному контролю
ЦПУ, CPU	Центральный процессор, central processing unit

1. Назначение средства

1.1. Область применения

Данное руководство предназначено для пользователей программного обеспечения «Deckhouse Platform Certified Security Edition» (далее по тексту – ПО «Deckhouse Platform», ПО).

1.2. Краткое описание возможностей

Объектом оценки является программное обеспечение ПО «Deckhouse Platform» назначением которого является управление Kubernetes-кластерами Deckhouse.

1.3. Уровень подготовки пользователя

Пользователи ПО «Deckhouse Platform» должны обладать базовыми навыками:

- наличие практических навыков работы с компьютерной техникой, операционным системами и Интернет-браузерами;

- знание технологических процессов обработки информации, выполняемых автоматизированным способом и знакомство с эксплуатационной документацией.

1.4. Перечень эксплуатационной документации, с которой необходимо ознакомиться пользователю

Пользователи обязаны до начала эксплуатации ПО «Deckhouse Platform» ознакомиться с эксплуатационной документацией, поставляемой с ПО «Deckhouse Platform», включая руководство пользователя.

2. Подготовка к работе

Для работы с ПО «Deckhouse Platform» пользователям требуется рабочее место, программа-клиент d8 (входит в поставку ПО «Deckhouse Platform») и файл конфигурации kubectl.

В рамках подготовки к работе с ПО «Deckhouse Platform» пользователям необходимо ознакомиться с данным руководством.

3. Режимы работы средства

ПО «Deckhouse Platform» функционирует в следующих режимах:

- Нормальный режим. Штатный режим функционирования. ПО «Deckhouse Platform» функционирует в заданных режимах и объемах обрабатываемой информации в соответствии с документацией. В этом режиме ПО «Deckhouse Platform» обеспечивает выполнение всех заявленных функций и обеспечивает режим работы — 24 часа в день, 7 дней в неделю (24x7);
- Сервисный режим. Режим, включаемый вручную в конфигурации модуля ПО «Deckhouse Platform» (ModuleConfig <МОДУЛЬ>) установкой параметра `maintenance: NoResourceReconciliation`. В этом режиме функционирование модуля ПО «Deckhouse Platform» полностью или частично приостанавливается, также, невозможно обновление ПО «Deckhouse Platform».

  При работе в сервисном режиме, в системе мониторинга появляется алерт `ModuleIsInMaintenanceMode`, следующего содержания:

```
Module `<MODULE_NAME>` is running in maintenance mode. In this mode, its state is not reconciled, which prevents configuration or hook changes from being applied automatically.

To switch the module back to normal mode, edit the corresponding ModuleConfig resource with the following command:
        d8 k patch moduleconfig <НАЗВАНИЕ_МОДУЛЯ> --type=json -p='[{"op": "remove", "path": "/spec/maintenance"}]'
```

Функционирование ПО «Deckhouse Platform» при отказах и сбоях серверного общесистемного и специального программного обеспечения, и оборудования, в том числе инфраструктурных узлов ПО «Deckhouse Platform», не предусматривается.

4. Функции и интерфейсы, доступные пользователю

ПО «Deckhouse Platform» предназначен для управления Kubernetes-кластерами Deckhouse.

Интерфейсы, доступные пользователю ПО «Deckhouse Platform», определяются в соответствии с назначенными ролями (см. Приложение А). В п.6.3 настоящего документа описано, как работать с этими интерфейсами. Дополнительный справочник приведен в электронном приложении к настоящему документу, каталог «Электронные приложения» - «Deckhouse Platform. Руководство администратора» - «Справочник администратора.pdf»).

5. Реализация функциональных возможностей средства согласно исполнениям ПО

В настоящем разделе представлено сопоставление исполнений ПО с функциональными возможностями, описанными в соответствующих разделах документа.

№ п/п	Разделы Руководства пользователя	Исполнения
6	**Описание операций**	Kubernetes+Virtualization; Virtualization; Kubernetes
6.1	Подключение к кластеру	Kubernetes+Virtualization; Kubernetes
6.2	Работа с кластером с помощью программы-клиента	Kubernetes+Virtualization; Kubernetes
6.3	Работа с веб-интерфейсом кластера	Kubernetes+Virtualization; Kubernetes
6.4	Настройки логирования	Kubernetes+Virtualization; Kubernetes
6.5	Работа с модулем виртуализации	Kubernetes+Virtualization; Virtualization;
7	**Доставка приложений**	**Kubernetes+Virtualization; Kubernetes**
7.1	Описание функциональных характеристик	Kubernetes+Virtualization; Kubernetes
7.2	Конфигурация проекта	Kubernetes+Virtualization; Kubernetes
7.3	Сборка	Kubernetes+Virtualization; Kubernetes

№ п/п	Разделы Руководства пользователя	Исполнения
7.4	Развертывание	Kubernetes+Virtualization; Kubernetes
7.5	Дистрибуция	Kubernetes+Virtualization; Kubernetes
7.6	Очистка	Kubernetes+Virtualization; Kubernetes

6. Описание операций

6.1. Подключение к кластеру

Для подключения к развернутому кластеру необходимо получить от администратора безопасности файл конфигурации клиента (далее – kubeconfig) и, при необходимости, учетные данные пользователя веб-интерфейсов кластера.

Подключение к кластеру осуществляется с помощью программы-клиента d8 (входит в поставку ПО «Deckhouse Platform»). Файл конфигурации предоставляется администратором безопасности, Программа-клиент предоставляется администратором информационной (автоматизированной) системы.

6.1.1. Проверка подключения к кластеру с помощью программы-клиента

Для проверки подключения к кластеру с помощью программы-клиента, выполните:

d8 k --kubeconfig <ФАЙЛ_КОНФИГУРАЦИИ> cluster-info,

где <ФАЙЛ_КОНФИГУРАЦИИ> – полученный от администратора безопасности файл конфигурации клиента, с учетом пути к файлу.

Пример вывода:

```
# d8 k --kubeconfig ~/.kube/config cluster-info
Kubernetes control plane is running at https://192.168.0.10:6445
To further debug and diagnose cluster problems, use 'd8 k cluster-info dump'.
```

6.1.2. Проверка доступа к веб-интерфейсу кластера

Проверка выполняется путем доступа к веб-интерфейсу Grafana.

Необходимо открыть в веб-браузере веб-интерфейс Grafana, доступный по адресу grafana.<ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> – строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate. Формат адреса подключения к Grafana может быть иным. Точный адрес подключения можно узнать у администратора информационной (автоматизированной) системы.

При первом входе в веб-интерфейс появится окно аутентификации (Рисунок 1.).

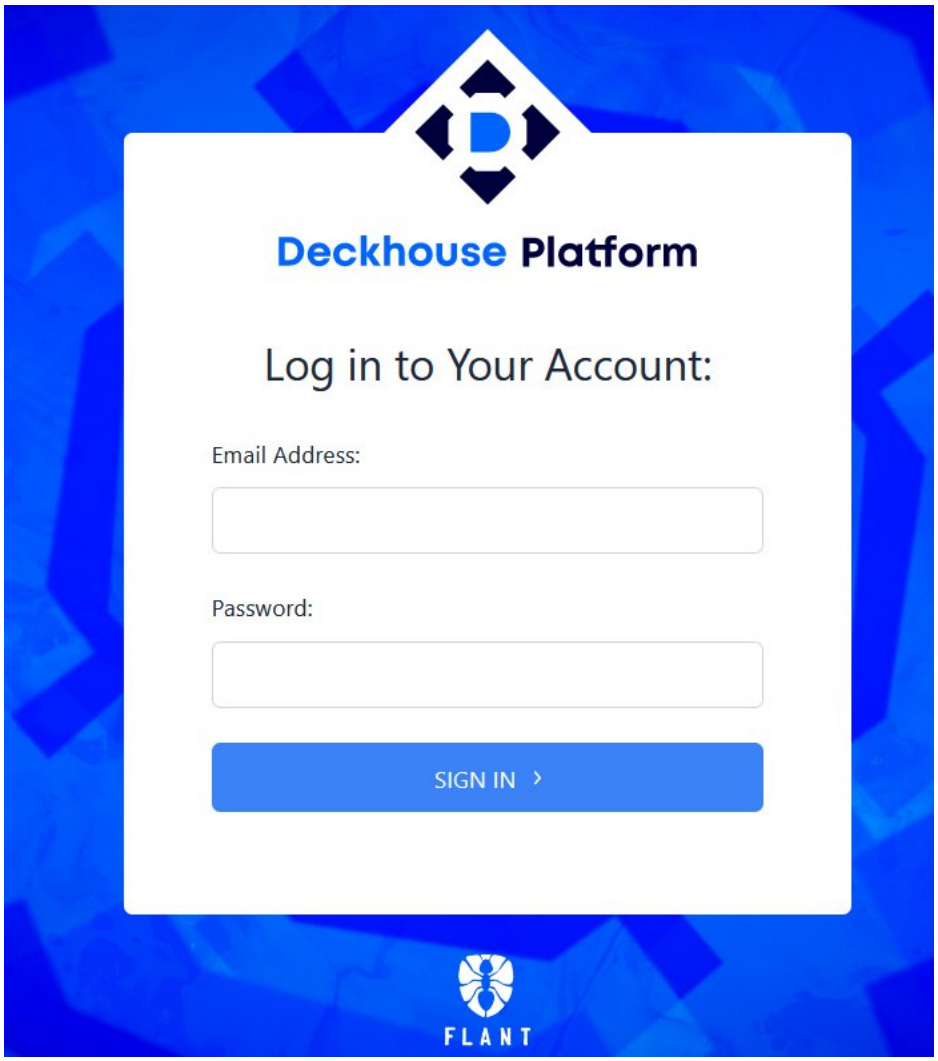


Рисунок 1 – Окно аутентификации веб-интерфейса.

Для аутентификации введите учетные данные, полученные от администратора безопасности.

При успешной аутентификации откроется страница веб-интерфейса Grafana.

6.2. Работа с кластером с помощью программы-клиента

С помощью программы d8 (входит в поставку ПО «Deckhouse Platform») можно выполнять различные операции в кластере Kubernetes, учитывая предоставленные разрешения. Работа с программой осуществляется в терминале.

Создание объектов в кластере, их модификация и удаление возможно выполнять как с использованием команд утилиты d8, так и с использованием подготовленного файла манифеста ресурсов (декларативный способ). Для вызова справки по параметрам программы-клиента d8 выполните: d8 help.

При составлении файла манифеста ресурсов, информацию о составе его полей, допустимых значениях, а также описание полей можно посмотреть в п. 6.2.3.

6.2.1. Создание объекта

Для создания объектов в кластере императивным способом используется команда *d8 k run*.

Пример создания объекта императивным способом:

```
d8 k run nginx --image=nginx
```

Для создания объектов в кластере с помощью файла манифеста, используется команда *d8 k create*.

Пример создания объекта с помощью d8 k и файла манифеста:

Файл манифеста nginx.yaml:

```
  apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Создание объекта с использованием файла манифеста nginx.yaml:

```
d8 k create -f nginx.yaml
```

6.2.2. Удаление объекта

Для удаления объектов в кластере императивным или декларативным способом используется команда *d8 k delete*.

Пример удаления объекта императивным способом, с использованием команд утилиты d8:

```
d8 k delete po nginx-abf4ef5
```

Пример удаления объекта с помощью d8 k и файла манифеста:

Файл манифеста nginx.yaml:

```
  apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Удаление объекта с использование файла манифеста nginx.yaml:

```
d8 k delete -f nginx.yaml
```

6.2.3. Получение информации об объекте

Для получения информации об объекте кластере используется команда *d8 k get*. С помощью нее можно получить информацию об объектах кластера в различных форматах, включая манифесты кластера, которые можно использовать для дальнейшего создания объектов с помощью команды d8 k create.

Пример получения информации об объекте кластера с помощью команды d8 k get:

d8 k get po nginx-abf4ef5

6.2.4. Обновление объекта

Для обновления объектов в кластере декларативным способом используется команда *d8 k apply*.

Пример создания объекта с помощью d8 k и файла манифеста:

Файл манифеста nginx.yaml:

```
  apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Создание объекта с использованием файла манифеста nginx.yaml:

d8 k apply -f nginx.yaml

6.3. Работа с веб-интерфейсом кластера

Интерфейс предназначен для просмотра состояния кластера, просмотра событий безопасности и журналов, автоматического получения параметров конфигурации d8 k для доступа к кластеру и просмотра локальной версии документации в соответствии с установленной версией Deckhouse Kubernetes Platform.

Выполните подключение к веб-интерфейсу кластера согласно п. 6.1. и п. 6.1.2

6.3.1. Веб-интерфейс системы мониторинга

В качестве веб-интерфейса системы мониторинга используется Grafana.

6.3.1.1. Главный экран

На главном экране Grafana расположена основная информация о кластере и его основных компонентах.

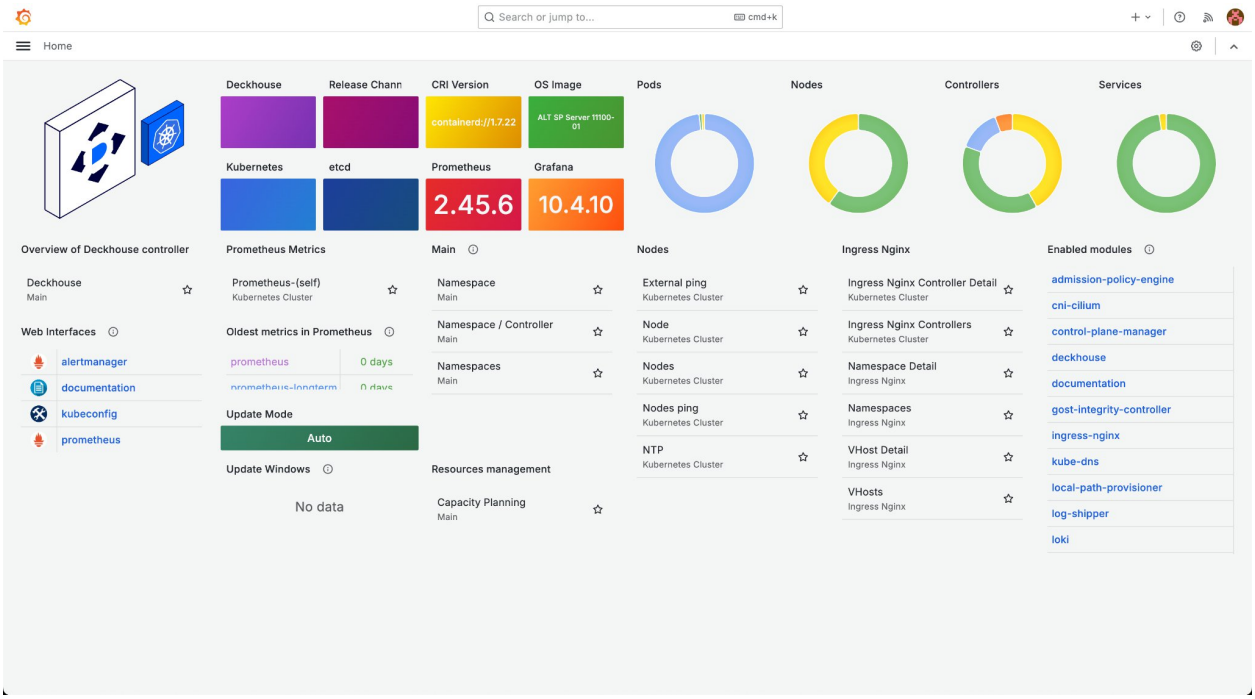


Рисунок 2 –Главный экран.

В левой верхней части экрана указаны характеристики основных компонентов кластера: версия containerd, дистрибутив ОС, на базе которого работает кластер, а также версии Grafana, Prometheus и т.д.

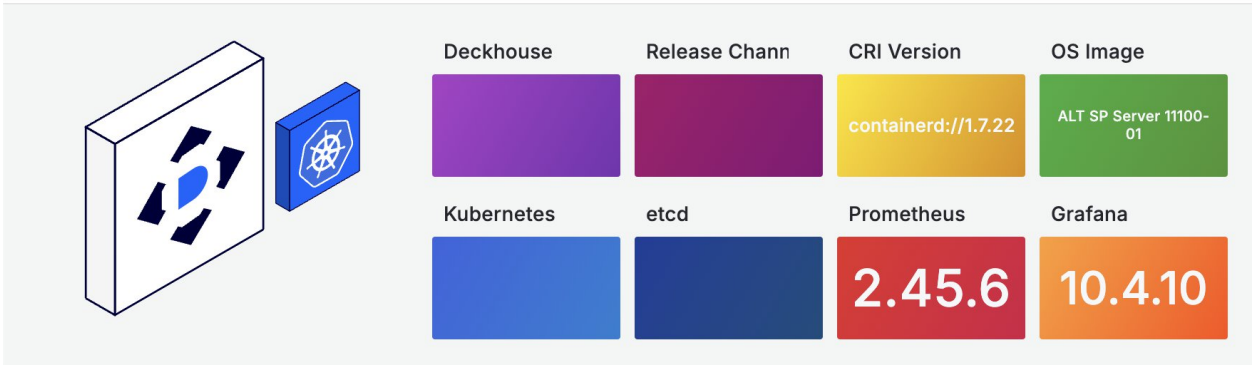


Рисунок 3 – Левая верхняя часть главного экрана.

В правой верхней части экрана расположены удобные графические обозначения для основных параметров — количества узлов кластере, количество запущенных в ней подов и других сущностей кластера.

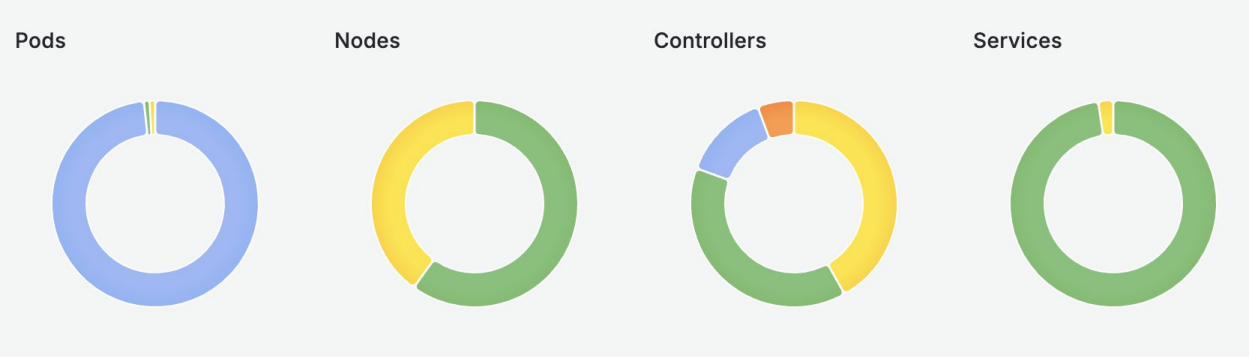


Рисунок 4 – Правая верхняя часть главного экрана.

Для получения более подробной информации можно навести на любой элемент курсор мыши, нужная информация отображается во всплывающей подсказке:

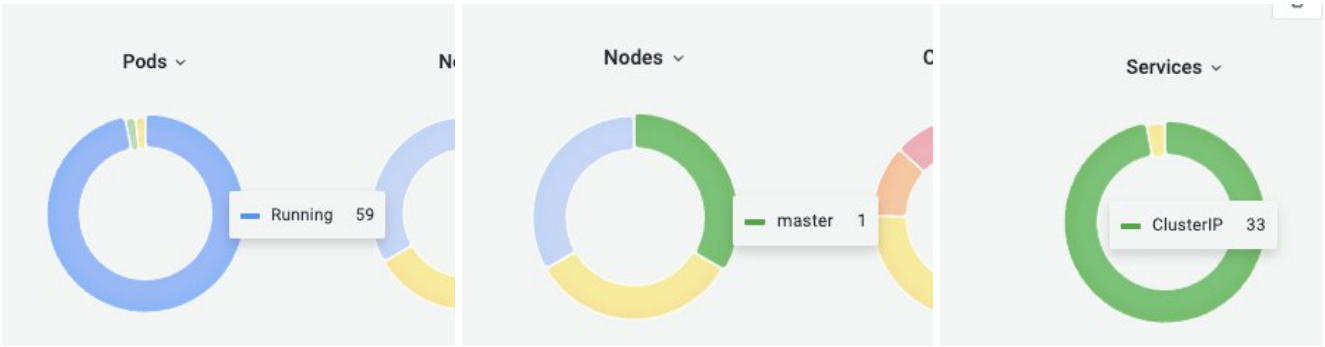


Рисунок 5 – Всплывающие подсказки.

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.

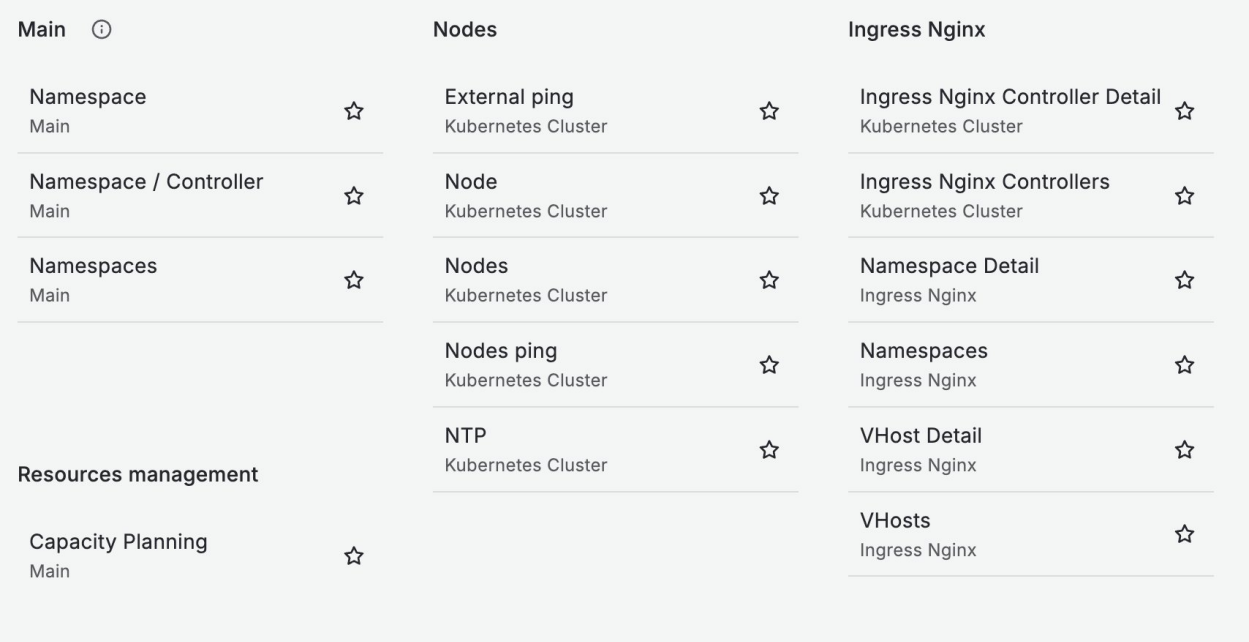


Рисунок 6 – Ссылки на дашборды

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).

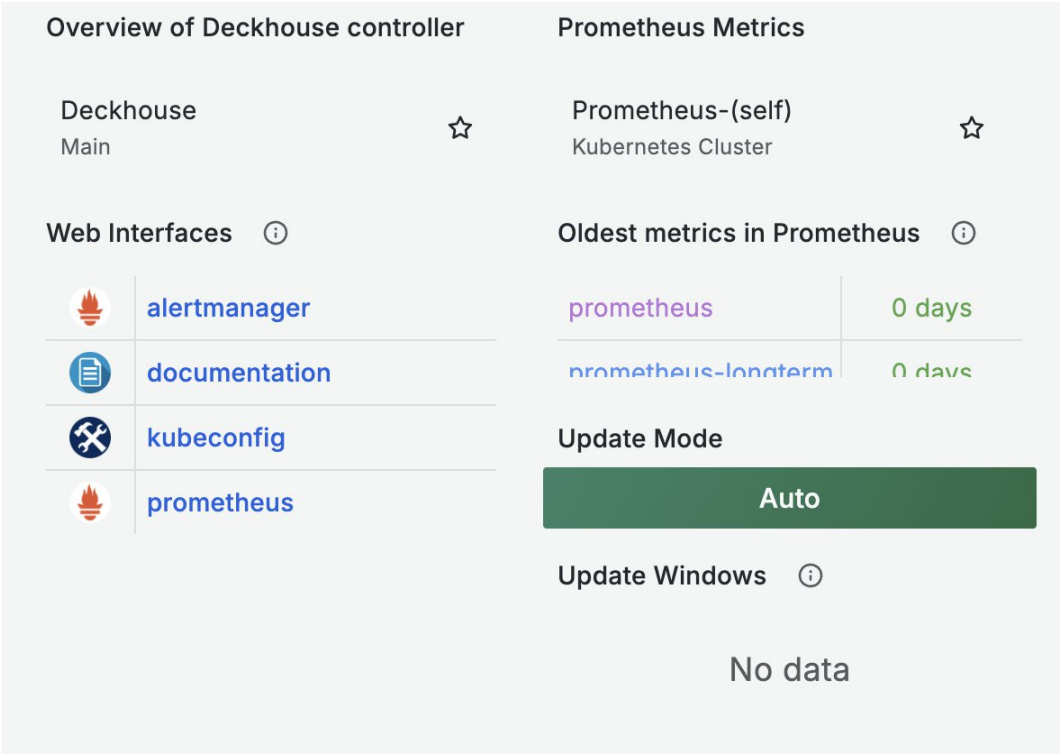


Рисунок 7 – Ссылки на веб-интерфейсы и информация о способе обновления кластера

В левом верхнем углу главного экрана расположена кнопка открытия бокового меню, в котором расположены ссылки на основные элементы Grafana.

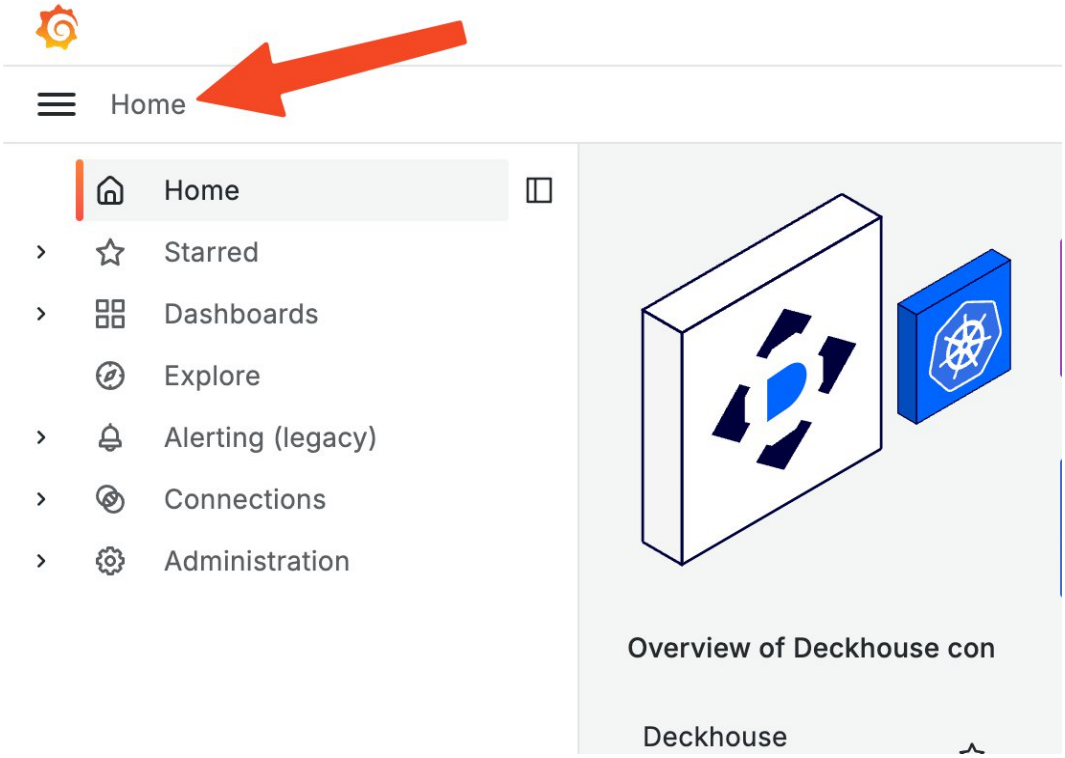


Рисунок 8 – Боковое меню

При переходе на вкладку «Dashboards» откроется список всех доступных дашбордов Deckhouse, сгруппированных по категориям.

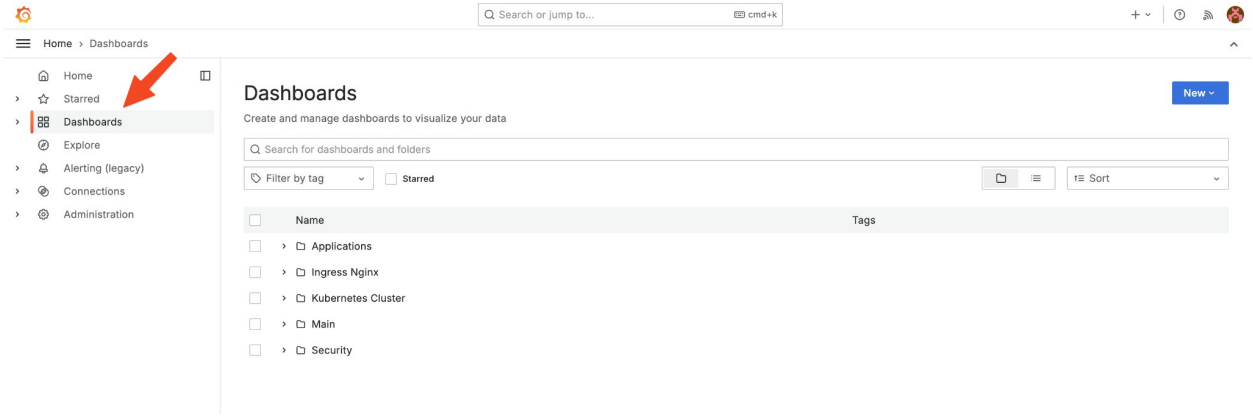


Рисунок 9 – Список доступных дашбордов

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).

6.3.1.2. Работа с дашбордами

Дашборд представляет собой экран с расположенными на нем таблицами и графиками, содержащими информацию о выбранном компоненте кластера.

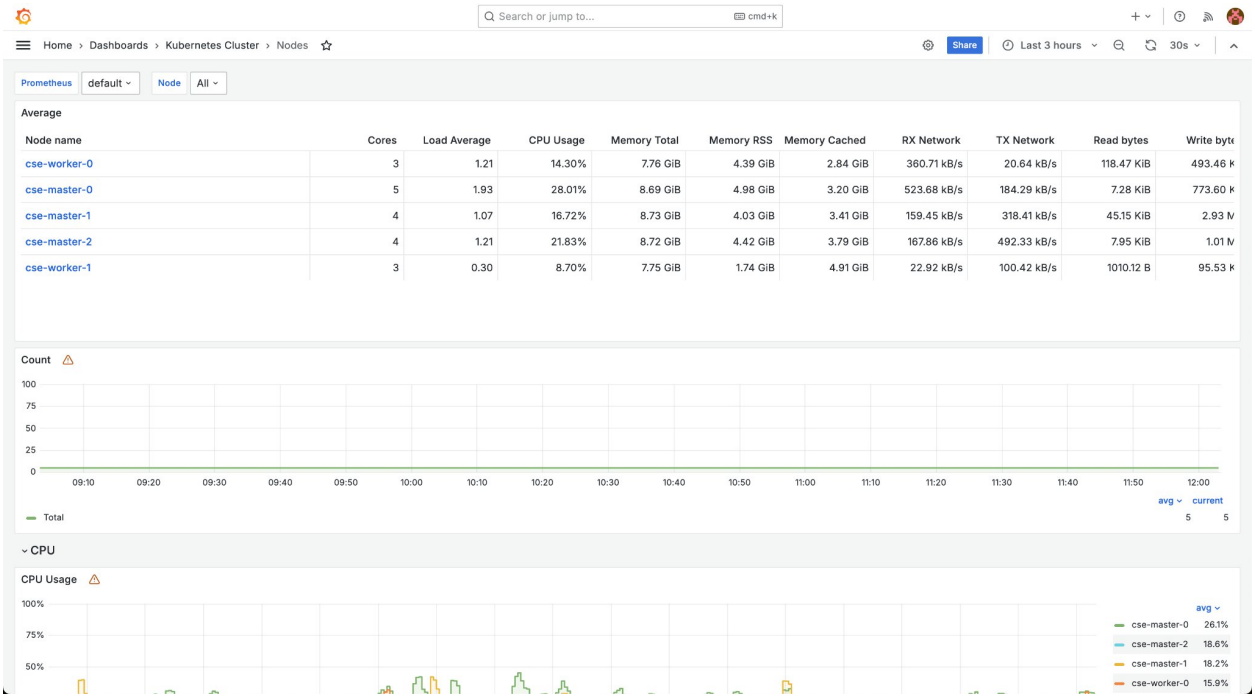


Рисунок 10 – Дашборд

6.3.1.3. 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.

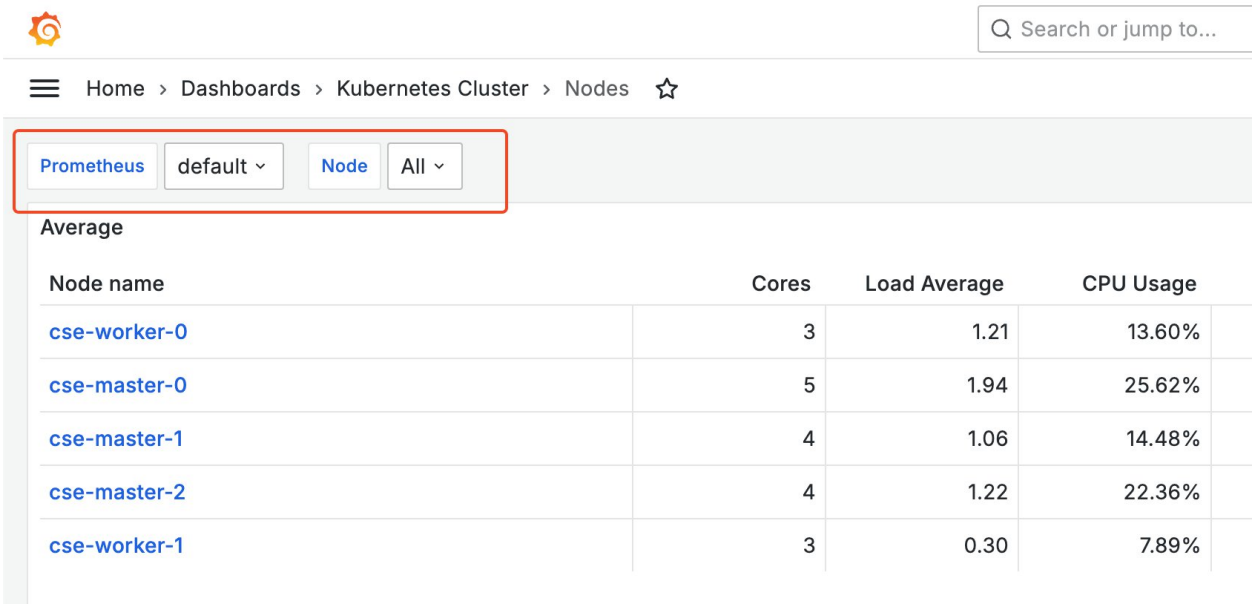


Рисунок 11 – Блок фильтров

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.

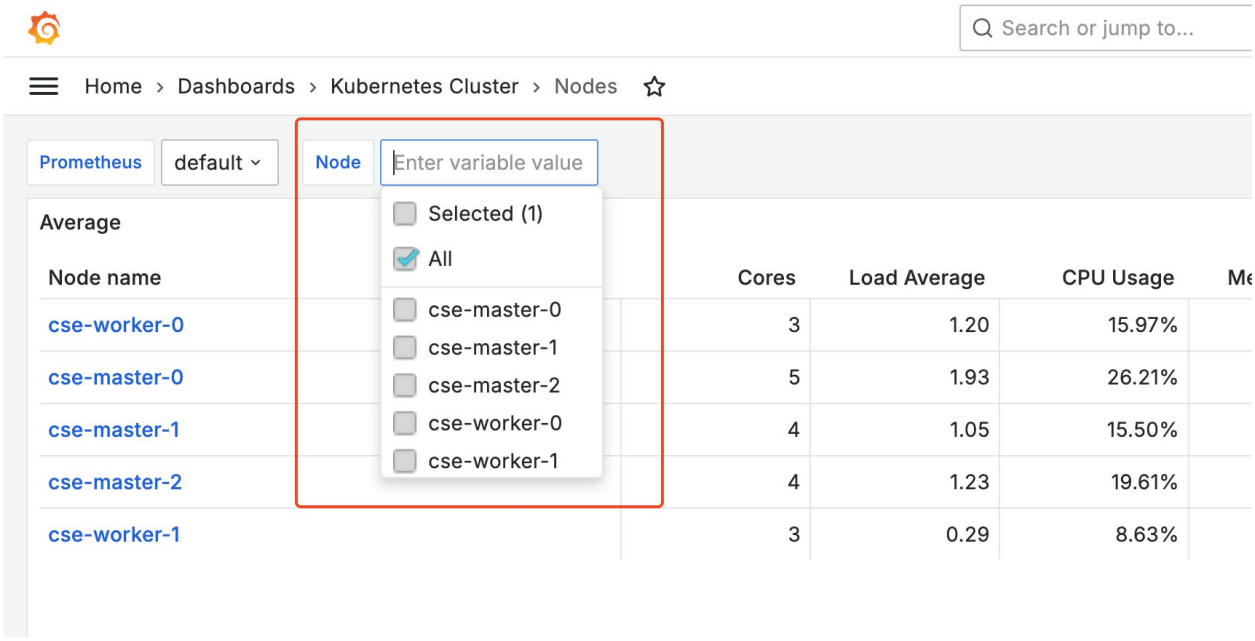


Рисунок 12 – Применение фильтров

После выбора в фильтре параметров дашборд сразу же изменится, и содержимое будет заменено на соответствующее заданным параметрам фильтрации.

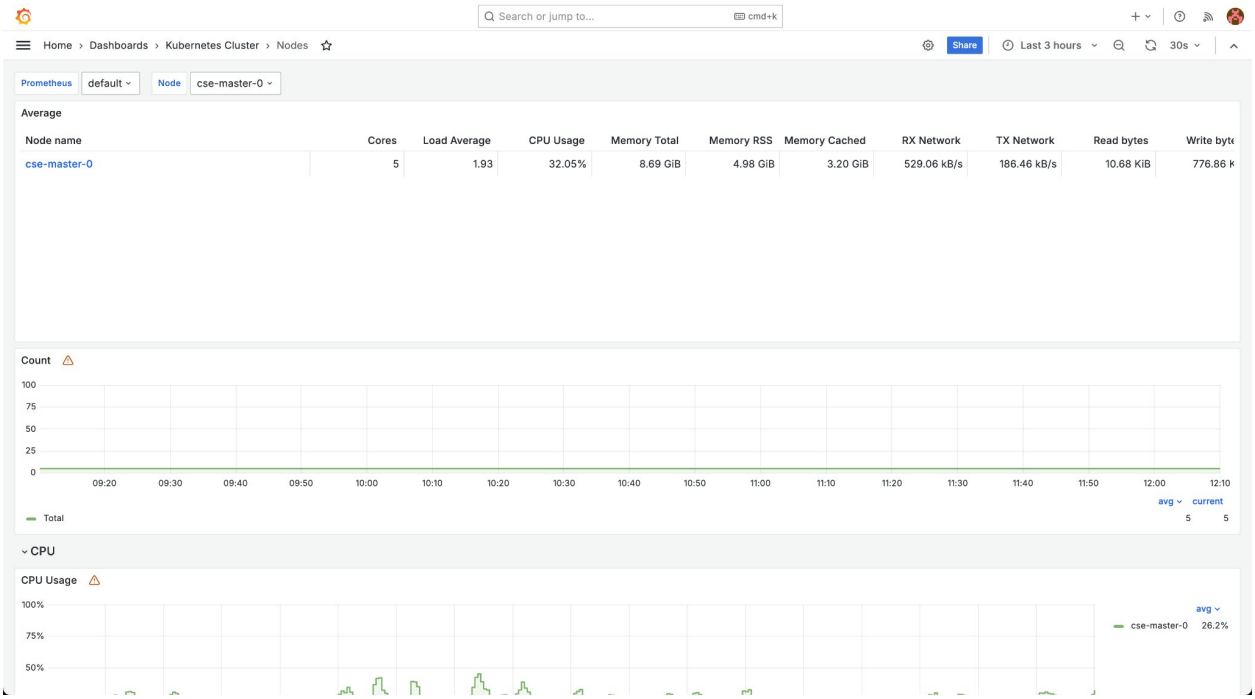


Рисунок 13 – Отображение информации после применения фильтров

6.3.1.4. 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».

Рисунок 14 – Вывод графика

Выбранный график откроется на весь экран.

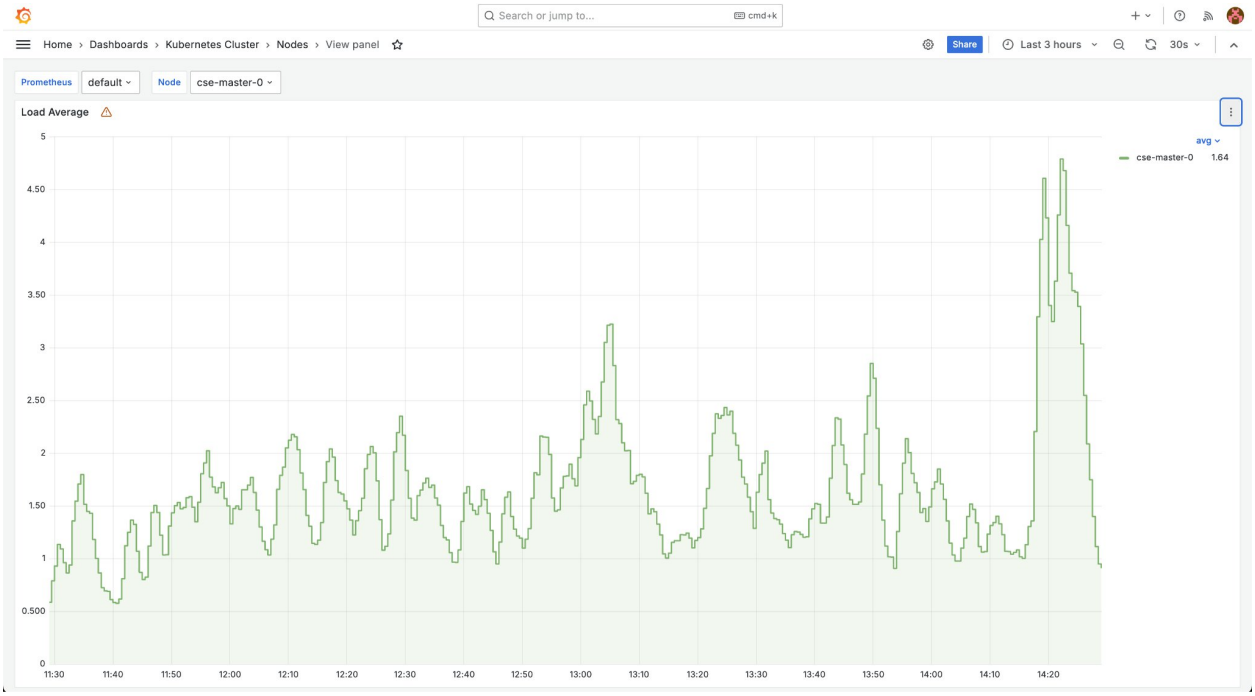


Рисунок 15 – Пример графика

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 временной меткой и значением графика на этот момент:

Рисунок 16 – График за определенный момент времени

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».

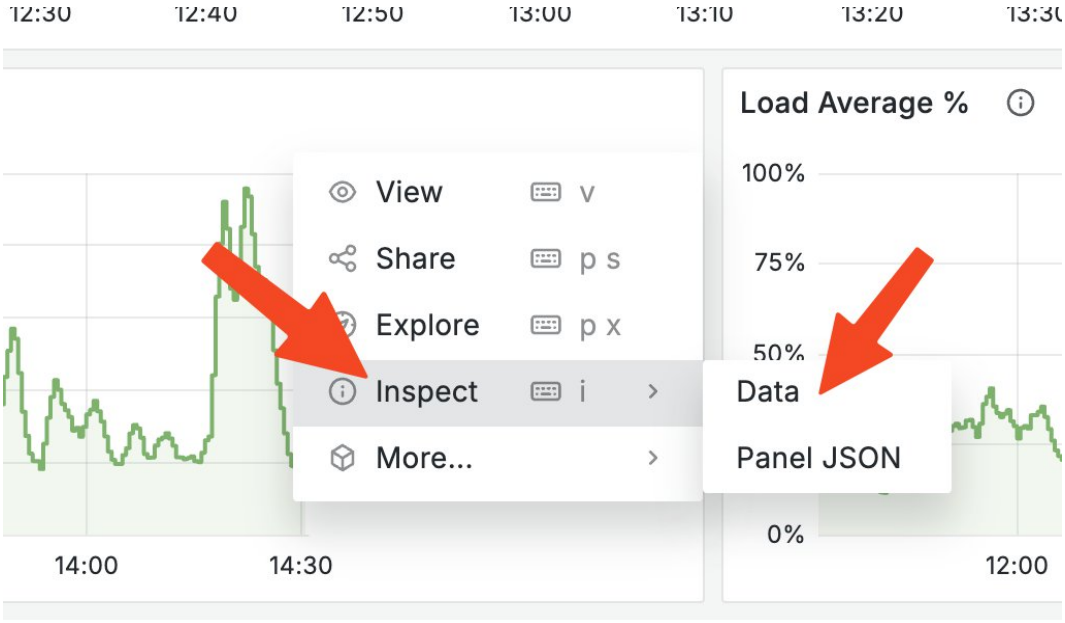


Рисунок 17 – Настройка просмотра списка записей из которых строится график

В правой части экрана откроется окно с подробным содержанием записей.

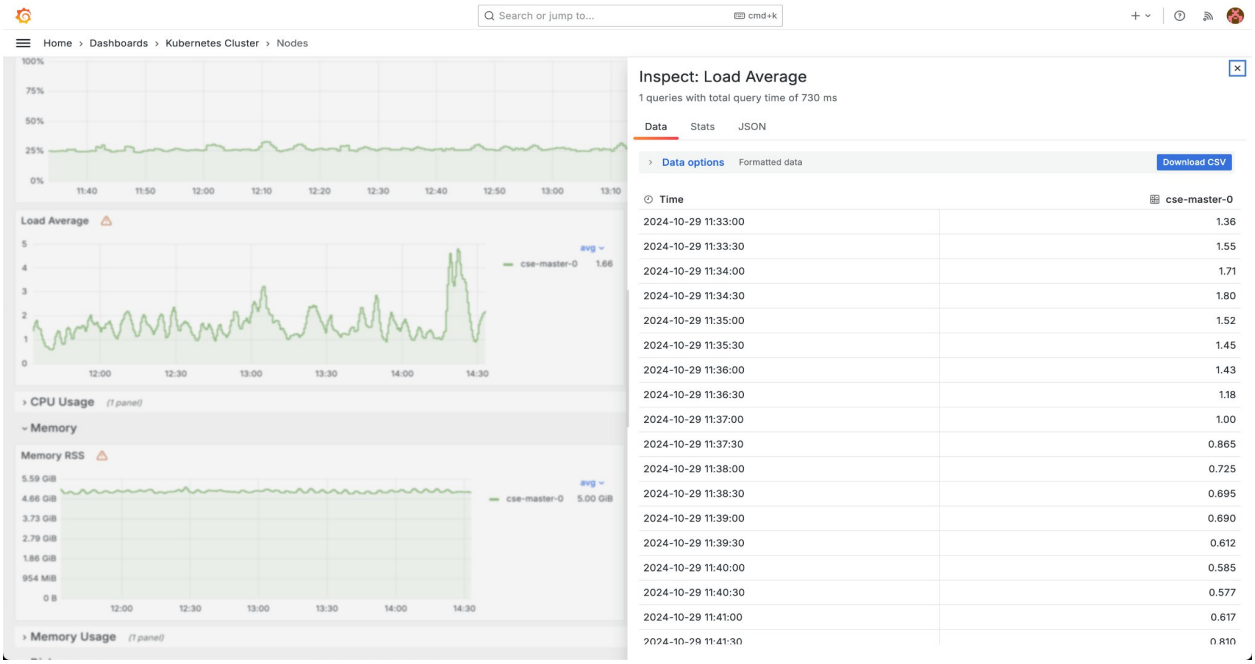


Рисунок 18 – Содержание записей из которых строится график

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.

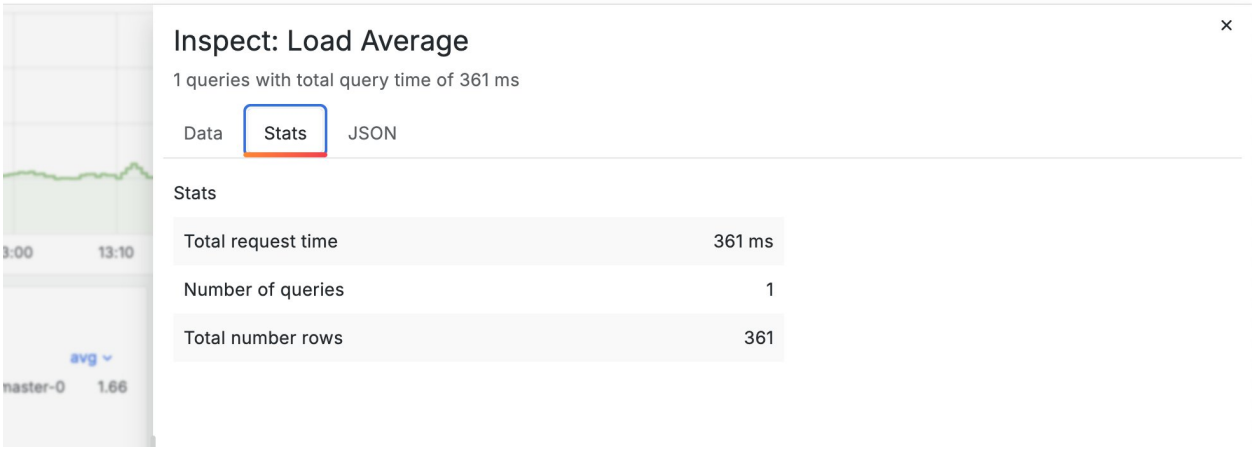


Рисунок 19 – Скачивание данных графика

6.3.1.5. Описание дашбордов

6.3.1.5.1. Дашборд «Applications – Log Shipper»

Состояние модуля log-shipper.

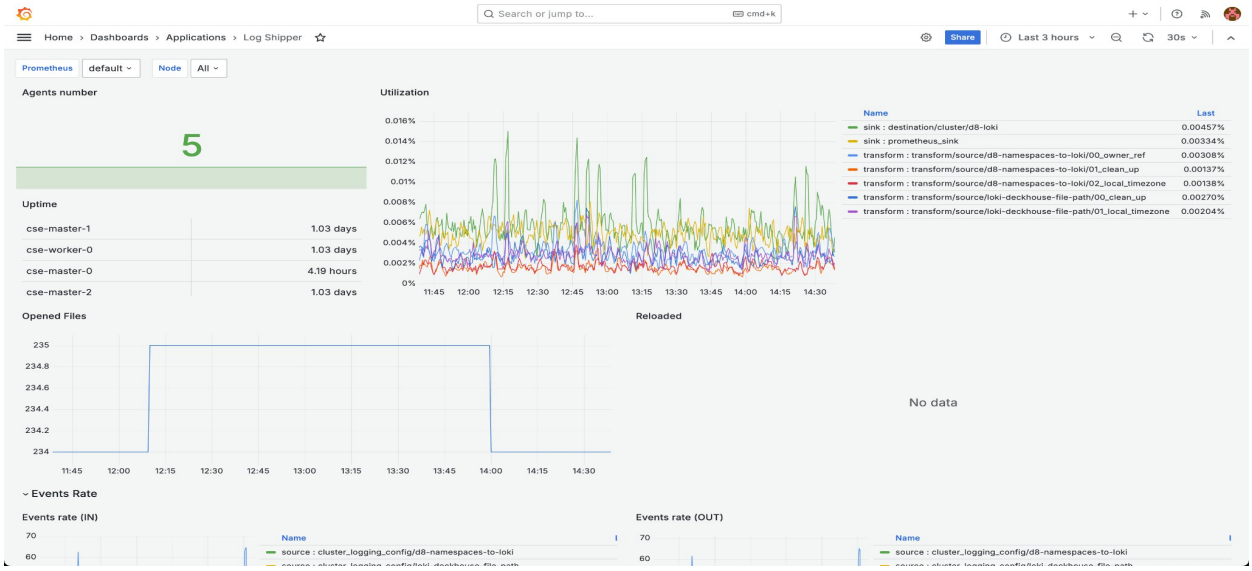


Рисунок 20 – Модуль log-shipper

Здесь представлено количество агентов модуля на узлах и их нагрузка.

6.3.1.5.2. Дашборд «Applications – Loki»

Состояние модуля loki.

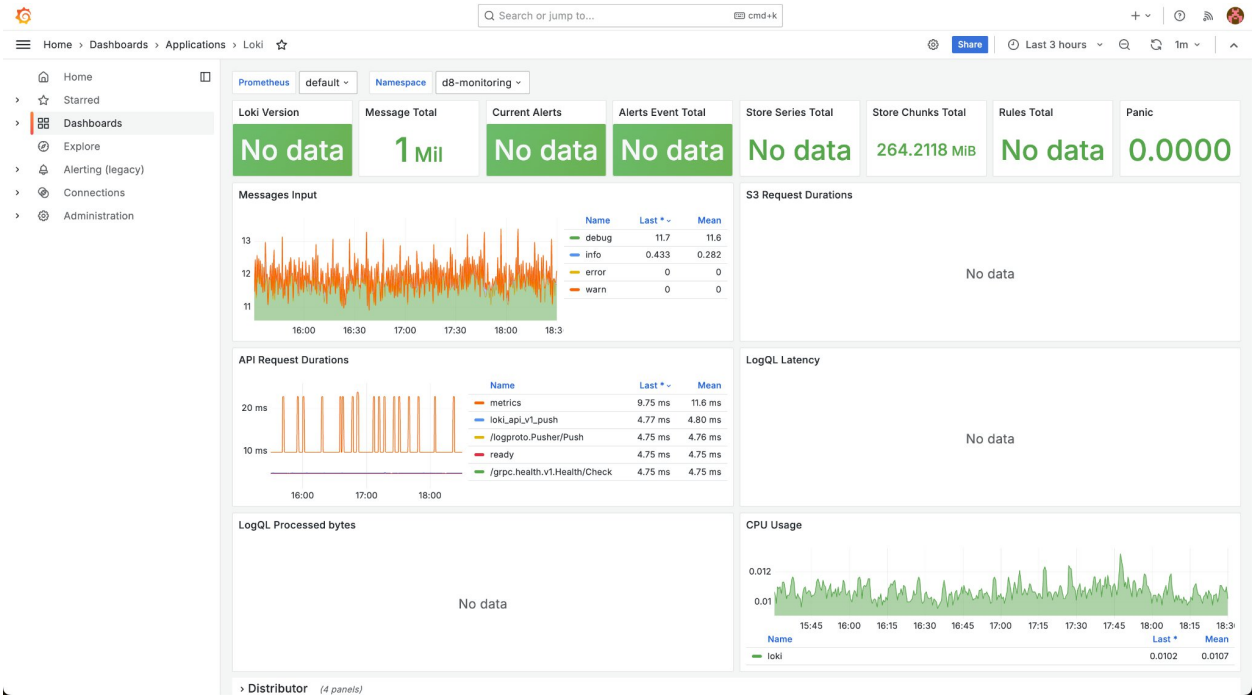


Рисунок 21 – Модуль loki

6.3.1.5.3. Дашборд «Applications – Loki Logs»

Логи модуля loki.

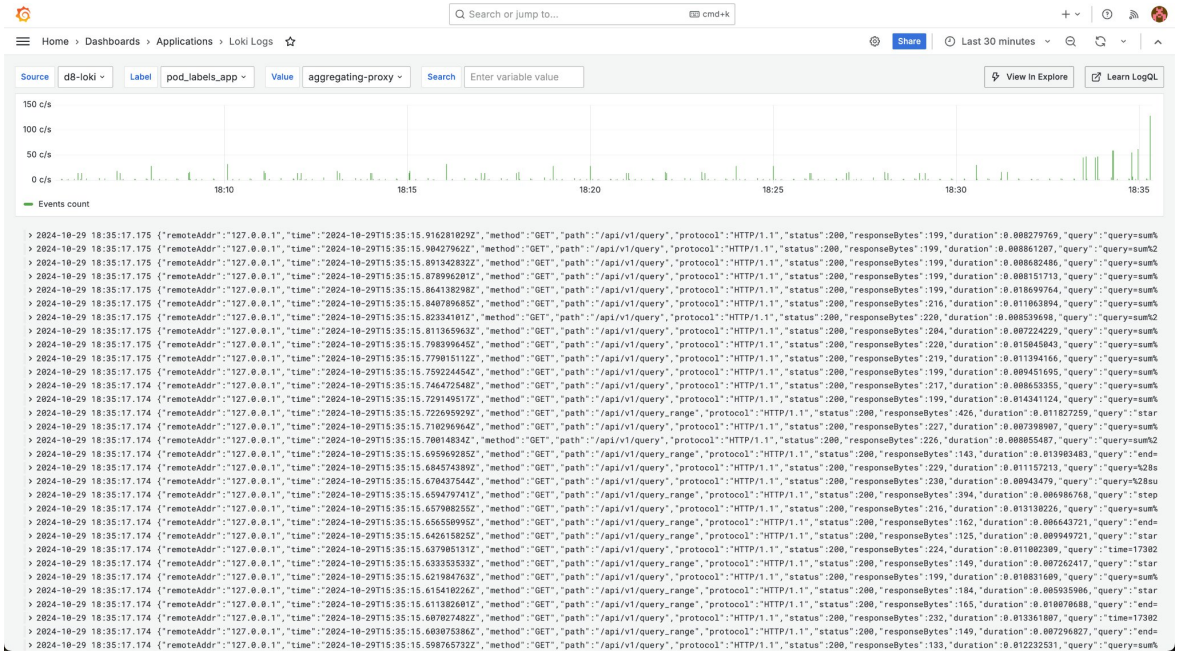


Рисунок 22 – Логи модуля loki

6.3.1.5.4. Дашборды группы Ingress Nginx

Дашборды, связанные с Ingress-контроллерами.

6.3.1.5.4.1. Дашборд «Namespace Detail»

На этом дашборде отображается детализация компонентов в пространстве имен.

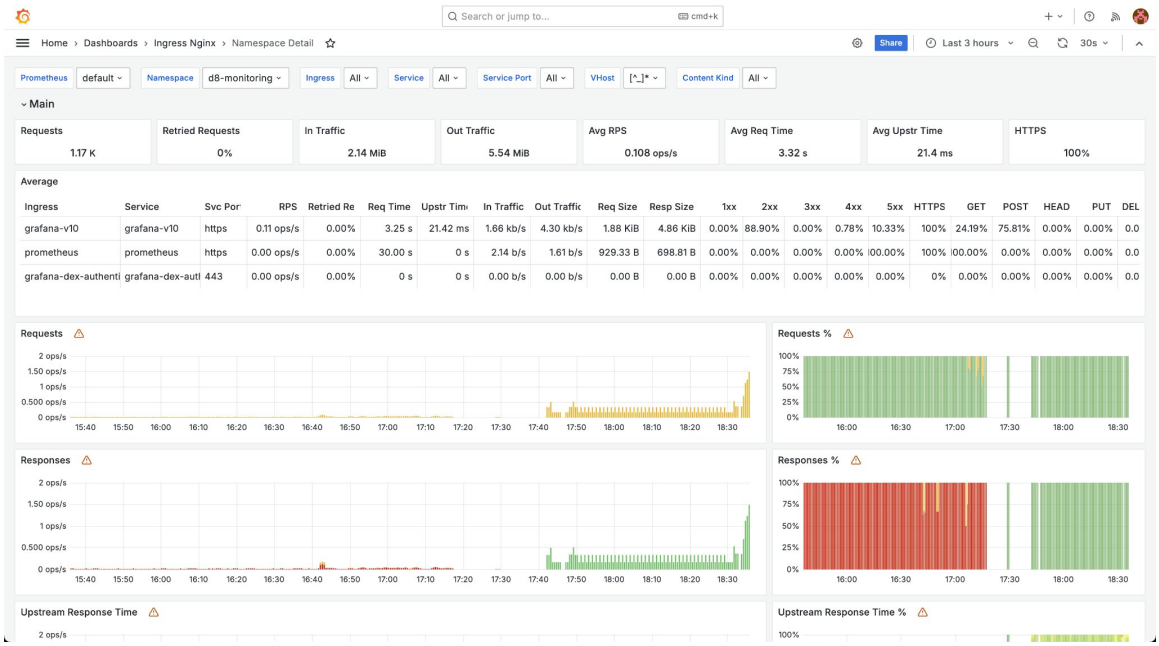


Рисунок 23 – Дашборд «Namespace Detail»

Детализация компонентов в пространстве имен. В фильтрах возможно выбрать конкретное пространство имен, Ingress, Service и другие параметры для отображения.

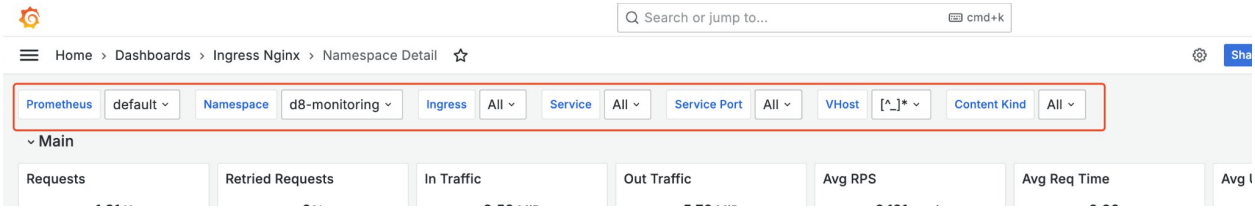


Рисунок 24 – Детализация компонентов в пространстве имен.

6.3.1.5.4.2. Дашборд «Namespaces»

Данные по Ingress-контроллеру в разрезе пространств имен кластера.

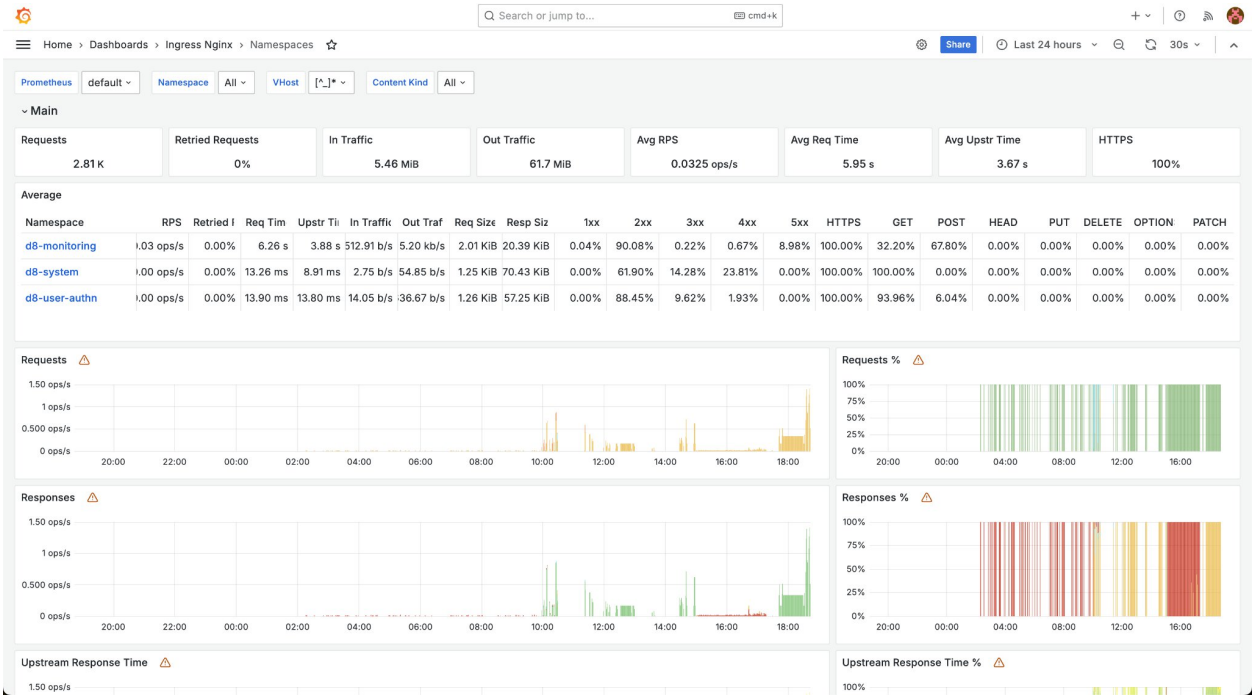


Рисунок 25 – Дашборд «Namespaces»

В фильтрах можно выбрать конкретное пространство имен, виртуальные хосты и тип контента.

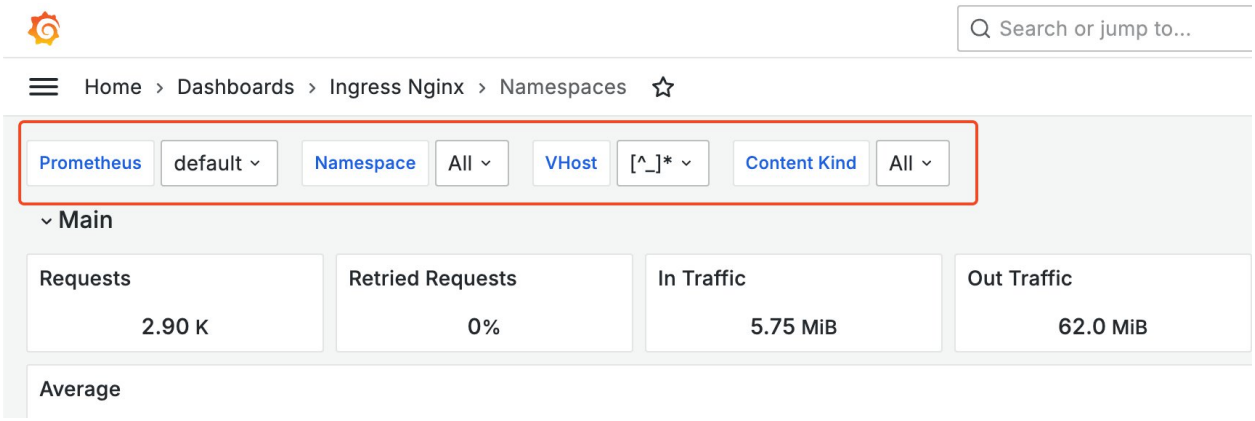


Рисунок 26 – Фильтры для дашборда «Namespaces»

6.3.1.5.4.3. Дашборд «VHost Detail»

Подробные данные по Ingress-контроллеру в разрезе виртуальных хостов.

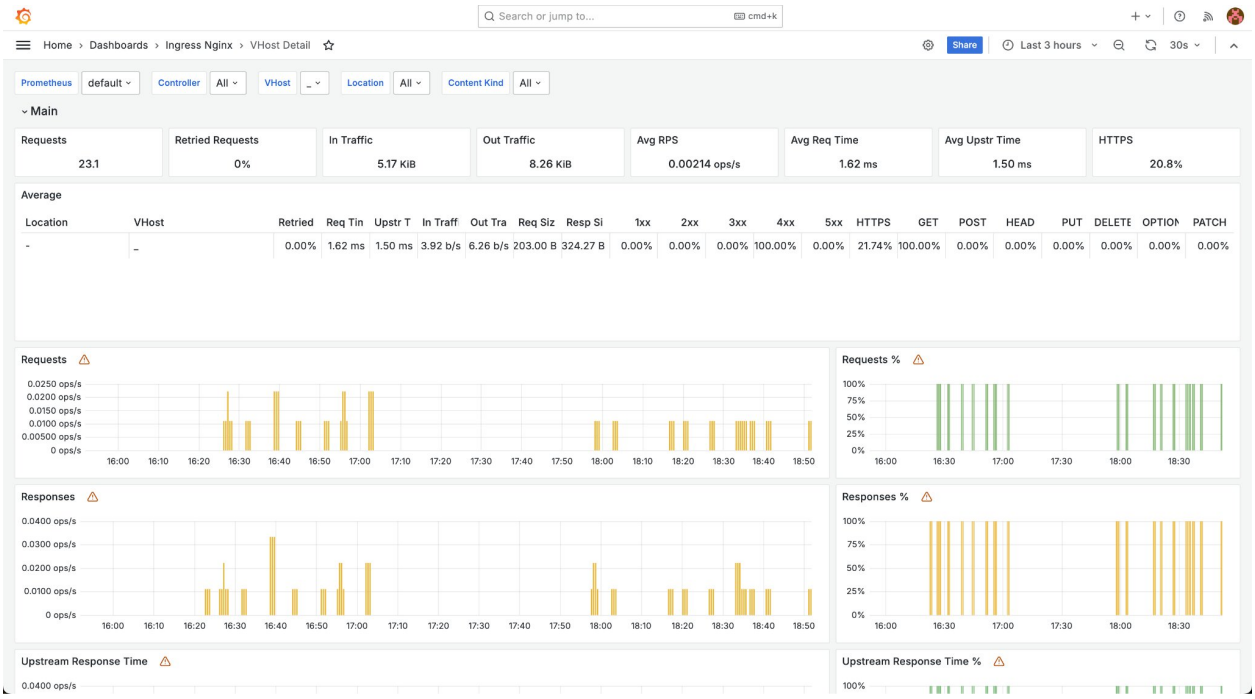


Рисунок 27 – Дашборд «VHost Detail»

6.3.1.5.4.4. Дашборд «VHost»

Сводные данные по Ingress-контроллеру в разрезе виртуальных хостов кластера.

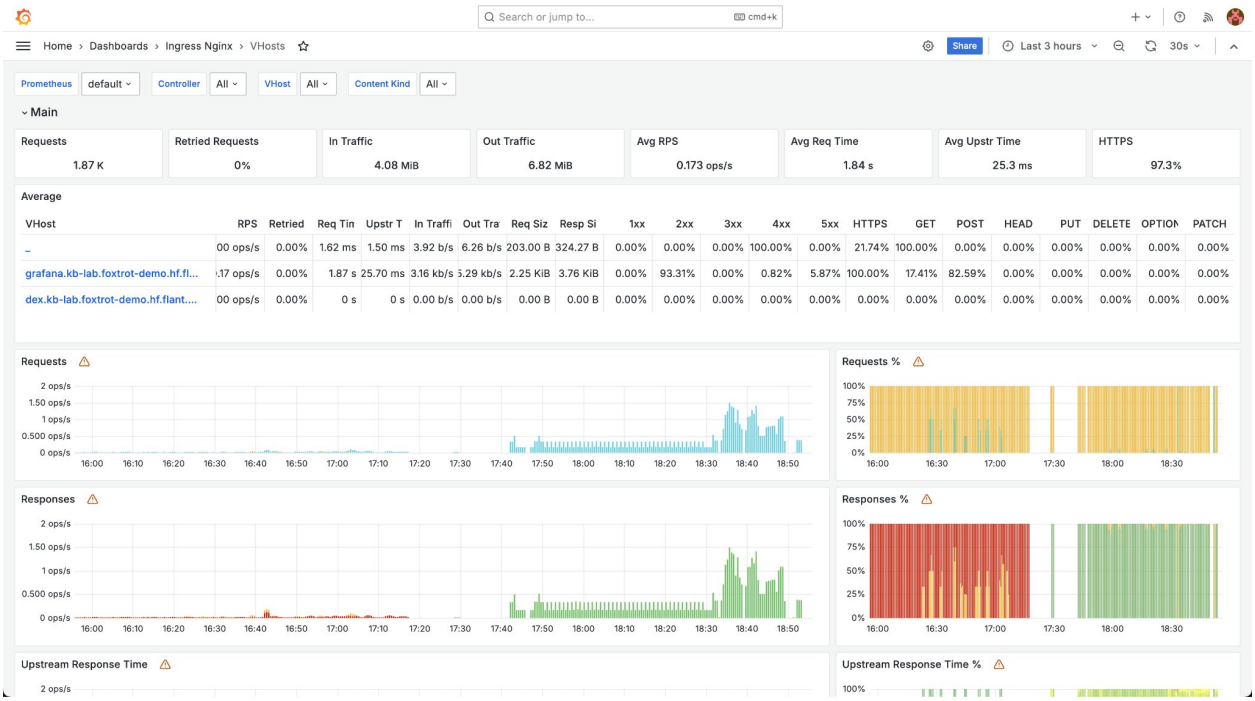


Рисунок 28 – Дашборд «VHost»

В фильтрации можно выбрать конкретный виртуальный хост.

6.3.1.5.5. Дашборды группы «Kubernetes Cluster»

Дашборды, связанные с кластером Kubernetes.

6.3.1.5.5.1. Дашборд «Aggregating Proxy Cache»

Сводная информация по потребляемым прокси-сервером ресурсам.

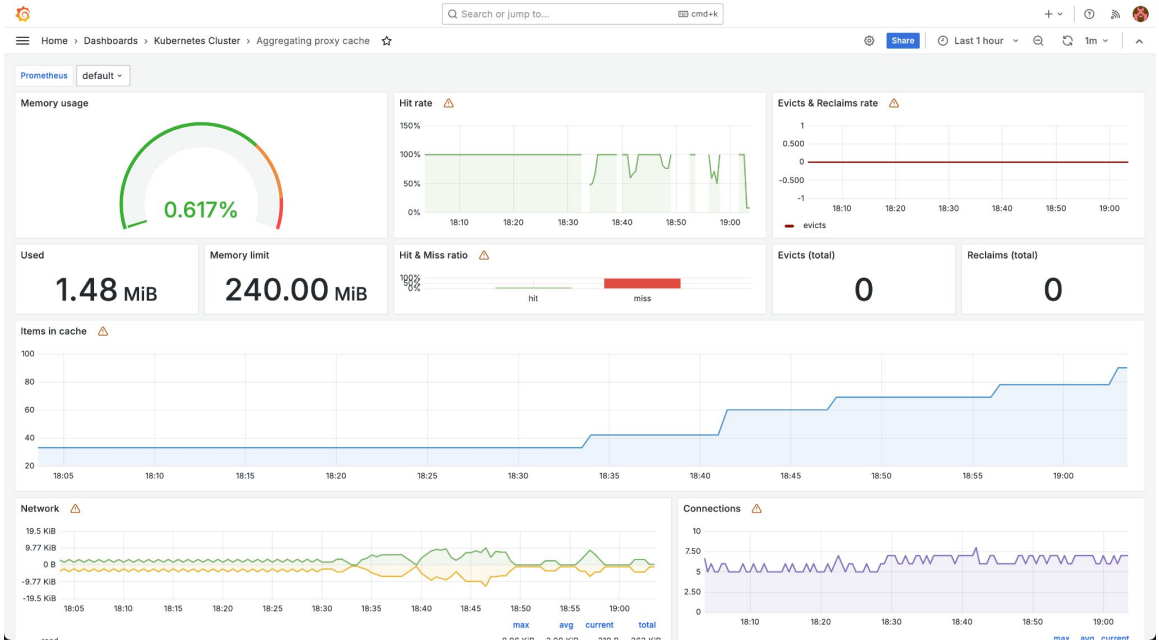


Рисунок 29 – Дашборд «aggregating-proxy cache»

6.3.1.5.5.2. Дашборд «Cilium Metrics»

Метрики модуля cni-cilium.

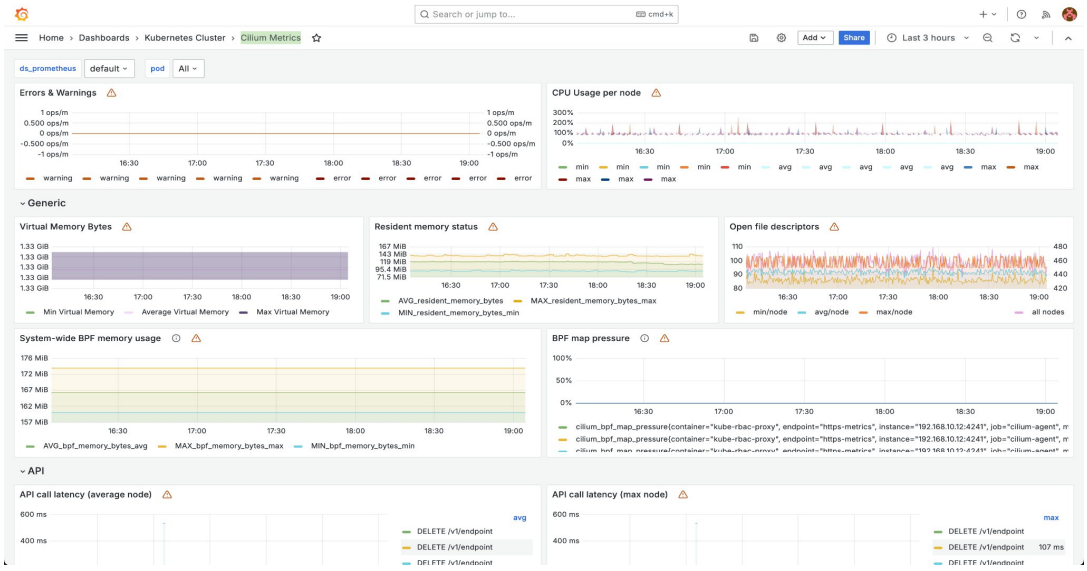


Рисунок 30 – Дашборд «Cilium Metrics»

6.3.1.5.5.3. Дашборд «Control Plane Status»

Состояние управляющего слоя кластера.

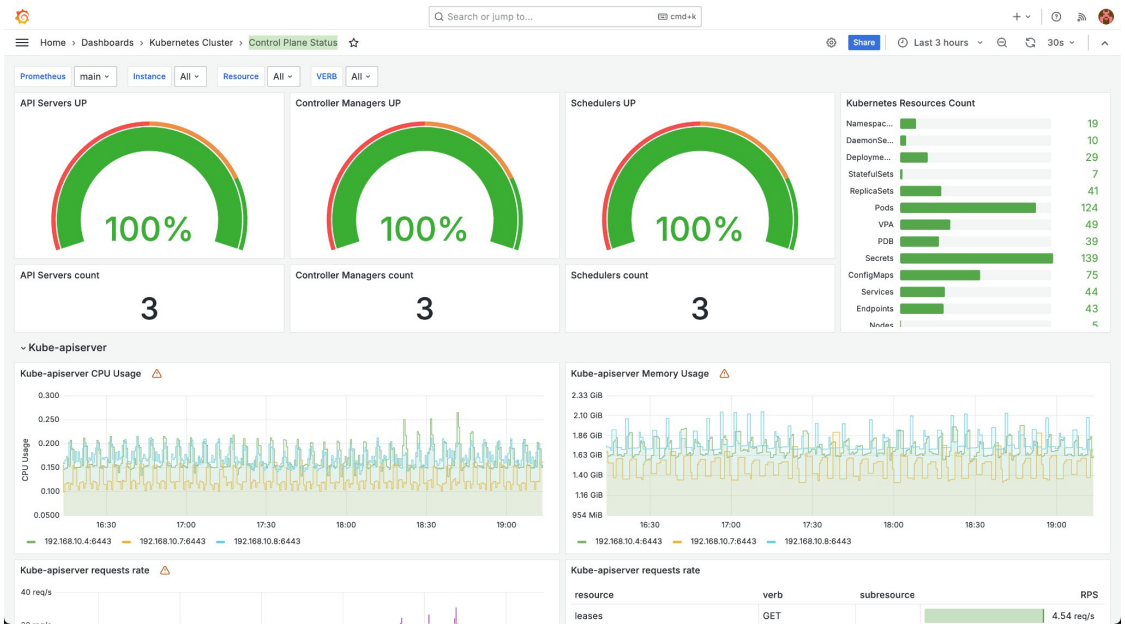


Рисунок 31 – Дашборд «Control Plane Status»

6.3.1.5.5.4. 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.

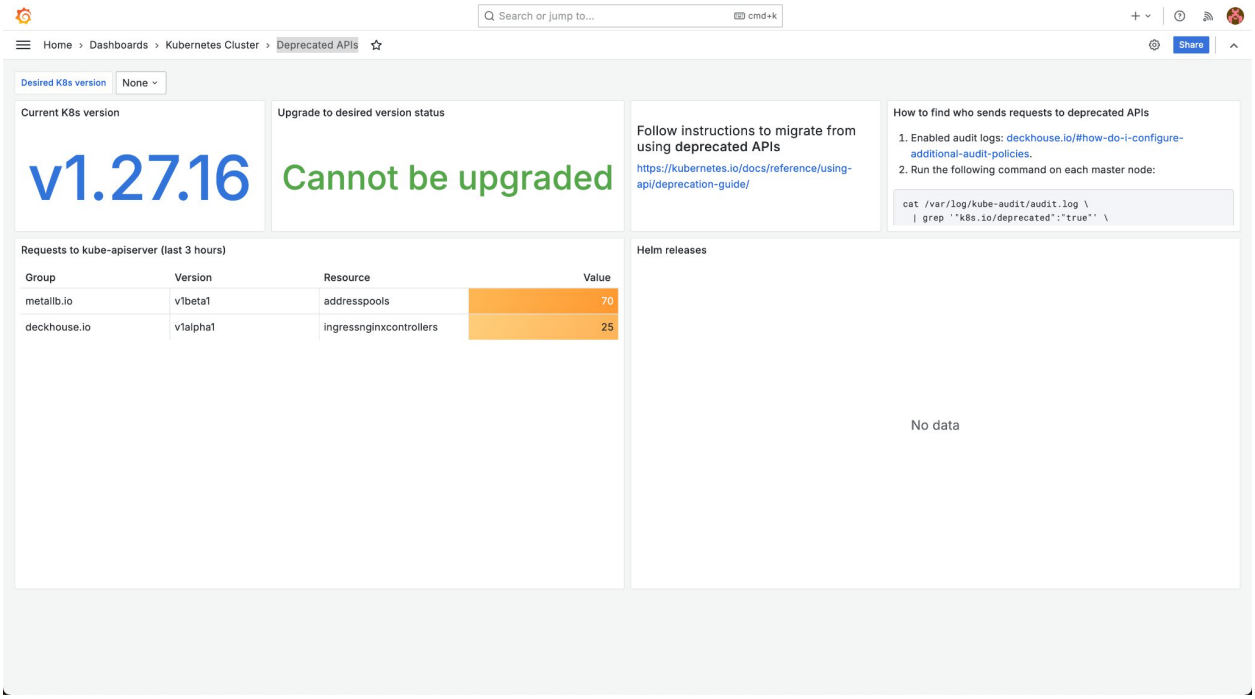


Рисунок 32 – Дашборд «Deprecated APIs»

6.3.1.5.5.5. Дашборд «DNS (coredns)»

Данные о работе компонента coredns.

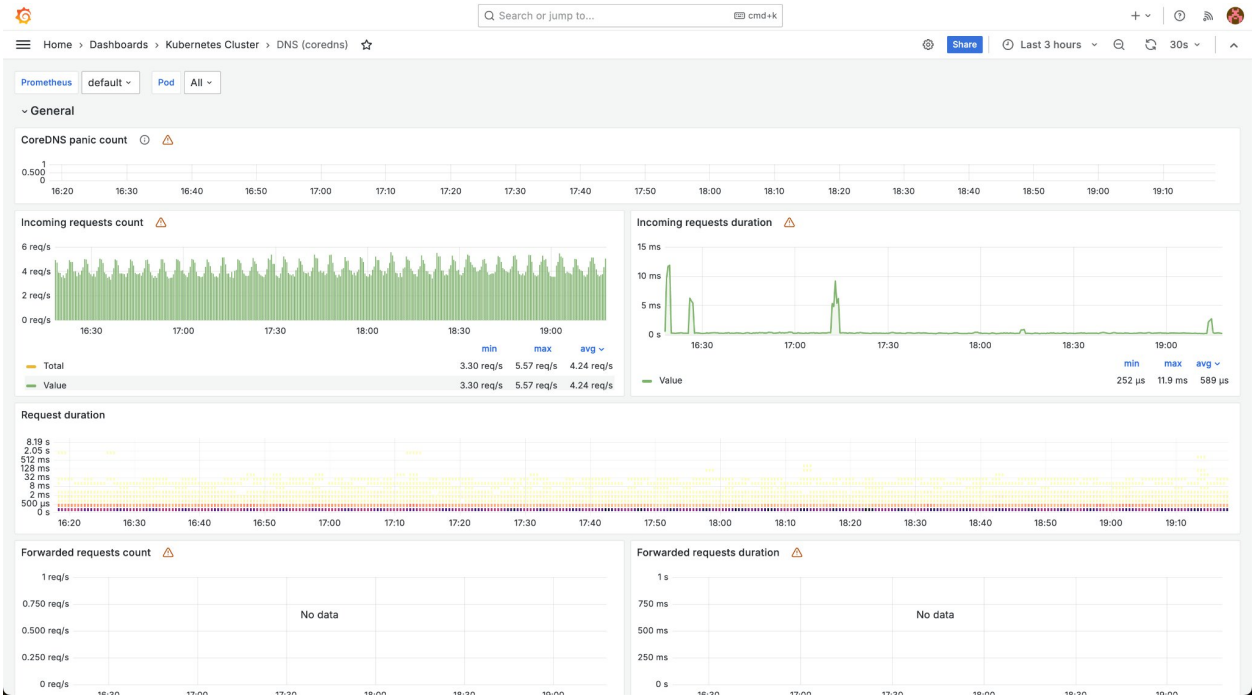


Рисунок 33 – Дашборд «DNS (coredns)»

6.3.1.5.5.6. Дашборд «etcd3»

Состояние базы данных etcd.

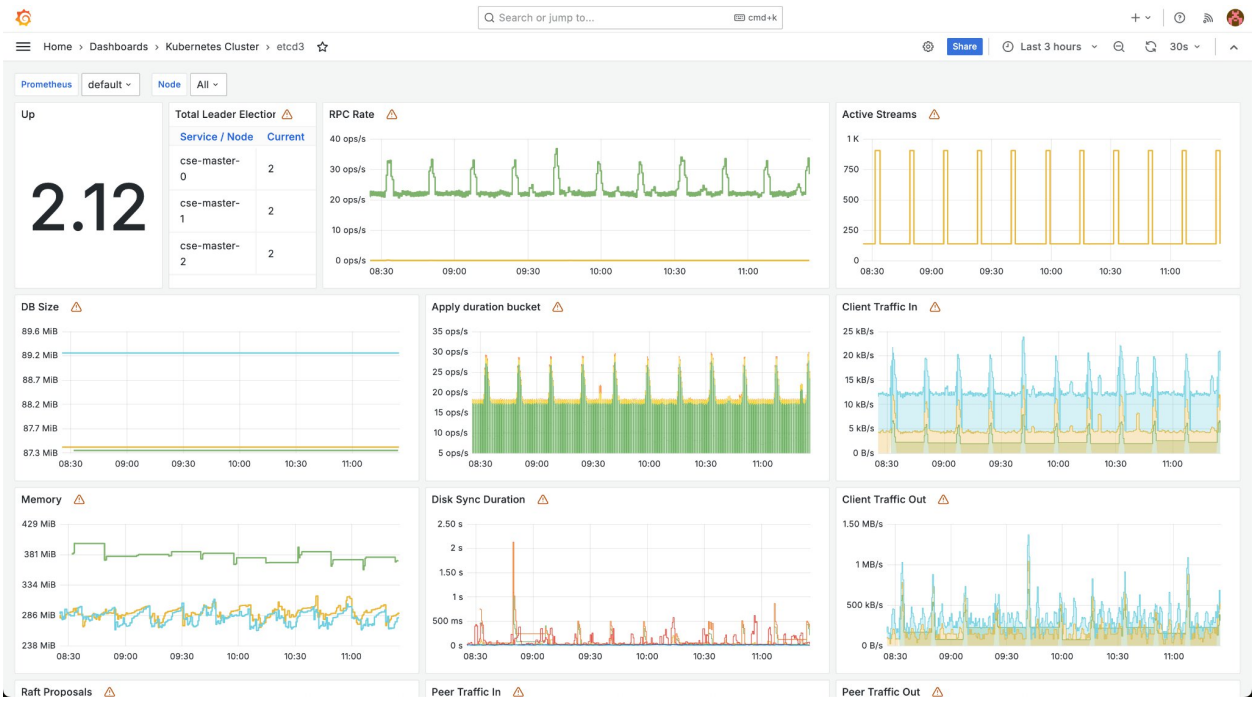


Рисунок 34 – Дашборд «etcd3»

6.3.1.5.5.7. Дашборд «External ping»

Статистика внешних запросов.

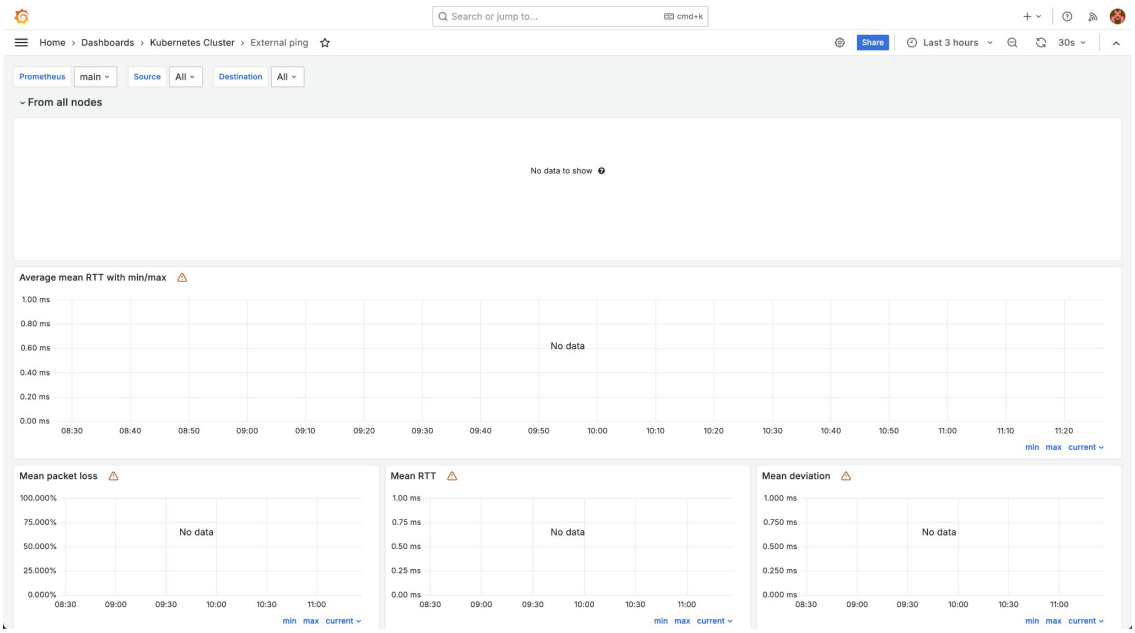


Рисунок 35 – Дашборд «External ping»

6.3.1.5.5.8. Дашборд «Ingress Nginx Controller Detail»

Параметры Ingress Nginx контроллера.

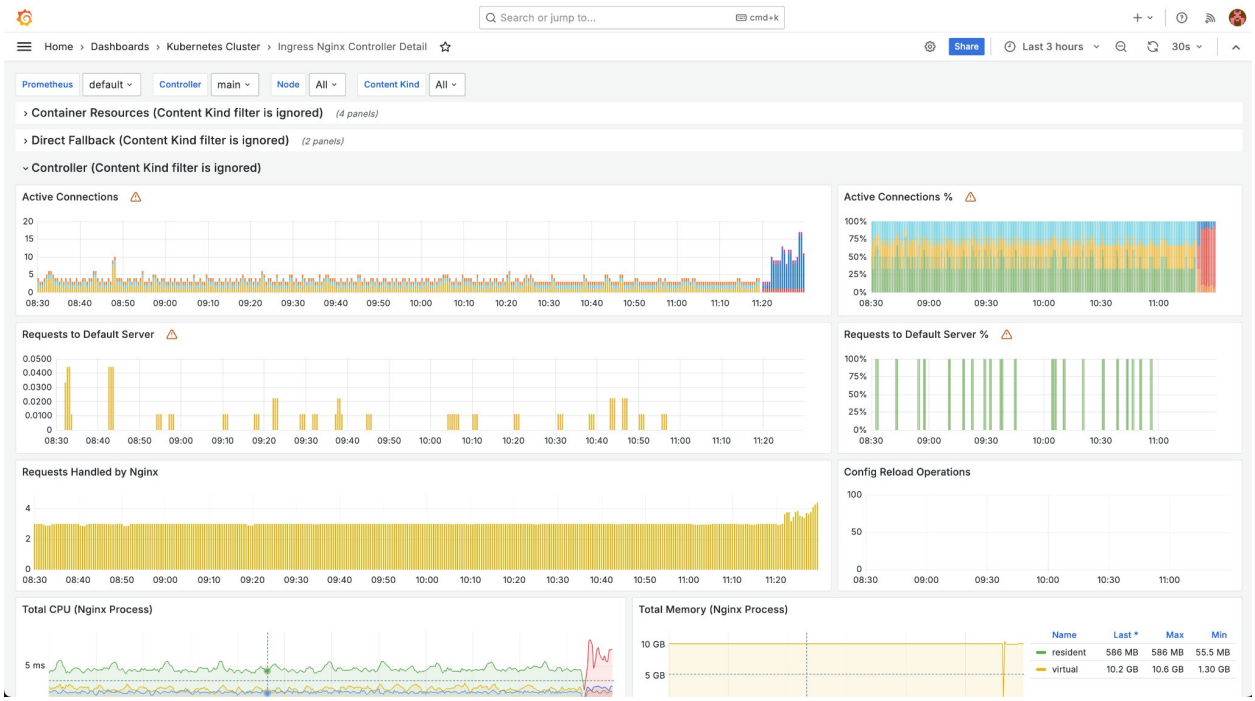


Рисунок 36 – Дашборд «Ingress Nginx Controller Detail»

6.3.1.5.5.9. Дашборд «Ingress Nginx Controllers»

Подробные данные Ingress-контроллеры кластера.

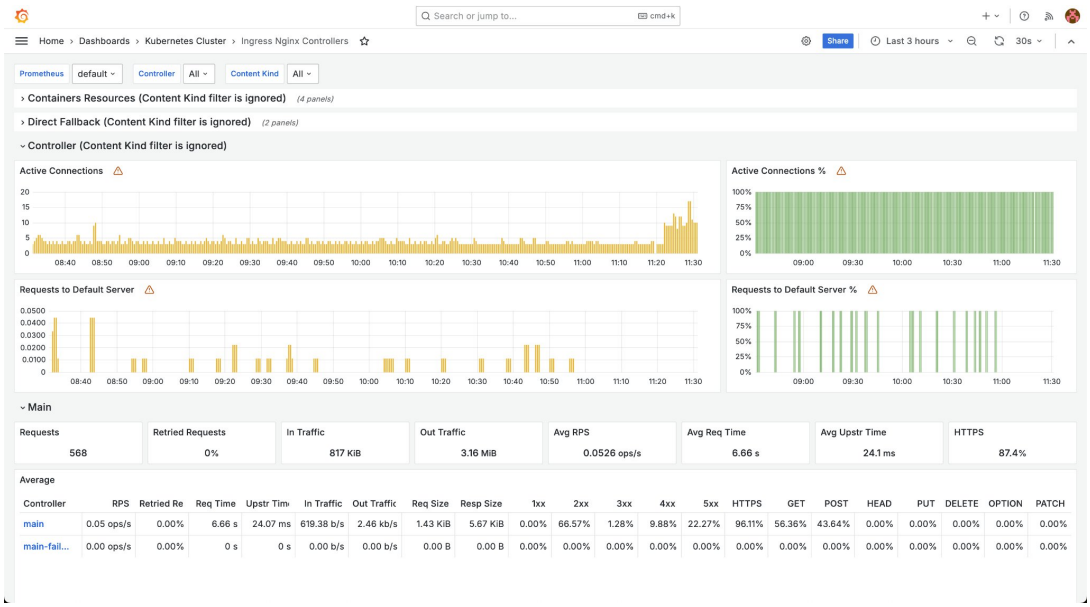


Рисунок 37 – Дашборд «Ingress Nginx Controllers»

6.3.1.5.5.10. Дашборд «Node»

Данные о работе узлов.

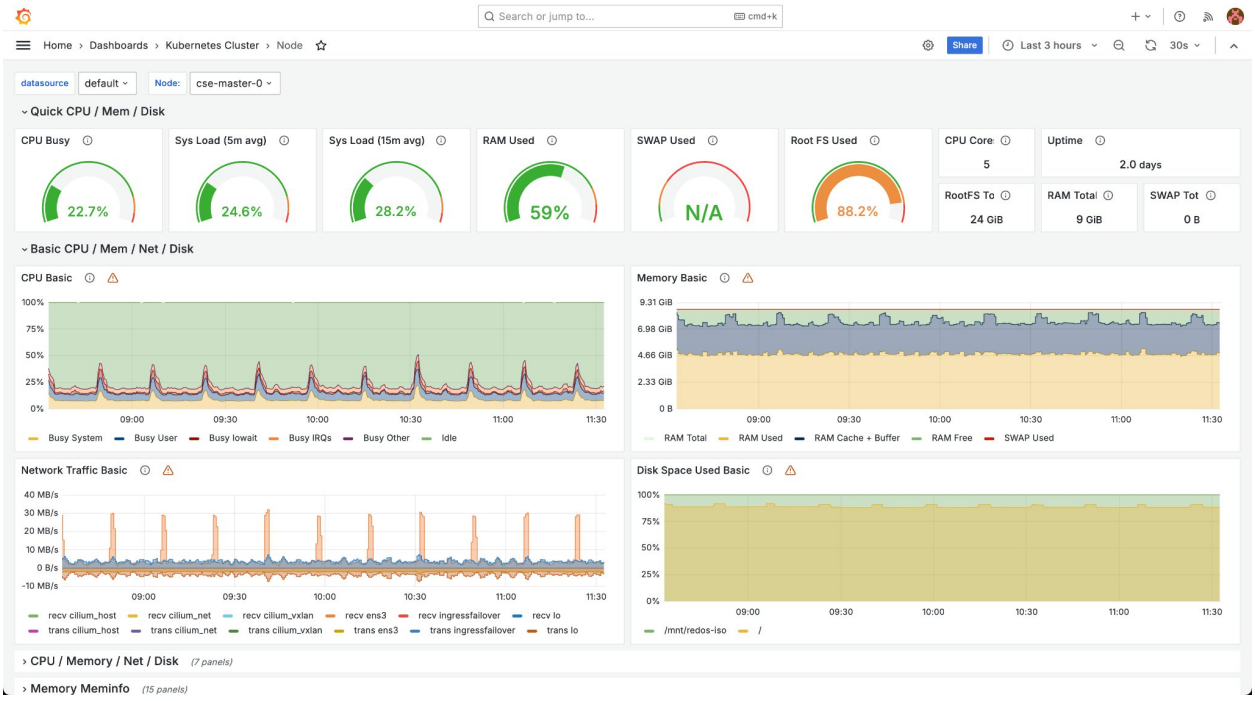


Рисунок 38 – Дашборд «Node»

В фильтрах можно выбрать целевой узел для отображения статистики.

6.3.1.5.5.11. Дашборд «Nodes»

Сводные данные о работе узлов кластера.

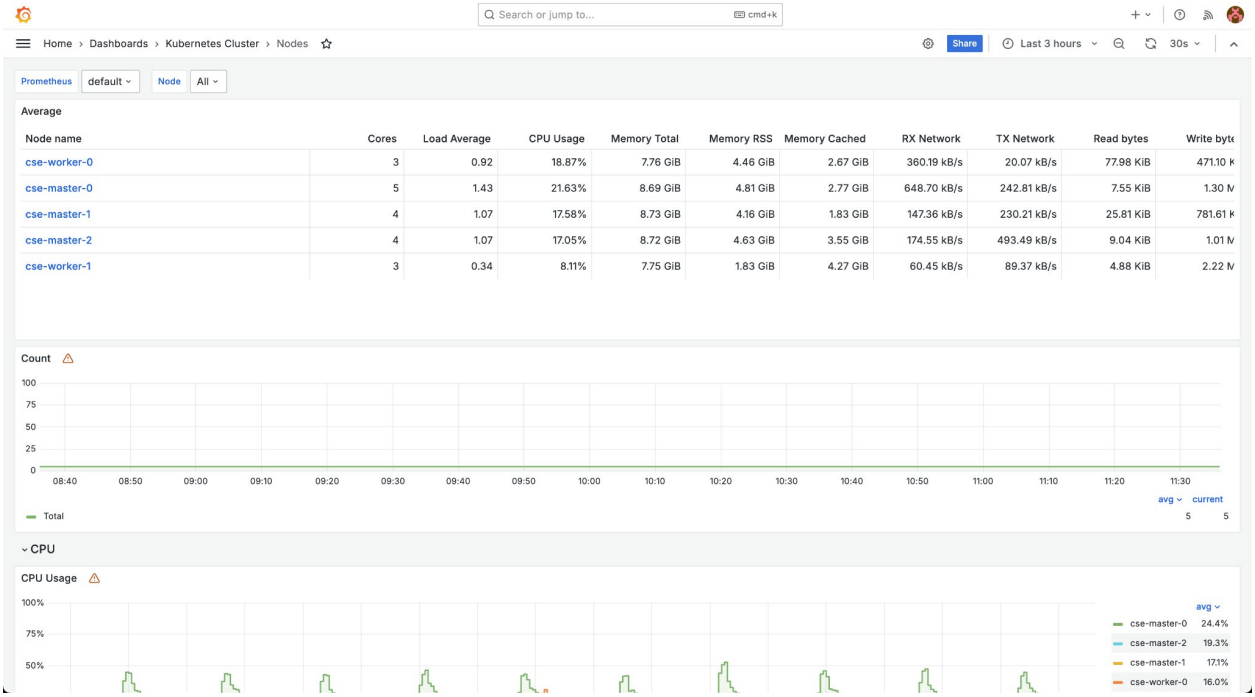


Рисунок 39 – Дашборд «Nodes»

В фильтрах можно выбрать конкретный узел.

6.3.1.5.5.12. Дашборд «Nodes ping»

Пинг до узлов кластера.

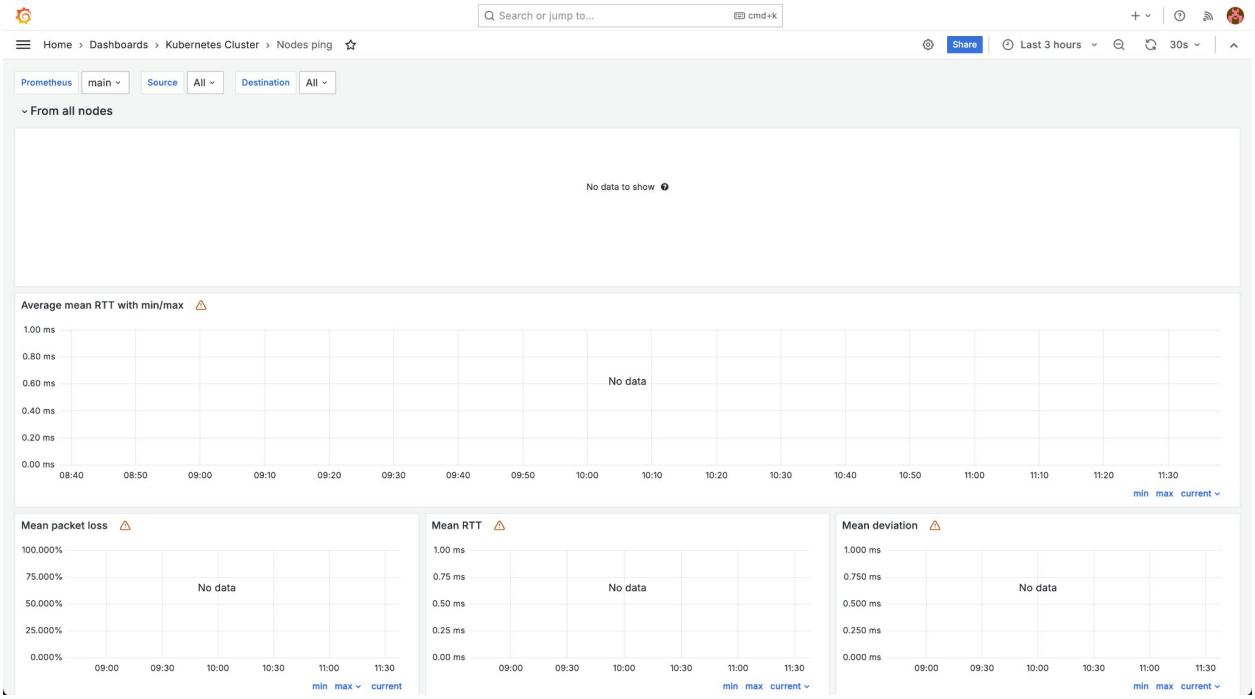


Рисунок 40 – Дашборд «Nodes ping»

6.3.1.5.5.13. Дашборд «NTP»

Состояние сервера времени.

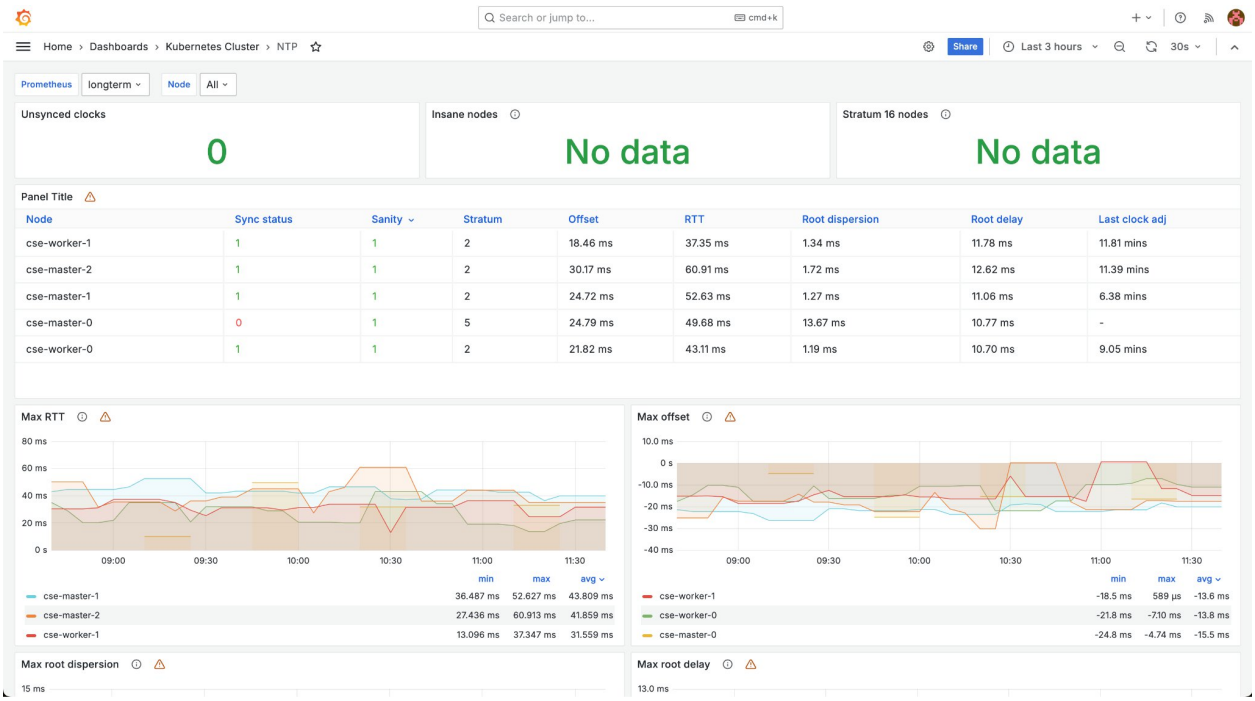


Рисунок 41 – Дашборд «NTP»

6.3.1.5.5.14. Дашборд «Prometheus Benchmark»

Статус prometheus.

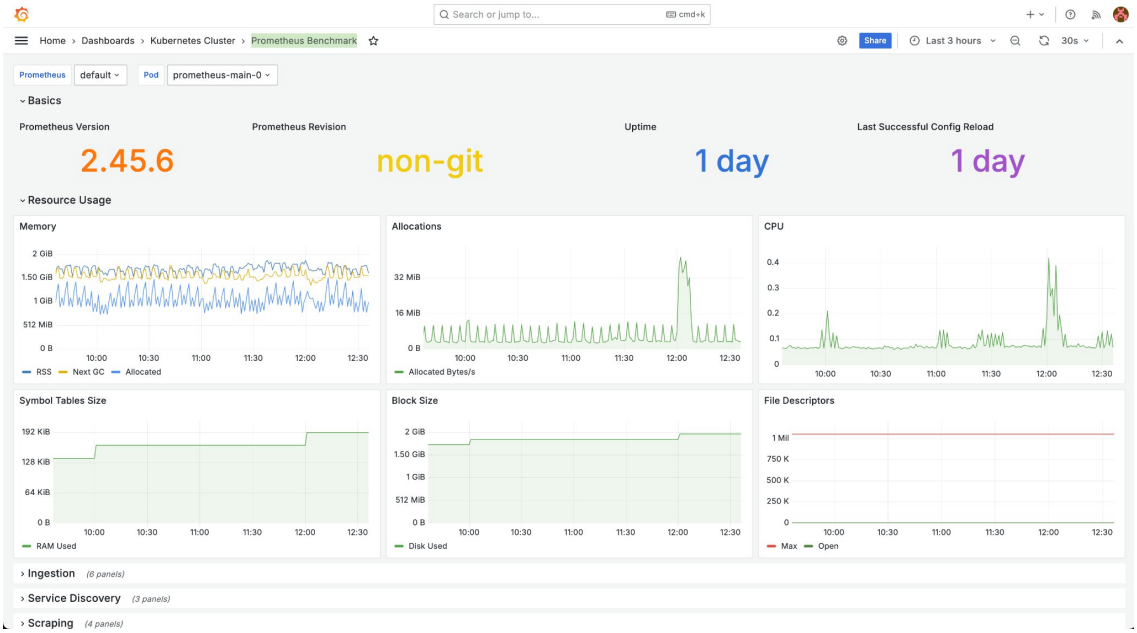


Рисунок 42 – Дашборд «Prometheus Benchmark»

6.3.1.5.5.15. Дашборд «Prometheus-(self)»

Сводная информация о состоянии prometheus.

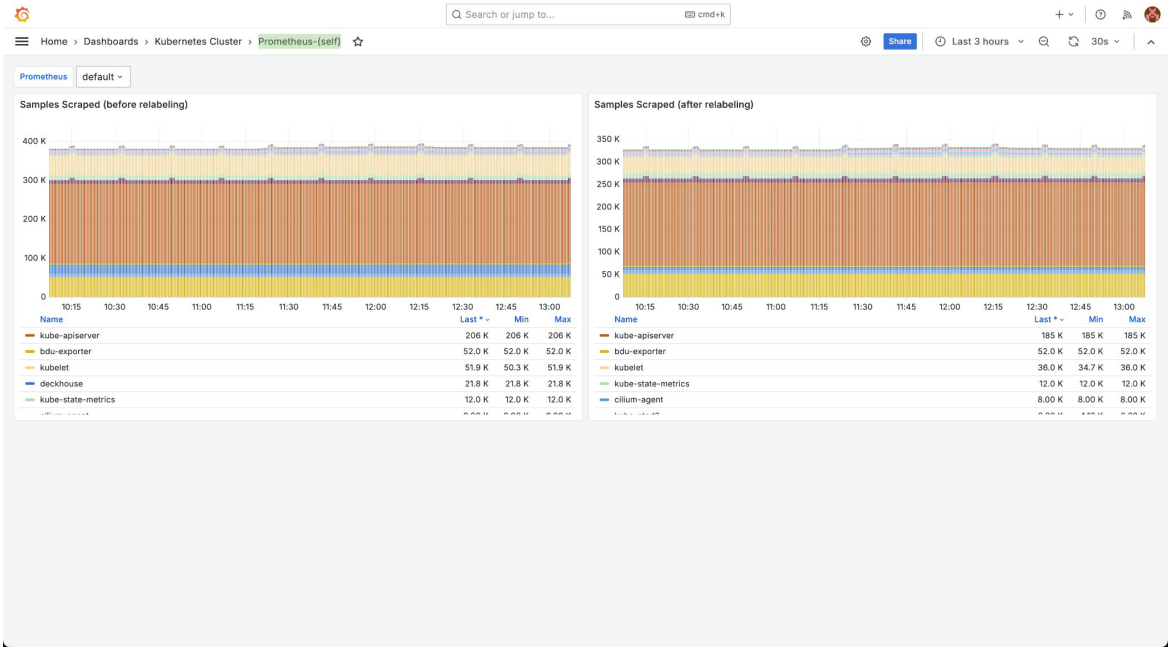


Рисунок 43 – Дашборд «Prometheus-(self)»

6.3.1.5.6. Дашборды группы «Main»

Дашборды с общими данными о состоянии кластера.

6.3.1.5.6.1. Дашборд «Capacity Planning»

Сводные данные о производительности кластера.

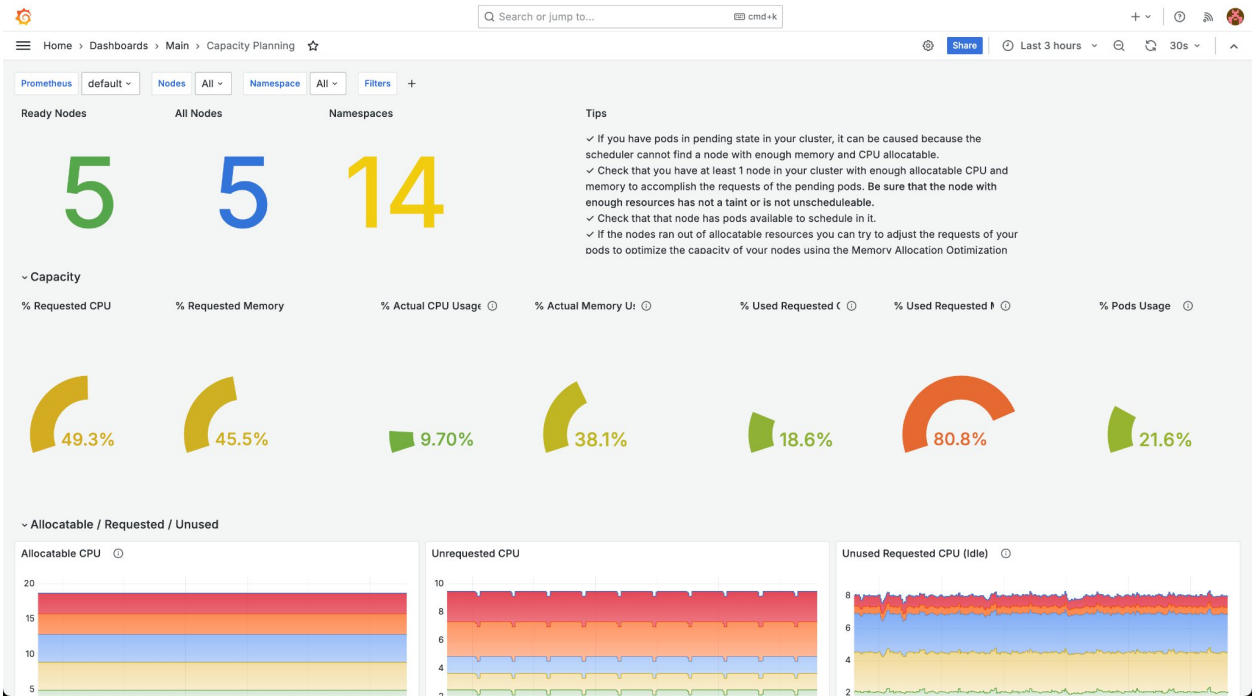


Рисунок 44 – Дашборд «Capacity Planning»

6.3.1.5.6.2. Дашборд «Deckhouse»

Сводная информация о состоянии главного компонента deckhouse.

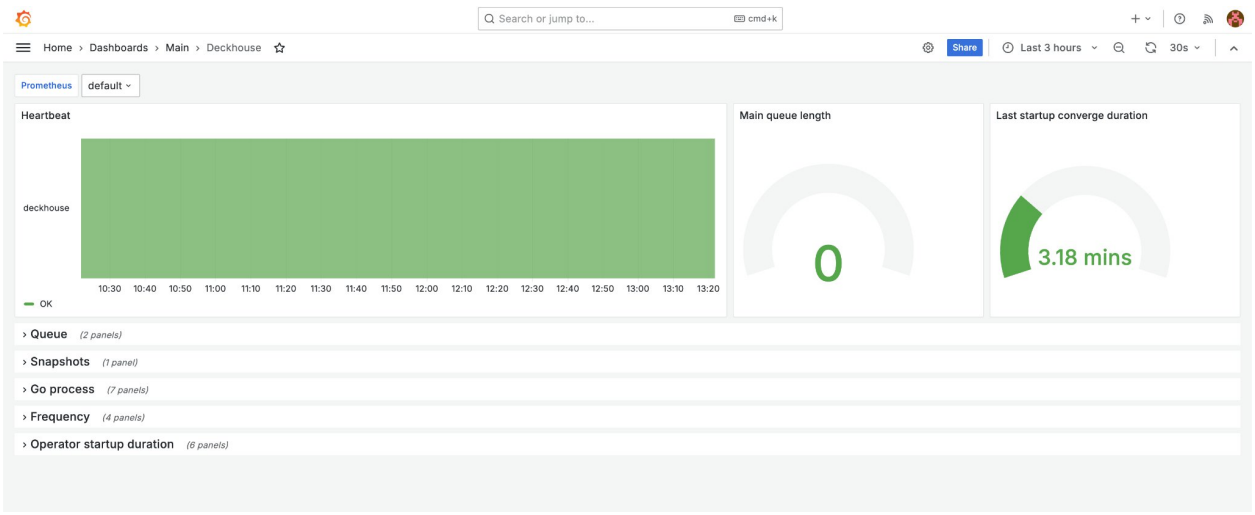


Рисунок 45 – Дашборд «Deckhouse»

6.3.1.5.6.3. Дашборд «Namespace»

Данные по конкретному пространству имен кластера.

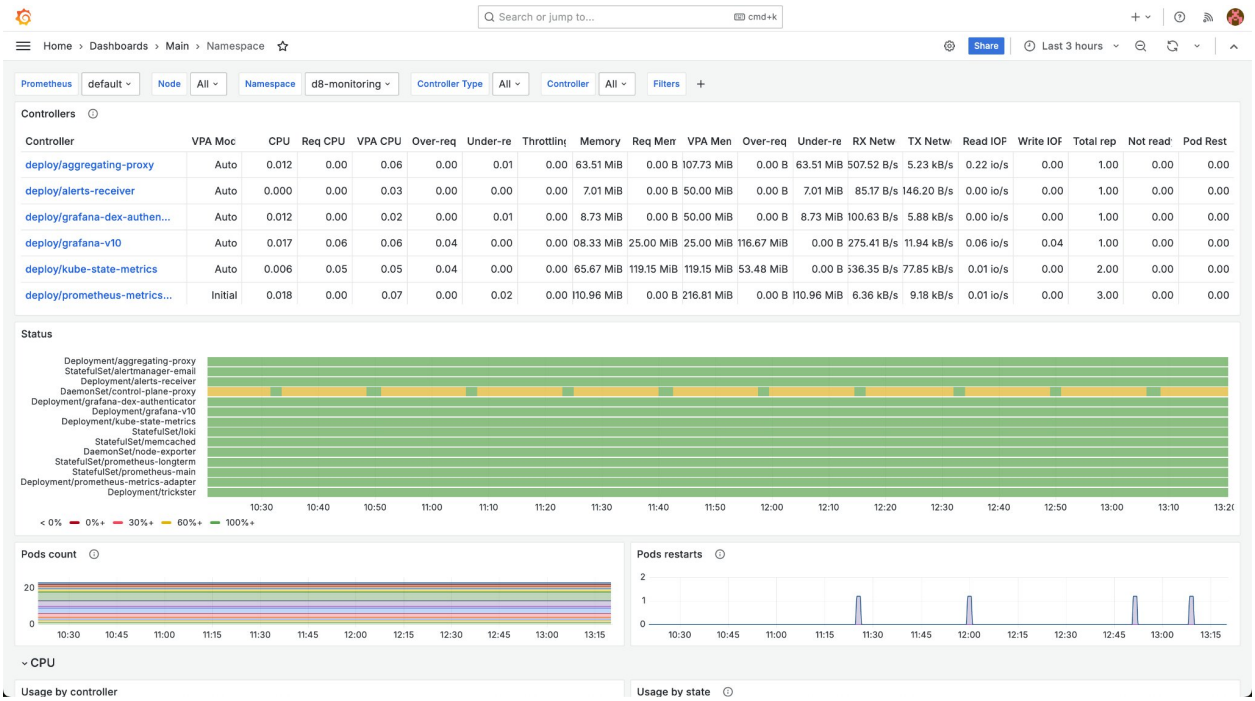


Рисунок 46 – Дашборд «Namespace»

6.3.1.5.6.4. Дашборд «Namespace / Controller»

Данные по контроллерам в пространствах имен.

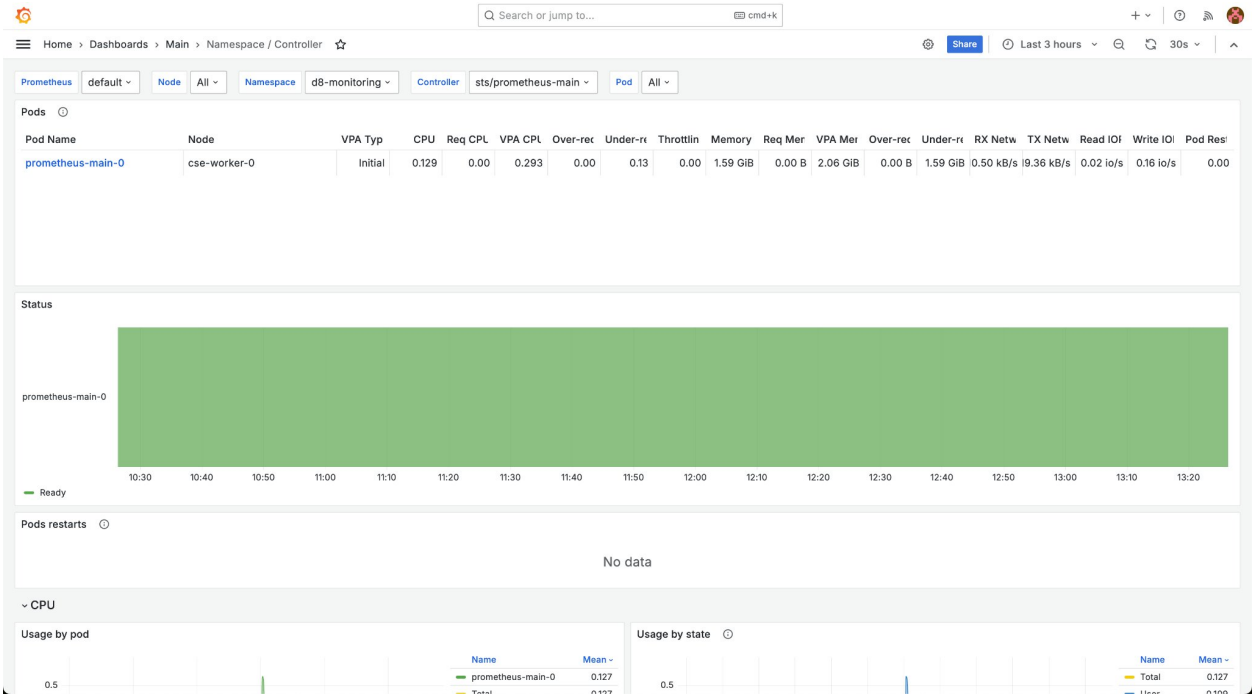


Рисунок 47 – Дашборд «Namespace / Controller»

В фильтрах можно выбрать конкретные пространства имен и контроллеры.

6.3.1.5.6.5. Дашборд «Namespace / Controller / Pod»

Данные по подам в пространствах имен.

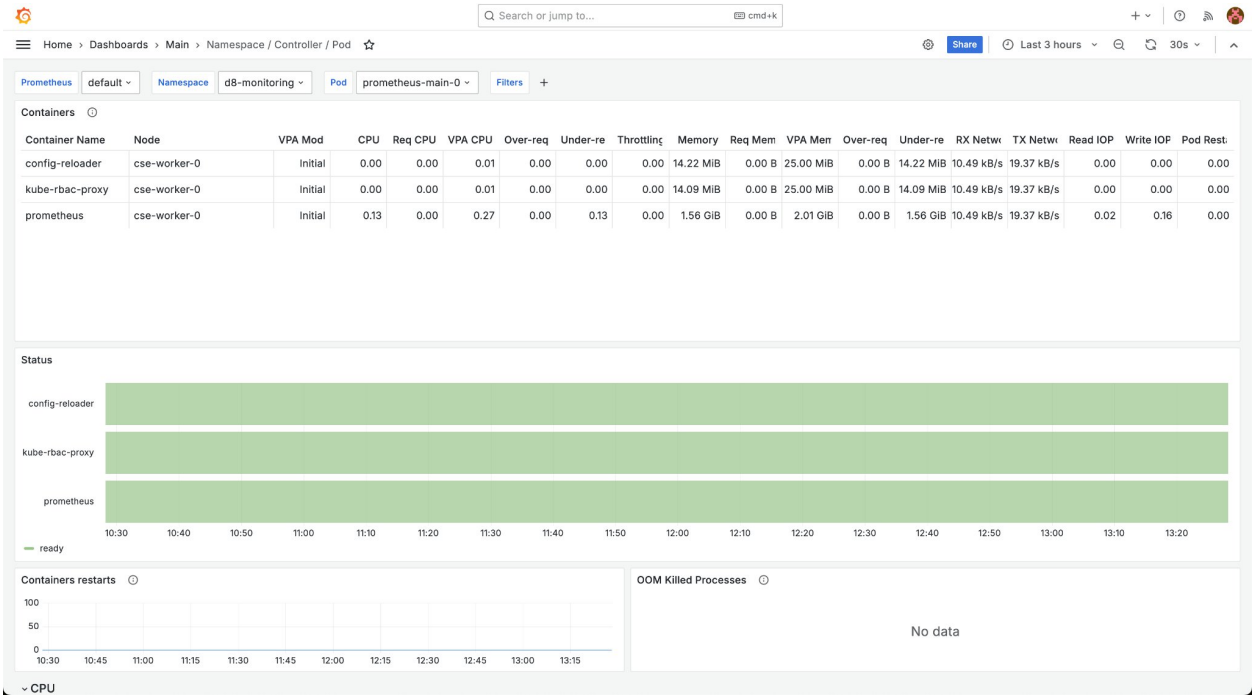


Рисунок 48 – Дашборд «Namespace / Controller / Pod»

В фильтрах можно выбрать определенные пространства имен и поды в них.

6.3.1.5.6.6. Дашборд «Namespaces»

Сводные данные в разрезе пространств имен в кластере.

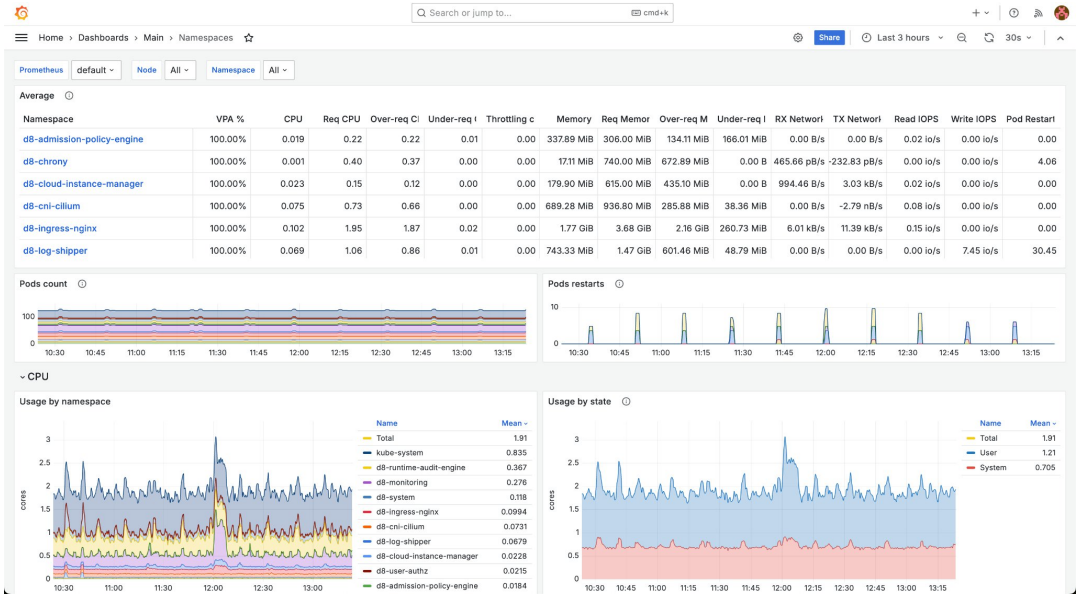


Рисунок 49 – Дашборд «Namespaces»

6.3.1.5.6.7. Дашборды группы Security

Работа с дашбордами группы Security описана в разделе 5.5. Просмотр журналов событий безопасности, Руководства администратора.

6.3.2. Веб-интерфейс документации

В поставку ПО «Deckhouse Platform» входит модуль, предоставляющий доступ к встроенной локальной копии документации платформы. Перейти в него можно по ссылке в левой части главного экрана Grafana.

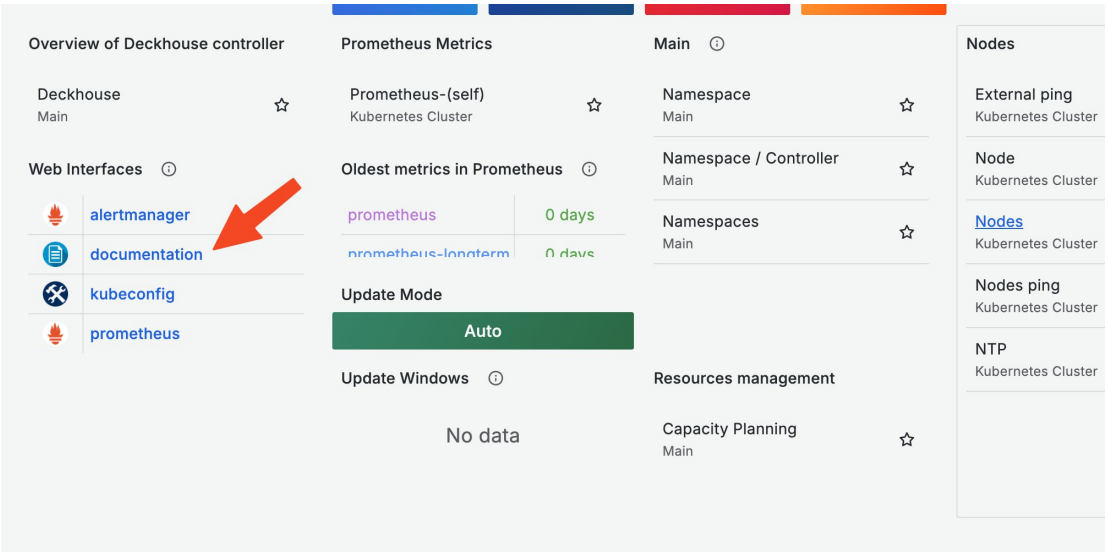


Рисунок 50 – Веб-интерфейс документации

При первом входе потребуется ввести учетные данные пользователя. После этого откроется главный экран документации.

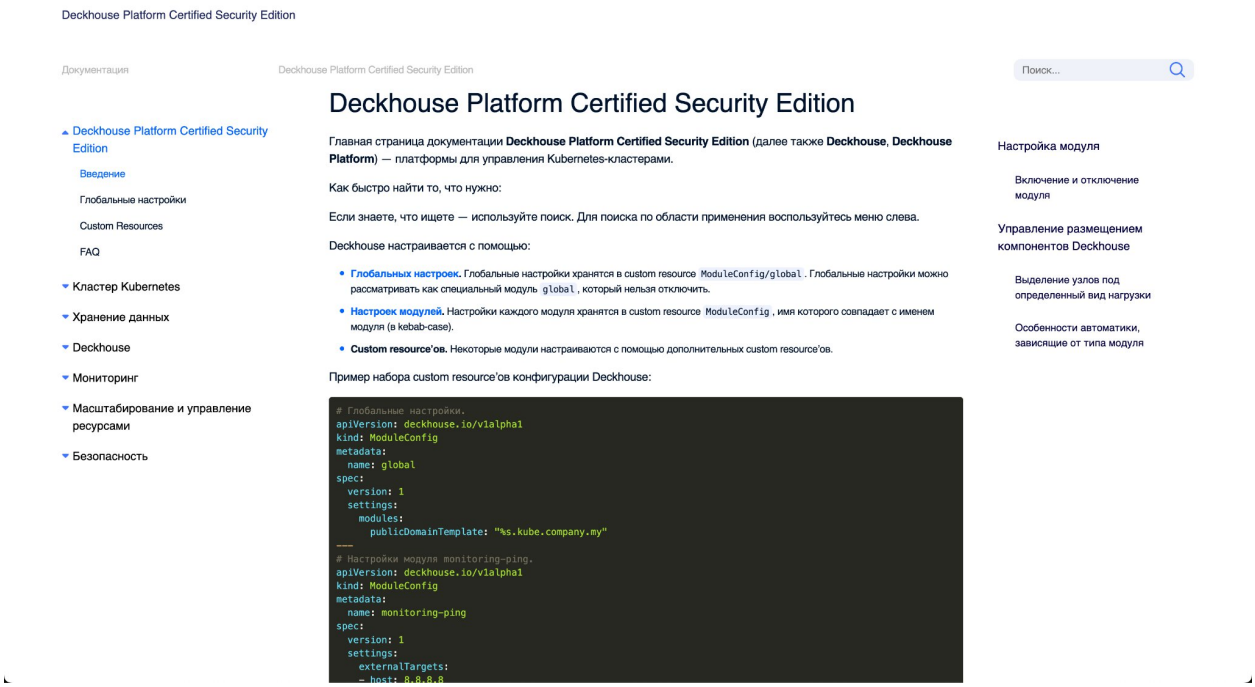


Рисунок 51 – Главный экран документации

Экран поделен на три части (слева направо): блок главного меню, содержащего ссылки на модули ПО «Deckhouse Platform», включенные в поставку, раздел данных, где отображается сама документация по модулями, и блок содержания страницы, в котором скомпонованы разделы текущей страницы.

Над блоком содержания расположено окно поиска, в котором можно осуществлять поиск по документации. Для этого необходимо ввести слово или название параметра, описание которого нужно найти, и нажать «Enter».

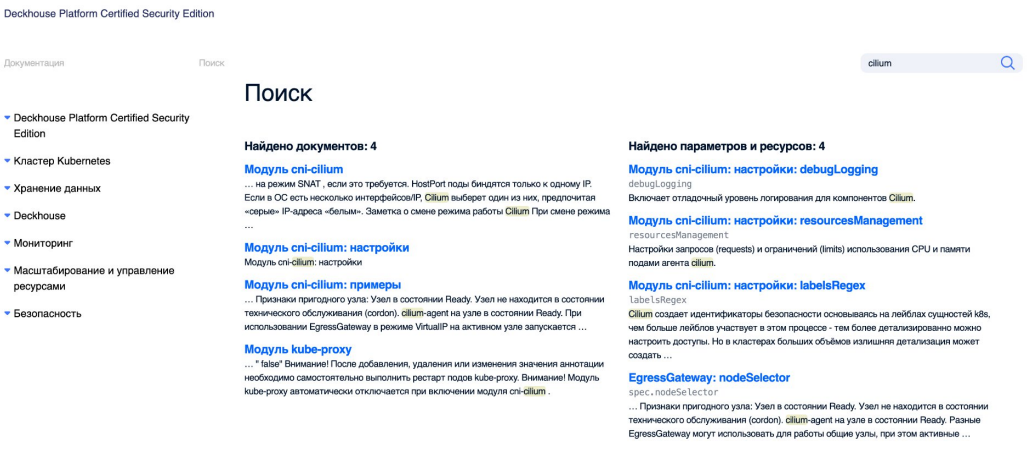


Рисунок 52 – Поиск по документации

6.3.3. Веб-интерфейс модуля alertmanager-email

Ссылка на веб-интерфейс модуля alertmanager-email расположена в левой части главного экрана Grafana.

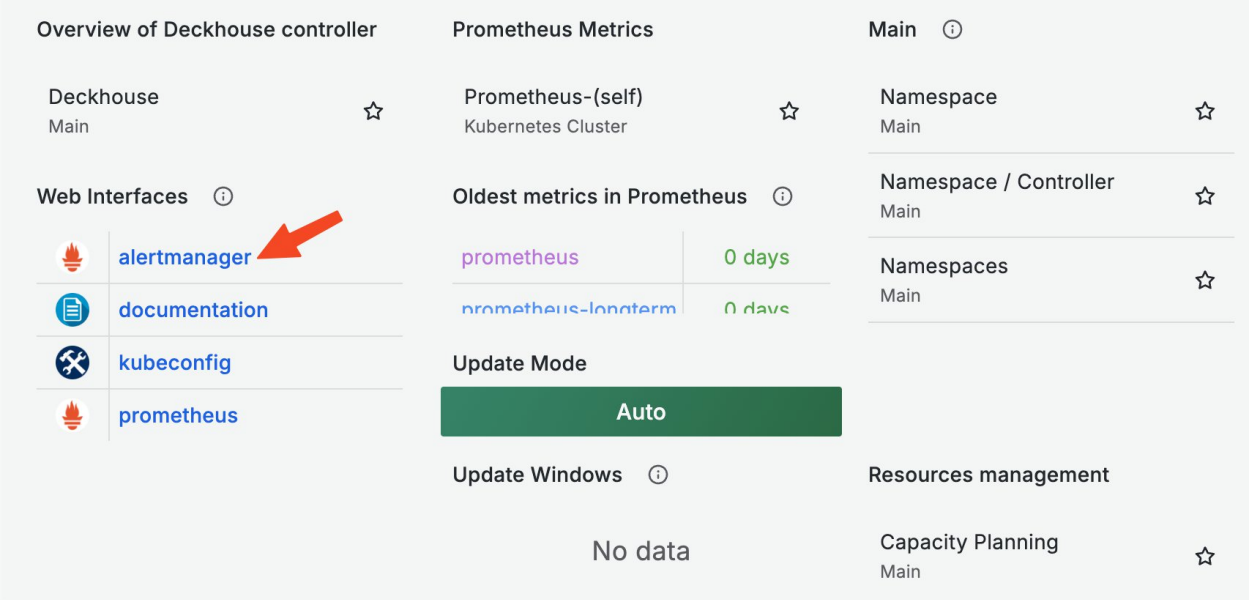


Рисунок 53 – Веб-интерфейс модуля alertmanager-email

При первом входе потребуется ввести учетные данные пользователя. После этого откроется главный экран документации.

На главном экране веб-интерфейса модуля располагается сводная информация во всем алертам, возникшим в кластере.

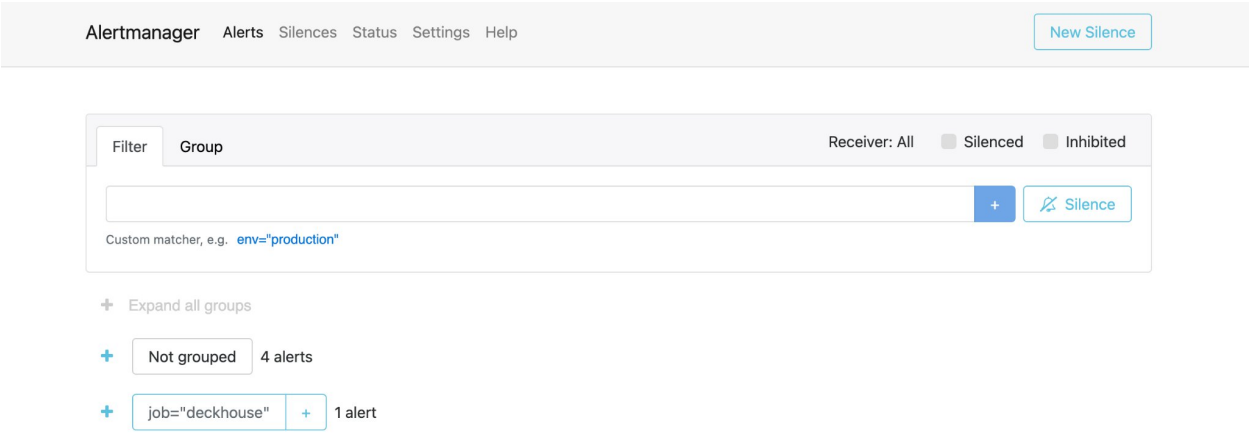


Рисунок 54 – Сводная информация во всем алертам, возникшим в кластере

Алерты сгруппированы по категориям. Чтобы раскрыть категорию нужно нажать на синюю иконку «+» слева от группы.

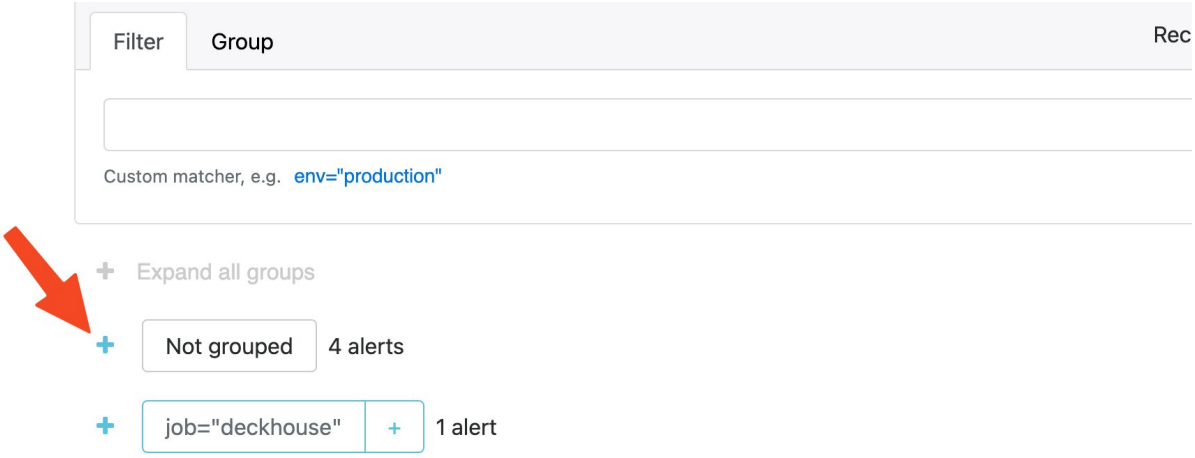


Рисунок 55 – Категории алертов

В раскрывшемся блоке будут отображаться все алерты группы.

Рисунок 56 – Алерты группы

6.3.4. Веб-интерфейс генератора kubeconfig

Ссылка на веб-интерфейс генератора kubeconfig расположена в левой части главного экрана Grafana.

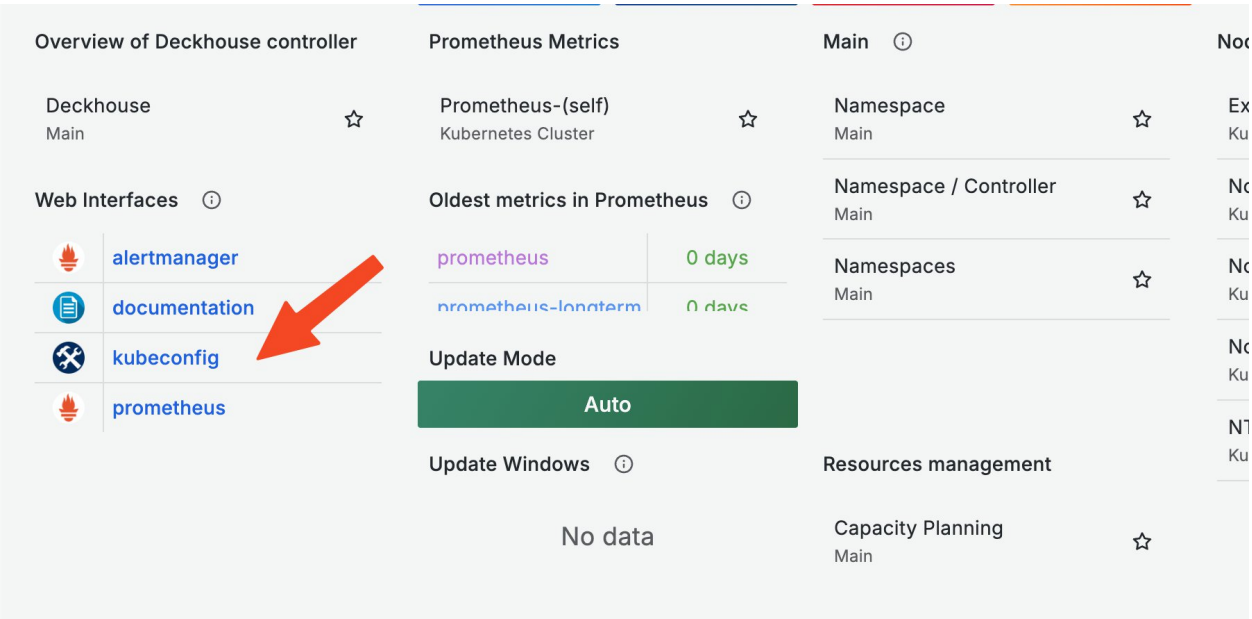


Рисунок 57 – Веб-интерфейс генератора kubeconfig

При первом входе потребуется ввести учетные данные пользователя. После этого откроется главный экран документации.

На главном экране веб-интерфейса сгруппированы команды, позволяющие получить конфигурационные настройки для доступа к кластеру с помощью утилиты kuberconfig.

В средней части экрана расположены вкладки, на которых можно выбрать целевую операционную систему, для которой будет генерироваться конфиг — Linux, macOS или Windows. В зависимости от выбранной ОС будут предложены команды, после выполнения которых в системе будет создан контекст для подключения к кластеру. Также можно выбрать вариант «сырого» конфигурационного файла, который можно вручную расположить в каталоге с настройками.

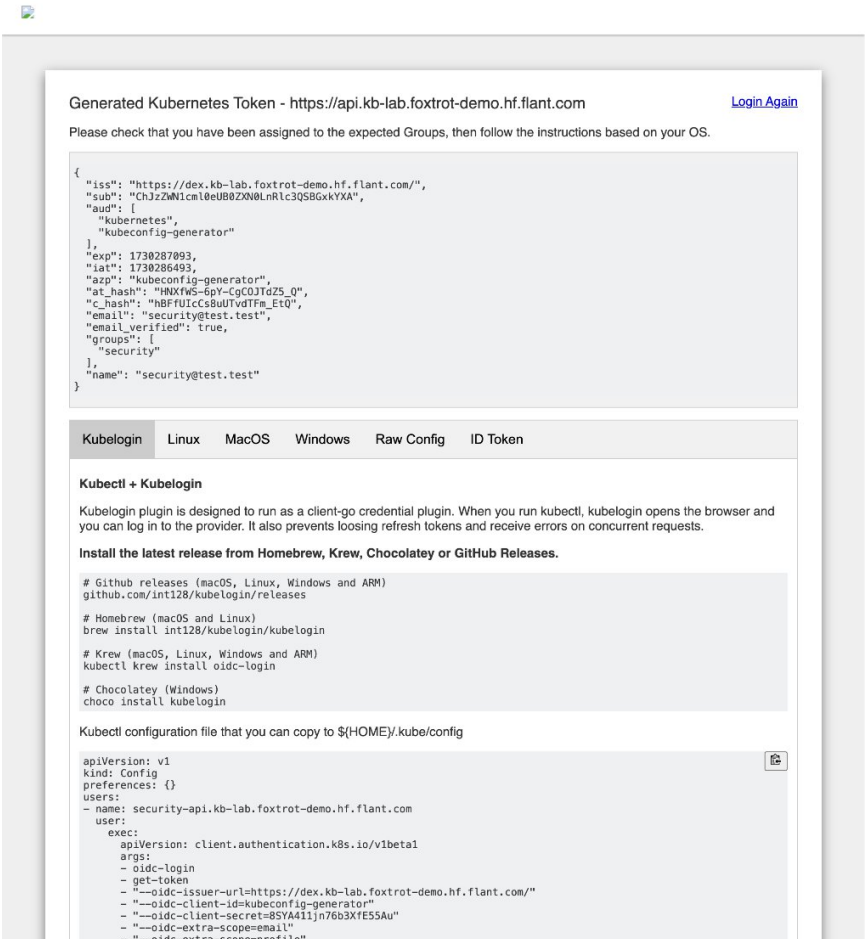


Рисунок 58 – Контекст для подключения к кластеру

6.3.5. Веб-интерфейс ПО «Deckhouse Platform»

Для управления кластером ПО «Deckhouse Platform» через веб-интерфейс (далее, веб-интерфейс, веб-интерфейс console, веб-интерфейс ПО «Deckhouse Platform») используется модуль console (он должен быть включен администратором в кластере).

Для получения доступа к веб-интерфейсу необходимо в адресной строке браузера ввести console. <ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> – строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate. Формат адреса подключения к console может быть иным. Точный адрес подключения можно узнать у администратора информационной (автоматизированной) системы.

При первом входе в веб-интерфейс появится окно аутентификации, где потребуется ввести учетные данные пользователя. После этого откроется главный экран документации.

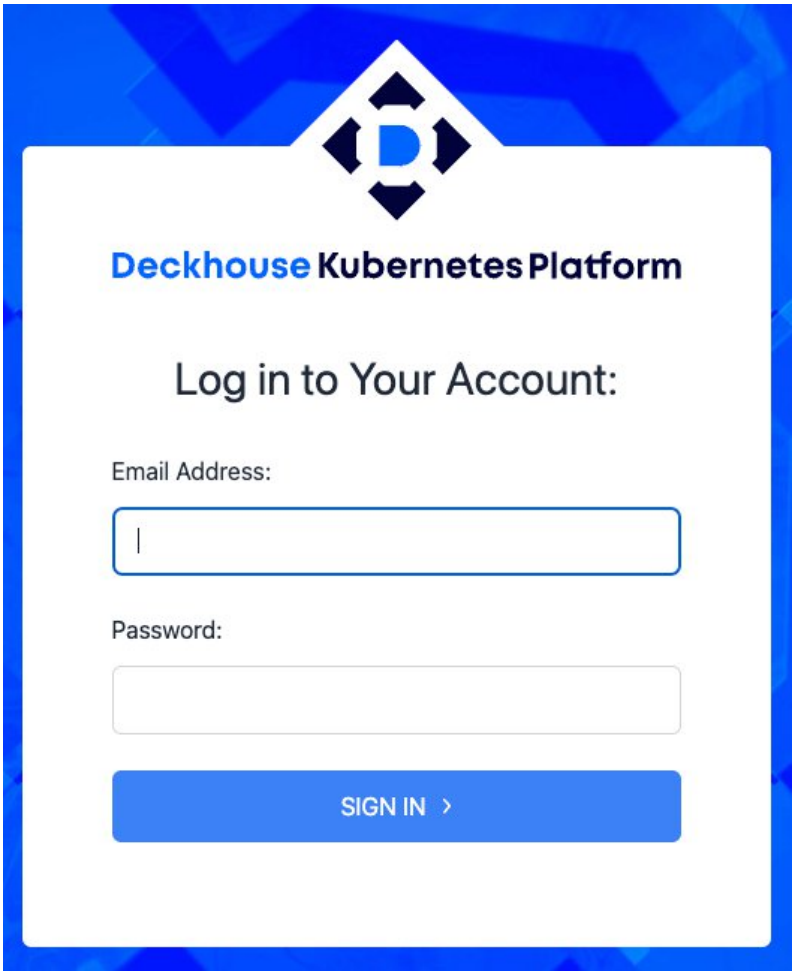


Рисунок 59 – Окно аутентификации для входа в веб-интерфейс

Для аутентификации введите учетные данные, полученные от администратора безопасности.

При успешной аутентификации откроется страница веб-интерфейса.

6.3.5.1. Раздел «Deckhouse»

6.3.5.1.1. Подраздел «Обзор»

В подразделе «Обзор» расположена основная информация о кластере и его основных компонентах.

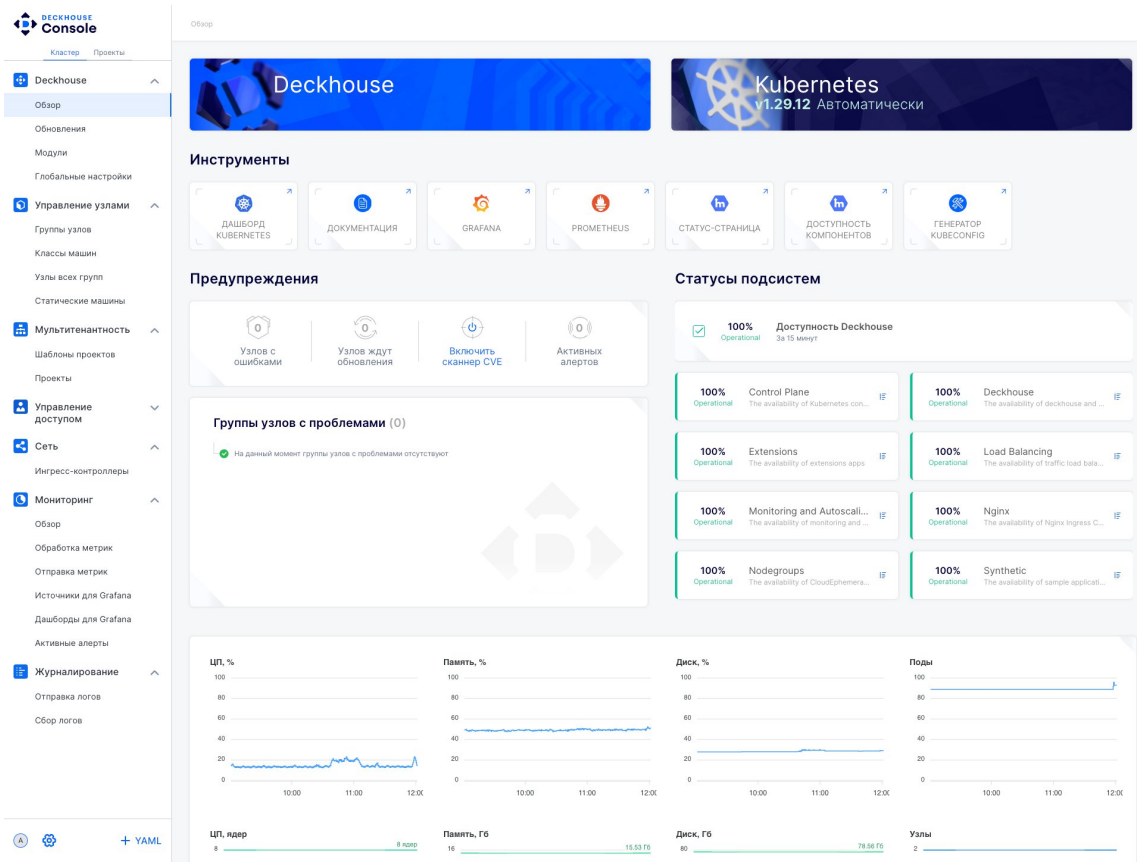


Рисунок 60 – Основная информация о кластере и его компонентах

В верхней части экрана находятся две панели: Deckhouse и Kubernetes. Панель Kubernetes показывает версию Kubernetes, который работает в кластере.

Ниже представлена панель «Инструменты», содержащая несколько кнопок:

- Дашборд Kubernetes – доступ к панели Kubernetes.
- Документация – доступ к справочным материалам.

- Grafana – мониторинг метрик.
- Prometheus – сбор и хранение метрик.
- Статус-страница – отображение статуса компонентов.
- Доступность компонентов – информация о доступности ключевых сервисов.
- Генератор kubeconfig – инструмент для создания конфигурационных файлов.

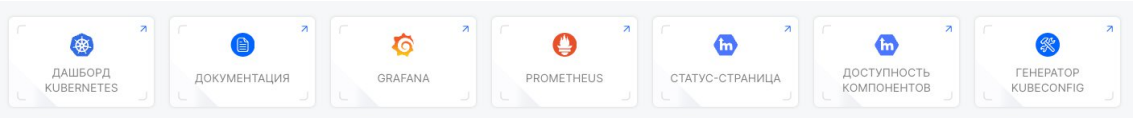


Рисунок 61 – Панель «Инструменты»

Далее представлена панель «Предупреждения», в которой содержатся ошибки, ожидаемые обновления, а также активные алерты.

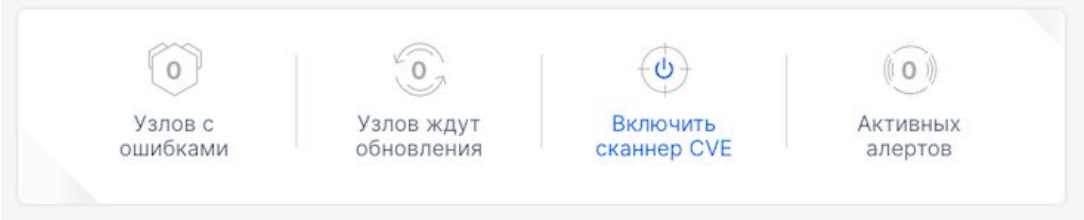


Рисунок 62 – Панель «Предупреждения»

Ниже представлена панель «Группы узлов с проблемами», которая анализирует работоспособность групп узлов и выводит список проблемных узлов, если таковые имеются.

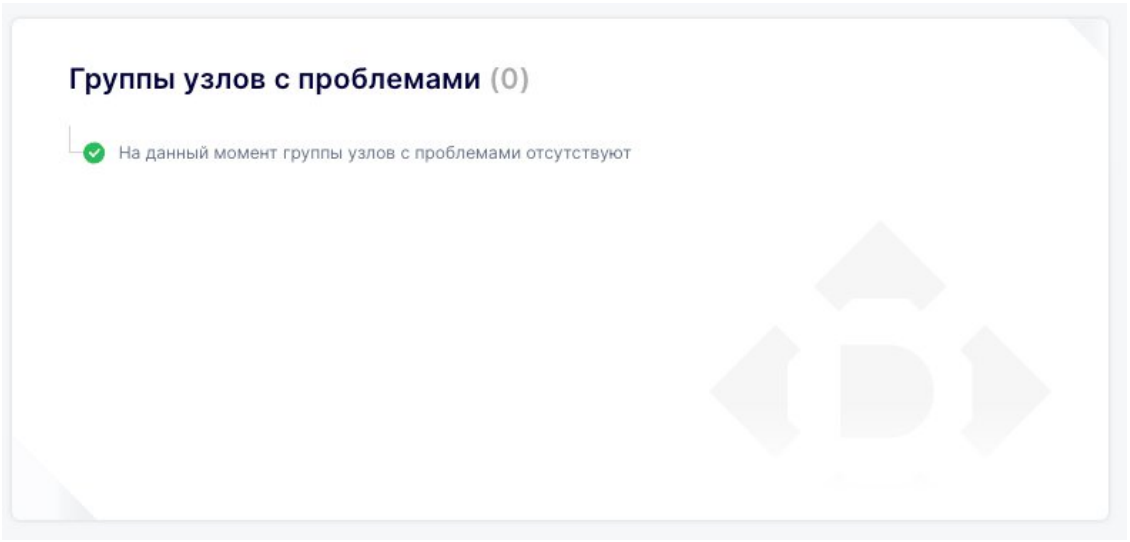


Рисунок 63 – Панель «Группы узлов с проблемами»

В правой части экрана находится панель «Статусы подсистем», где отображается статус различных сервисов, которые работают в кластере.

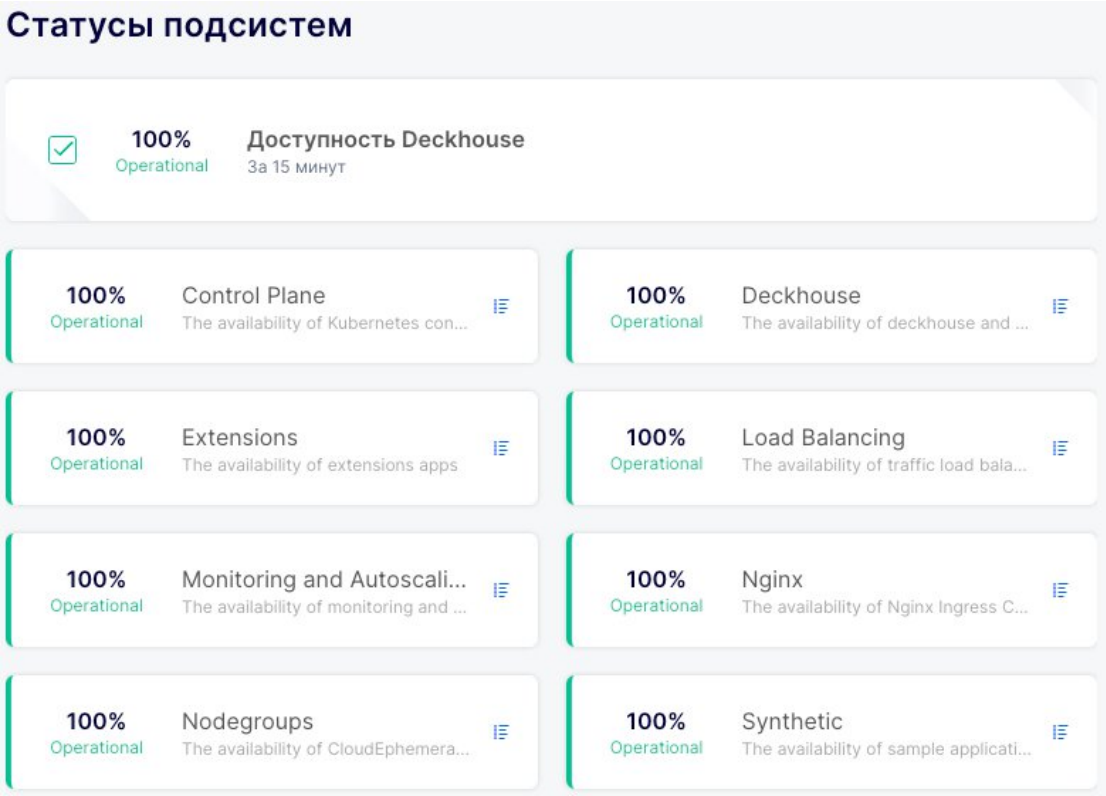


Рисунок 64 – Панель «Статусы подсистем»

Внизу экрана размещены графики и показатели мониторинга ресурсов, которые отображают текущие изменения нагрузки и позволяют отслеживать производительность кластера.

Рисунок 65 – Графики и показатели мониторинга ресурсов.

Слева представлено боковое меню с основными разделами. Некоторые разделы меню могут не отображаться из-за отсутствия доступа к ним. При необходимости получения доступа к отсутствующим разделам требуется обратиться к администратору.

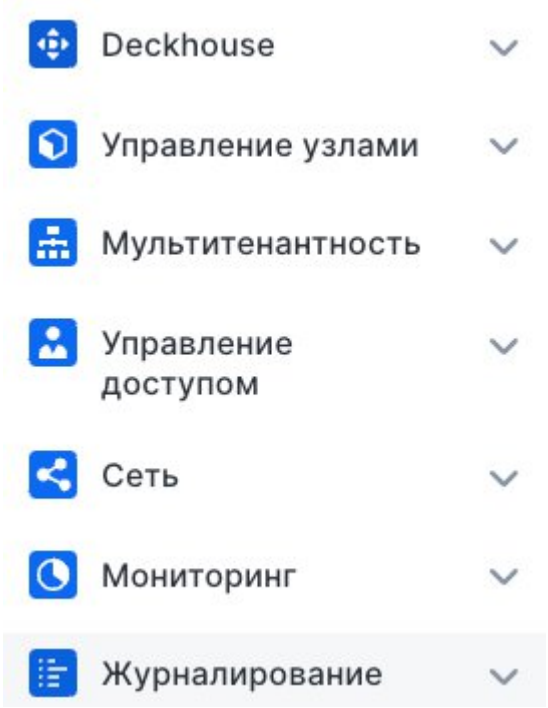


Рисунок 66 – Боковое меню с основными разделами

Снизу слева находится меню с профилем пользователя, настройками и добавлением YAML-файла.

Всплывающее меню пользователя позволяет увидеть текущего пользователя и выйти из системы при необходимости.

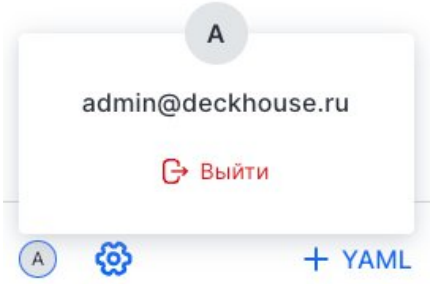


Рисунок 67 – Меню с профилем пользователя, настройками и добавлением YAML-файла.

Всплывающее меню настроек позволяет изменять системные параметры и отображает текущую версию модуля Console.

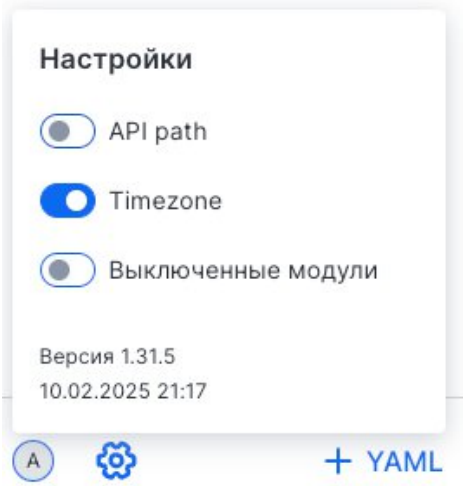


Рисунок 68 – Меню настроек.

Всплывающее меню добавления YAML-файла вызывает редактор YAML, который используется для управления конфигурациями в Kubernetes.

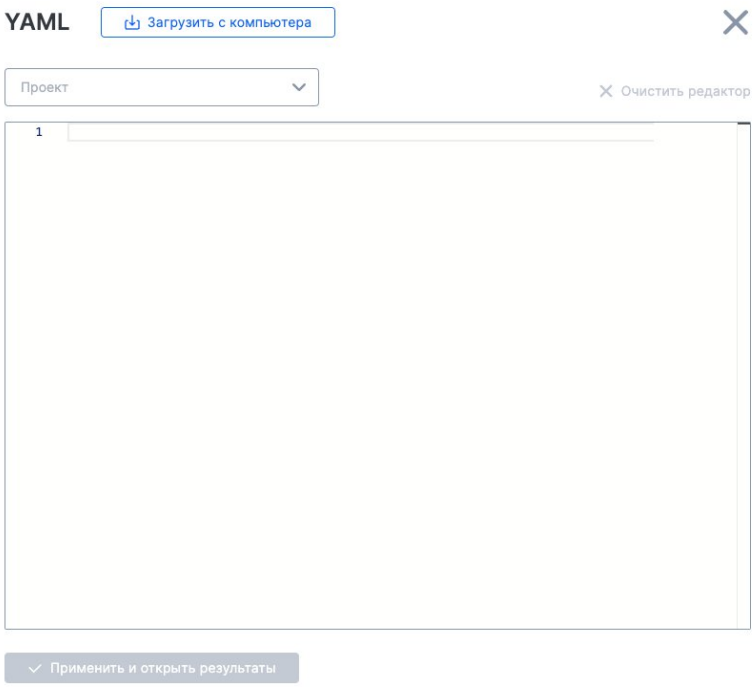


Рисунок 69 – Редактор YAML.

6.3.5.1.2. Подраздел «Модули»

В подразделе «Модули» перечислены запущенные и отключенные модули. Для поиска необходимого модуля можно воспользоваться фильтром.

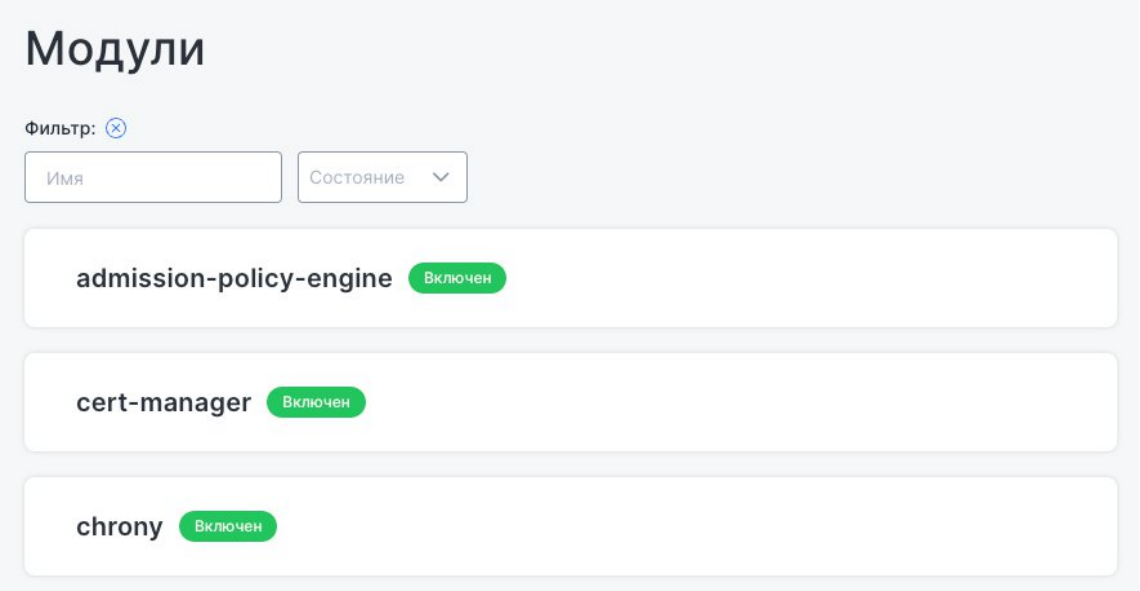


Рисунок 70 – Подраздел «Модули»

6.3.5.1.3. Подраздел «Глобальные настройки»

В подразделе «Глобальные настройки» представлены критически важные настройки для работы кластера.

- Глобальные настройки кластера – управление DNS-именами и toleration.
- Глобальные настройки модулей – отказоустойчивость, Ingress-класс и StorageClass.
- Режим работы HTTPS – настройка сертификатов.
- Ресурсы control plane – выделение CPU и памяти для управляющих компонентов.

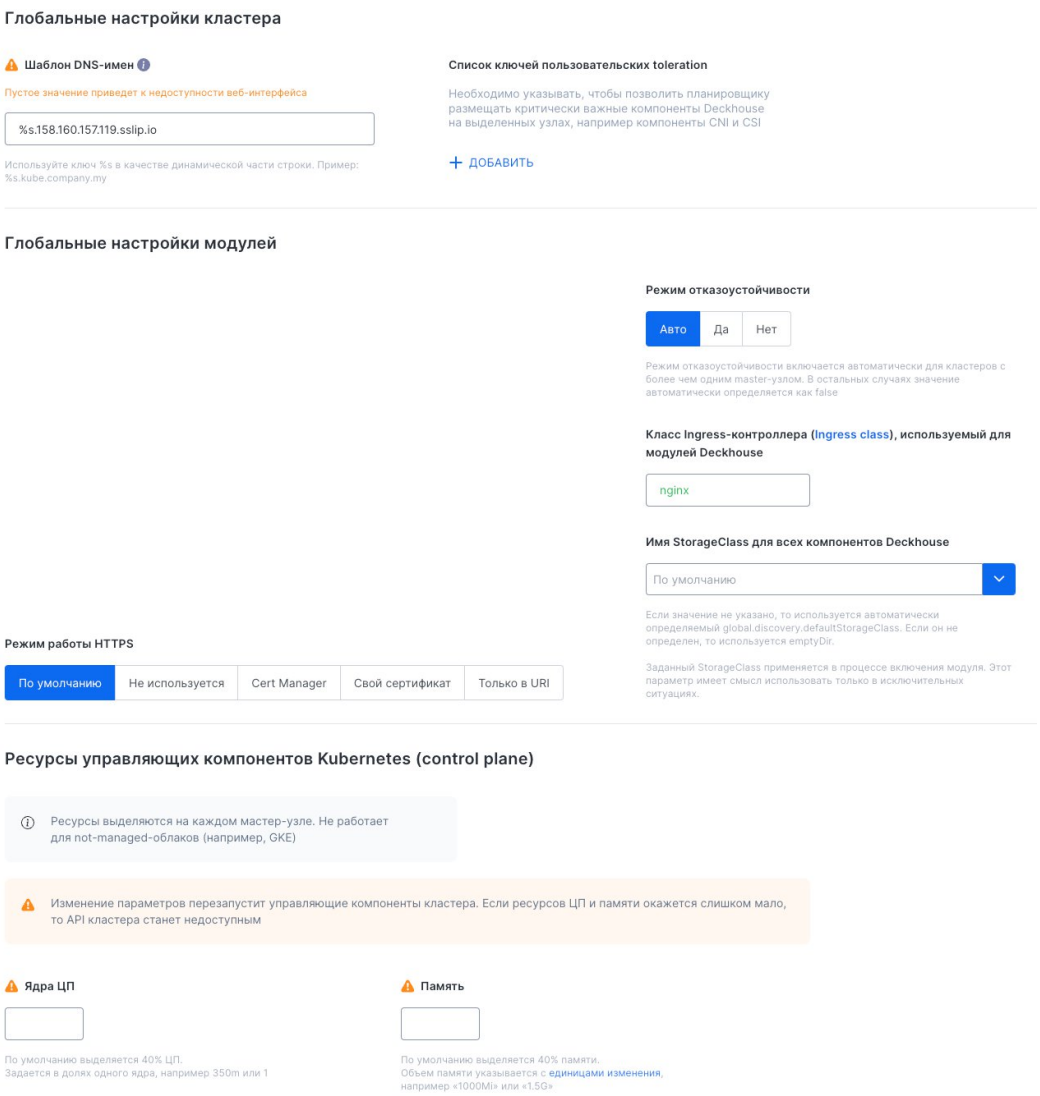


Рисунок 71 – Подраздел «Глобальные настройки»

Эти параметры влияют на стабильность, безопасность и отказоустойчивость кластера, поэтому их изменение требует осторожности.

6.3.5.2. Раздел «Управление узлами»

6.3.5.2.1. Подраздел «Группы узлов»

Подраздел «Группы узлов» предназначен для управления группами узлов Kubernetes-кластера. Он позволяет просматривать, фильтровать и добавлять узлы, а также следить за их состоянием и загрузкой ресурсов. Для добавления новой группы узлов можно воспользоваться кнопкой «Добавить».

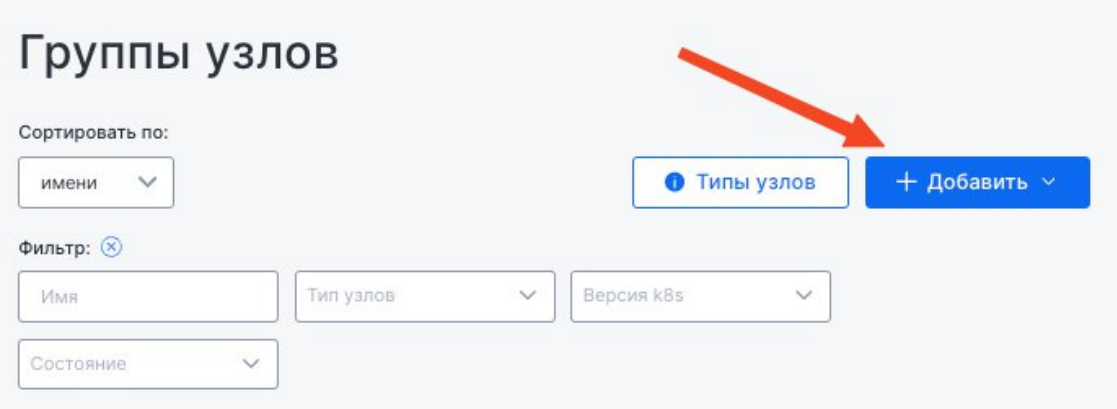


Рисунок 72 – Подраздел «Группы узлов»

Форма добавления группы узлов позволяет задать имя, количество узлов и настроить селектор статических машин по лейблам, который после создания группы становится недоступным для редактирования. Также предусмотрена возможность добавления выражений 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.

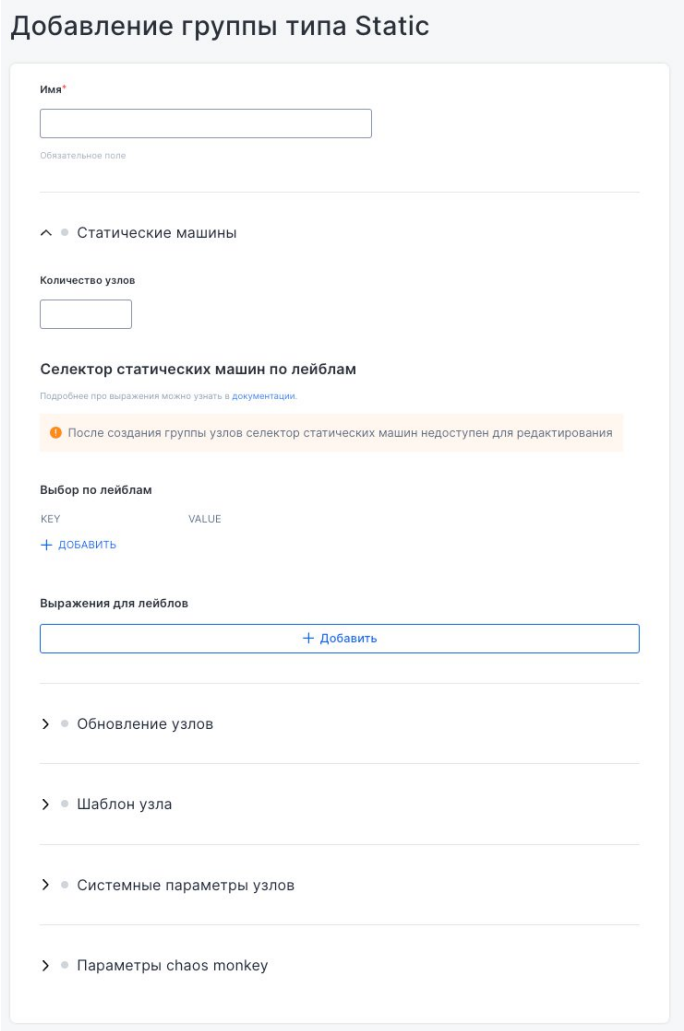


Рисунок 73 – Форма добавления группы узлов

Карточка группы узлов отображает информацию о типе узлов и версии Kubernetes, текущем состоянии узлов, включая общее количество, готовность и актуальность. Также представлены графики мониторинга нагрузки на ресурсы, такие как процессор, память и диск, позволяющие отслеживать их использование. Дополнительно указываются тейнты и лейблы, которые используются для управления назначением подов и организации работы узлов.

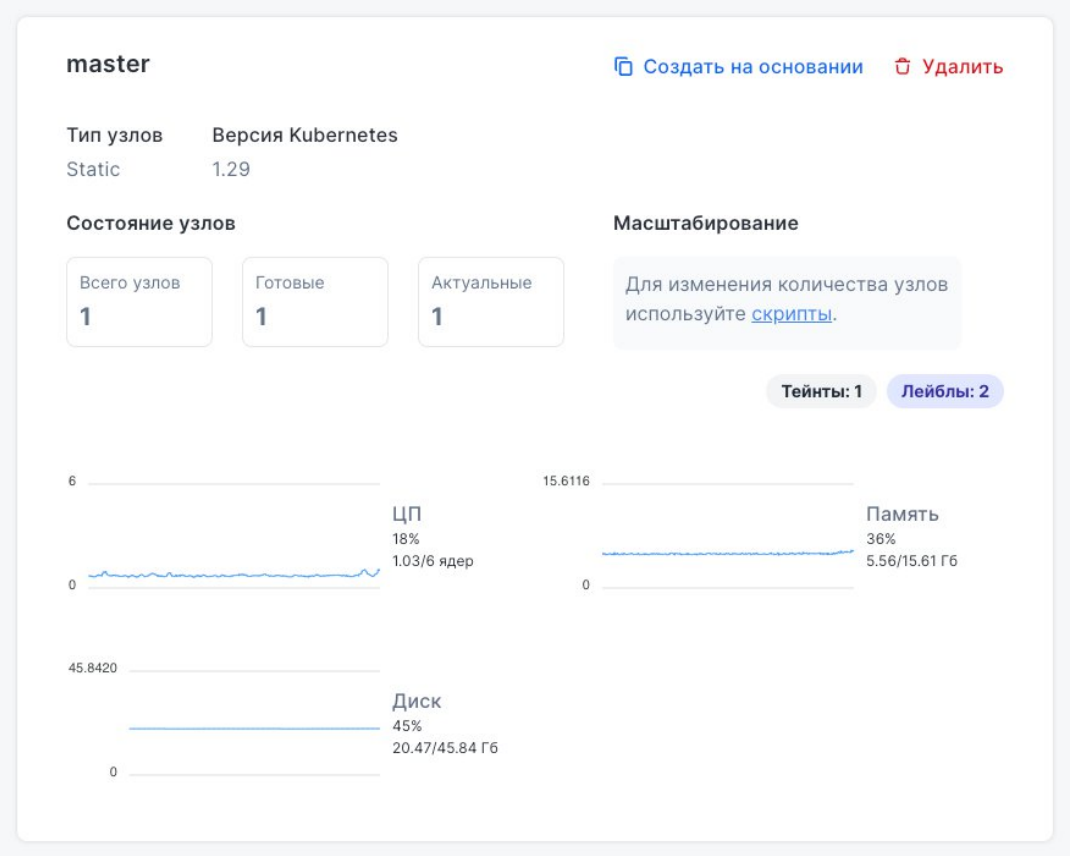


Рисунок 74 – Карточка группы узлов

6.3.5.2.2. Подраздел «Классы машин»

Подраздел «Классы машин» позволяет управлять конфигурациями машин, используемых в кластере, с возможностью сортировки списка. В карточке класса отображаются основные характеристики, включая количество процессорных ядер, объем памяти и дискового пространства, а также дополнительные параметры, такие как наличие GPU, внешний IP и основная сеть. Доступны опции создания нового класса машин, клонирования существующего и удаления. Внизу указано, в каких группах узлов используется данный класс, что помогает отслеживать его применение в кластере.

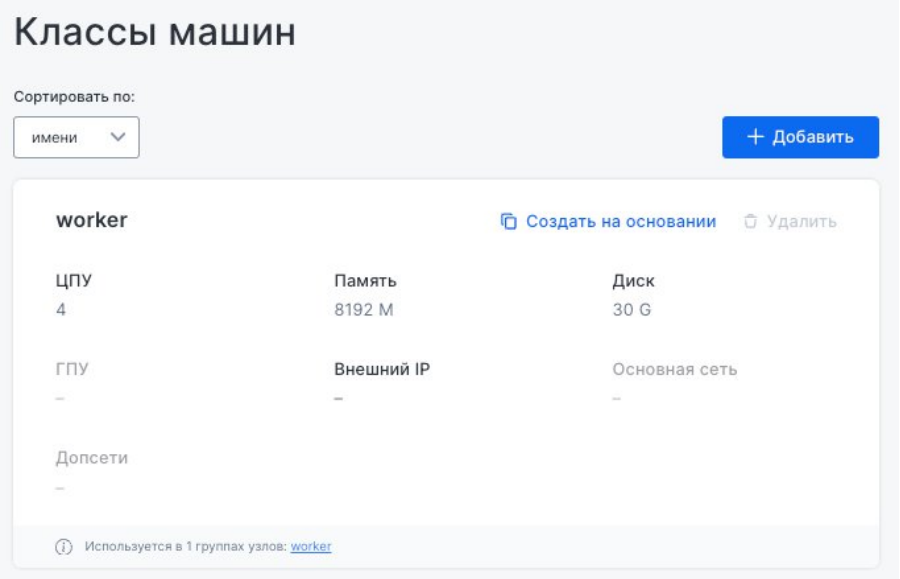


Рисунок 75 – Подраздел «Классы машин»

Меню «Добавление класса машин» позволяет задать параметры новой конфигурации машин для кластера. В разделе конфигурации указывается имя класса, а в блоке ресурсов настраиваются количество виртуальных ядер, платформа процессора, объем памяти, базовый уровень производительности, число графических адаптеров и идентификатор образа. Дополнительно можно включить поддержку прерываемых виртуальных машин, задать размер и тип диска. В разделе сети выбирается основная подсеть, тип сети и возможность использования публичного IP. Также предусмотрена возможность добавления дополнительных подсетей и лейблов для более гибкой настройки инфраструктуры.

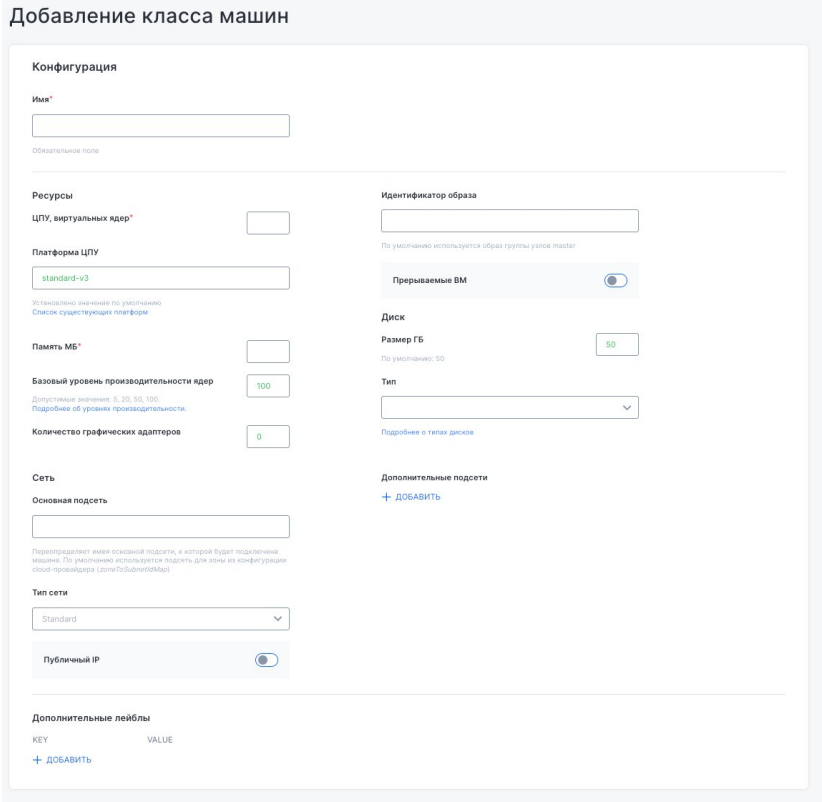


Рисунок 76 – Меню «Добавление класса машин»

6.3.5.2.3. Подраздел «Узлы всех групп»

Подраздел «Узлы всех групп» предоставляет информацию о всех узлах Kubernetes-кластера с возможностью сортировки и фильтрации по имени, зоне, версии ОС, CRI, kubelet и состоянию. В карточке узла отображается его текущее состояние, группа, дата и время, зона размещения, внутренний и внешний IP-адреса, используемый контейнерный рантайм (CRI), версия ядра, версия kubelet и операционная система. Также представлены графики загрузки процессора, памяти, диска и сетевого трафика, что позволяет отслеживать производительность узла. Доступны кнопки «Cordon» и «Cordon+Drain» для управления доступностью узла в кластере.

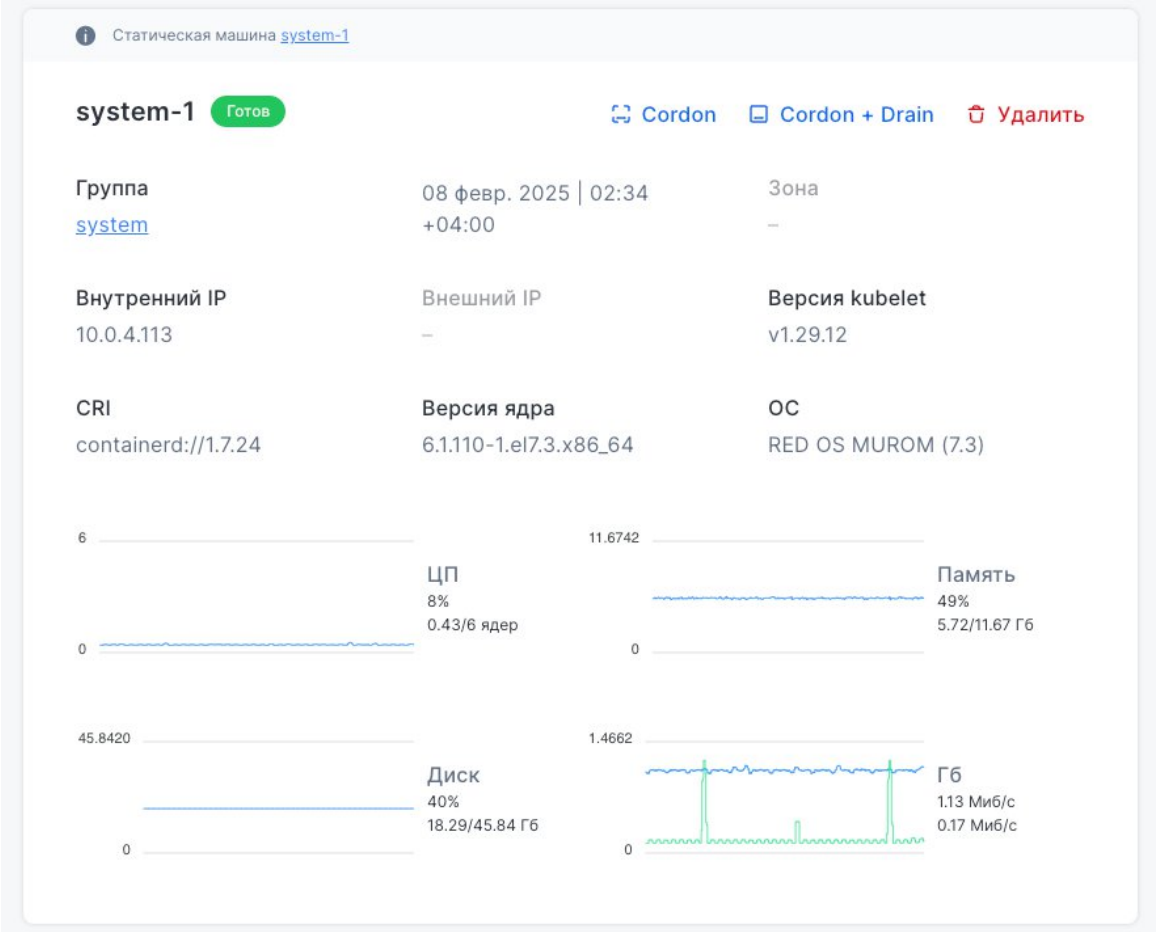


Рисунок 77 – Подраздел «Узлы всех групп»

6.3.5.2.4. Подраздел «Статические машины»

Подраздел «Статические машины» предоставляет возможность управления статическими узлами в кластере и включает две вкладки: «Машины» для работы со статическими машинами и «SSH-доступы» для настройки авторизации. Этот интерфейс позволяет быстро находить и управлять статическими машинами в инфраструктуре.

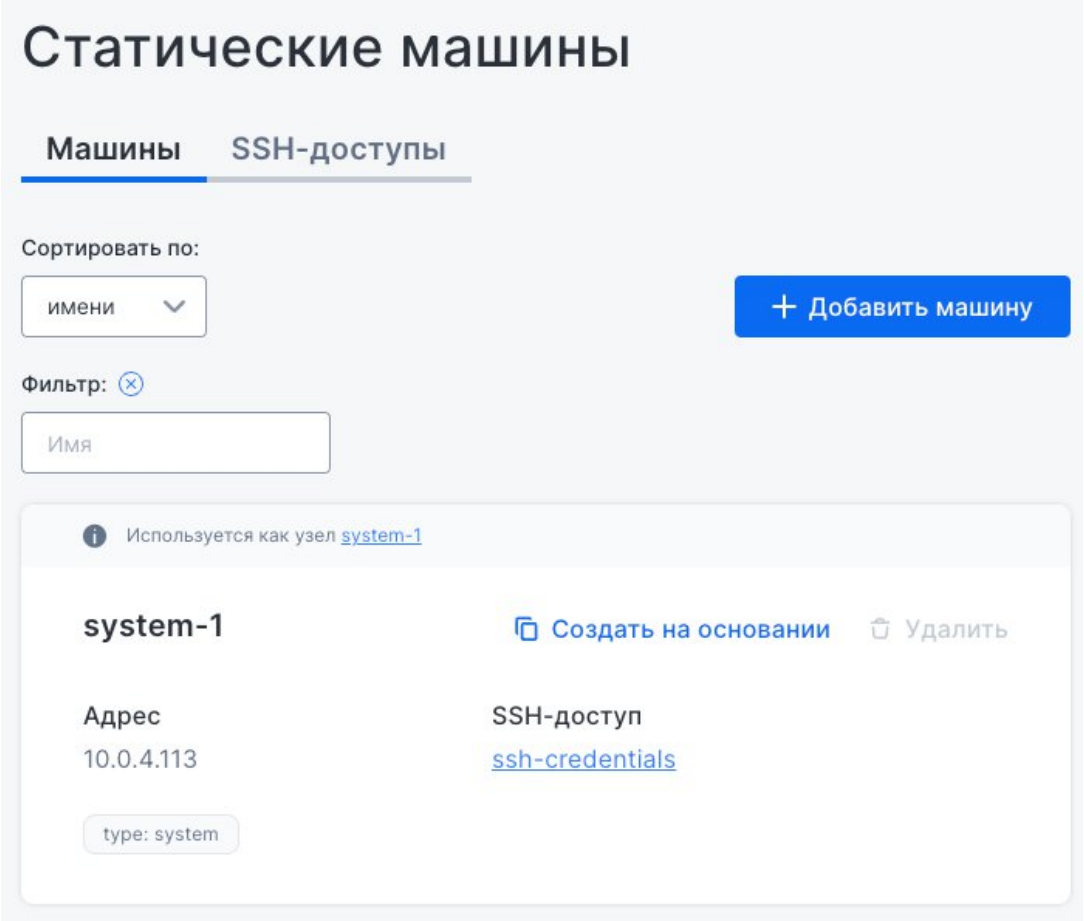


Рисунок 78 – Подраздел «Статические машины»

Кнопка «Добавить машину» во вкладке «Машины» предназначена для добавления новой машины в кластер и включает обязательные поля конфигурации. Пользователь должен указать имя машины, ее адрес и выбрать способ SSH-доступа из выпадающего списка. Дополнительно можно задать лейблы, добавляя ключи и значения для дальнейшей идентификации и управления.

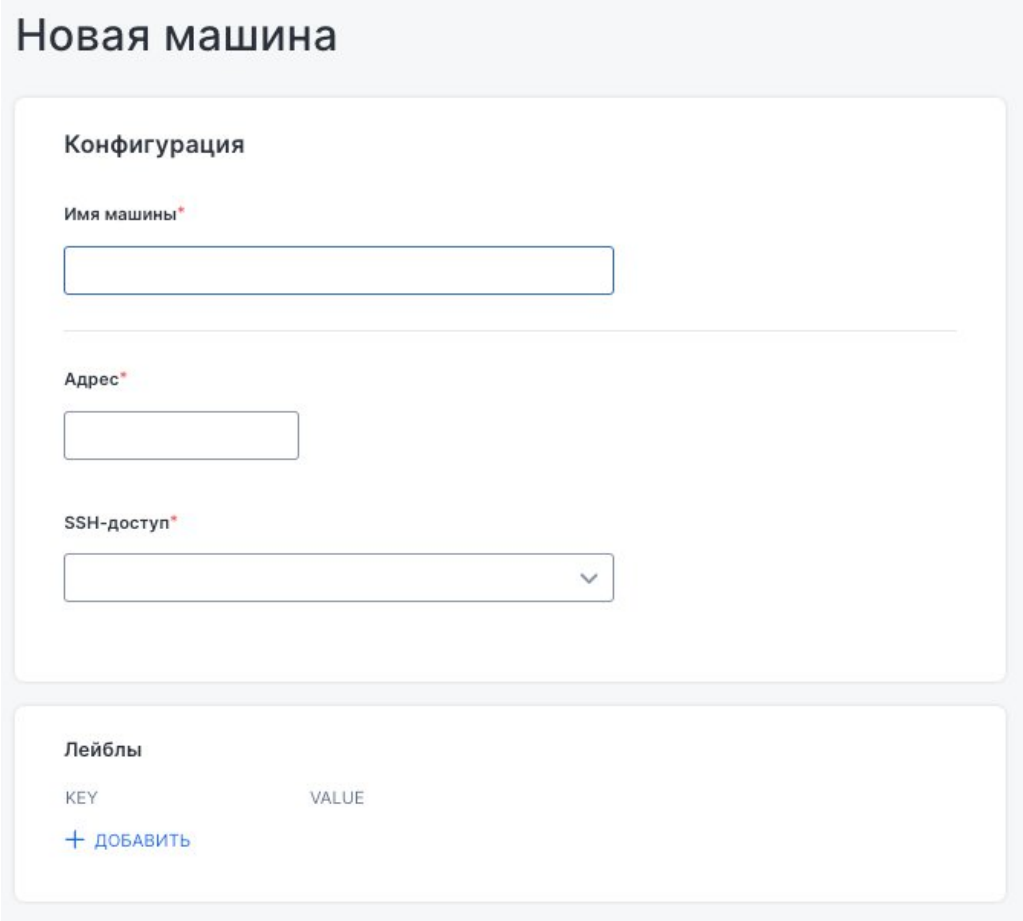


Рисунок 79 – Добавление новой машины

Кнопка «Добавить SSH-доступ» во вкладке «SSH-доступы» предназначена для настройки подключения к узлам через SSH. Пользователь должен задать имя доступа, имя пользователя и приватный SSH-ключ, а также может указать пароль sudo для выполнения привилегированных команд. Дополнительно доступны поля для изменения SSH-порта и добавления дополнительных аргументов SSH.

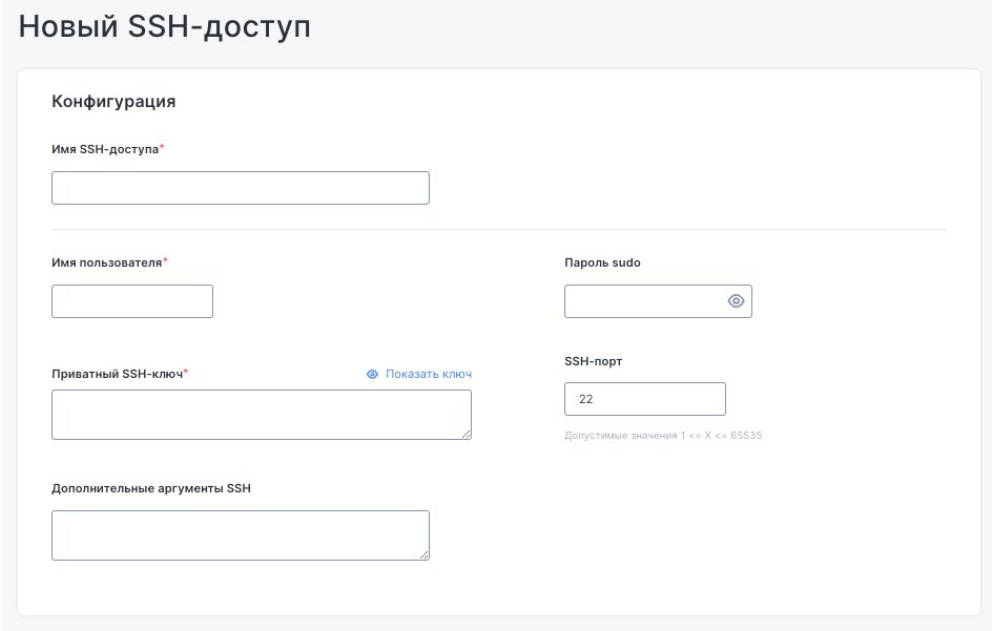


Рисунок 80 – Добавление SSH-доступа

6.3.5.3. Раздел «Мультитенантность»

Мультитенантность позволяет создавать проекты в кластере Kubernetes. Проект — это изолированное окружение, в котором можно развернуть приложения.

6.3.5.3.1. Подраздел «Шаблоны проектов»

Подраздел «Шаблоны проектов» предназначен для создания шаблонов проектов. Шаблоны проектов по умолчанию включают базовые сценарии использования и служат примером возможностей шаблонов. Для добавления нового шаблона используется кнопка «Создать шаблон проекта».

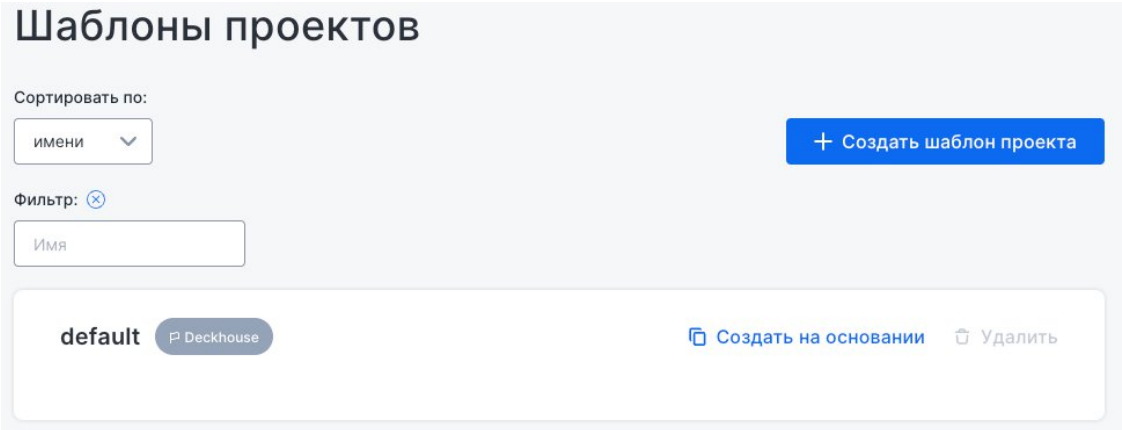


Рисунок 81 – Подраздел «Шаблоны проектов»

Форма «Новый шаблон проекта» позволяет задать имя проекта, а также добавить лейблы и аннотации для его идентификации. В разделе представлены две вкладки: «Схема openAPI», предназначенная для описания спецификации значений в формате JSON, и «Шаблон ресурсов проекта», где можно определить ресурсы, совместимые с Helm, для управления окружением проекта.

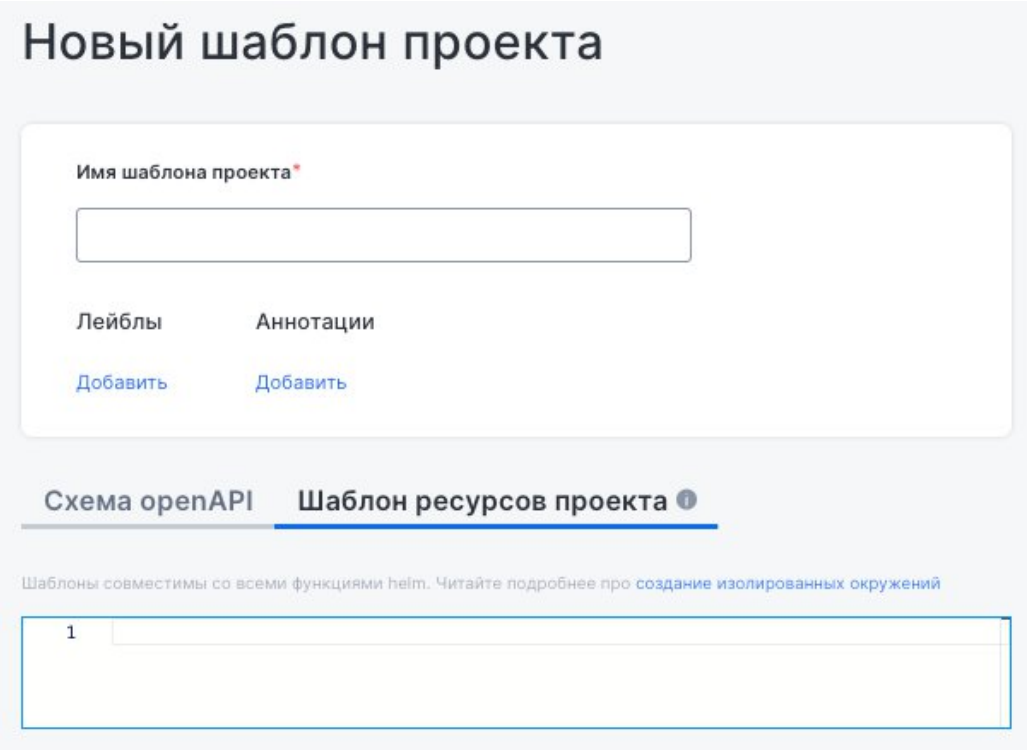


Рисунок 82 – Форма «Новый шаблон проекта»

6.3.5.3.2. Подраздел «Проекты»

Подраздел «Проекты» предназначен для формирования нового проекта на основе заранее подготовленного шаблона, который определяет создаваемые ресурсы и их параметры. В процессе создания происходит валидация параметров по OpenAPI, рендеринг шаблона через Helm и развертывание всех описанных ресурсов внутри автоматически создаваемого Namespace. Проект использует механизмы Kubernetes для контроля доступа, ограничения ресурсов и настройки сетевой изоляции, что позволяет управлять безопасностью и нагрузкой в рамках Namespace. Это меню предоставляет удобный интерфейс для настройки проекта, выбора шаблона и передачи параметров для корректной интеграции ресурсов. Для создания проекта используется кнопка «Создать проект».

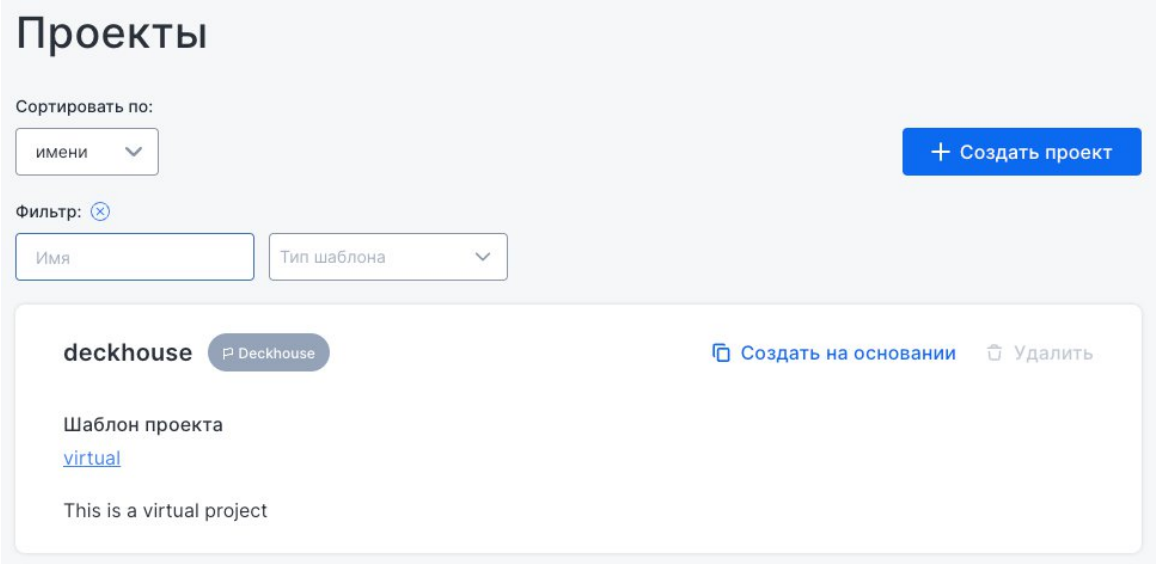


Рисунок 83 – Подраздел «Проекты»

Форма «Новый проект» предназначена для создания проекта на основе выбранного шаблона. Пользователь должен задать имя проекта, а также при необходимости добавить лейблы и аннотации. В центральной части формы выбирается шаблон проекта, на основе которого будут созданы необходимые ресурсы, и можно оставить комментарий. В нижнем блоке предусмотрены поля для ввода параметров, требуемых для работы шаблона, а также для отображения его структуры. Этот интерфейс позволяет удобно настраивать новый проект, обеспечивая его соответствие заданному шаблону.

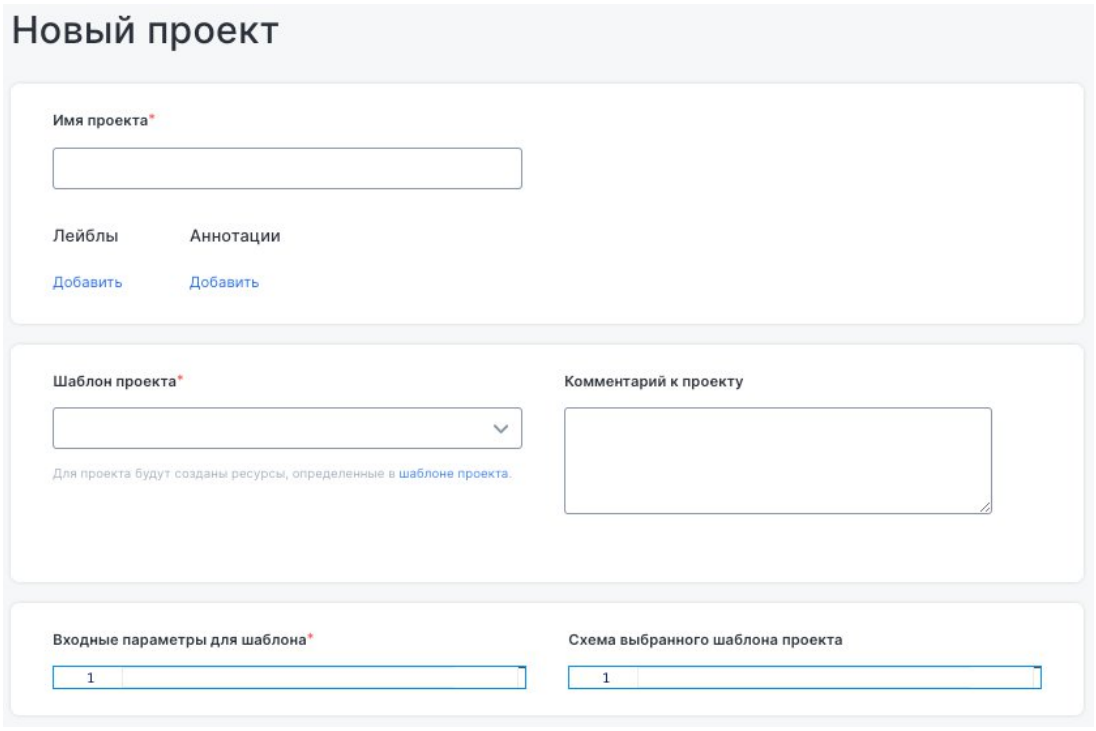


Рисунок 84 – Форма «Новый проект»

6.3.5.4. Раздел «Сеть»

6.3.5.4.1. Подраздел «Ингресс-контроллеры»

Подраздел «Ингресс-контроллеры» предоставляет информацию о текущих ингресс-контроллерах, обеспечивающих маршрутизацию трафика внутри кластера. Интерфейс позволяет сортировать список контроллеров и добавлять новые. В карточке контроллера «nginx» отражены его основные параметры, такие как тип входящего подключения (LoadBalancer), IP-адрес, класс ингресса (nginx) и уровень доступа к балансировщику. Также указан селектор узлов, определяющий, на каких нодах работает контроллер. В нижней части представлены графики мониторинга загрузки процессора, 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 ингресс-контроллера.

Кнопка «Добавить» предоставляет пользователю возможность выбрать тип нового входящего подключения (инлета) для ингресс-контроллера. Доступны несколько вариантов: порт хоста, порт хоста с Proxy Protocol, порт хоста с резервным контроллером, Балансировщик и Балансировщик с Proxy Protocol.

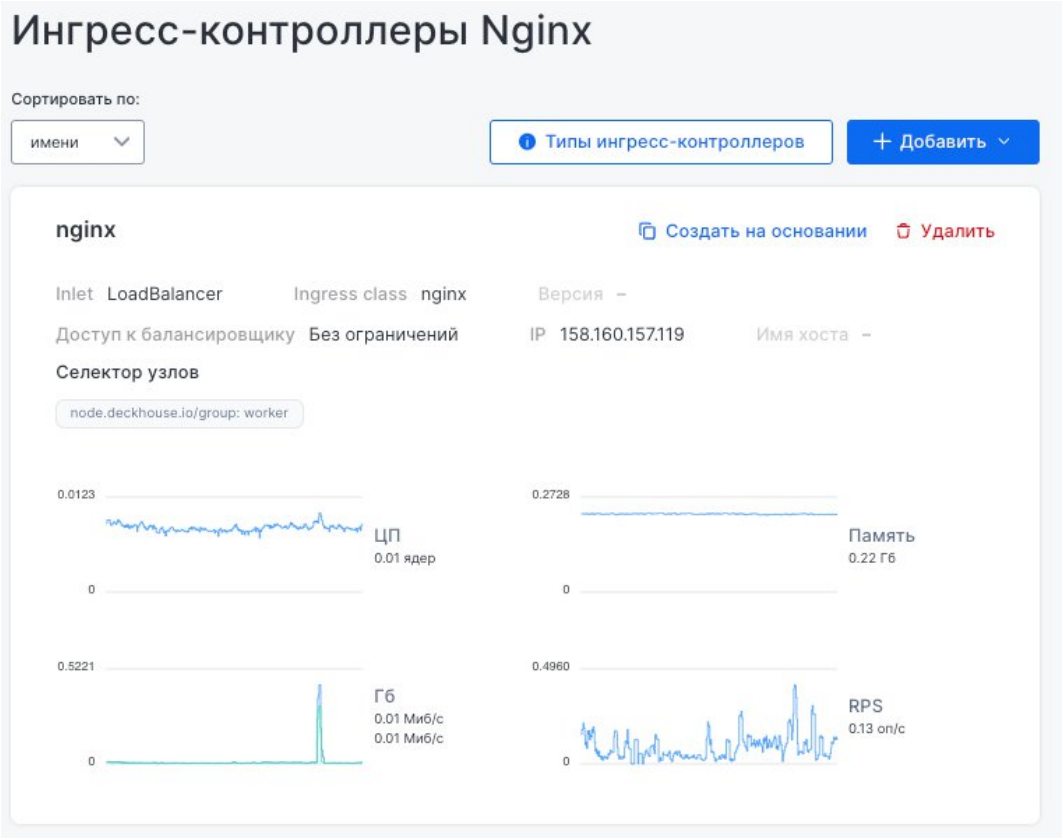


Рисунок 85 – Подраздел «Ингресс-контроллеры»

6.3.5.5. Раздел «Безопасность»

6.3.5.5.1. Подраздел «Сканер CVE»

Подраздел «Сканер CVE» предназначен для проверки контейнерных образов на наличие уязвимостей (CVE) в кластере.

Вкладка «Отчеты об уязвимостях» — отображает результаты последних сканирований. Здесь представлена информация о проверенном объекте, включая его имя, пространство имен, тип и имя ресурса, контейнер, а также используемый образ. Если уязвимости не обнаружены, отображается зеленый индикатор. Также есть возможность выполнить повторное сканирование нажатием кнопки «Пересканировать».

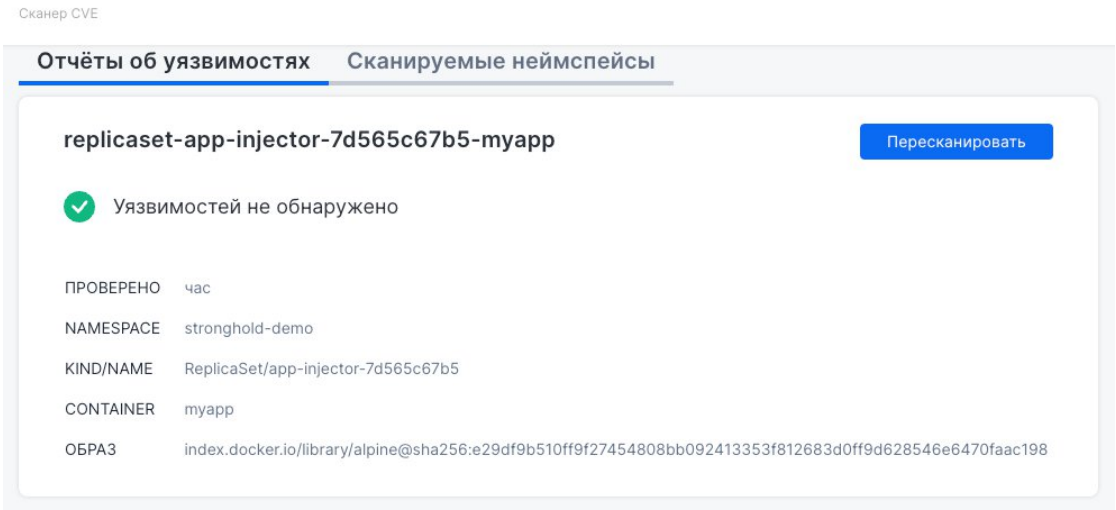


Рисунок 86 – Вкладка «Отчеты об уязвимостях»

Вкладка «Сканируемые неймспейсы» — позволяет управлять пространствами имен, которые подлежат сканированию. Интерфейс поддерживает сортировку по имени и параметрам сканируемости. Опционально можно скрыть системные пространства имен. Пользователь может вручную выбрать нужные пространства имен для сканирования и запустить процесс проверки кнопкой «Пересканировать», а также просмотреть отчеты по каждому объекту.

6.3.5.6. Раздел «Мониторинг»

6.3.5.6.1. Подраздел «Обзор»

Подраздел «Обзор» включает две вкладки: «Состояние» и «Конфигурация», предназначенные для мониторинга и настройки экземпляров Prometheus. Вкладка «Состояние» отображает 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. Также присутствует возможность удаления пода.

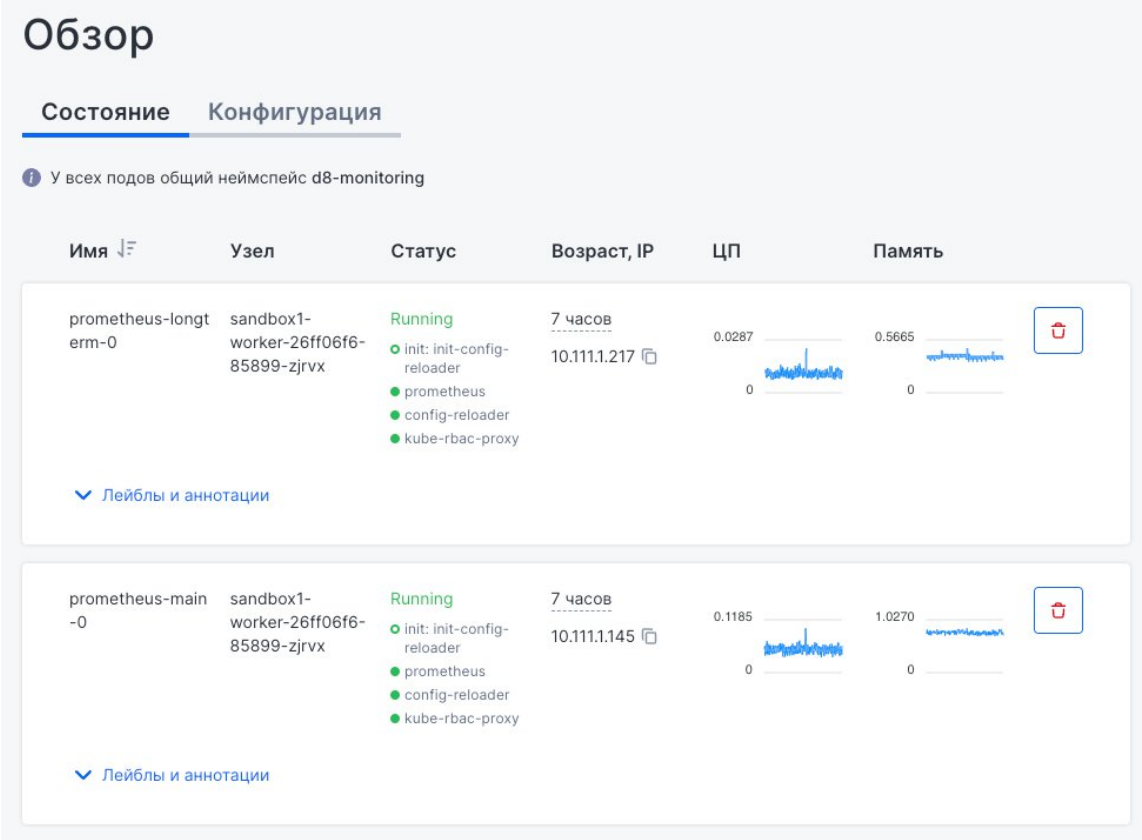


Рисунок 87 – Подраздел «Обзор»

Во 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 позволяет пользователям следить за состоянием метрик в реальном времени и гибко настраивать интеграцию с другими сервисами.

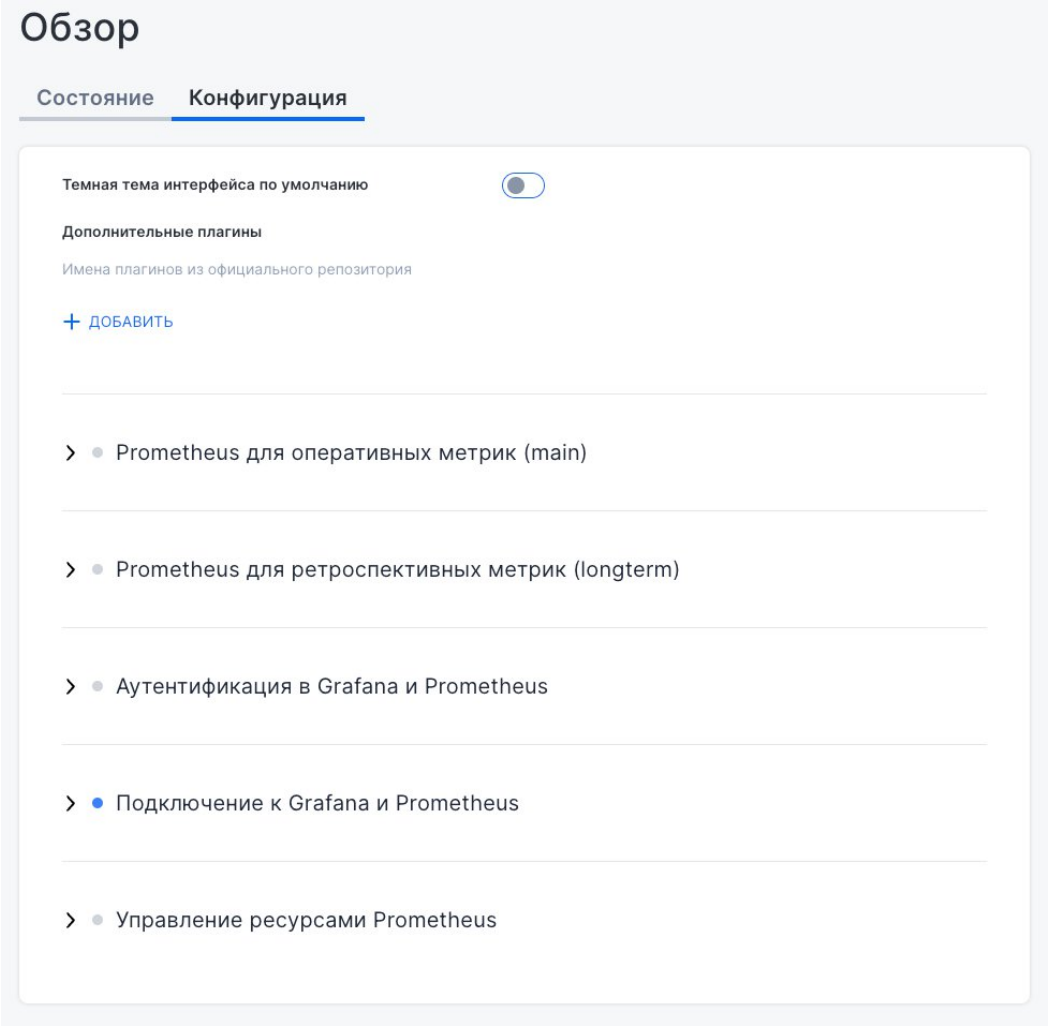


Рисунок 88 – Вкладка «Конфигурация» подраздела «Обзор»

6.3.5.6.2. Подраздел «Обработка метрик»

Подраздел «Обработка метрик» позволяет создавать и управлять правилами обработки метрик. При добавлении нового правила требуется задать его название и указать группу обработки. Это дает возможность организовывать и модифицировать поступающие метрики перед их дальнейшей передачей.

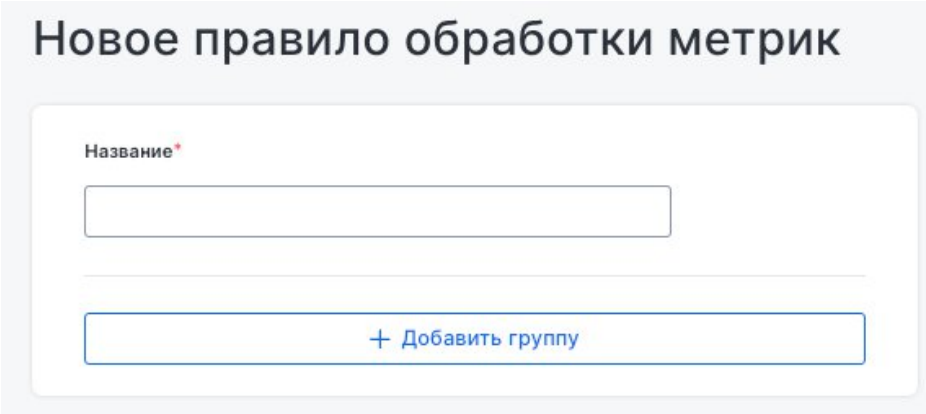


Рисунок 89 – Подраздел «Обработка метрик»

6.3.5.6.3. Подраздел «Отправка метрик»

Подраздел «Отправка метрик» используется для настройки экспорта данных в локальный или внешний сервер Prometheus. В процессе добавления нового ресурса указывается URL для отправки метрик, а также настраиваются параметры TLS, аутентификация и возможность предварительной обработки метрик перед отправкой.

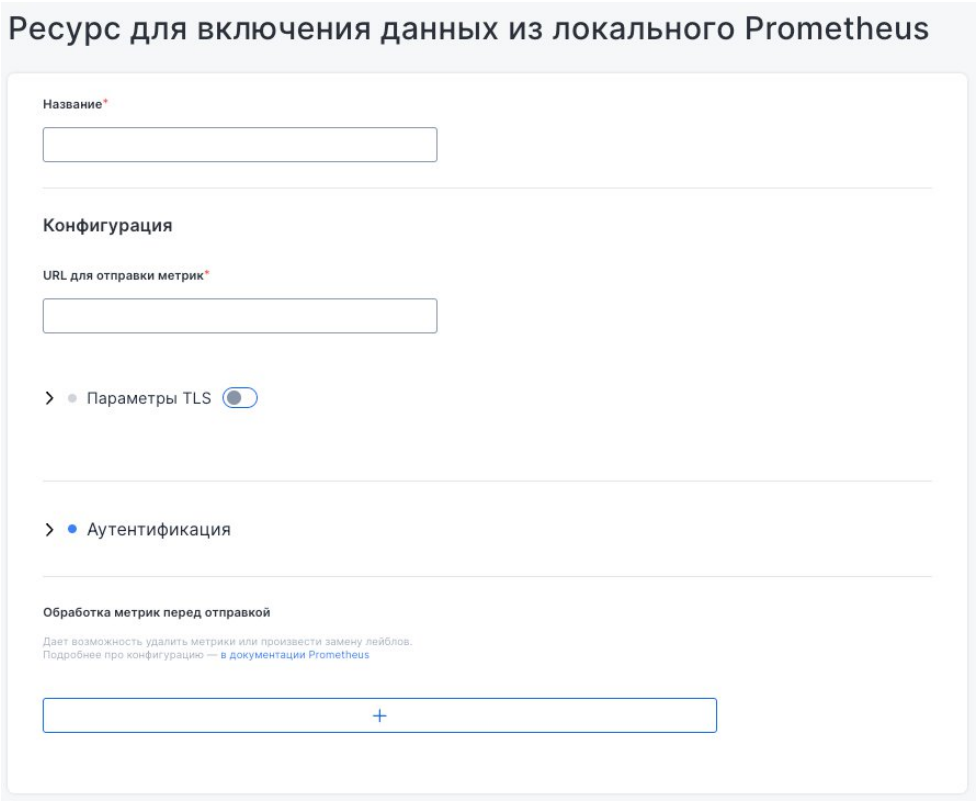


Рисунок 90 – Подраздел «Отправка метрик»

6.3.5.6.4. Подраздел «Источники для Grafana»

Подраздел «Источники для Grafana» предоставляет возможность интеграции с различными источниками данных, используемыми в дашбордах.

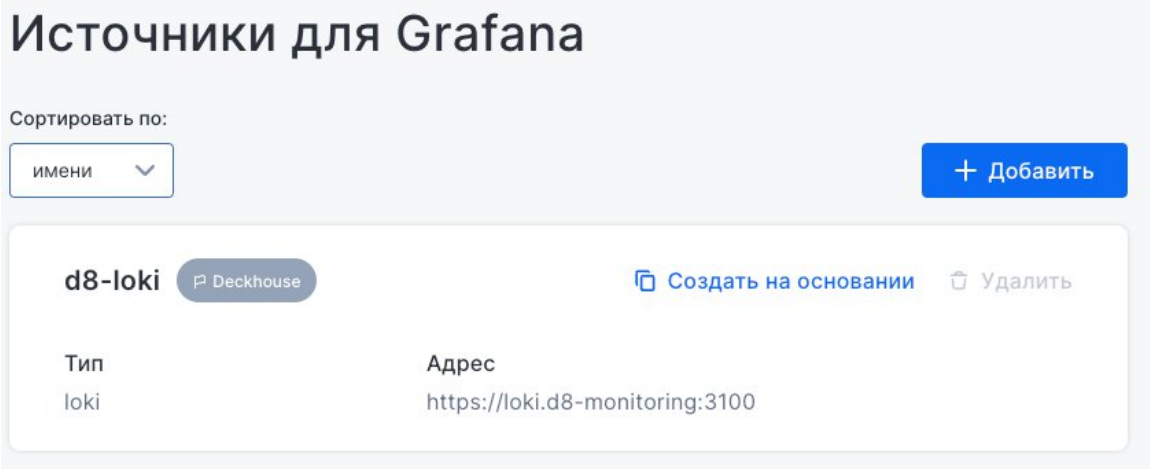


Рисунок 91 – Подраздел «Источники для Grafana»

При создании нового источника данных необходимо задать его название, тип, URL, параметры доступа и настройки аутентификации. Это позволяет подключать Grafana к нужным хранилищам метрик и визуализировать данные.

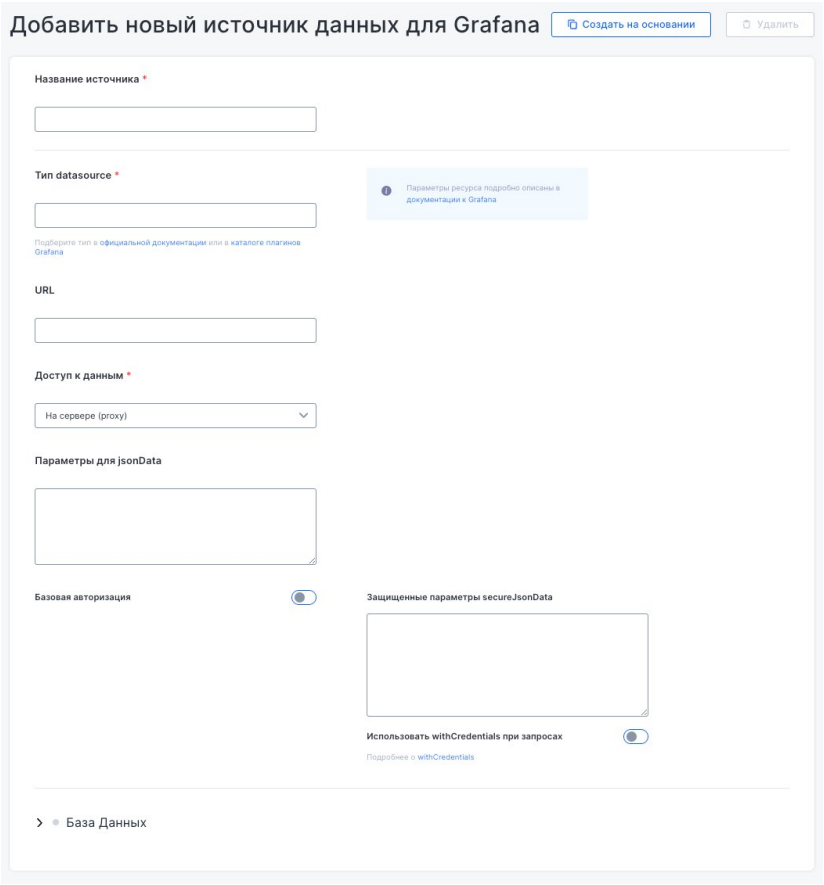


Рисунок 92 – Добавление нового источника для Grafana

6.3.5.6.5. Подраздел «Дашборды для Grafana»

Подраздел «Дашборды для Grafana» предназначен для управления дашбордами, используемыми для визуализации метрик. В основном интерфейсе отображается список доступных дашбордов с возможностью сортировки по времени создания, фильтрации по имени или каталогу. Каждый дашборд имеет название, принадлежность к папке и возможность создания на его основе нового экземпляра или удаления.

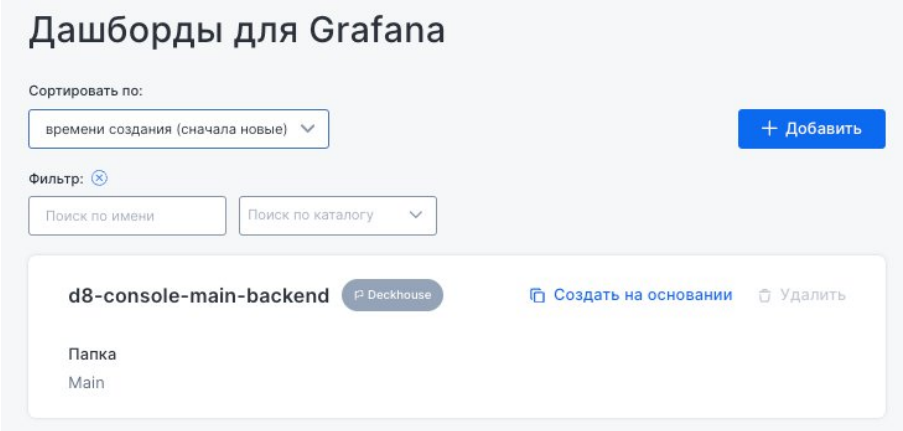


Рисунок 93 – Подраздел «Дашборды для Grafana»

При добавлении нового дашборда требуется задать его название и папку, в которой он будет храниться (если папка не существует, она будет автоматически создана). Внизу формы присутствует поле для ввода JSON-манифеста, содержащего описание конфигурации дашборда. Важно, чтобы в манифесте не было локального id, кроме uid, так как это может повлиять на корректность отображения в Grafana.

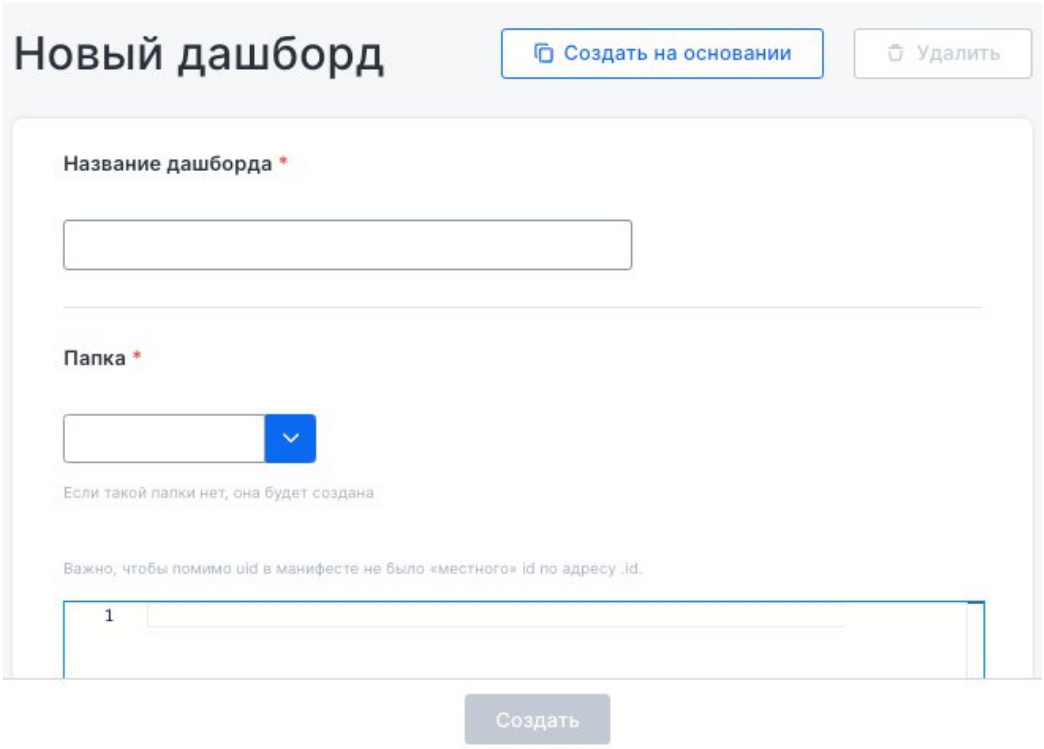


Рисунок 94 – Добавление нового дашборда для Grafana

6.3.5.6.6. Подраздел «Активные алерты»

Подраздел «Активные алерты» отображает список текущих предупреждений в системе мониторинга. Интерфейс позволяет сортировать алерты по имени и фильтровать их по статусу или названию, что упрощает поиск нужного уведомления.

Каждый алерт содержит название, уровень критичности, информацию о времени создания и последнего обновления. Также указываются связанные компоненты и модули, что помогает определить источник проблемы.

Для получения подробной информации по алерту доступна кнопка «Читать описание», а внизу карточки присутствует пояснение о причине срабатывания уведомления. Этот раздел предназначен для оперативного мониторинга проблем в кластере и быстрого реагирования на критические события.

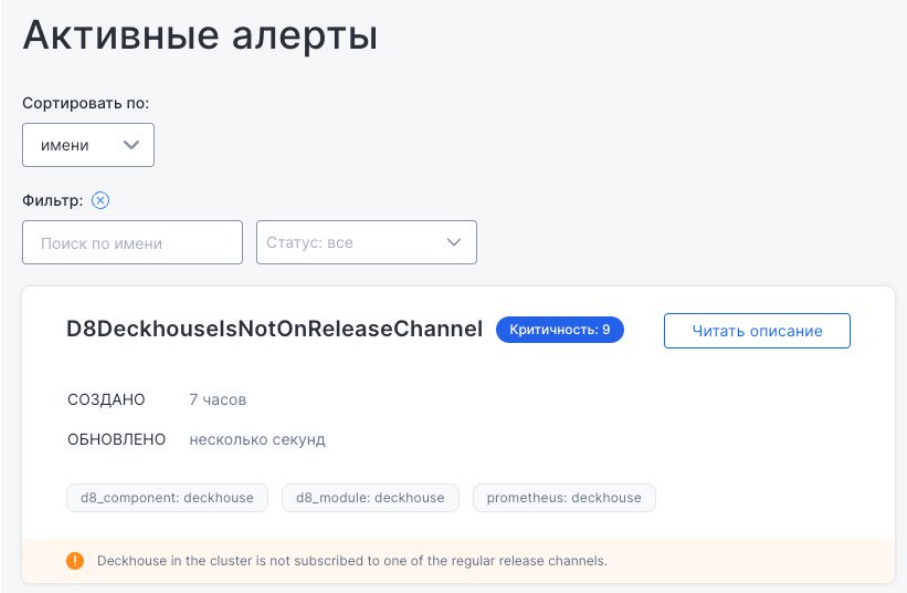


Рисунок 95 – Подраздел «Активные алерты»

6.3.5.7. Раздел «Журналирование»

6.3.5.7.1. Подраздел «Отправка логов»

Подраздел «Отправка логов» предназначен для управления логированием и настройкой отправки логов в различные хранилища. В основном интерфейсе отображается список доступных конфигураций, с возможностью сортировки по имени и фильтрации по типу. Кнопка «Добавить» открывает выпадающее меню с выбором целевого хранилища, включая Loki, ElasticSearch, Logstash, Vector, Kafka и Splunk.

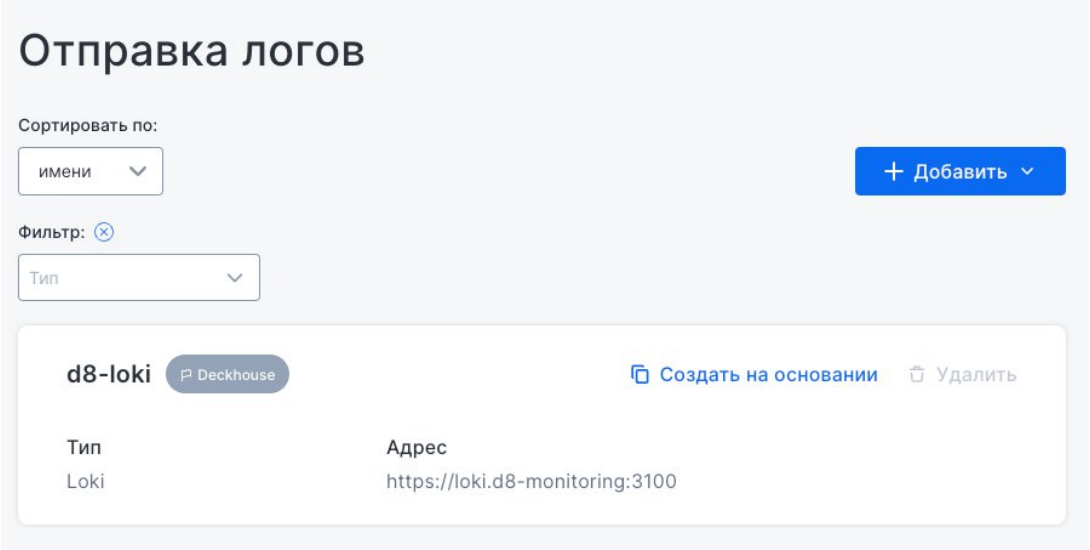


Рисунок 96 – Подраздел «Отправка логов»

При добавлении нового хранилища логов (на примере Loki) необходимо задать его название и адрес подключения. Доступны 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, которые позволяют фильтровать определенные логи.

6.3.5.7.2. Подраздел «Сбор логов»

Подраздел «Сбор логов» предназначен для настройки источников логов, которые затем могут быть отправлены в целевые хранилища. Интерфейс позволяет сортировать и фильтровать существующие правила сбора логов, а также добавлять новые источники. В выпадающем меню кнопки «Добавить» представлены два типа источников: File (сбор логов из файловой системы) и KubernetesPods (сбор логов из подов Kubernetes).

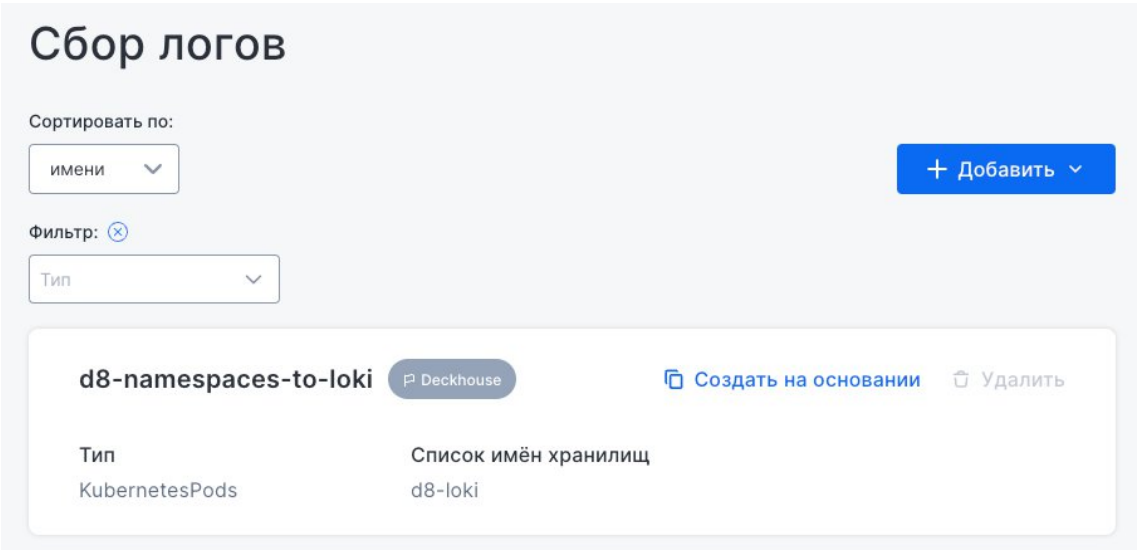


Рисунок 97 – Подраздел «Сбор логов»

При добавлении нового правила сбора логов (например, File) требуется задать название, а затем настроить параметры:

- Хранилище — указывается, куда будут отправляться собранные логи.
- Фильтр файлов — задаются пути к файлам логов, которые необходимо или, наоборот, не нужно считывать. Поддерживаются подстановки (wildcards).
- Разделитель строк — можно задать символ, разделяющий записи в файле.

- Фильтрация логов — можно добавить правила по лейблам и фильтры, чтобы сохранялись только нужные записи.

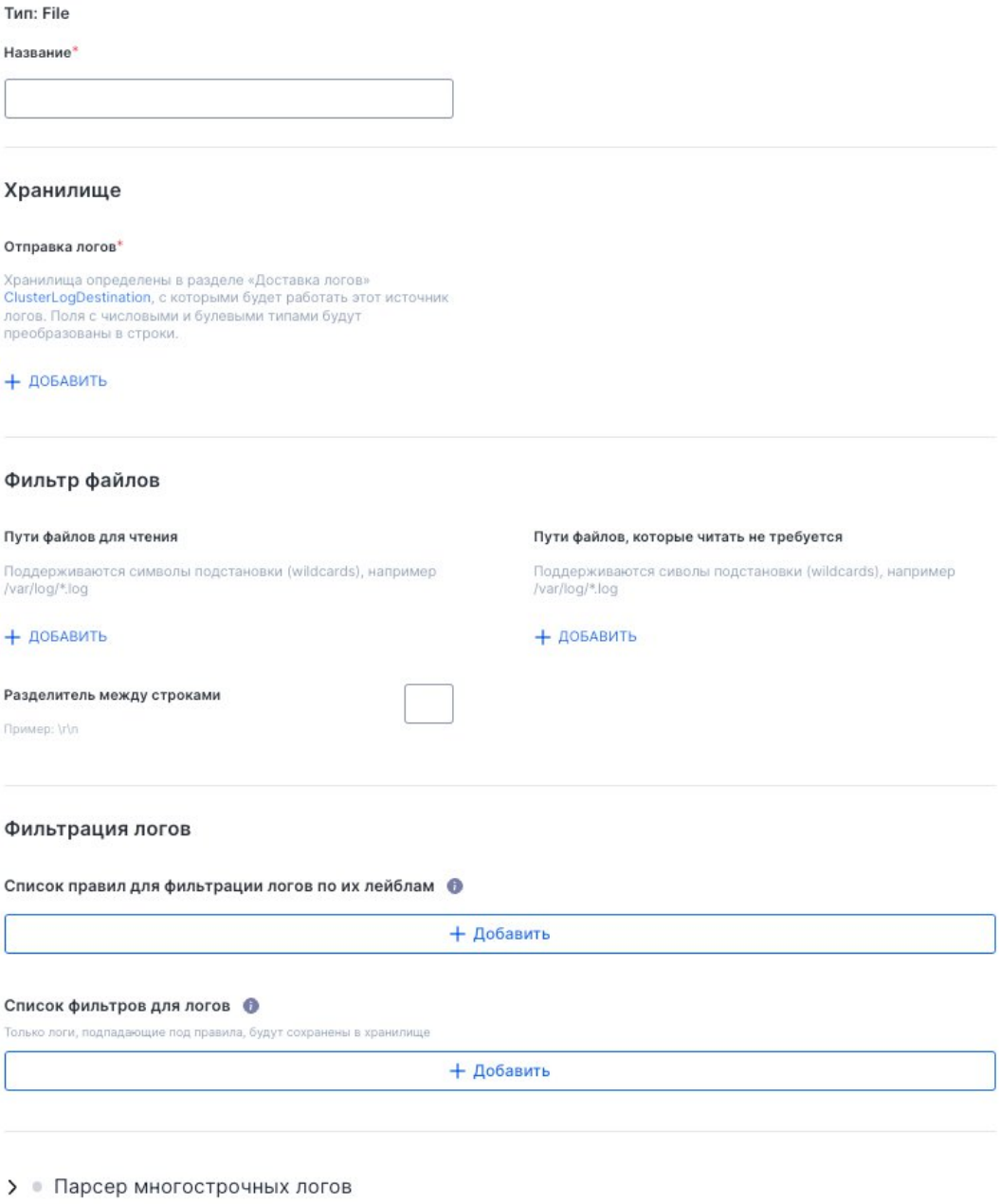


Рисунок 98 – Добавление нового правила сборки логов

6.3.6. Веб-интерфейс модуля deckhouse-tools

Этот модуль создает веб-интерфейс со ссылками для скачивания утилит ПО «Deckhouse Platform» для различных операционных систем.

Для получения доступа к веб-интерфейсу deckhouse-tools необходимо в адресной строке браузера ввести tools. <ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> –

строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate. Формат адреса подключения к deckhouse-tools может быть иным. Точный адрес подключения можно узнать у администратора безопасности ПО «Deckhouse Platform».

При первом входе в веб-интерфейс появится окно аутентификации, где потребуется ввести учетные данные пользователя. После этого откроется главный экран документации.

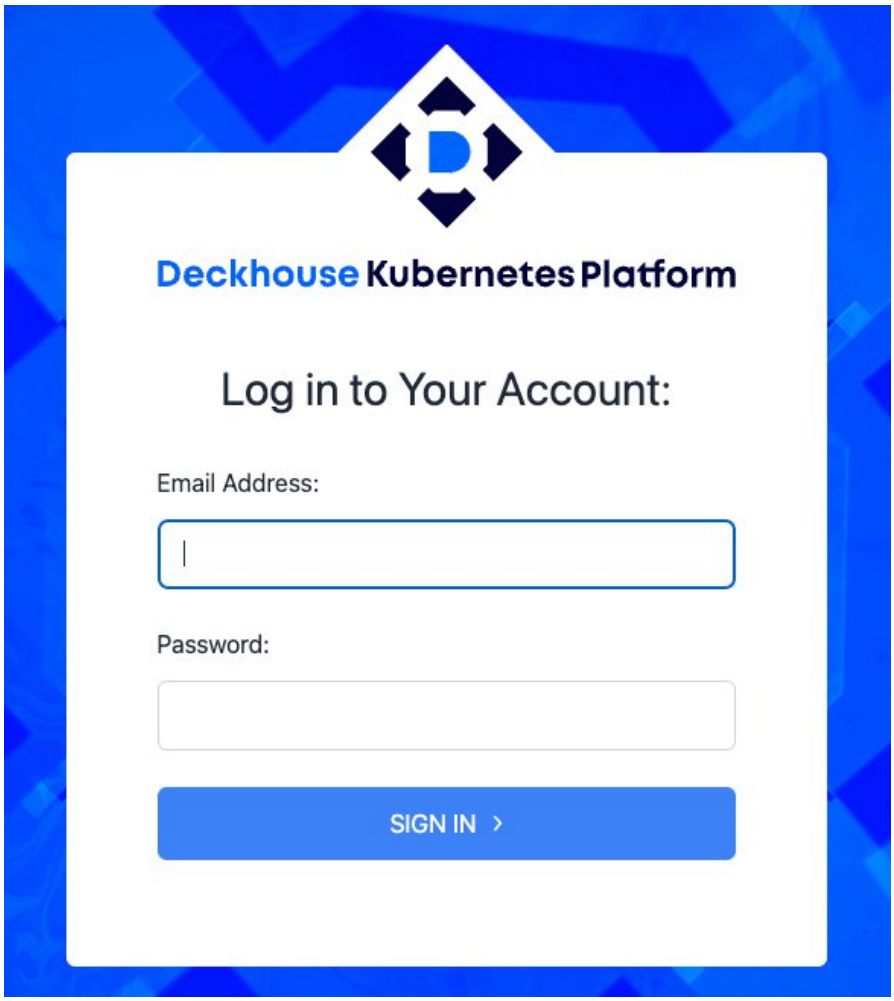


Рисунок 99 – Окно аутентификации веб-интерфейса

Для аутентификации введите учетные данные, полученные от администратора безопасности.

При успешной аутентификации откроется страница веб-интерфейса deckhouse-tools, на которой доступны для загрузки утилиты Deckhouse CLI под разные версии операционных систем.

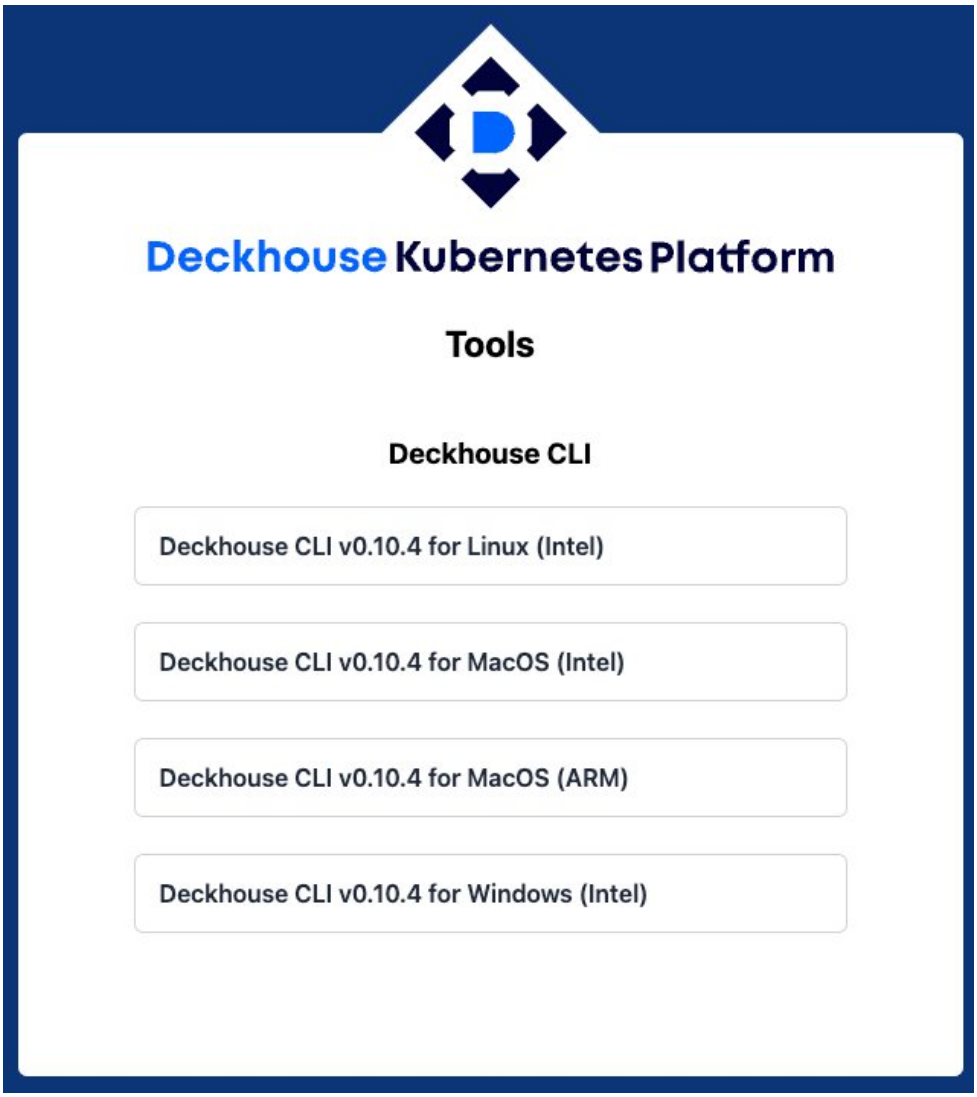


Рисунок 100 – Страница веб-интерфейса deckhouse-tools

6.3.7. Веб-интерфейс модуля stronghold

Интерфейс stronghold доступен по адресу stronghold.<ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> – строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate.

При первом входе потребуется ввести учетные данные пользователя. После этого откроется главный экран Stronghold.

6.3.7.1. Главный экран и работа с механизмами секретов

При переходе по адресу stronghold.<ШАБЛОН_ИМЕН_КЛАСТЕРА> открывается раздел интерфейса для работы с механизмами секретов. Он же — главный экран.

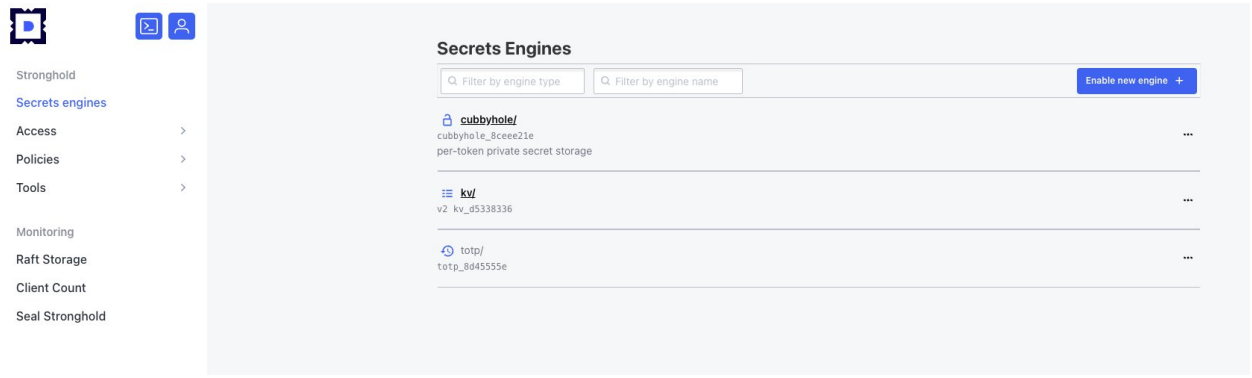


Рисунок 101 – Веб-интерфейс модуля stronghold

В левой части экрана находится окно навигации по основным разделам пользовательского интерфейса. В центре — список механизмов секретов, используемых в кластере и кнопка для добавления нового механизма секретов.

6.3.7.1.1. Просмотр информации о механизме секретов

Кликнув по названию механизма секретов, можно посмотреть информацию о нем и о добавленных в систему секретах. В окне с информацией отображаются вкладки: «Secrets» — со списком секретов (ролей, ключей и т.д., в зависимости от механизма секретов), «Configuration» — с конфигурацией механизма и кнопка для добавления секрета (роли, ключа и т.д., в зависимости от механизма секретов).

Например, для механизма «KV» («Ключ-значение») доступна следующая информация и элементы управления:

- список секретов;
- конфигурация механизма;
- кнопка добавления секрета.

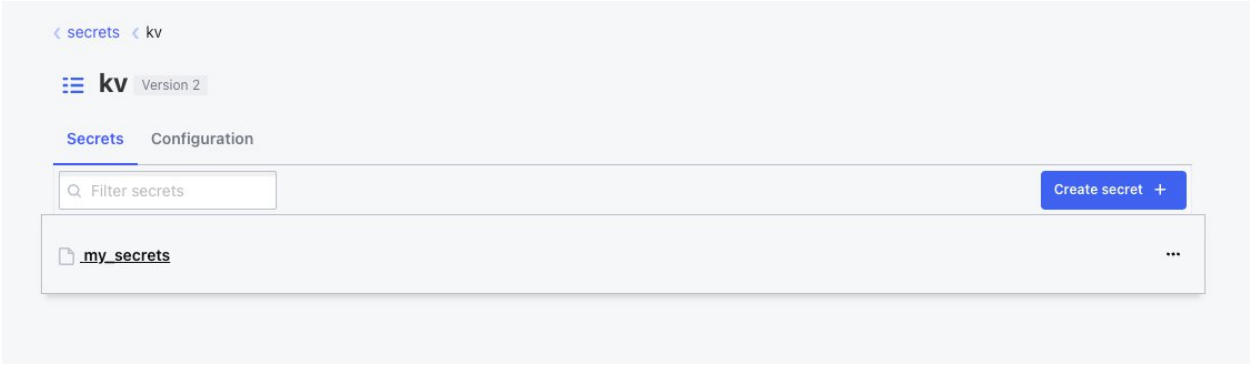


Рисунок 102 – Просмотр информации о механизме секретов

Для просмотра конфигурации механизма секретов необходимо кликнуть по вкладке «Configuration». Содержимое вкладки зависит от просматриваемого механизма секретов.

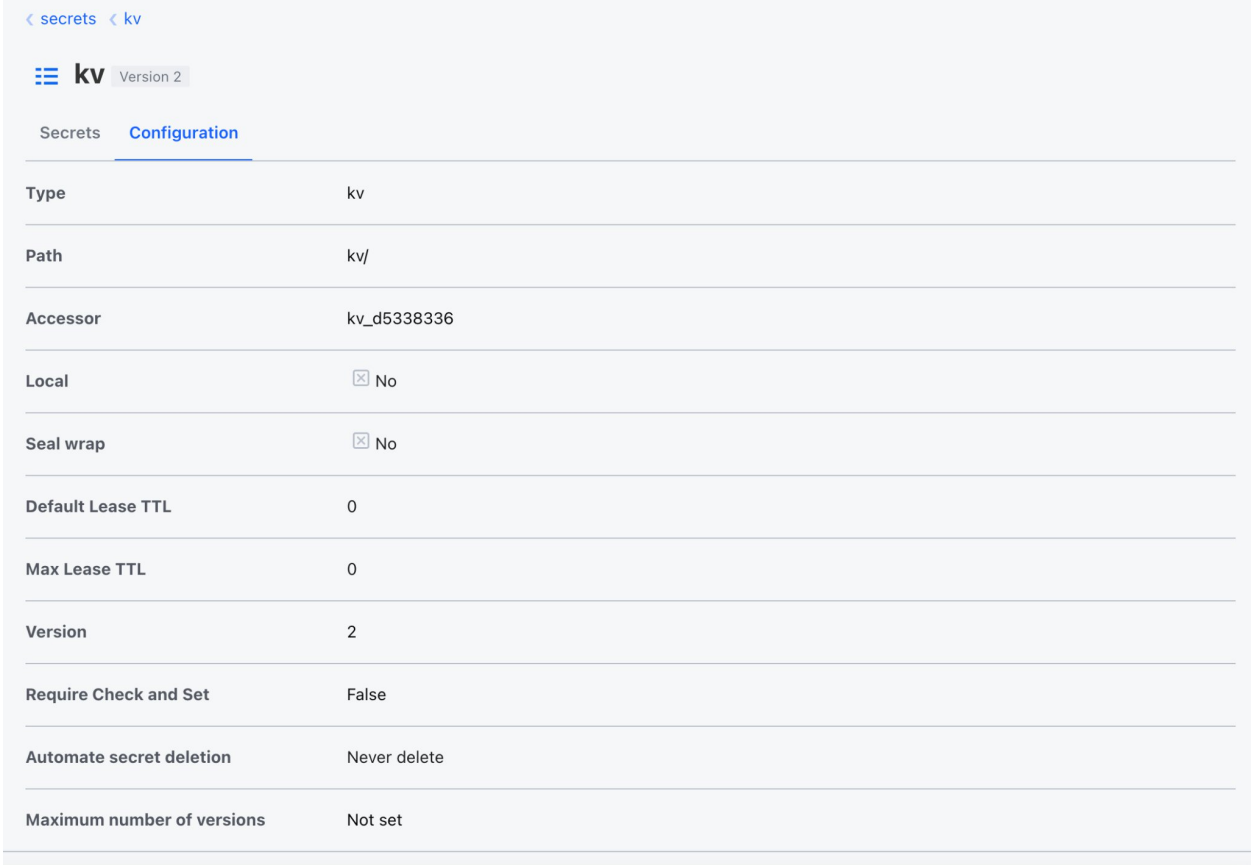


Рисунок 103 – Просмотр конфигурации механизма секретов

6.3.7.1.1.1. Просмотр информации о секрете и его версиях (на примере механизма «Ключ-значение»)

Посмотреть информацию о секрете можно, кликнув по его названию в окне информации о механизме секретов. В окне с информацией о секрете отображается две вкладки: вкладка с общей информацией о секрете и его версиях и вкладка с метаданными секрета.

На вкладке «Secret» с общей информацией о секрете отображается переключатель для просмотра сведений о секрете в формате JSON.

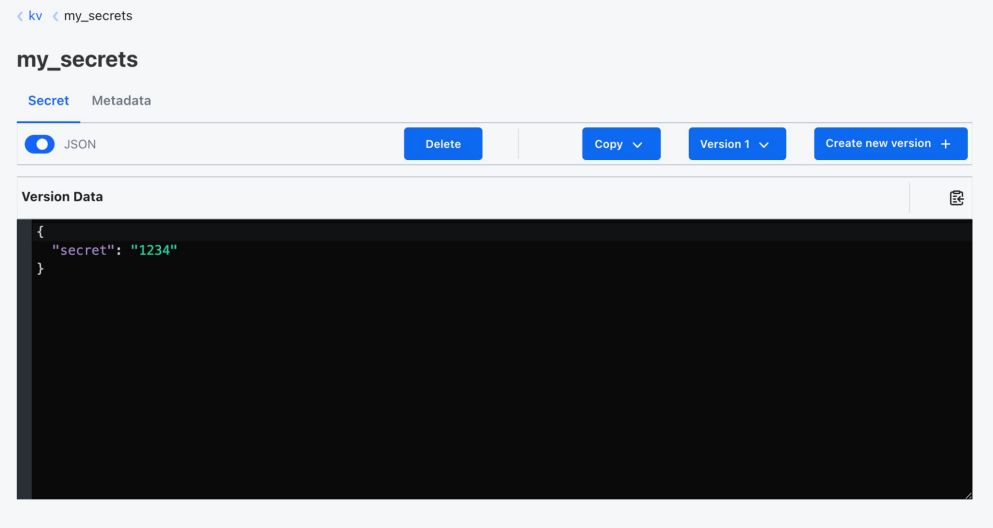


Рисунок 104 – Просмотр информации о секрете и его версиях

Также на вкладке «Secret» с общей информацией о секрете отображаются кнопки для работы с секретом и его версиями:

- удаление;
- копирование;
- выбор версии (не для всех механизмов секретов);
- добавление новой версии.

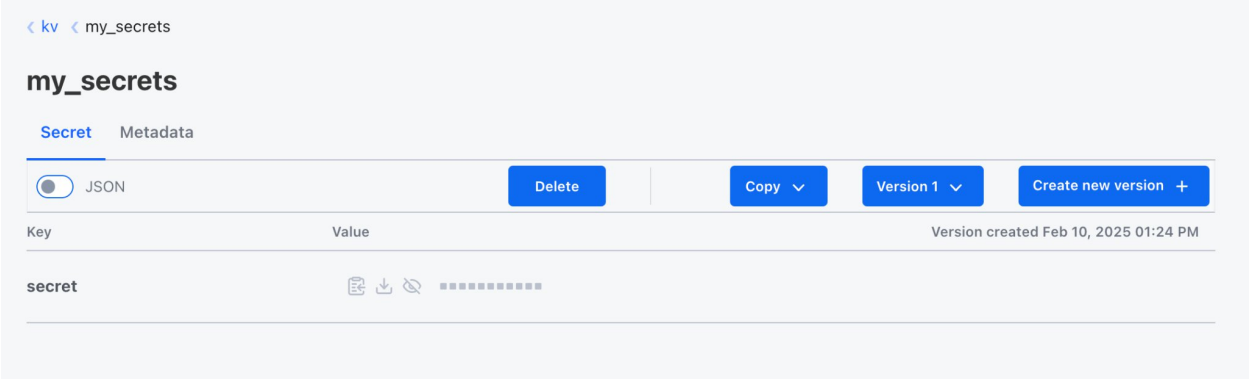


Рисунок 105 – Вкладка «Secret»

Для просмотра метаданных секрета необходимо кликнуть по вкладке «Metadata». После этого отобразится окно для просмотра и редактирования метаданных секрета. На вкладке отображаются метаданные секрета, кнопка для их редактирования и ссылка для добавления пользовательских метаданных.

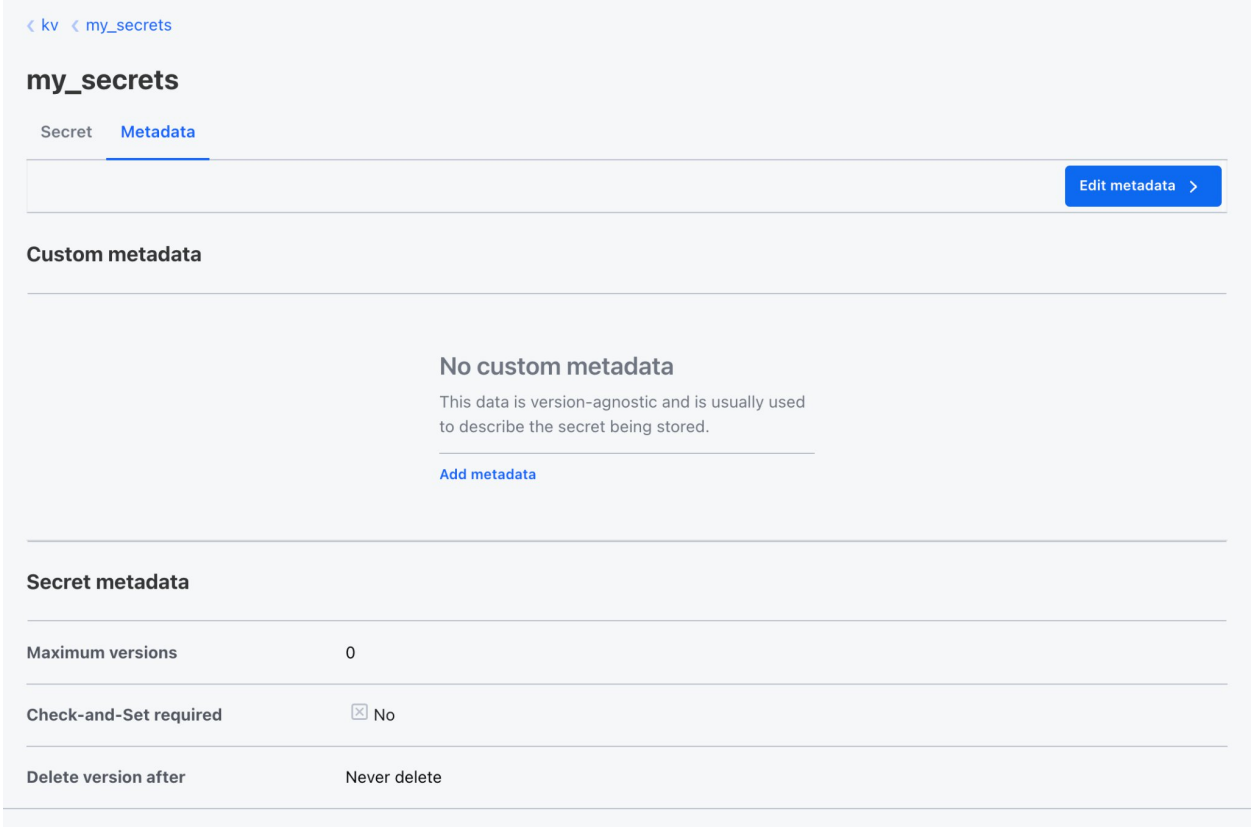


Рисунок 106 – Просмотр метаданных секрета

6.3.7.1.1.2. Добавление секрета

Добавить новый секрет можно, кликнув по кнопке для добавления секрета (роли, ключа и т.д. — название кнопки зависит от механизма секретов) в окне с информацией о механизме секретов.

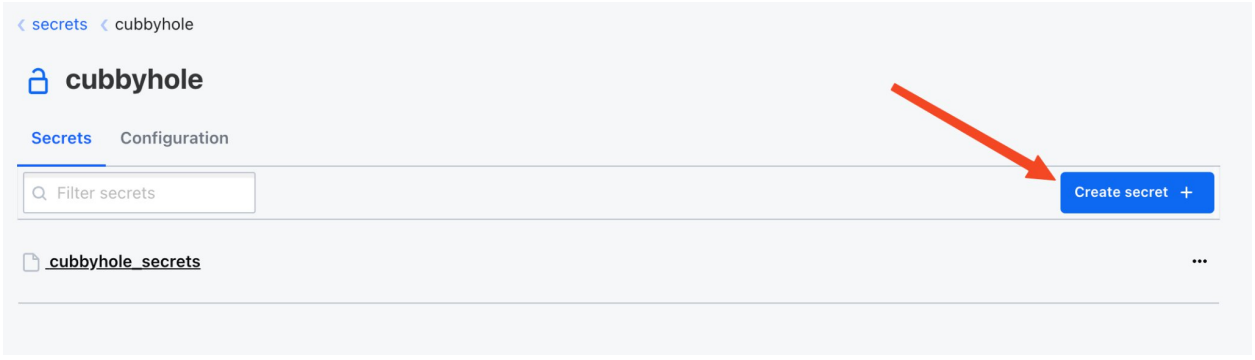


Рисунок 107 – Добавление секрета

После нажатия кнопки откроется форма добавления секрета (роли, ключи и т.д. — название кнопки зависит от механизма секретов). Состав формы 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»).

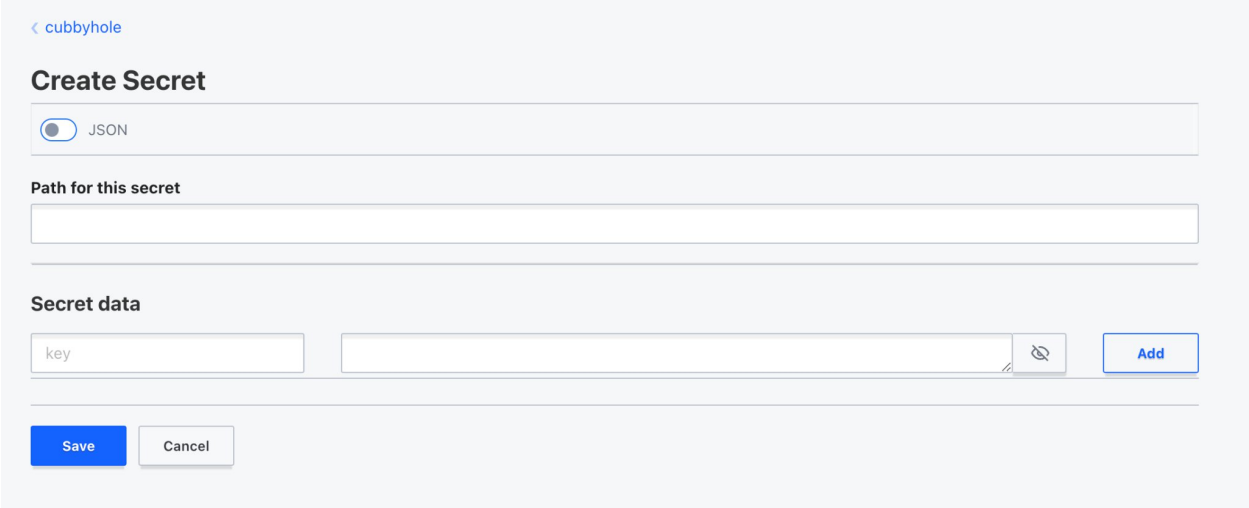


Рисунок 108 – Форма добавления секрета

6.3.7.1.2. Добавление механизма секретов

Чтобы добавить механизм секретов, необходимо нажать кнопку «Enable new engine» на главном экране. После этого откроется экран выбора типа добавляемого механизма секретов. На нем необходимо выбрать нужный механизм и нажать кнопку «Next».

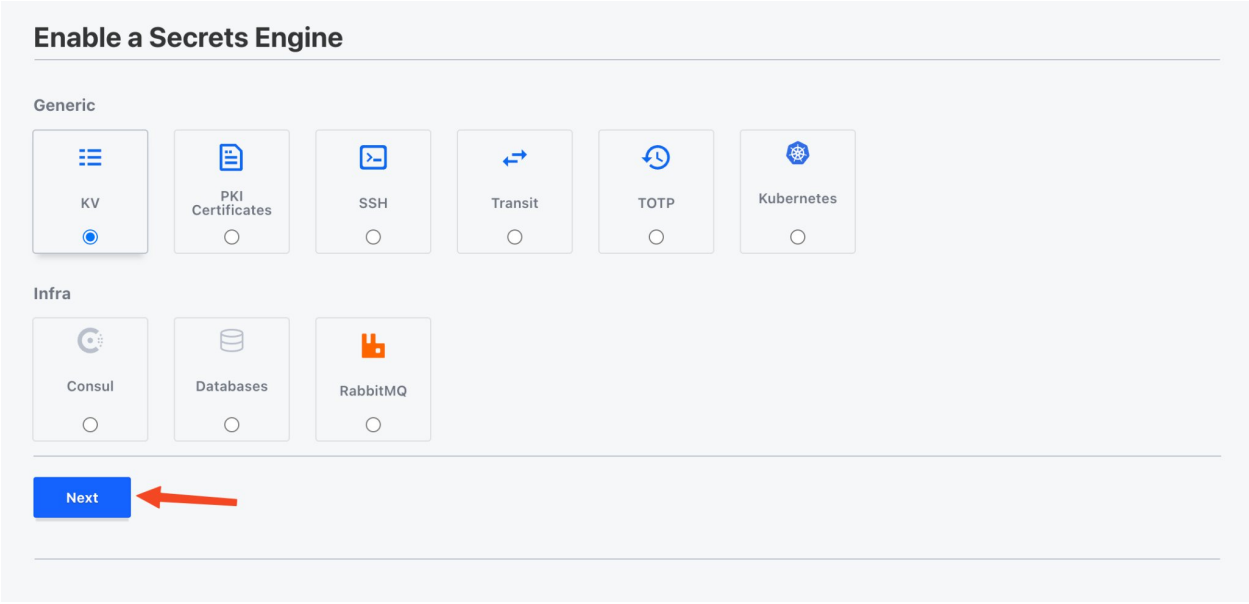


Рисунок 109 – Добавление механизма секретов

После этого откроется окно с настройками добавляемого механизма секретов. Оно состоит из двух блоков: основных настроек (различаются в зависимости от добавляемого механизма секретов) и опций («Method options» — по умолчанию блок свернут, чтобы открыть его, нужно кликнуть по его названию). Внизу окна находятся кнопки «Enable Engine» — для сохранения механизма секретов после его настройки и «Back» для возврата без сохранения на экран выбора механизма секретов.

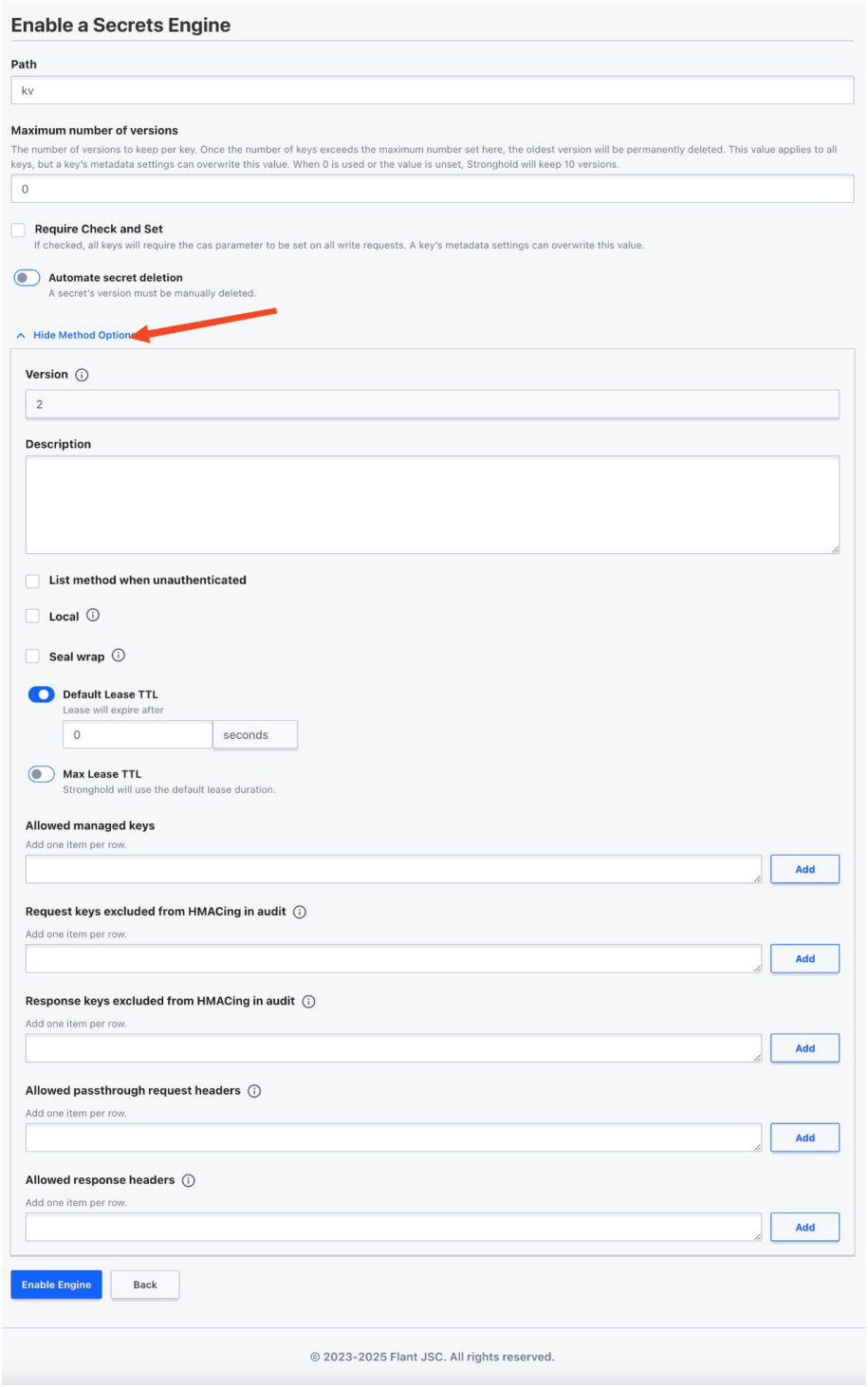


Рисунок 110 – Окно с настройками добавляемого механизма секретов

6.3.7.2. Управление доступом к данным и функциям stronghold

Управление доступом к данным и функциям stronghold осуществляется в разделе «Access». Перейти в него можно, кликнув по пункту меню «Access» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). 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»)).

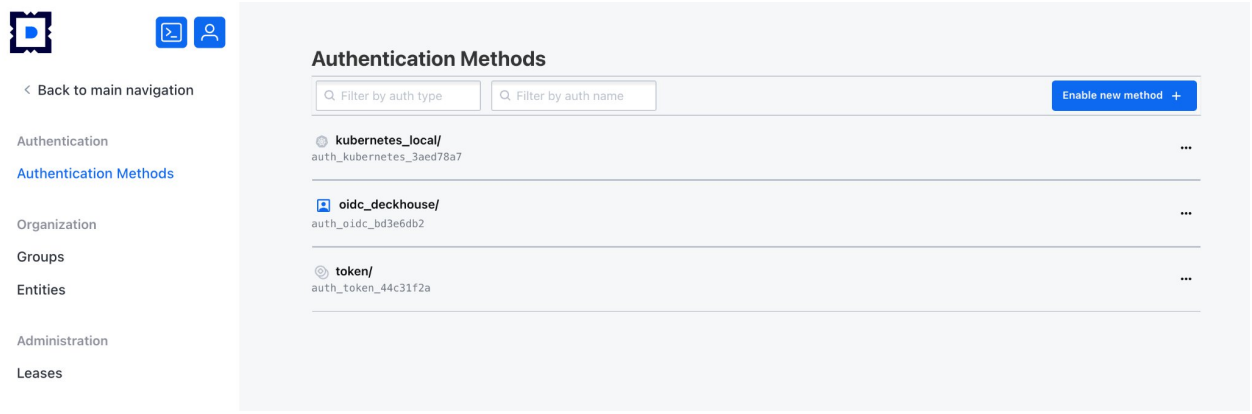


Рисунок 111 – Управление доступом к данным и функциям stronghold

6.3.7.2.1. Работа с методами аутентификации

Подраздел для работы с методами аутентификации открывается по умолчанию при переходе в раздел «Access» с главного экрана. Для перехода в подраздел из других подразделов необходимо кликнуть по пункту «Authentication Methods» в меню слева.

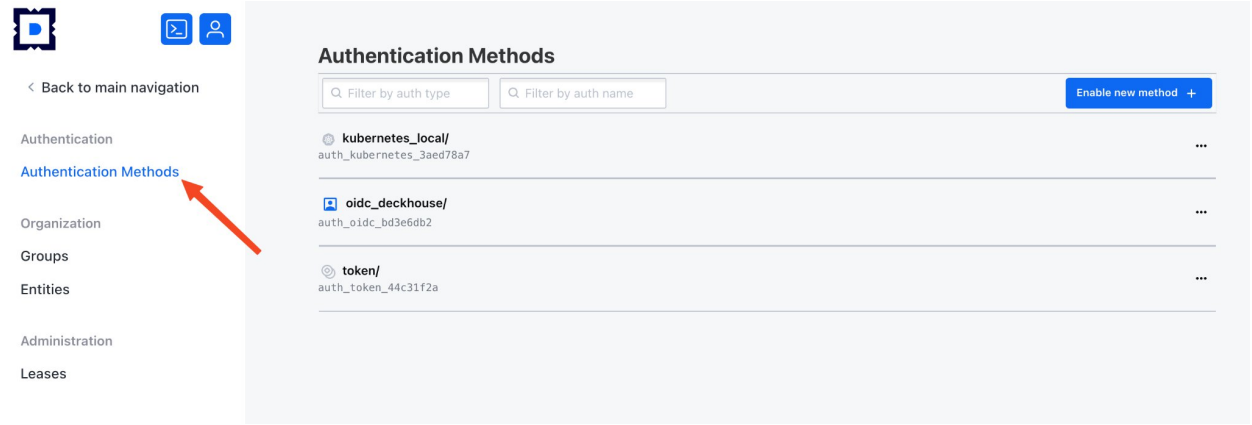


Рисунок 112 – Работа с методами аутентификации

В центре экрана находится список методов аутентификации, используемых в кластере, поля для фильтрации элементов списка и кнопка для добавления нового метода.

Для методов из списка доступны следующие действия:

- просмотр конфигурации;
- изменение конфигурации;
- удаление метода.

Выбрать нужное действие можно, кликнув по кнопке с тремя точками, которая находится в конце строки с названием метода.

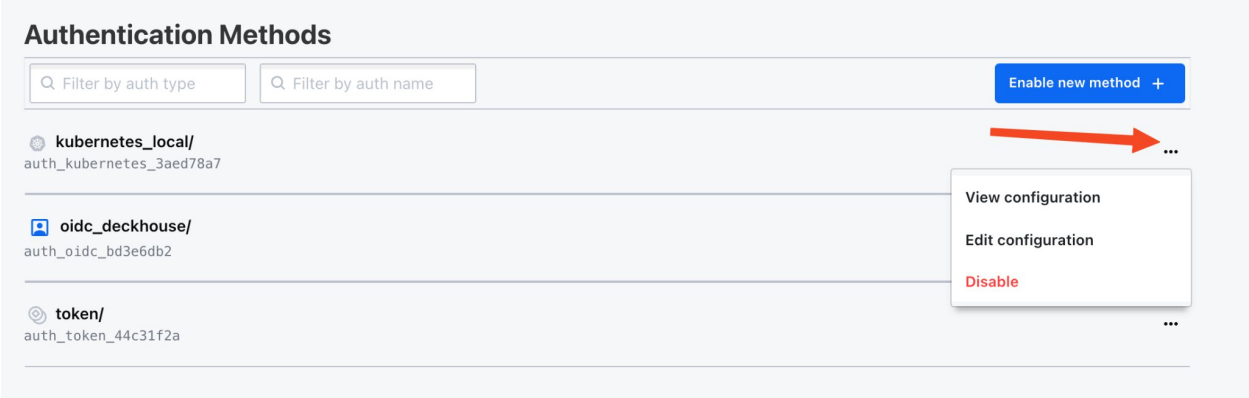


Рисунок 113 – Список методов аутентификации

6.3.7.2.1.1. Просмотр информации о методе аутентификации

Информацию о методе аутентификации можно посмотреть, кликнув по его названию или выбрав пункт «View configuration» (кликнув по кнопке с тремя точками, которая находится в конце строки с названием метода). В окне с информацией о методе аутентификации отображается одна или две вкладки (количество и содержимое вкладок зависит от метода аутентификации) 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».

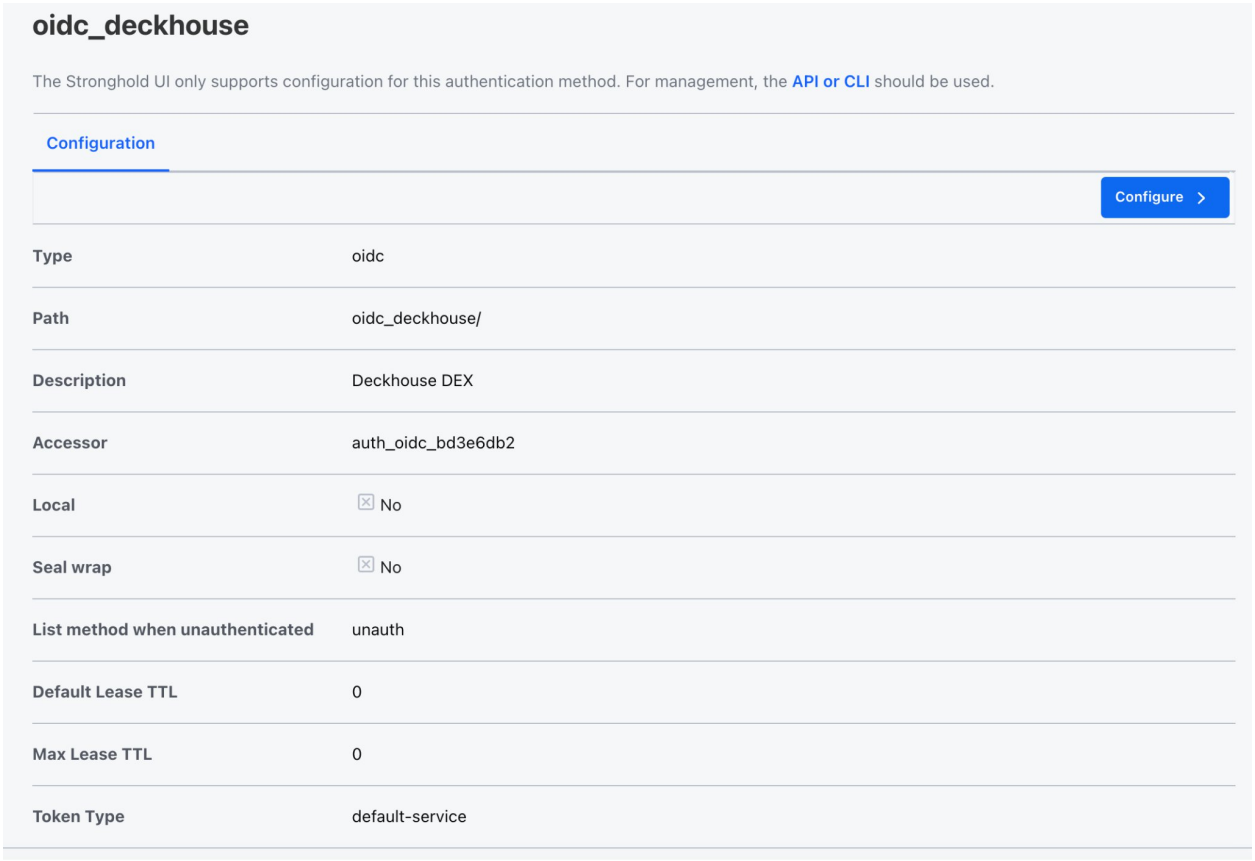


Рисунок 114 – Просмотр информации о методе аутентификации

6.3.7.2.1.2. Добавление метода аутентификации

Добавить метод аутентификации можно, нажав кнопку для добавления метода в окне для работы с методами аутентификации (п. 6.3.7.2.1).

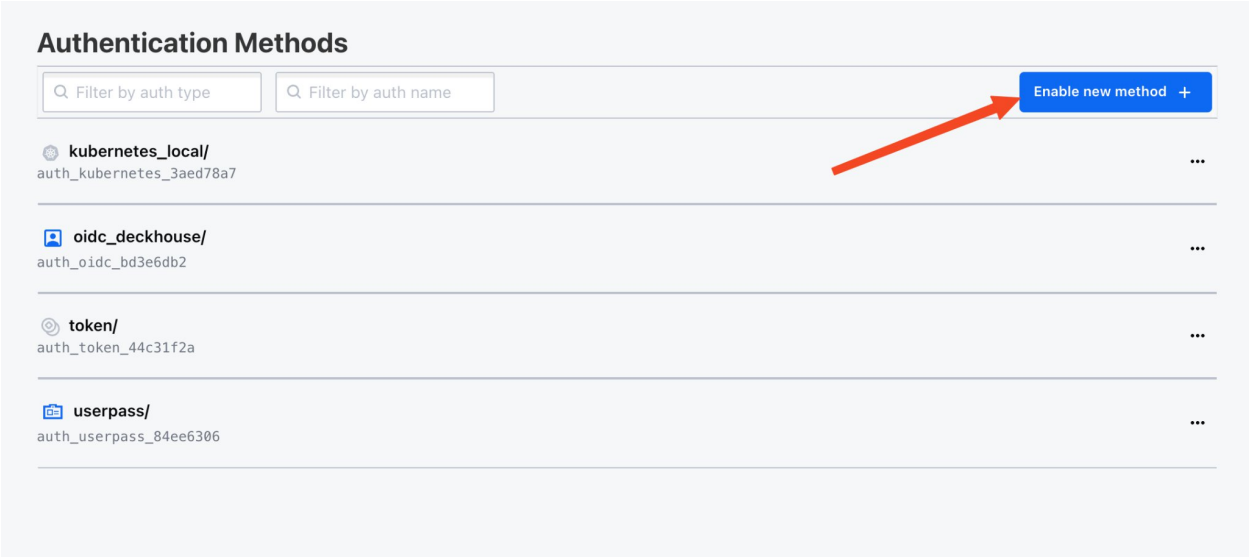


Рисунок 115 – Добавление метода аутентификации

После этого откроется экран выбора добавляемого метода аутентификации. На нем необходимо выбрать нужный метод и нажать кнопку «Next».

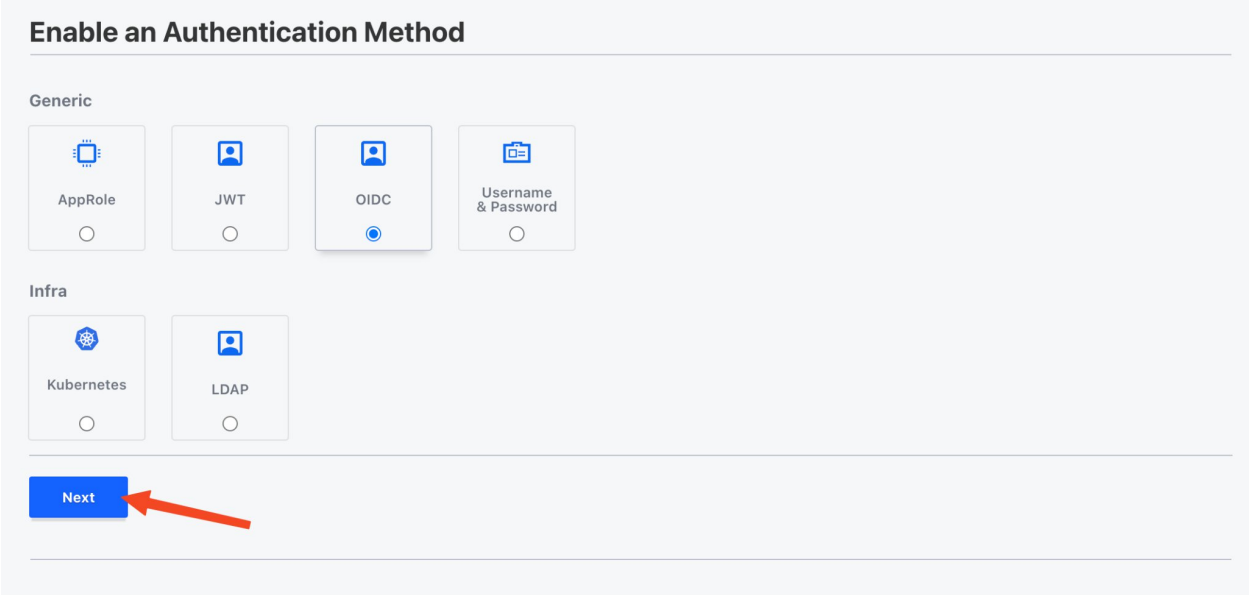


Рисунок 116 – Выбор добавляемого метода аутентификации

После этого откроется окно с настройками добавляемого метода аутентификации. Оно состоит из двух блоков: поле «Path» и опции («Method options» — по умолчанию блок свернут, чтобы открыть его, нужно кликнуть по его названию). Внизу окна находятся кнопки «Enable Method» — для сохранения метода после его настройки и «Back» для возврата без сохранения на экран выбора метода аутентификации.

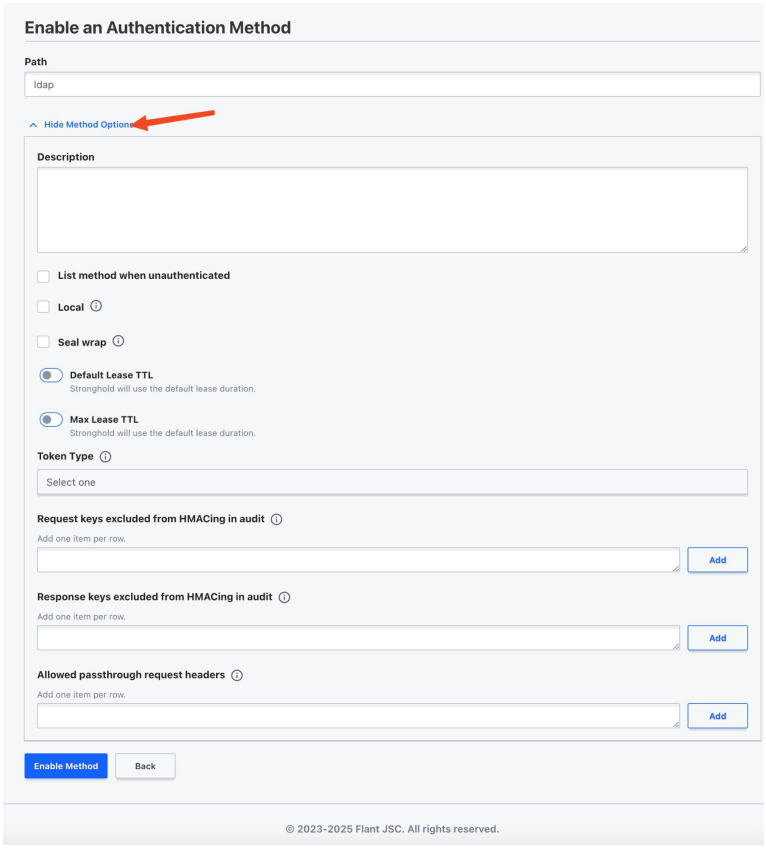


Рисунок 117 – Настройки добавляемого метода аутентификации

6.3.7.2.2. Работа с группами пользователей

Для перехода в подраздел необходимо кликнуть по пункту «Groups» в меню слева.

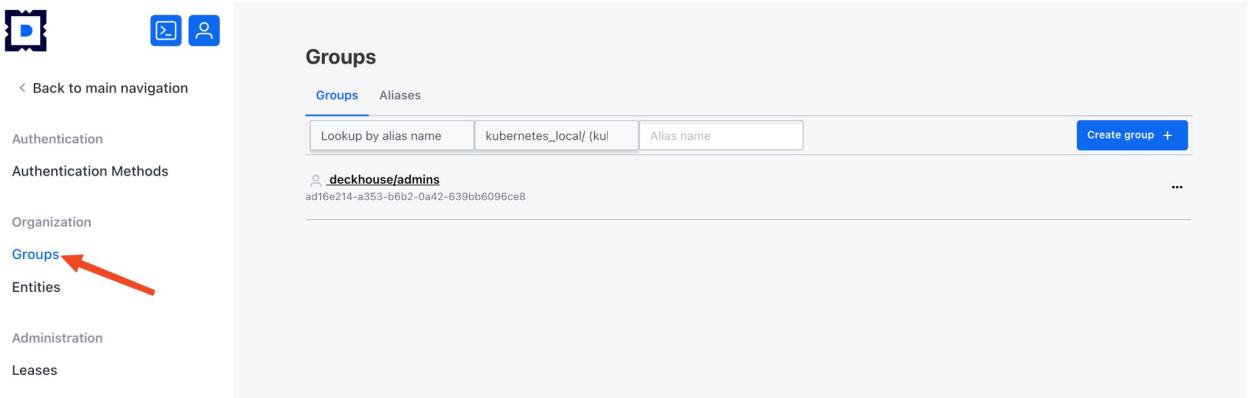


Рисунок 118 – Работа с группами пользователей

В центре экрана находится список групп пользователей, имеющихся в кластере, поля для фильтрации элементов списка и кнопка для добавления новой группы.

Для групп из списка доступны следующие действия:

- просмотр детальной информации о группе;
- изменение настроек группы;

– удаление групп.

Выбрать нужное действие можно, кликнув по кнопке с тремя точками, которая находится в конце строки с названием группы.

Рисунок 119 – Список групп пользователей

6.3.7.2.2.1. Просмотр информации о группе пользователей

Информацию о группе пользователей можно посмотреть, кликнув по ее названию или выбрав пункт «Details» (кликнув по кнопке с тремя точками, которая находится в конце строки с названием группы). В окне с информацией о группе отображаются вкладки с разными видами информации и кнопка для редактирования группы.

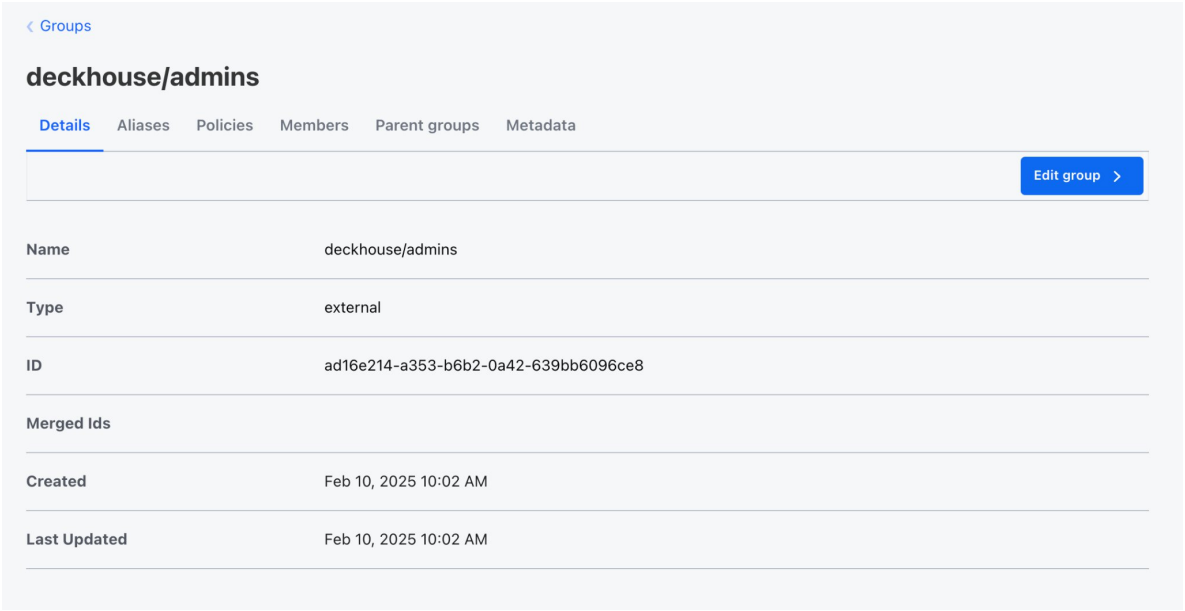


Рисунок 120 – Просмотр информации о группе пользователей

6.3.7.2.2.2. Добавление группы пользователей

Добавить группу пользователя можно, нажав кнопку для добавления группы («Create group») в окне для работы с группами.

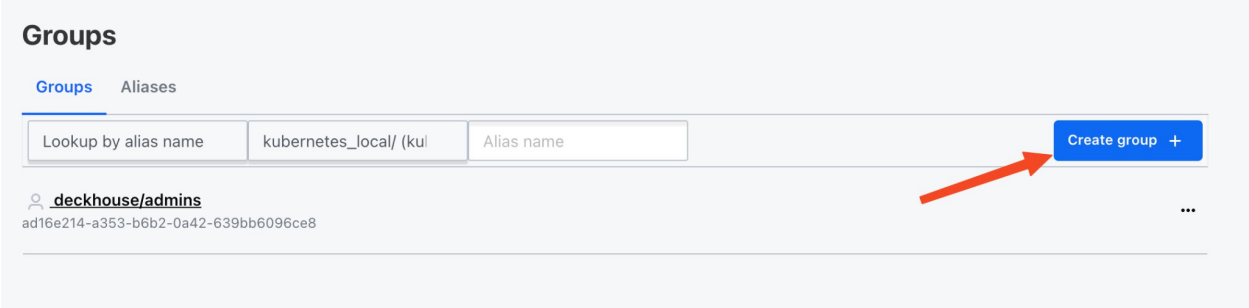


Рисунок 121 – Добавление группы пользователей

После этого откроется форма для создания группы. Под формой находятся кнопки «Create» — для сохранения группы «Back» для возврата без сохранения на экран со списком групп.

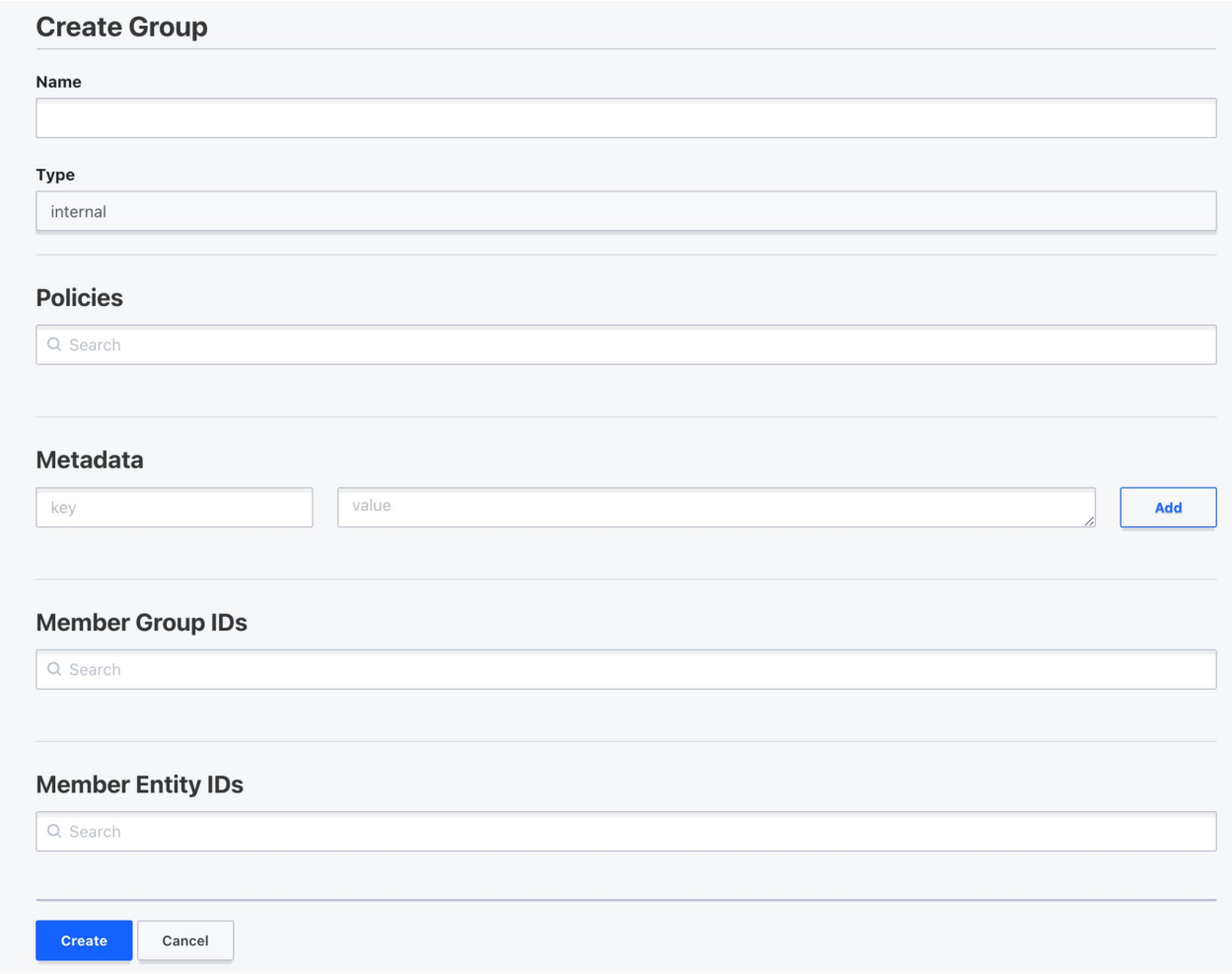


Рисунок 122 – Форма для создания группы пользователей

6.3.7.2.3. Работа с сущностями и алиасами

Сущности (Entities) в Stronghold представляют собой абстракцию пользователя или приложения, объединяющие несколько методов аутентификации под одним логическим идентификатором.

Для перехода в подраздел для работы с сущностями и алиасами необходимо кликнуть по пункту «Entities» в меню слева раздела для работы с доступами (п. 6.3.7.2).

В центре экрана находится две вкладки: «Entities» (список сущностей) и «Aliases» (список алиасов), поля для фильтрации элементов списка, кнопка объединения сущностей и кнопка для добавления новой сущности.

Для сущностей из списка на вкладке «Entities» доступны следующие действия:

- просмотр детальной информации о сущности;
- создание алиаса.

Выбрать нужное действие можно, кликнув по кнопке с тремя точками, которая находится в конце строки с названием сущности.

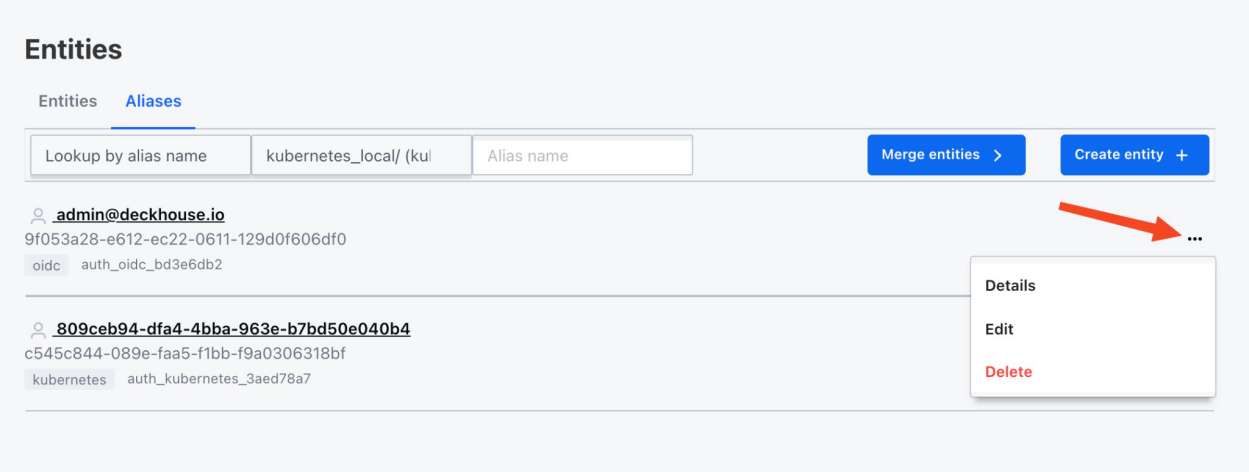


Рисунок 123 – Работа с сущностями и алиасами

Для алиасов из списка на вкладке «Aliases» доступны следующие действия:

- просмотр детальной информации об алиасе;
- редактирование алиаса;
- удаление алиаса.

Выбрать нужное действие можно, кликнув по кнопке с тремя точками, которая находится в конце строки с названием алиаса.

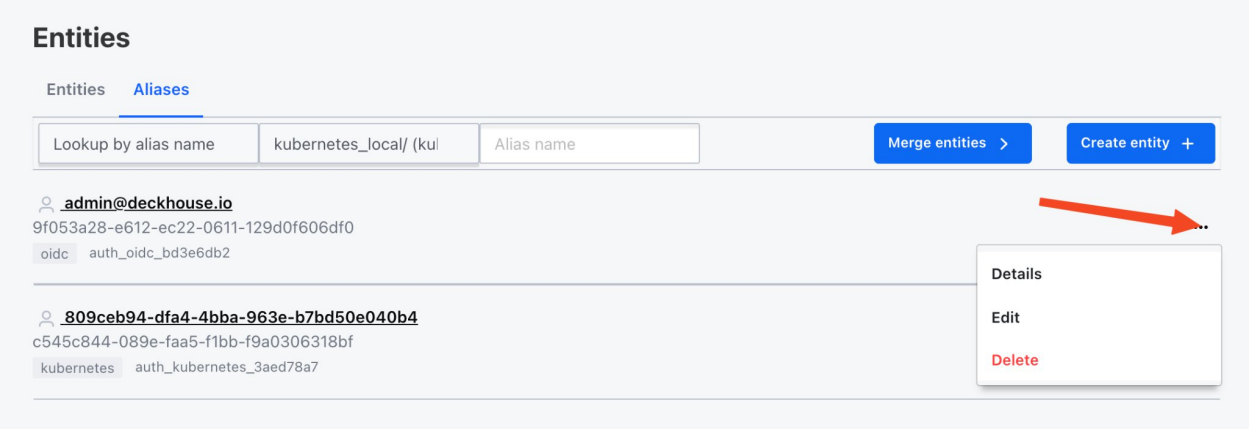


Рисунок 124 – Вкладка «Aliases»

6.3.7.2.3.1. Просмотр информации о сущности

Информацию о сущности можно посмотреть, кликнув по ее названию в списке на вкладке «Entities» окна для работы с сущностями (п. 6.3.7.2.3) или выбрав пункт «Details» (кликнув по кнопке с тремя точками, которая находится в конце строки с названием сущности). В окне с информацией о сущности отображаются вкладки с разными видами информации, кнопка добавления сущности и кнопка для редактирования сущности.

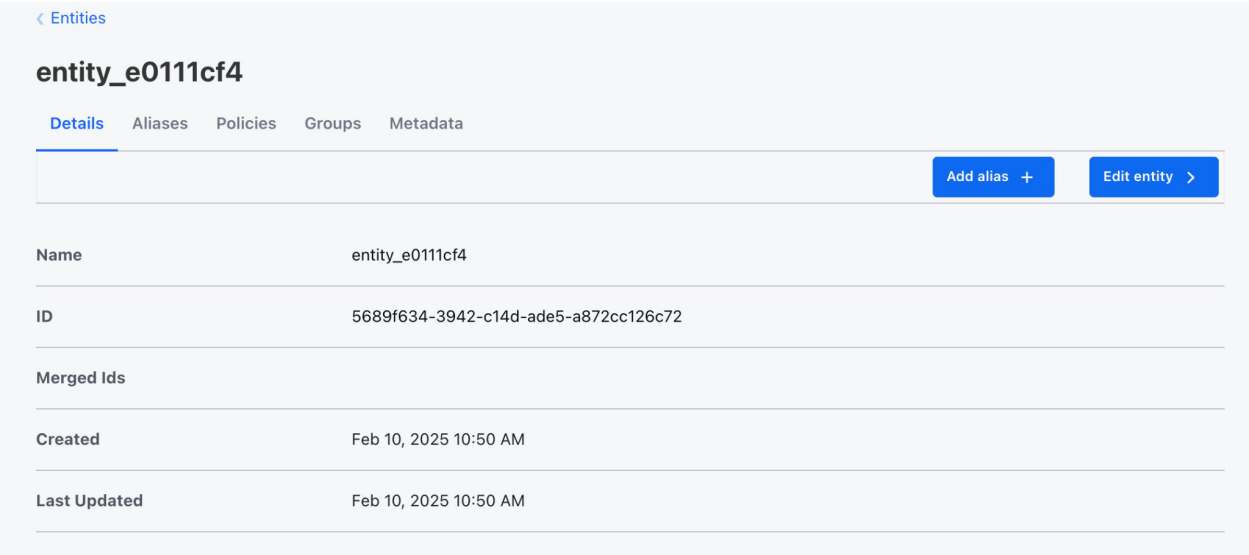


Рисунок 125 – Просмотр информации о сущности

6.3.7.2.3.2. Просмотр информации об алиасе

Информацию об алиасе можно посмотреть, кликнув по его названию в списке на вкладке «Aliases» окна для работы с сущностями (п. 6.3.7.2.3) или выбрав пункт «Details» (кликнув по кнопке с тремя точками, которая находится в конце строки с названием алиаса). В окне с информацией об алиасе отображаются вкладки с общей информацией, метадатой и кнопка для редактирования алиаса.

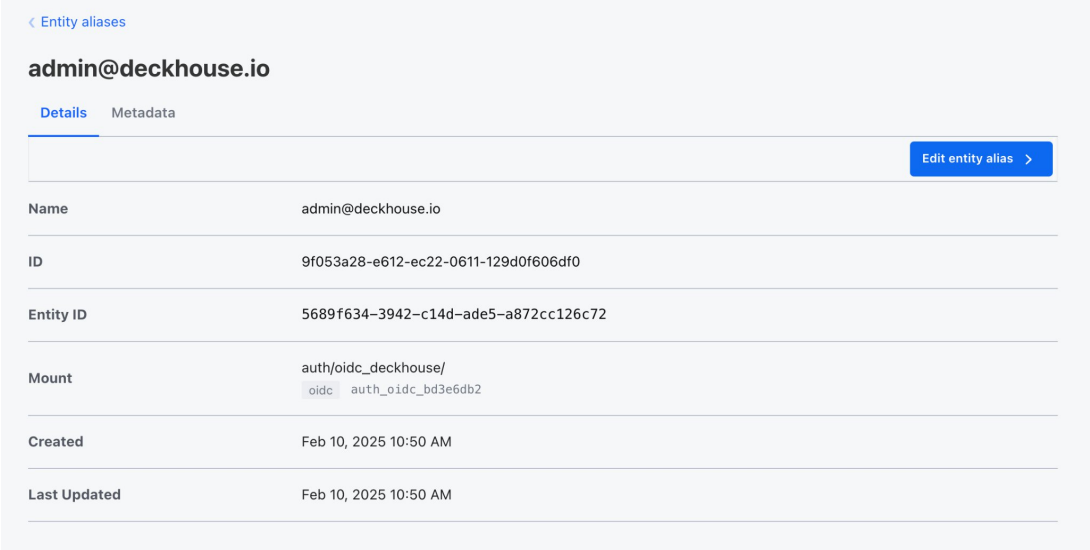


Рисунок 126 – Просмотр информации об алиасе

6.3.7.2.3.3. Создание сущности

Создать сущность можно, нажав кнопку для добавления («Create entity») в окне для работы с сущностями и алиасами (п. 6.3.7.2.3).

Рисунок 127 – Создание сущности

После этого откроется форма для создания сущности. Под формой находятся кнопки «Create» — для сохранения сущности и «Back» — для возврата без сохранения на экран со списком сущностей.

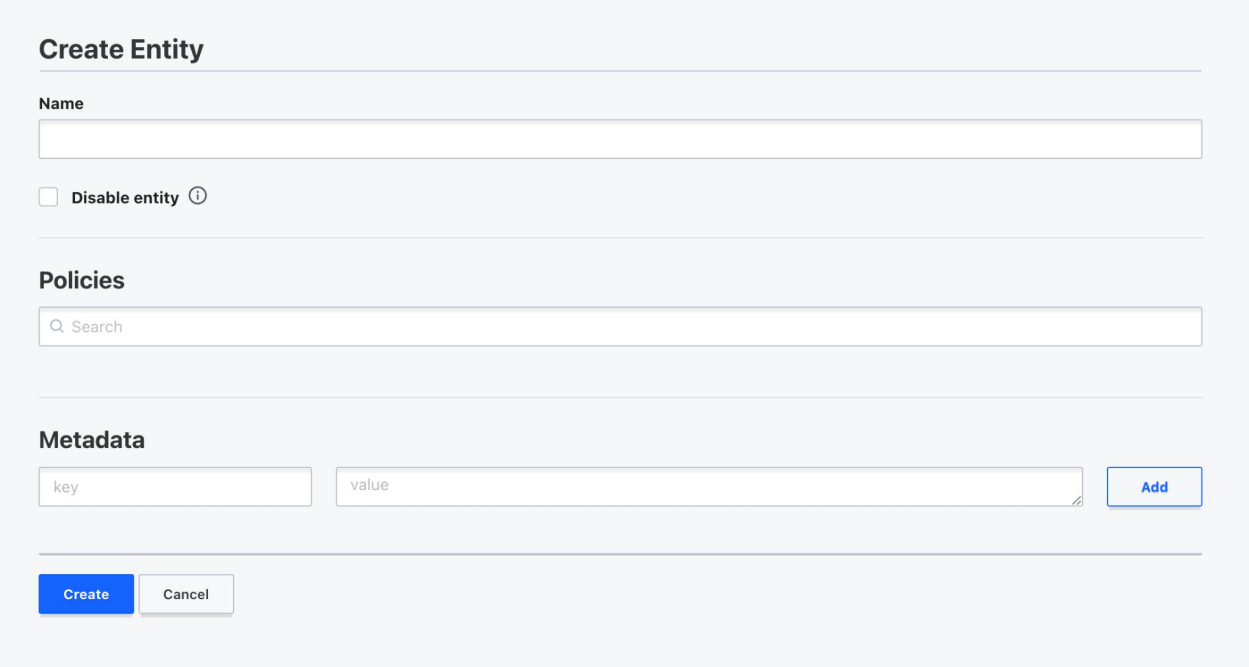


Рисунок 128 – Форма для создания сущности

6.3.7.2.3.4. Создание алиаса

Добавить алиас для сущности можно, нажав на кнопку с тремя точками, которая находится в конце строки с названием сущности, в окне для работы с сущностями и алиасами (п 6.3.7.2.3) и выбрав пункт «Create alias».

После этого откроется форма для создания алиаса. Под формой находятся кнопки «Create» — для сохранения сущности и «Cancel» — для отмены.

Рисунок 129 – Создание алиаса

6.3.7.2.3.5. Объединение сущностей

Объединить сущности можно, нажав кнопку «Merge entities» в окне для работы с сущностями и алиасами (п. 6.3.7.2.5). После этого откроется форма для объединения сущностей.

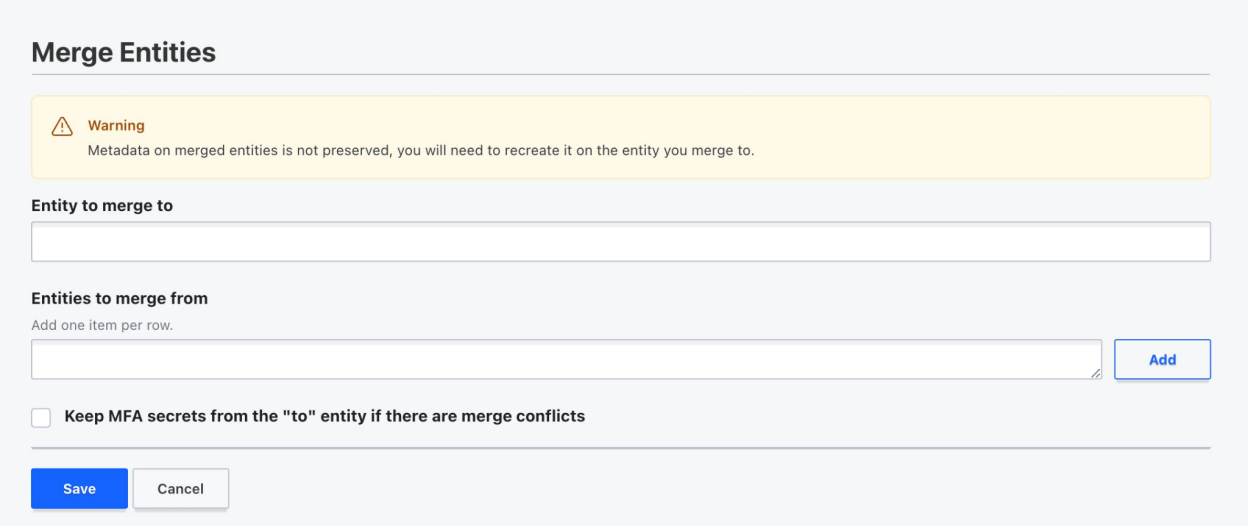


Рисунок 130 – Объединение сущностей

6.3.7.2.4. Управление временными правами доступа к секретам и ресурсам (Leases)

Для перехода в подраздел для управления временными правами доступа к секретам и ресурсам (Leases) необходимо кликнуть по пункту «Leases» в меню слева в разделе для работы с доступами (п. 6.3.7.2). Откроется окно для поиска информации об аренде по ее идентификатору.

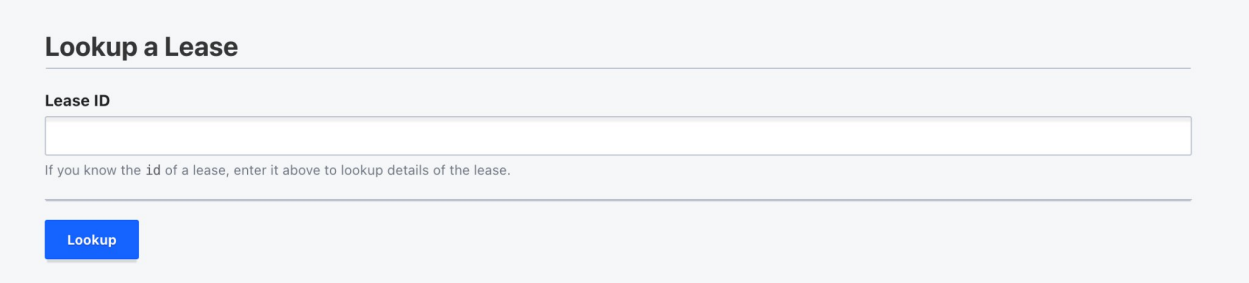


Рисунок 131 – Управление временными правами доступа к секретам и ресурсам (Leases)

6.3.7.3. Работа с политиками контроля доступа

Работа с политиками контроля доступа в stronghold осуществляется в разделе «Policies». Перейти в него можно, кликнув по пункту меню «Access» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). В левой части экрана раздела для работы с политиками находится окно навигации, вверху которого расположена ссылка для быстрого перехода на главный экран веб-интерфейса stronghold. В центре размещен список политик, фильтр для поиска нужной и кнопка для добавления новой политики.

Для политик из списка доступны следующие действия:

- просмотр детальной информации о политике;
- редактирование политики;
- удаление политики.

Выбрать нужное действие можно, кликнув по кнопке с тремя точками, которая находится в конце строки с названием политики.

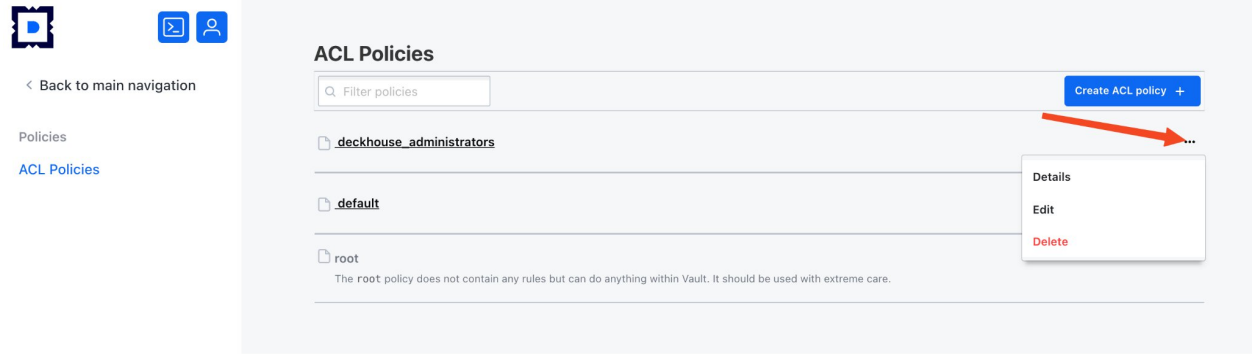


Рисунок 132 – Работа с политиками контроля доступа

6.3.7.3.1. Просмотр информации о политике

Информацию о политике можно посмотреть, кликнув по ее названию или выбрав пункт «Details» (кликнув по кнопке с тремя точками, которая находится в конце строки с названием политики). В окне с информацией о политике отображаются сведения о ней в формате HCL, а также кнопка для загрузки данных на компьютер и кнопка для редактирования политики.

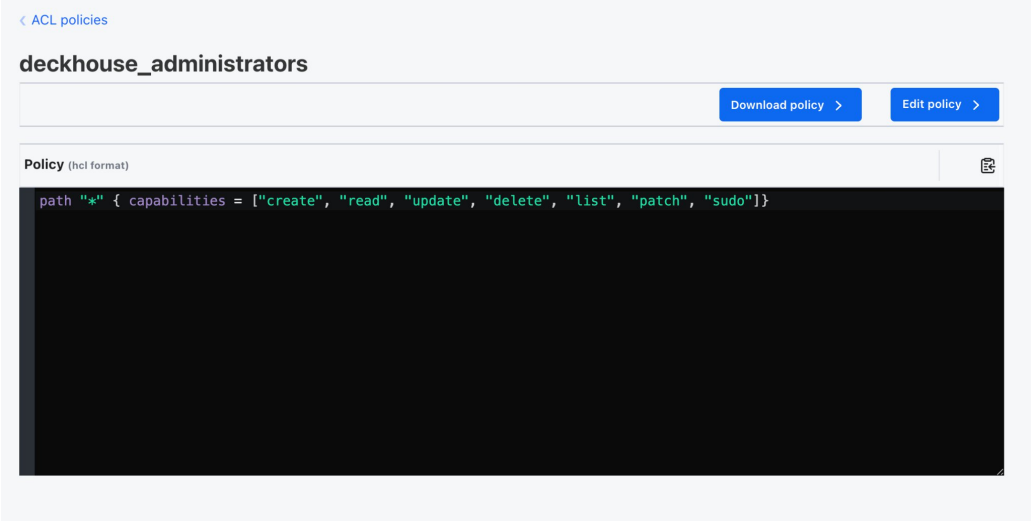


Рисунок 133 – Просмотр информации о политике

6.3.7.3.2. Добавление политики

Чтобы добавить политику, необходимо нажать кнопку «Create ACL policy» на экране для работы с политиками (п. 6.3.7.3). После этого откроется форма с полями для ввода имени политики и ее описания в формате HCL.

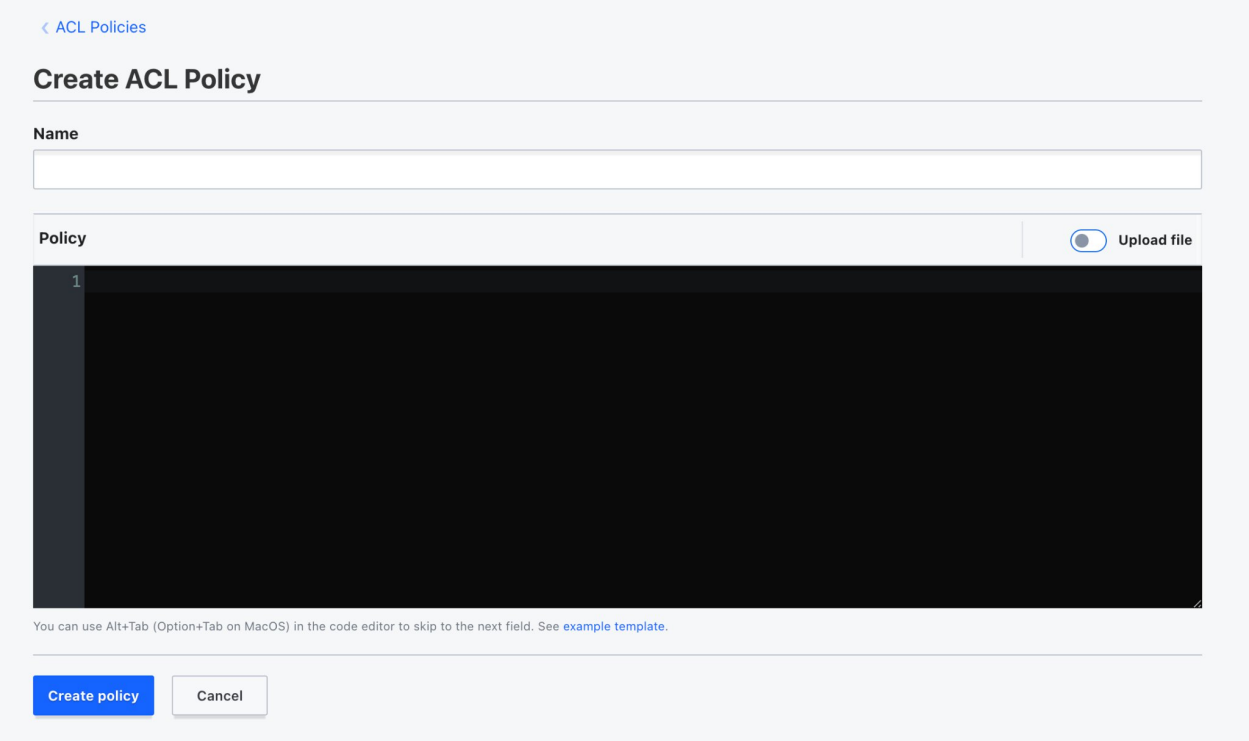


Рисунок 134 – Добавление политики

6.3.7.4. Работа с дополнительными инструментами

Работа с дополнительными инструментами в stronghold осуществляется в разделе «Tools». Перейти в него можно, кликнув по пункту меню «Tools» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). В левой части раздела находится окно навигации по инструментам, вверху которого расположена ссылка для быстрого перехода на главный экран веб-интерфейса stronghold. В центре отображаются поля выбранного инструмента.

6.3.7.4.1. Инструмент «Wrap»

Инструмент «Wrap» предназначен 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.

Для доступа к инструменту кликните по пункту меню «Wrap» раздела «Tools».

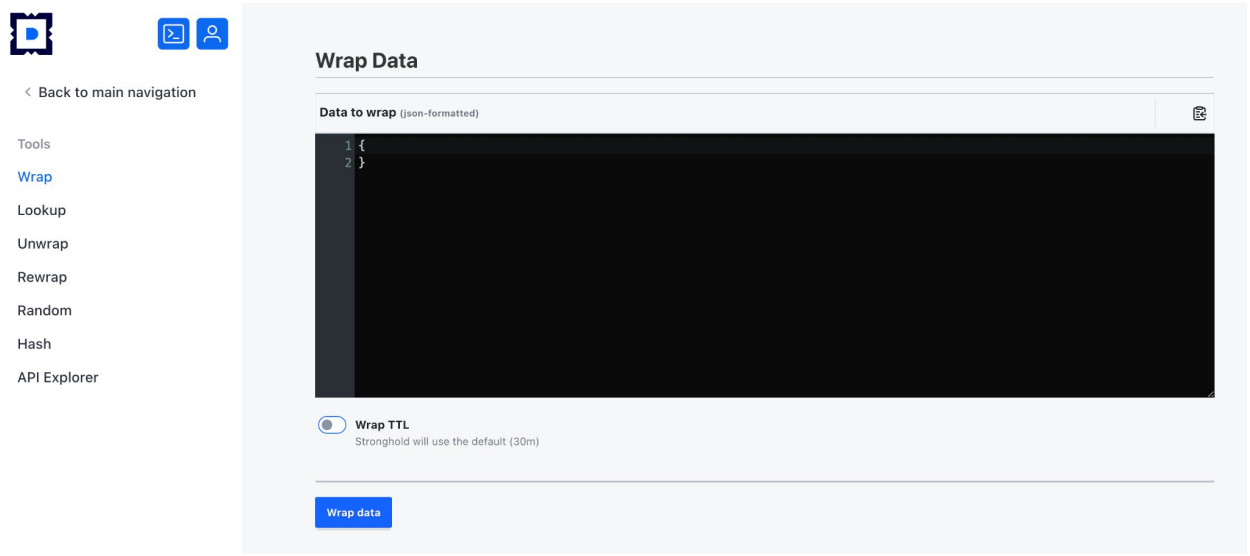


Рисунок 135 – Инструмент «Wrap»

6.3.7.4.2. Инструмент «Lookup»

Инструмент «Lookup» используется 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.

Для доступа к инструменту кликните по пункту меню «Lookup» раздела «Tools».

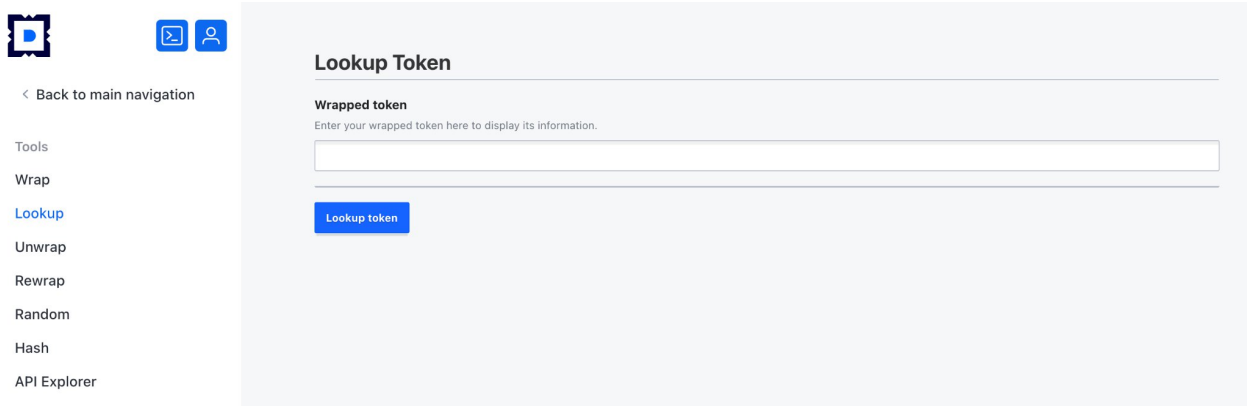


Рисунок 136 – Инструмент «Lookup»

6.3.7.4.3. Инструмент «Unwrap»

Инструмент «Unwrap» предназначен для распаковки wrapping token (токена обертки) и получения доступа к «упакованным» данным

Для доступа к инструменту кликните по пункту меню «Unwrap» раздела «Tools».

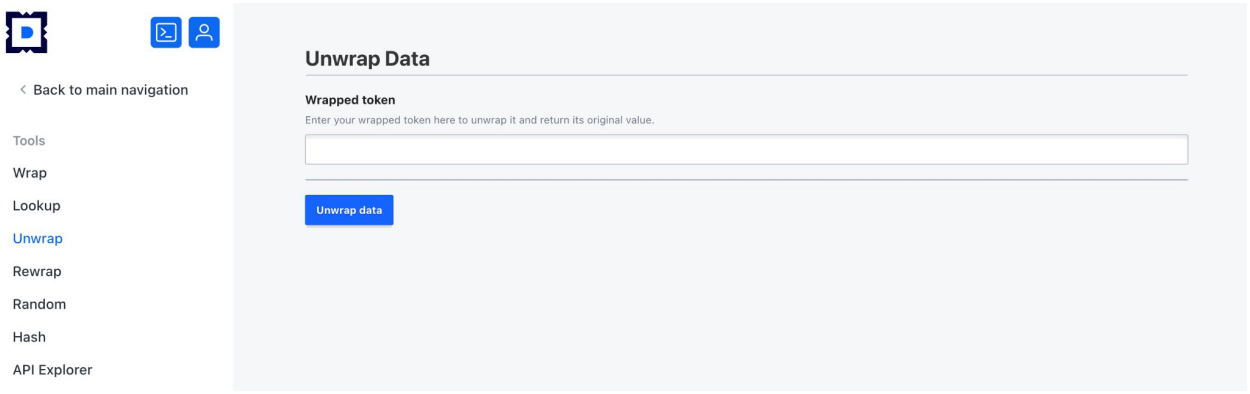


Рисунок 137 – Инструмент «Unwrap»

6.3.7.4.4. Инструмент «Rewrap»

Инструмент «Rewrap» предназначен 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.

Для доступа к инструменту кликните по пункту меню «Rewrap» раздела «Tools».

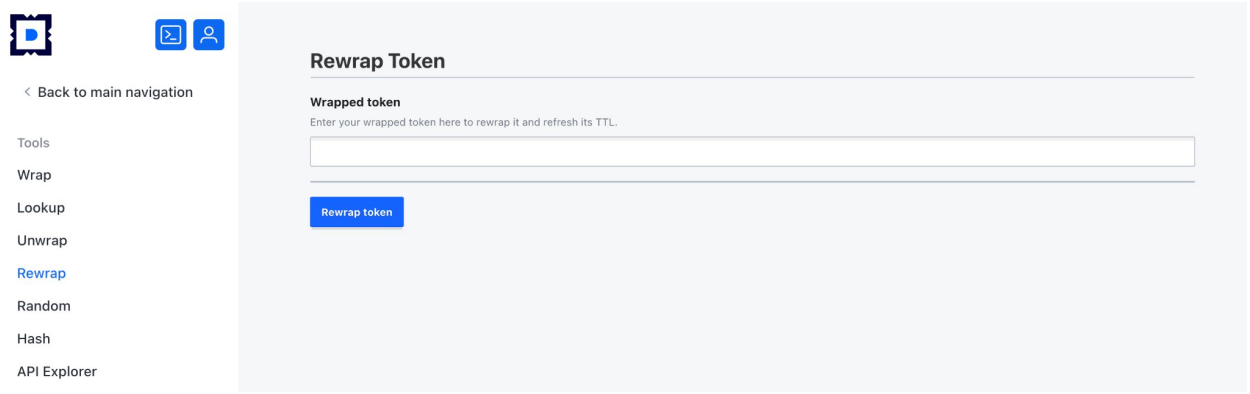


Рисунок 138 – Инструмент «Rewrap»

6.3.7.4.5. Инструмент «Random»

Инструмент «Random» предназначен для генерации криптографически безопасных случайных данных для создания уникальных идентификаторов, токенов, паролей или иных данных, для которых важна высокая степень случайности и безопасности.

Для доступа к инструменту кликните по пункту меню «Random» раздела «Tools».

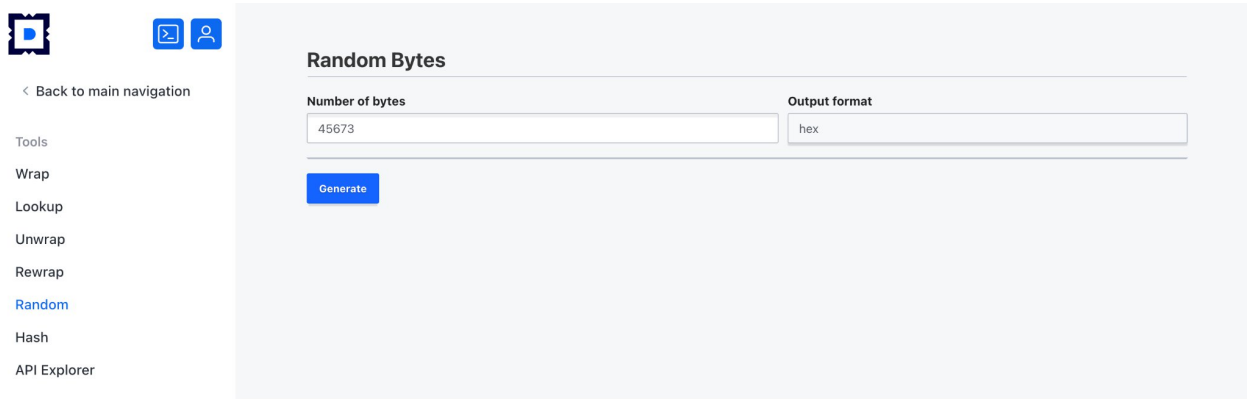


Рисунок 139 – Инструмент «Random»

6.3.7.4.6. Инструмент «Hash»

Инструмент «Hash» предназначен для генерации хешей для различных данных. Поддерживается несколько алгоритмов кэширования.

Для доступа к инструменту кликните по пункту меню «Hash» раздела «Tools».

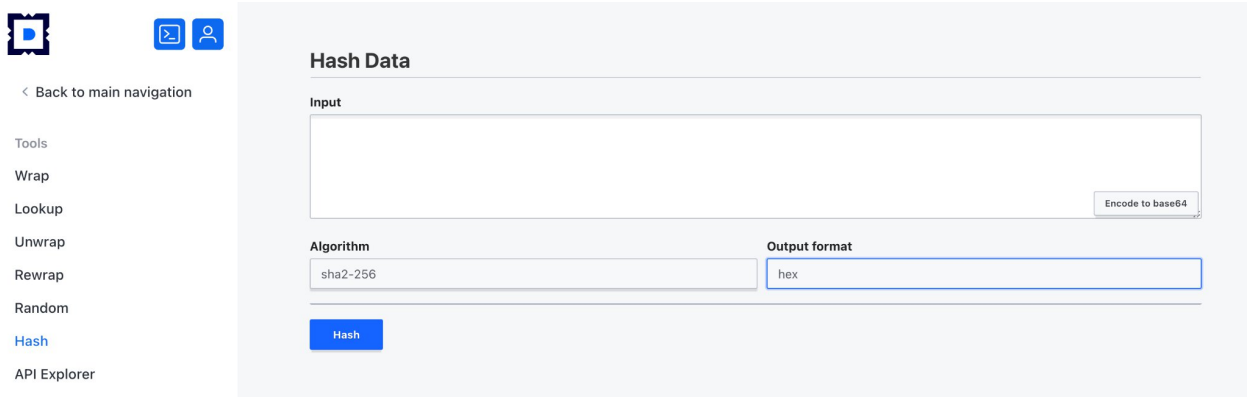


Рисунок 140 – Инструмент «Hash»

6.3.7.4.7. Инструмент «API Explorer»

Инструмент «API Explorer» предоставляет пользователям удобный способ взаимодействия с API stronghold через графический интерфейс.

Для доступа к инструменту кликните по пункту меню «API Explorer» раздела «Tools».

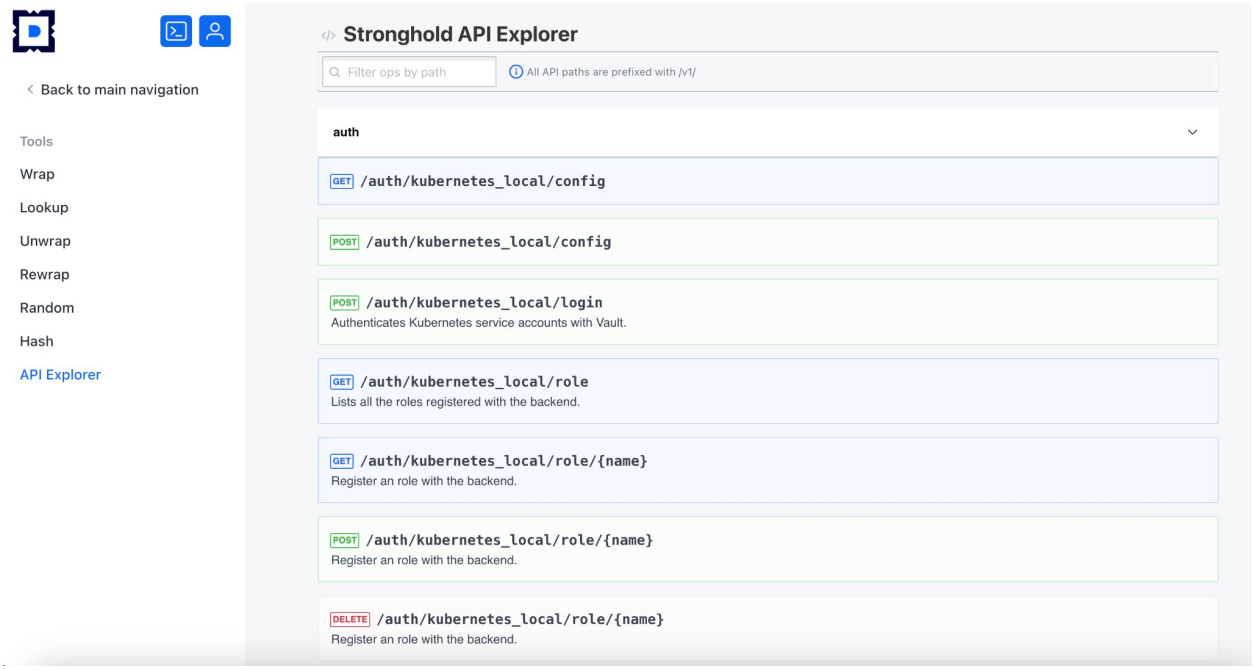


Рисунок 141 – Инструмент «API Explorer»

6.3.7.5. Мониторинг состояния Raft кластера stronghold

Мониторинг состояния Raft кластера stronghold осуществляется в разделе «Raft Storage». Перейти в него можно, кликнув по пункту меню «Raft Storage» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). В левой части интерфейса находится окно навигации по разделам. В центре отображается информация о лидере и узлах кластера, а также кнопка «Snapshots» для создания резервной копии данных Raft кластера stronghold и восстановления данных из резервной копии.

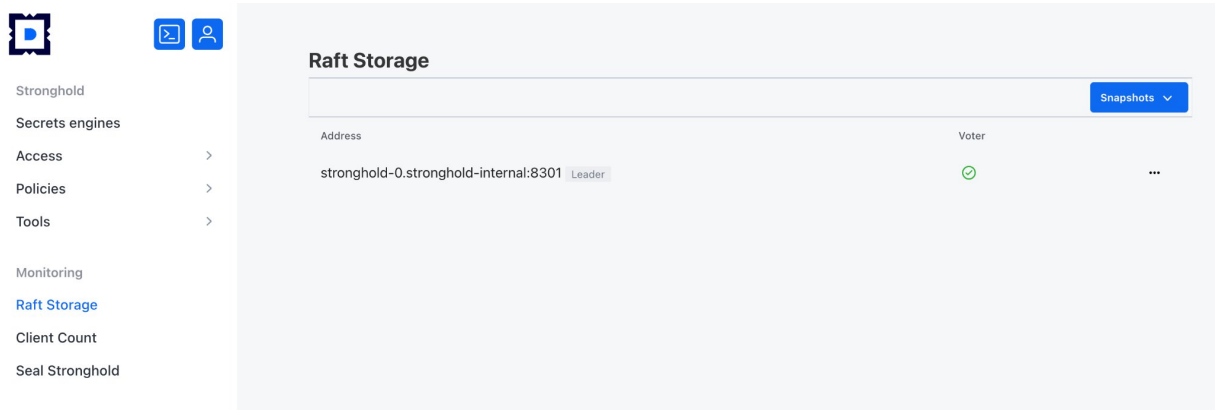


Рисунок 142 – Мониторинг состояния Raft кластера stronghold

6.3.7.6. Мониторинг активности и оценка нагрузки на stronghold

Мониторинг активности и оценка нагрузки на stronghold осуществляется в разделе «Client Count». Перейти в него можно, кликнув по пункту меню «Client Count» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). В левой части интерфейса находится окно навигации по

разделам. В центре размещены две вкладки. Первая — «Dashboard» с информацией о количестве уникальных клиентов за текущий месяц и кнопками выбора другого периода.

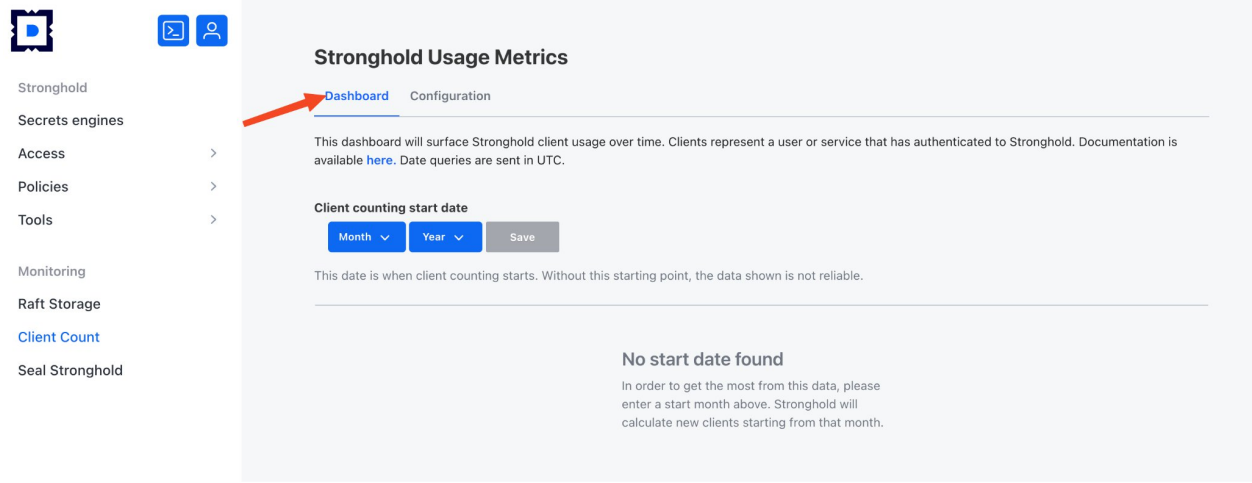


Рисунок 143 – Мониторинг активности и оценка нагрузки на stronghold

Вторая вкладка — «Configuration». Здесь можно посмотреть настройки сбора метрик и отредактировать их (для этого необходимо нажать кнопку «Edit configuration»).

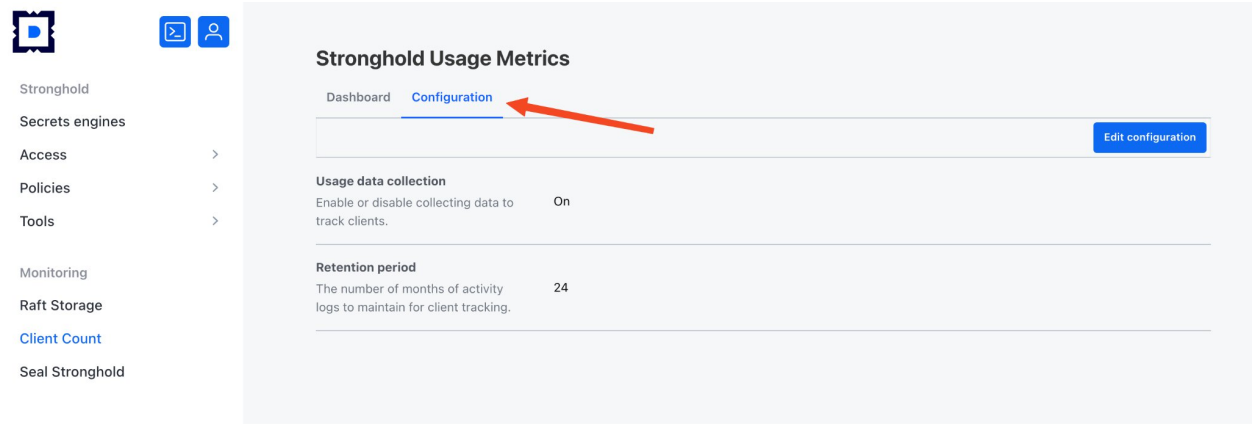


Рисунок 144 – Вкладка «Configuration»

6.3.7.7. Запечатывание и распечатывание хранилища секретов

Запечатывание и распечатывание хранилища секретов осуществляется в разделе «Seal Stronghold». Перейти в него можно, кликнув по пункту меню «Seal Stronghold» на главном экране веб-интерфейса stronghold (п. 6.3.7.1). В левой части интерфейса находится окно навигации по разделам. В центре отображается кнопка для запечатывания и распечатывания хранилища секретов (в зависимости от его текущего состояния).

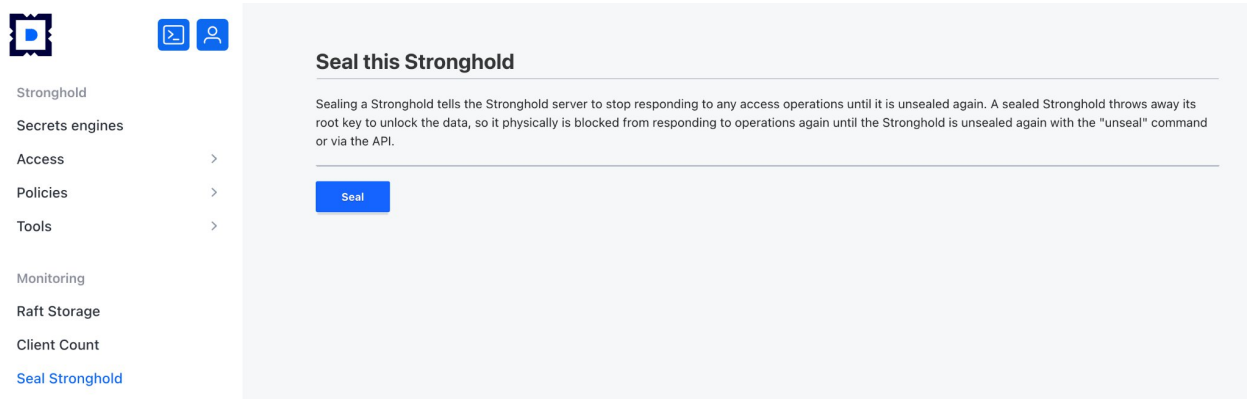


Рисунок 145 – Запечатывание и распечатывание хранилища секретов

Когда хранилище находится в состоянии «запечатано» (sealed), он не может обрабатывать запросы на чтение или запись секретов.

6.3.7.8. Работа со stronghold CLI

stronghold CLI — инструмент для взаимодействия со stronghold, который позволяет выполнять различные операции по управлению секретами, настройке политик, управлению пользователями и т.д. Вызвать stronghold CLI можно, находясь в любом из разделов интерфейса. Для его запуска необходимо нажать кнопку в левом верхнем углу окна.

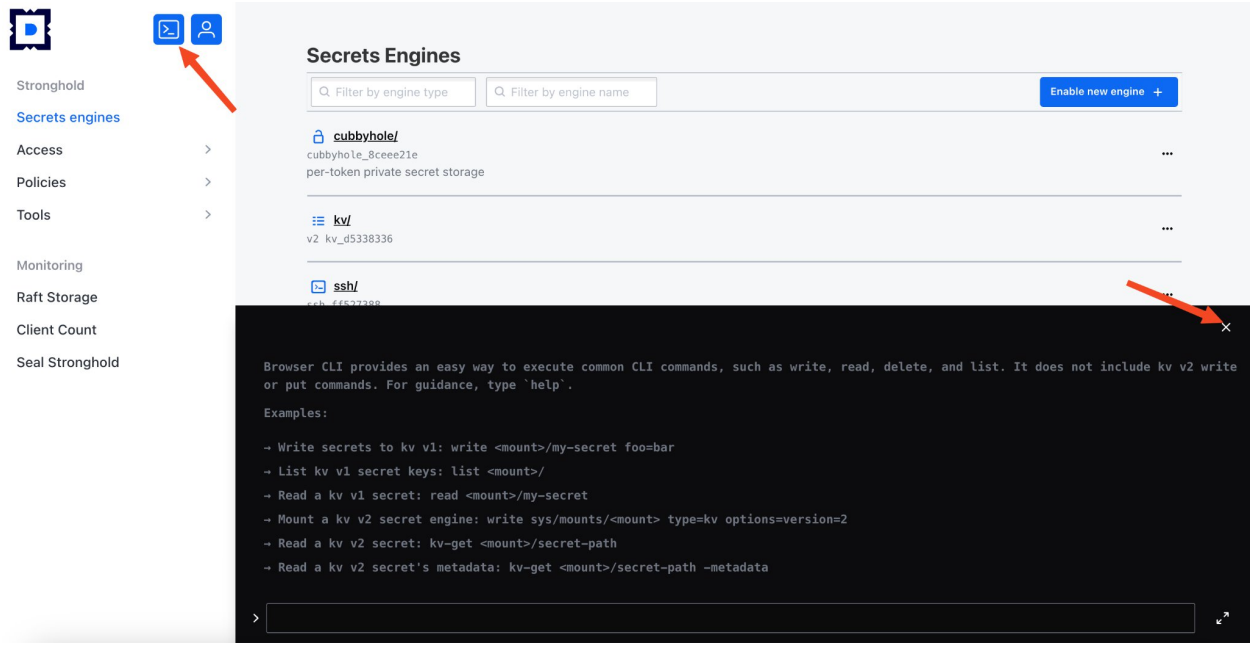


Рисунок 146 – Работа со stronghold CLI

Закрыть stronghold CLI можно, нажав на крестик в правом верхнем углу окна инструмента.

6.3.8. Веб-интерфейс модуля cilium-hubble

Веб-интерфейс Hubble позволяет визуализировать сетевой стек кластера, отслеживать сетевые взаимодействия между подами, сервисами и внешними ресурсами, анализировать сетевую активность и выявлять проблемы с сетью.

Веб-интерфейс Hubble доступен по адресу hubble.<ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> – строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate.

При первом входе потребуется ввести учетные данные пользователя.

6.3.8.1. Экран выбора пространства имен

При переходе по адресу hubble.<ШАБЛОН_ИМЕН_КЛАСТЕРА> откроется экран выбора пространства имен, для которого будет визуализирован сетевой стек. Выбрать пространство имен можно с помощью выпадающего списка в левой верхней части экрана или кликнув по названию нужного пространства имен в списке в центре экрана.

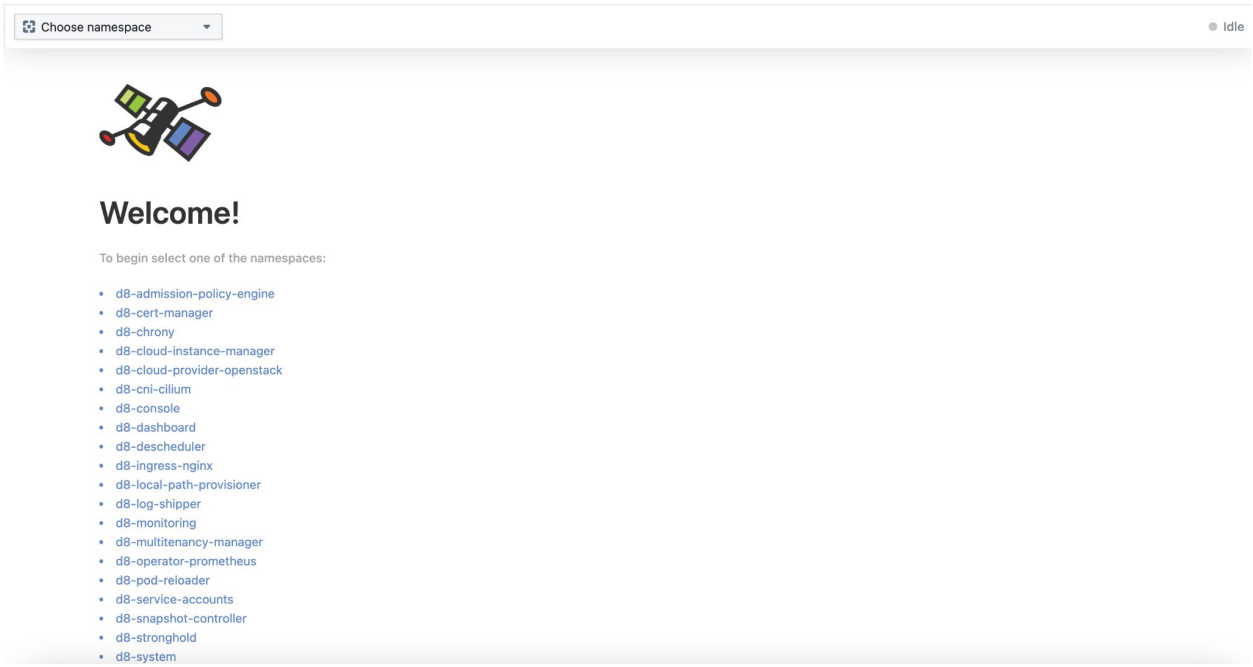


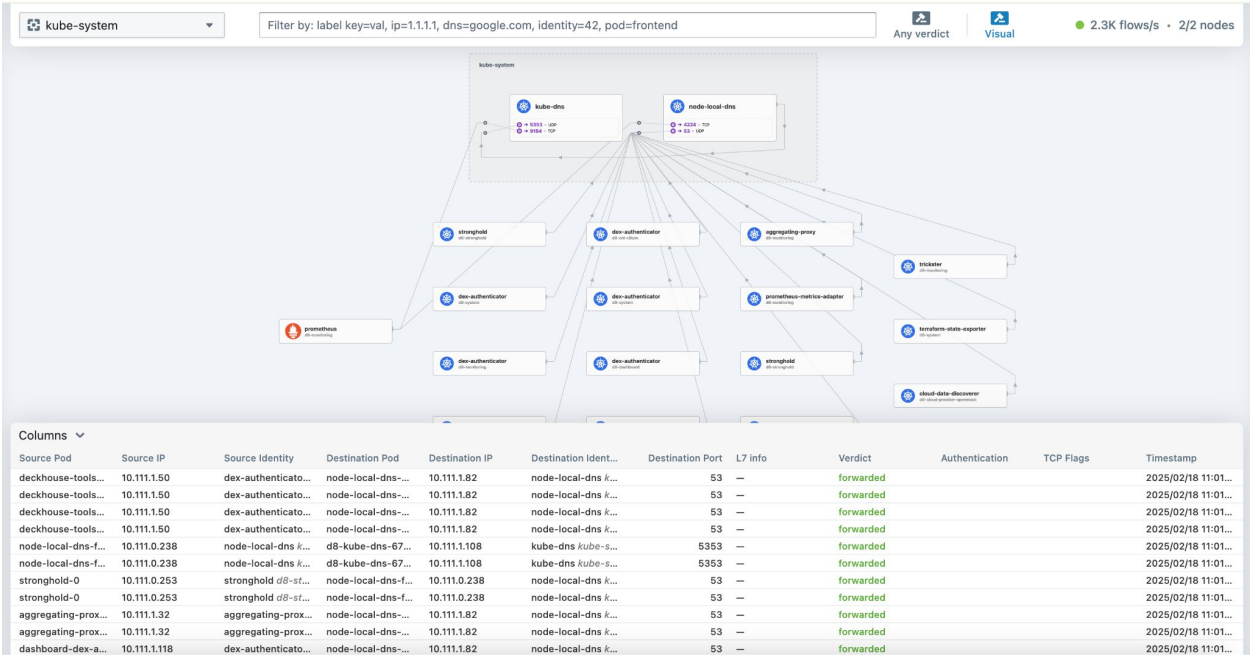
Рисунок 147 – Экран выбора пространства имен

После выбора пространства имен откроется экран с визуализацией сетевого стека и средствами анализа.

6.3.8.2. Визуализация сетевого стека и анализ сетевых взаимодействий

Экран с визуализацией сетевого стека и средствами анализа состоит из следующих частей:

- верхняя панель с фильтрами и краткой сводкой по кластеру (количество потоков и количество узлов);
- схема сетевых потоков;
- таблица сетевых потоков и событий.



Source Pod	Source IP	Source Identity	Destination Pod	Destination IP	Destination Ident…	Destination Port	L7 info	Verdict	Authentication	TCP Flags	Timestamp
deckhouse-tools…	10.111.1.50	dex-authenticato…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
deckhouse-tools…	10.111.1.50	dex-authenticato…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
deckhouse-tools…	10.111.1.50	dex-authenticato…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
deckhouse-tools…	10.111.1.50	dex-authenticato…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
node-local-dns-f…	10.111.0.238	node-local-dns *k…*	d8-kube-dns-67…	10.111.1.108	kube-dns *kube-s…*	5353	—	forwarded			2025/02/18 11:01…
node-local-dns-f…	10.111.0.238	node-local-dns *k…*	d8-kube-dns-67…	10.111.1.108	kube-dns *kube-s…*	5353	—	forwarded			2025/02/18 11:01…
stronghold-0	10.111.0.253	stronghold *d8-st…*	node-local-dns-f…	10.111.0.238	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
stronghold-0	10.111.0.253	stronghold *d8-st…*	node-local-dns-f…	10.111.0.238	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
aggregating-prox…	10.111.1.32	aggregating-prox…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
aggregating-prox…	10.111.1.32	aggregating-prox…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…
dashboard-dex-a…	10.111.1.118	dex-authenticato…	node-local-dns-…	10.111.1.82	node-local-dns *k…*	53	—	forwarded			2025/02/18 11:01…

Рисунок 148 – Визуализация сетевого стека и анализ сетевых взаимодействий

Данные на схеме и таблице сетевых потоков отображаются в реальном времени.

6.3.8.2.1. Фильтрация отображаемых данных

Отфильтровать отображаемые данные о сетевом стеке и потоках можно с помощью верхней панели с фильтрами. Здесь расположены фильтры:

- для выбора пространства имен (выпадающий список в левой части панели);

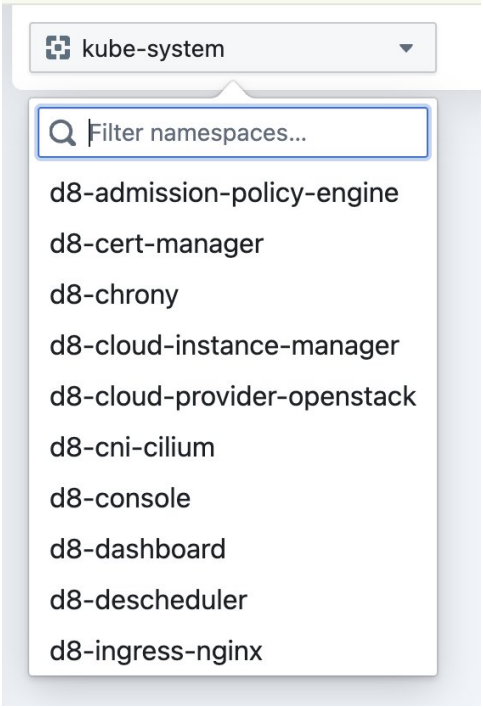
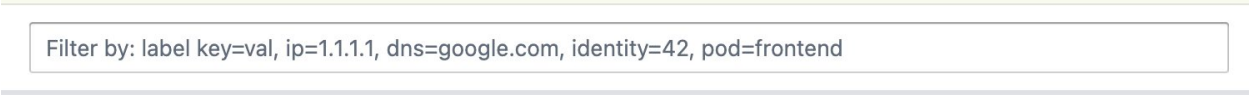


Рисунок 149 – Фильтрация отображаемых данных

- 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;

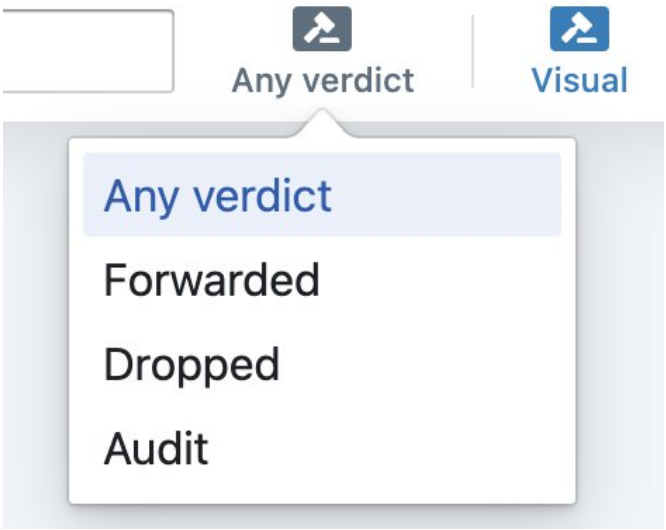


Рисунок 150 – Выбор сетевых потоков на основе решения («вердикта»)

- для выбора элементов схемы анализируемого пространства имен.

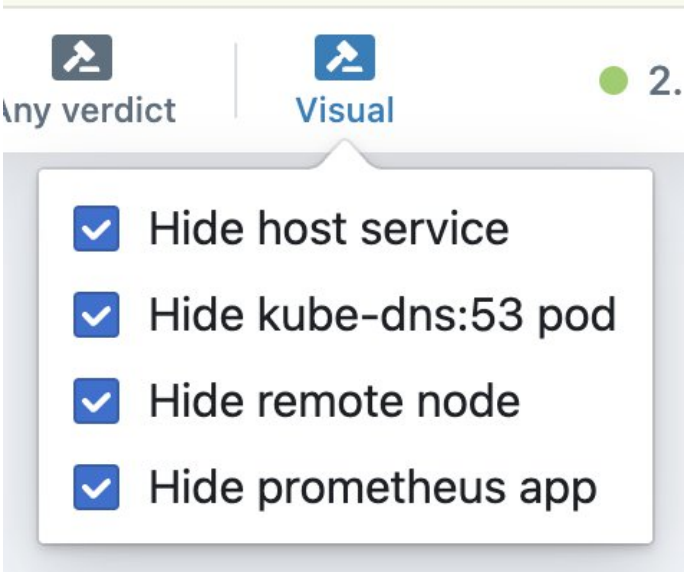


Рисунок 151 – Выбор элементов схемы анализируемого пространства имен

6.3.8.2.2. 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.

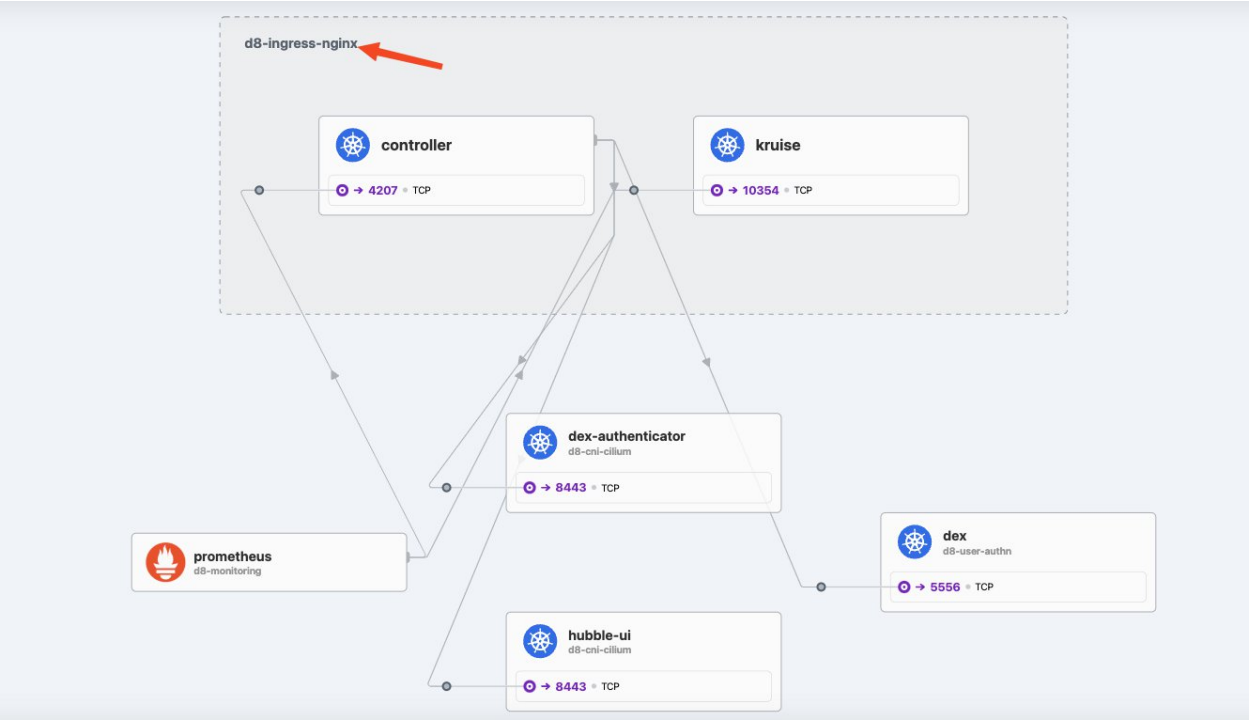


Рисунок 152 – Работа со схемой сетевых потоков

Посмотреть детальную информацию по конкретному ресурсу (список лейблов, сетевые взаимодействия и т.д.), можно, кликнув по нему.

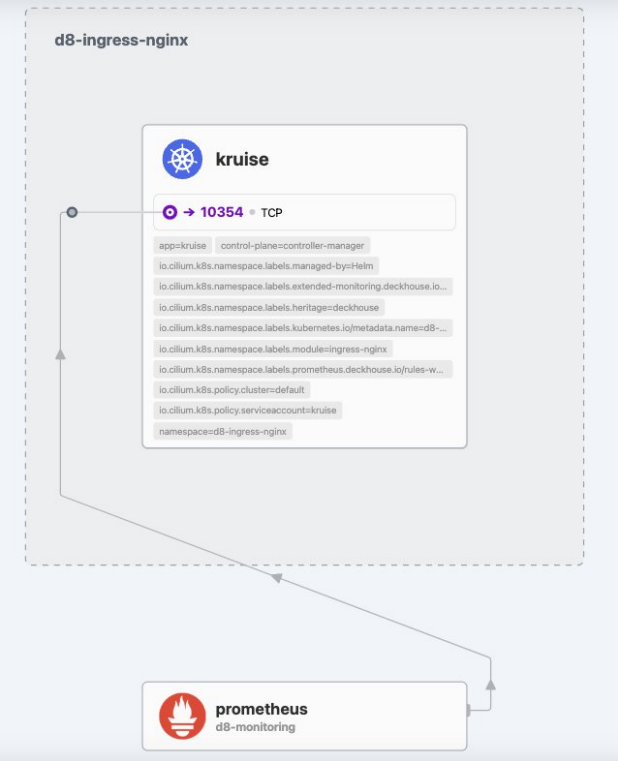


Рисунок 153 – Детальная информация по ресурсу

6.3.8.2.3. Работа с таблицей сетевых потоков и событий

Таблица сетевых потоков и событий отображается в нижней части экрана с визуализацией сетевого стека и средствами анализа. 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»);

- временная метка потока (столбец «Timestamp»).

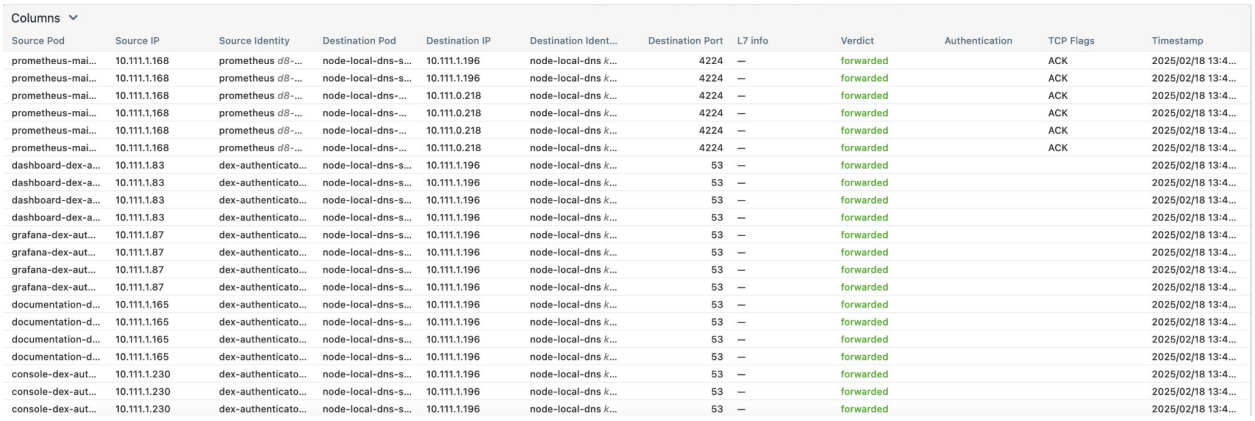


Рисунок 154 – Таблица сетевых потоков и событий

Набором столбцов, отображаемых в таблице, можно управлять. Чтобы выбрать нужные, кликните по кнопке «Columns» в левой верхней части таблицы.

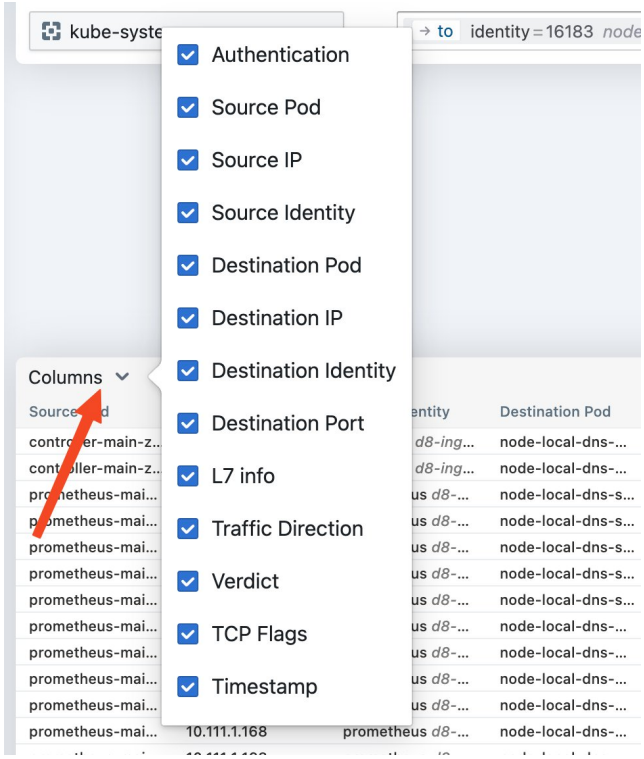


Рисунок 155 – Управление набором столбцов

Чтобы посмотреть информацию о записи таблицы в текстовом виде, кликните 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.

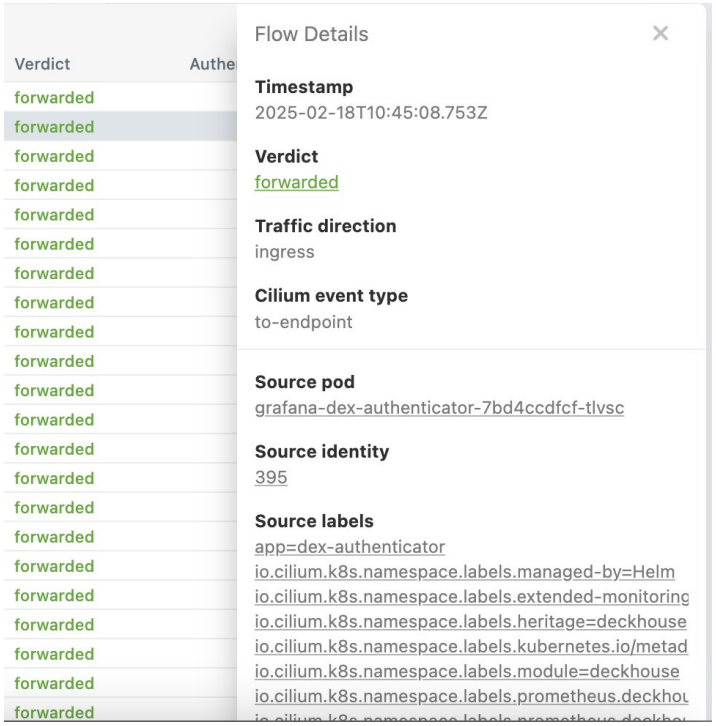


Рисунок 156 – Просмотр информации о записи таблицы в текстовом виде

6.3.9. Веб-интерфейс Kiali

Веб-интерфейс Kiali предоставляет возможность управлять и наблюдать за ресурсами и пользовательскими сервисами, работающими под управлением Istio. С помощью Kiali можно визуализировать взаимодействие между сервисами, анализировать сетевой трафик, а также выполнять диагностику проблемных связей и состояния control plane.

Веб-интерфейс Kiali доступен по адресу istio.<ШАБЛОН_ИМЕН_КЛАСТЕРА>, где <ШАБЛОН_ИМЕН_КЛАСТЕРА> — строка, соответствующая шаблону DNS-имен кластера, указанному в глобальном параметре modules.publicDomainTemplate.

При первом входе потребуется ввести учетные данные пользователя.

6.3.9.1. Общая информация

По умолчанию при открытии веб-интерфейса Kiali отображается дашборд «Overview» с общей информацией о состоянии инфраструктуры service mesh (далее — сервис-меш). На дашборде представлены данные о текущем состоянии конфигурации Istio, а также о состоянии отдельных компонентов и сетевом трафике.

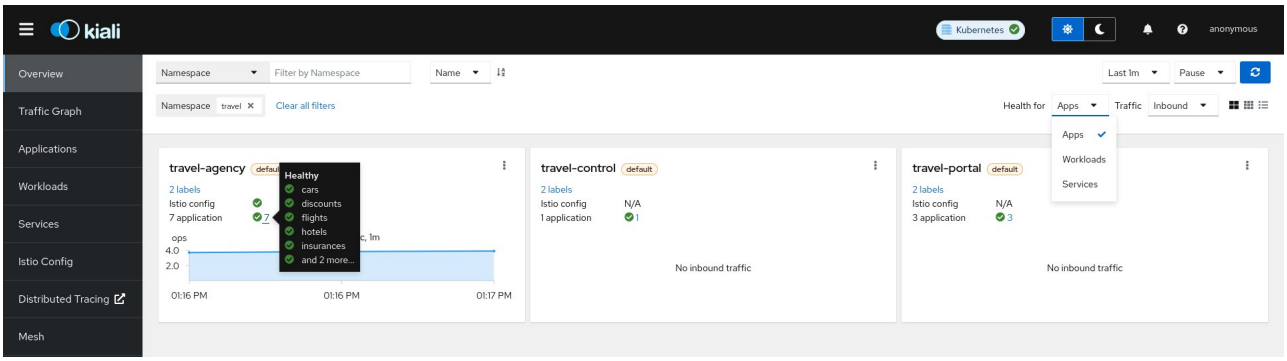


Рисунок 157 – Просмотр общей информации о состоянии сервис-меша

Над дашбордом располагается панель фильтров. В левой части панели представлены фильтры, позволяющие ограничить отображаемые данные по следующим категориям: название пространства имён («Namespace»), состояние («Health»), статус mTLS («mTLS») и лейбл пространства имён («Namespace Label»).

В правой части панели располагаются дополнительные параметры отображения данных. Здесь можно задать частоту обновления данных на дашборде, выбрать типы компонентов, по которым будет выводиться информация (приложения («Apps»), нагрузки («Workloads») или сервисы («Services»)), направление сетевого трафика (входящий («Inbound») или исходящий («Outbound»)), а также формат представления данных (в виде карточек или списком).

6.3.9.2. Схема движения трафика

На дашборде «Traffic Graph» представлена схема движения трафика в сервис-меше. Узлы и маршруты на схеме подсвечиваются в зависимости от их состояния. Оранжевым и красным цветом выделены проблемные участки инфраструктуры, требующие внимания. В правой части дашборда представлена сводная информация об успешности выполнения HTTP-запросов. Переключаясь между вкладками, можно выбрать объём отображаемых данных в зависимости от типа сетевого трафика (входящий («Inbound»), исходящий («Outbound»), совокупный («Total»)).

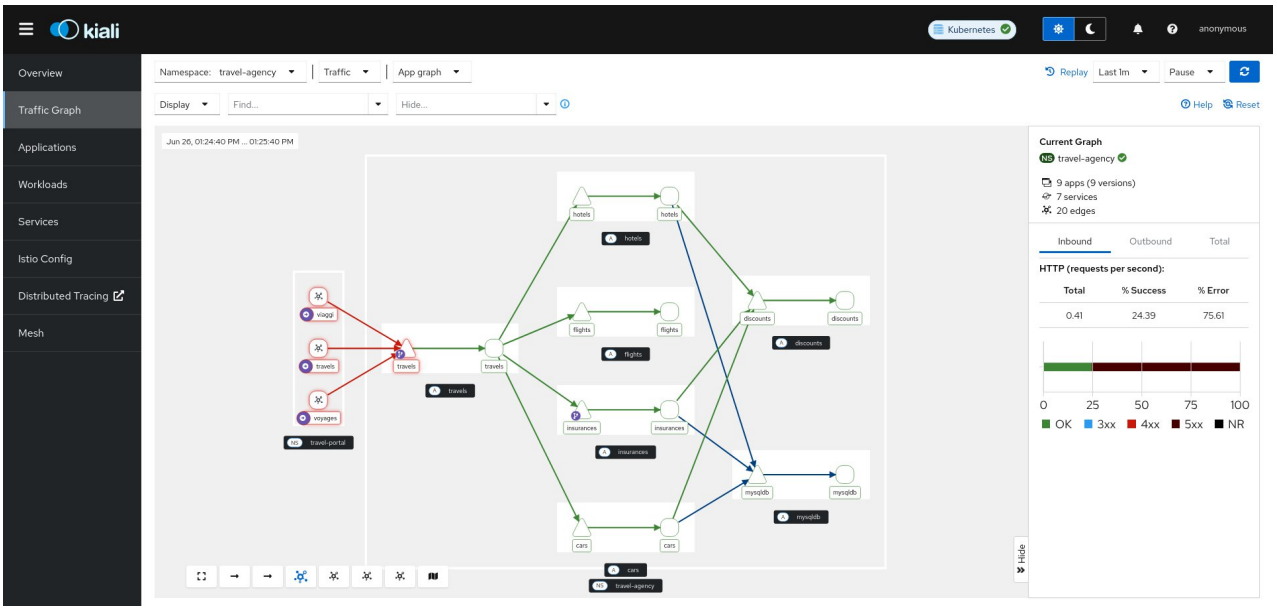


Рисунок 158 – Просмотр информации о движении сетевого трафика

Над дашбордом располагается панель фильтров. В левой части представлены фильтры для настройки отображения данных. Используя выпадающие списки в верхнем ряду, можно отфильтровать данные по пространствам имён («Select Namespaces»), типу трафика («Traffic») и отображаемым компонентам («XXX graph»). Выпадающие списки в нижнем ряду предназначены для более тонкой фильтрации по отдельным компонентам инфраструктуры («Display»), отображаемым («Find») и скрываемым («Hide») элементам. В правой части панели настраивается частота обновления данных на дашборде.

6.3.9.3. Информация по компонентам сервис-меша

На дашбордах «Applications», «Workloads», «Services» и «Istio config» представлены сводные данные по соответствующим компонентам сервис-меша. Все дашборды имеют схожую структуру в виде таблиц с информацией о состоянии компонентов, связанных с ними пространствах имён, лейблах и состоянии конфигурации.

6.3.9.3.1. Applications

На дашборде «Applications» представлена таблица со сводными данными о приложениях сервис-меша. Информация в таблице разбита на следующие столбцы: состояние приложения («Health»), имя приложения («Name»), пространство имён («Namespace»), лейблы («Labels»), подробная информация («Details»). Выпадающий список над таблицей позволяет отфильтровать данные по имени приложения, типу конфигурации Istio и другим параметрам.

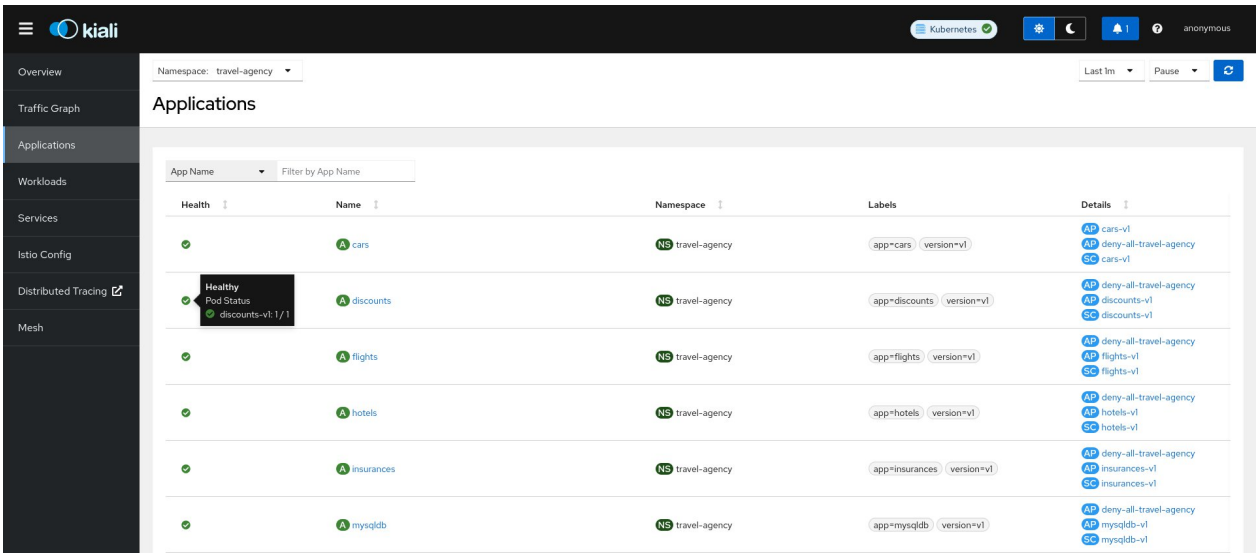


Рисунок 159 – Просмотр информации о состоянии приложений

6.3.9.3.2. Workloads

На дашборде «Workloads» представлена таблица со сводными данными о нагрузках сервис-меша. Информация в таблице разбита на следующие столбцы: состояние нагрузки («Health»), имя нагрузки («Name»), пространство имён («Namespace»), тип нагрузки («Type»), лейблы («Labels»), подробная информация («Details»). Выпадающий список над таблицей позволяет отфильтровать данные по имени и типу нагрузки, а также по другим параметрам.

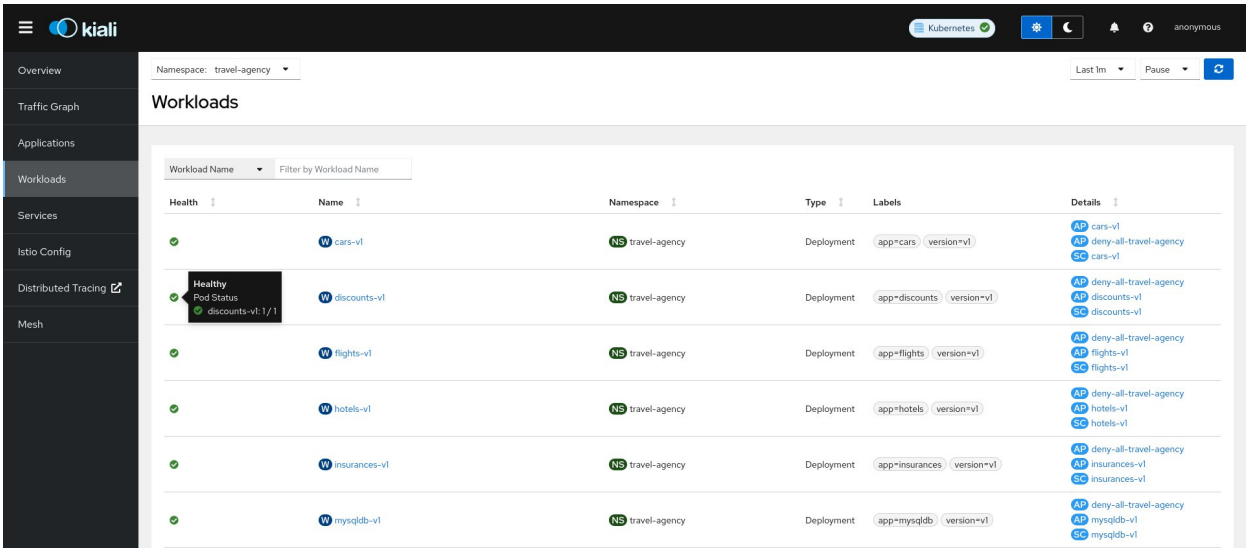


Рисунок 160 – Просмотр информации о состоянии нагрузок

6.3.9.3.3. Services

На дашборде «Services» представлена таблица со сводными данными о сервисах сервис-меша. Информация в таблице разбита на следующие столбцы: состояние сервиса («Health»), имя сервиса («Name»), пространство имён («Namespace»), лейблы («Labels»), состояние конфигурации сервиса («Configuration»), подробная информация («Details»). Выпадающий список над таблицей позволяет отфильтровать данные по имени и типу сервиса, а также по другим параметрам.

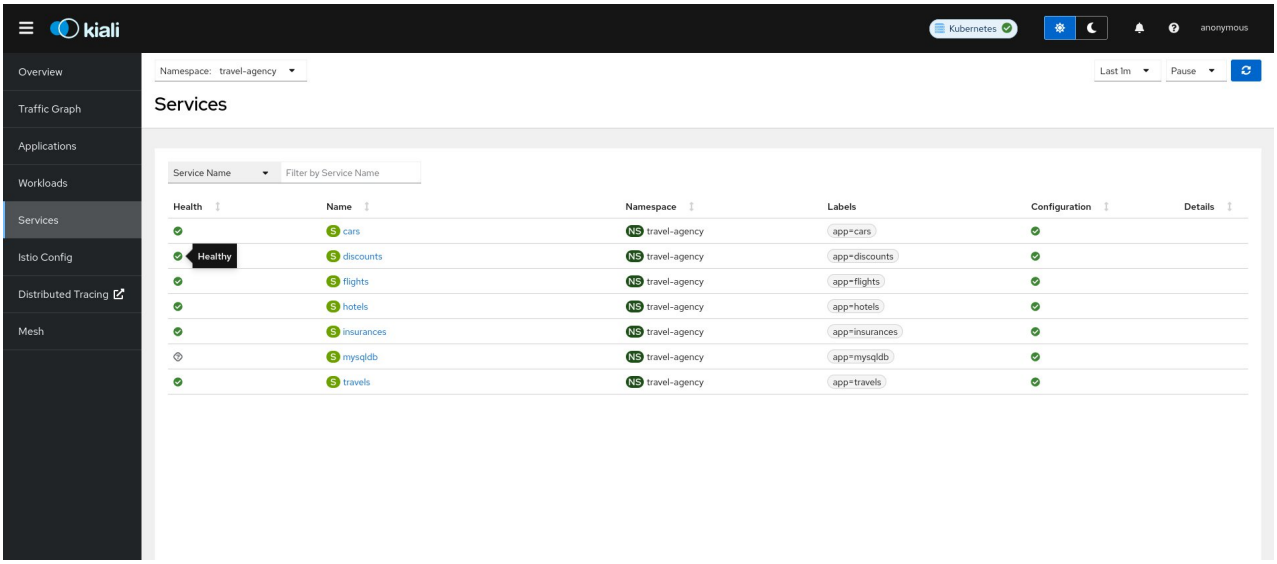


Рисунок 161 – Просмотр информации о состоянии сервисов

6.3.9.3.4. Istio config

На дашборде «Istio config» представлена таблица со сводными данными об объектах конфигурации Istio, используемых в сервис-меше. Информация в таблице разбита на следующие столбцы: имя объекта конфигурации («Name»), пространство имён («Namespace»), тип объекта («Type»), состояние конфигурации («Configuration»). При выборе одного из объектов откроется соответствующая YAML-конфигурация. Выпадающий список над таблицей позволяет отфильтровать отображаемые данные по типу конфигурации и другим параметрам.

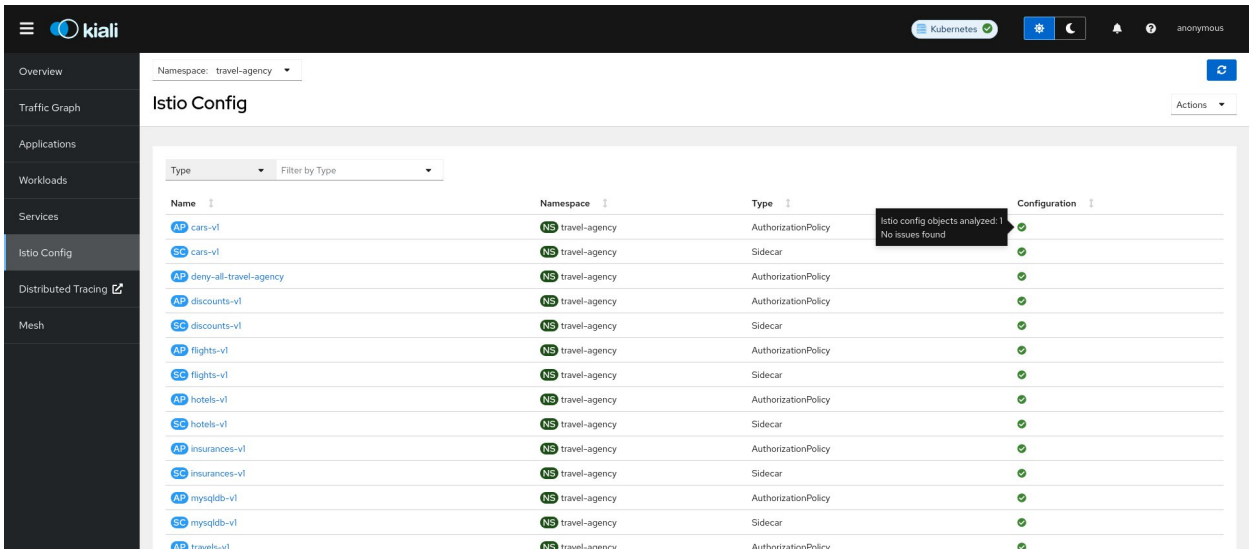


Рисунок 162 – Просмотр информации о состоянии объектов конфигурации Istio

6.3.9.4. Информация о трассировке

При интеграции Kiali с одной из поддерживаемых платформ для работы с распределёнными трассировками (Jaeger или Grafana Tempo) вы можете открыть дашборд на соответствующей платформе, выбрав в боковом меню пункт «Distributed Tracing».

6.4. Настройки логирования

В ПО «Deckhouse Platform» предусмотрен сбор и доставка логов из узлов и подов кластера в системы хранения. Предусмотрена возможность:

- собирать логи из всех или отдельных подов и пространств имен;
- 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.

Пользователям доступна настройка параметров сбора логов из приложения с помощью ресурса PodLoggingConfig, который описывает источник логов в рамках заданного пространства имен, включая правила сбора, фильтрации и парсинга.

6.4.1. Настройка сбора логов из приложения

Для настройки сбора логов из приложения следует выполнить следующие шаги:

1. Уточните у администратора ПО «Deckhouse Platform», 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 своем пространстве имен.

В примере ниже логи собираются со всех подов указанного пространства имен 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 определенных 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
```

6.4.2. Пример создания source в пространстве имен и чтение логов всех подов в нем с направлением их в Loki

Следующий pipeline создает source в пространстве имен test-whispers, читает логи всех подов в нем 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
```

6.4.3. Пример чтения подов в указанном пространстве имен с определенным лейблом

Пример настройки чтения подов с лейблом app=booking в пространстве имен 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
```

6.5. Работа с модулем виртуализации

6.5.1. Быстрый старт по созданию виртуальной машины

Рассмотрим пример создания виртуальной машины с Astra Linux.

```
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: root-disk-astra
spec:
  dataSource:
    objectRef:
      kind: VirtualImage
      name: astra-1-8-1
    type: ObjectRef
  persistentVolumeClaim:
    size: 40Gi
---
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: astra-1-8-1
spec:
  dataSource:
    http:
      url: https://download.astralinux.ru/artifactory/mg-
generic/alse/cloudinit/alse-1.8.1uu1-base-cloudinit-mg15.0.2-amd64.qcow2
    type: HTTP
  storage: ContainerRegistry
---
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: astra
spec:
  virtualMachineClassName: generic
  cpu:
    cores: 2
  memory:
    size: 4Gi
  provisioning:
    type: UserData
```

```
    userData: |
      #cloud-config
      users:
        - name: cloud
          passwd: !!!password!!!
          shell: /bin/bash
          sudo: ALL=(ALL) NOPASSWD:ALL
          chpasswd: {expire: False}
          lock_passwd: false
          ssh_authorized_keys:
            - !!!ssh-pub-key!!!
  blockDeviceRefs:
  - kind: VirtualDisk
    name: root-disk-astra
```

Проверьте с помощью команды, что образ и диск созданы, а виртуальная машина - запущена. Ресурсы создаются не мгновенно, поэтому прежде чем они придут в готовое состояние потребуется подождать какое-то время.

```
d8 k  get vi,vd,vm
```

Подключитесь с помощью консоли к виртуальной машине (для выхода из консоли необходимо нажать Ctrl+]), с помощью команды:

```
d8 v console linux-vm
```

Пример вывода:

```
Successfully connected to linux-vm console. The escape sequence is ^]
#
linux-vm login: cloud
Password: cloud
...
cloud@linux-vm:~$
```

Для удаления созданных ранее ресурсов используйте следующие команды:

```
d8 k delete vm astra
d8 k delete vd root-disk-astra
d8 k delete vi astra
```

6.5.2. Образы

Ресурс VirtualImage предназначен для загрузки образов виртуальных машин и их последующего использования для создания дисков виртуальных машин. Данный ресурс доступен только в неймспейсе или проекте в котором он был создан.

При подключении к виртуальной машине доступ к образу предоставляется в режиме «только чтение».

Процесс создания образа включает следующие шаги:

- Пользователь создает ресурс VirtualImage.

- После создания образ автоматически загружается из указанного в спецификации источника в хранилище (DVCR).
- После завершения загрузки, ресурс становится доступным для создания дисков.

Существуют различные типы образов:

ISO-образ — установочный образ, используемый для начальной установки операционной системы. Такие образы выпускаются производителями ОС и используются для установки на физические и виртуальные серверы.

Образ диска с предустановленной системой — содержит уже установленную и настроенную операционную систему, готовую к использованию после создания виртуальной машины. Готовые образы можно получить на ресурсах разработчиков дистрибутива, либо создать самостоятельно.

Примеры ресурсов для получения образов виртуальной машины:

Дистрибутив	Пользователь по умолчанию
AlmaLinux	almalinux
AlpineLinux	alpine
AltLinux	altlinux
AstraLinux	astra
CentOS	cloud-user
Debian	debian
Rocky	rocky
Ubuntu	ubuntu

Поддерживаются следующие форматы образов с предустановленной системой: qcow2, raw, vmdk, vdi.

Также файлы образов могут быть сжаты одним из следующих алгоритмов сжатия: gz, xz.

После создания ресурса, тип и размер образа определяются автоматически и эта информация отражается в статусе ресурса.

Образы могут быть загружены из различных источников, таких как HTTP-серверы, где расположены файлы образов, или контейнерные реестры. Также доступна возможность загрузки образов напрямую из командной строки с использованием утилиты curl.

Образы могут быть созданы из других образов и дисков виртуальных машин.

Проектный образ поддерживает два варианта хранения:

ContainerRegistry - тип по умолчанию, при котором образ хранится в DVCR.

PersistentVolumeClaim - тип, при котором в качестве хранилища для образа используется PVC. Этот вариант предпочтителен, если используется хранилище с поддержкой быстрого клонирования PVC, что позволяет быстрее создавать диски из образов.

6.5.2.1. Создание образа с HTTP-сервера

Пример создания образа с вариантом хранения в DVCR.

Выполните следующую команду для создания VirtualImage (укажите URL образа):

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: myimage
spec:
  # Сохраним образ в DVCR.
  storage: ContainerRegistry
  # Источник для создания образа.
  dataSource:
    type: HTTP
    http:
      url: https://<IMAGE_URL>.img
EOF
```

Проверьте результат создания VirtualImage:

```
d8 k get virtualimage myimage
```

Пример вывода:

```
NAME           PHASE   CDROM   PROGRESS   AGE
myimage        Ready   false   100%       23h
```

После создания ресурс VirtualImage может находиться в следующих состояниях (фазах):

Pending - ожидание готовности всех зависимых ресурсов, требующихся для создания образа.

WaitForUserUpload - ожидание загрузки образа пользователем (фаза присутствует только для type=Upload).

Provisioning - идет процесс создания образа.

Ready - образ создан и готов для использования.

Failed - произошла ошибка в процессе создания образа.

Terminating - идет процесс удаления Образа. Образ может «зависнуть» в данном состоянии, если он еще подключен к виртуальной машине.

До тех пор пока образ не перешел в фазу Ready, содержимое всего блока .spec допускается изменять. При изменении процесс создании диска запустится заново. После перехода в фазу Ready содержимое блока .spec менять нельзя!

Диагностика проблем с ресурсом осуществляется путем анализа информации в блоке .status.conditions.

Отследить процесс создания образа можно путем добавления ключа -w к предыдущей команде:

```
d8 k get vi myimage -w
```

Пример вывода:

```
NAME      PHASE          CDROM   PROGRESS   AGE
myimage   Provisioning   false              4s
myimage   Provisioning   false   0.0%       4s
myimage   Provisioning   false   28.2%      6s
myimage   Provisioning   false   66.5%      8s
myimage   Provisioning   false   100.0%     10s
myimage   Provisioning   false   100.0%     16s
myimage   Ready          false   100%       18s
```

В описании ресурса VirtualImage можно получить дополнительную информацию о скачанном образе:

```
d8 k describe vi myimage
```

Теперь рассмотрим пример создания образа с хранением его в PVC (укажите URL образа):

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: myimage-pvc
spec:
  # Настройки хранения проектного образа.
  storage: PersistentVolumeClaim
  persistentVolumeClaim:
    # Подставьте ваше название StorageClass.
    storageClassName: i-sds-replicated-thin-r2
  # Источник для создания образа.
  dataSource:
    type: HTTP
    http:
      url: https://<IMAGE_URL>.img
EOF
```

Проверьте результат создания VirtualImage:

d8 k get vi myimage-pvc

Пример вывода:

```
NAME         PHASE   CDROM   PROGRESS   AGE
myimage-pvc  Ready   false   100%       23h
```

Если параметр .spec.persistentVolumeClaim.storageClassName не указан, то будет использован StorageClass по умолчанию на уровне кластера, либо для образов, если он указан в настройках модуля.

6.5.2.2. Создание образа из Container Registry

Образ, хранящийся в Container Registry, имеет определенный формат. Рассмотрим на примере:

1) Загрузите образ локально (укажите URL образа):

```
curl -L https://<IMAGE_URL>.img -o myimage.img
```

2) Создайте Dockerfile со следующим содержимым:

```
FROM scratch
COPY myimage.img /disk/myimage.img
```

3) Соберите образ и загрузите его в container registry. В качестве container registry в примере ниже использован адрес registry.my (используйте свой).

```
docker build -t registry.my/<username>/myimage:latest
```

где username — имя пользователя в container registry registry.my.

4) Загрузите созданный образ в container registry:

```
docker push registry.my/<username>/myimage:latest
```

5) Чтобы использовать этот образ, создайте в качестве примера ресурс:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: myimage
spec:
  storage: ContainerRegistry
  dataSource:
    type: ContainerImage
    containerImage:
      image: registry.my/<username>/myimage:latest
EOF
```

Как создать образ из Container Registry в веб-интерфейсе:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Образы дисков».
- Нажмите «Создать образ».
- Из списка выберите «Загрузить данные из образа контейнера».
- В открывшейся форме в поле «Имя образа» введите имя образа.
- В поле «Хранилище» выберите ContainerRegistry.
- В поле «Образ в реестре контейнеров» укажите URL образа (например, registry.my/<username>/myimage:latest).

- Нажмите кнопку «Создать».
- Статус образа отображается слева вверху, под именем образа.

6.5.2.3. Загрузка образа из командной строки

Чтобы загрузить образ из командной строки, предварительно создайте ресурс, как представлено ниже на примере VirtualImage:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: some-image
spec:
  # Настройки хранения проектного образа.
  storage: ContainerRegistry
  # Настройки источника образа.
  dataSource:
    type: Upload
EOF
```

После создания, ресурс перейдет в фазу WaitForUserUpload, а это значит, что он готов для загрузки образа.

Доступно два варианта загрузки с узла кластера и с произвольного узла за пределами кластера:

```
d8 k get vi some-image -o jsonpath="{.status.imageUploadURLs}"  | jq
Пример вывода:
```

{

```
  "external":
"https://virtualization.example.com/upload/g2OuLgRhdAWqlJsCMyNvcdt4o5ERIwmm",
  "inCluster": "http://10.222.165.239/upload"
}
```

В качестве примера загрузите образ Cirros:

curl -L http://download.cirros-cloud.net/0.5.1/cirros-0.5.1-x86_64-disk.img -o cirros.img

Выполните загрузку образа с использованием следующей команды:

```
curl
https://virtualization.example.com/upload/g2OuLgRhdAWqlJsCMyNvcdt4o5ERIwmm  --
progress-bar -T cirros.img | cat
```

После завершения загрузки образ должен быть создан и перейти в фазу Ready:

```
d8 k get vi some-image
```

Пример вывода:

```
NAME         PHASE   CDROM   PROGRESS   AGE
some-image   Ready   false   100%       1m
```

Как загрузить образ из командной строки в веб-интерфейсе:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Образы дисков».

- Нажмите «Создать образ», далее в выпадающем меню выберите «Загрузить с компьютера».
- В поле «Имя образа» введите имя образа.
- В поле «Загрузить файл» нажмите ссылку «Выберите файл на вашем компьютере».
- Выберите файл в открывшемся файловом менеджере.
- Нажмите кнопку «Создать».
- Дождитесь пока образ перейдет в состояние Ready.

6.5.2.4. Создание образа из диска

Существует возможность создать образ из диска. Для этого необходимо выполнить одно из следующих условий:

- Диск не подключен ни к одной из виртуальных машин.
- Виртуальная машина, к которой подключен диск, находится в выключенном состоянии.

Пример создания образа из диска:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: linux-vm-root
spec:
  storage: ContainerRegistry
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualDisk
      name: linux-vm-root
EOF
```

Как в веб-интерфейсе создать образ из диска:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Образы дисков".
- Нажмите «Создать образ».
- Из списка выберите «Записать данные из диска».
- В открывшейся форме в поле «Имя образа» введите linux-vm-root.
- В поле «Хранилище» выберите ContainerRegistry.
- В поле «Диск» выберите из выпадающего списка необходимый диск.
- Нажмите кнопку «Создать».
- Статус образа отображается слева вверху, под его именем.

6.5.2.5. Создание образа из снимка диска

Можно создать образ из снимка. Для этого необходимо чтобы снимок диска находился в фазе готовности.

Пример создания образа из моментального снимка диска:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualImage
metadata:
  name: linux-vm-root
spec:
  storage: ContainerRegistry
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualDiskSnapshot
      name: linux-vm-root-snapshot
EOF
```

6.5.3. Диски

Диски в виртуальных машинах необходимы для записи и хранения данных, они обеспечивают полноценное функционирование приложений и операционных систем. Хранилище для этих дисков предоставляет платформа.

В зависимости от свойств хранилища, поведение дисков при создании и виртуальных машин в процессе эксплуатации может отличаться:

Свойство VolumeBindingMode:

Immediate - Диск создается сразу после создания ресурса (предполагается, что диск будет доступен для подключения к виртуальной машине на любом узле кластера).

WaitForFirstConsumer - Диск создается только после того как будет подключен к виртуальной машине и будет создан на том узле, на котором будет запущена виртуальная машина.

Режимы доступа AccessMode:

ReadWriteOnce (RWO) - доступ к диску предоставляется только одному экземпляру виртуальной машины. Живая миграция виртуальных машин с такими дисками невозможна.

ReadWriteMany (RWX) - множественный доступ к диску. Живая миграция виртуальных машин с такими дисками возможна.

При создании диска контроллер самостоятельно определит наиболее оптимальные параметры поддерживаемые хранилищем.

Внимание: Создать диски из iso-образов - нельзя!

Чтобы узнать доступные варианты хранилищ на платформе, выполните следующую команду:

```
d8 k  get storageclass
```

Пример вывода:

```
NAME                          PROVISIONER                     RECLAIMPOLICY
VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
```

```
i-sds-replicated-thin-r1  (default)       replicated.csi.storage.deckhouse.io
Delete          Immediate              true                    48d
i-sds-replicated-thin-r2          replicated.csi.storage.deckhouse.io   Delete
Immediate              true                    48d
i-sds-replicated-thin-r3          replicated.csi.storage.deckhouse.io   Delete
Immediate              true                    48d
sds-replicated-thin-r1            replicated.csi.storage.deckhouse.io   Delete
WaitForFirstConsumer   true                    48d
sds-replicated-thin-r2            replicated.csi.storage.deckhouse.io   Delete
WaitForFirstConsumer   true                    48d
sds-replicated-thin-r3            replicated.csi.storage.deckhouse.io   Delete
WaitForFirstConsumer   true                    48d
nfs-4-1-wffc                       nfs.csi.k8s.io                       Delete
WaitForFirstConsumer   true                    30d
```

6.5.3.1. Создание пустого диска

Пустые диски обычно используются для установки на них ОС, либо для хранения каких-либо данных.

Создайте диск:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: blank-disk
spec:
  # Настройки параметров хранения диска.
  persistentVolumeClaim:
    # Подставьте ваше название StorageClass.
    storageClassName: i-sds-replicated-thin-r2
    size: 100Mi
EOF
```

После создания ресурс VirtualDisk может находиться в следующих состояниях (фазах):

Pending - ожидание готовности всех зависимых ресурсов, требующихся для создания диска.

Provisioning - идет процесс создания диска.

Resizing - идет процесс изменения размера диска.

WaitForFirstConsumer - диск ожидает создания виртуальной машины, которая будет его использовать.

WaitForUserUpload - диск ожидает от пользователя загрузки образа (type: Upload).

Ready - диск создан и готов для использования.

Failed - произошла ошибка в процессе создания.

PVCLost - системная ошибка, PVC с данными утерян.

Terminating - идет процесс удаления диска. Диск может «зависнуть» в данном состоянии, если он еще подключен к виртуальной машине.

До тех пор пока диск не перешел в фазу Ready содержимое всего блока .spec допускается изменять. При изменении процесс создании диска запустится заново.

Диагностика проблем с ресурсом осуществляется путем анализа информации в блоке .status.conditions.

Если параметр .spec.persistentVolumeClaim.storageClassName не указан, то будет использован StorageClass по умолчанию на уровне кластера, либо для образов, если он указан в настройках модуля.

Проверьте состояние диска после создания командой:

```
d8 k get vd blank-disk
```

Пример вывода:

```
NAME       PHASE   CAPACITY   AGE
blank-disk   Ready   100Mi      1m2s
```

Как создать пустой диск в веб-интерфейсе (данный шаг можно пропустить и выполнить при создании ВМ):

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Диски ВМ».
- Нажмите «Создать диск».
- В открывшейся форме в поле «Имя диска» введите blank-disk.
- В поле «Размер» задайте размер с единицами измерений 100Mi.
- В поле «Имя StorageClass» можно выбрать StorageClass или оставить выбранный по умолчанию.
- Нажмите кнопку «Создать».
- Статус диска отображается слева вверху, под именем диска.

6.5.3.2. Создание диска из образа

Диск также можно создавать и заполнять данными из ранее созданных образов ClusterVirtualImage и VirtualImage.

При создании диска можно указать его желаемый размер, который должен быть равен или больше размера распакованного образа. Если размер не указан, то будет создан диск с размером, соответствующим исходному образу диска.

На примере ранее созданного проектного образа VirtualImage, рассмотрим команду, позволяющую определить размер распакованного образа:

```
d8 k get cvi myimage -o wide
```

Пример вывода:

```
NAME           PHASE   CDROM   PROGRESS   STOREDSIZE   UNPACKEDSIZE   REGISTRY
URL                                                                    AGE
```

```
myimage     Ready    false    100%        285.9Mi       2.5Gi           dvcr.d8-
virtualization.svc/cvi/myimage:eac95605-7e0b-4a32-bb50-cc7284fd89d0   122m
```

Искомый размер указан в колонке UNPACKEDSIZE и равен 2.5Gi.

Создайте диск из этого образа:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: linux-vm-root
spec:
  # Настройки параметров хранения диска.
  persistentVolumeClaim:
    # Укажем размер больше чем значение распакованного образа.
    size: 10Gi
    # Подставьте ваше название StorageClass.
    storageClassName: i-sds-replicated-thin-r2
  # Источник из которого создается диск.
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualImage
      name: myimage
EOF
```

А теперь создайте диск, без явного указания размера:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: linux-vm-root-2
spec:
  # Настройки параметров хранения диска.
  persistentVolumeClaim:
    # Подставьте ваше название StorageClass.
    storageClassName: i-sds-replicated-thin-r2
  # Источник из которого создается диск.
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualImage
      name: myimage
EOF
```

Проверьте состояние дисков после создания:

```
d8 k get vd
Пример вывода:
NAME           PHASE   CAPACITY   AGE
linux-vm-root    Ready   10Gi       7m52s
linux-vm-root-2  Ready   2590Mi     7m15s
```

Как создать диск из образа в веб-интерфейсе (данный шаг можно пропустить и выполнить при создании ВМ):

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Диски ВМ».
- Нажмите «Создать диск».

- В открывшейся форме в поле «Имя диска» введите linux-vm-root.
- В поле «Источник» убедитесь, что установлен чек-бокс «Проектные».
- Из выпадающего списка выберите интересующий Вас образ.
- В поле «Размер» можете изменить размер на больший или оставить выбранный по умолчанию.
- В поле «Имя StorageClass» можно выбрать StorageClass или оставить выбранный по умолчанию.
- Нажмите кнопку «Создать».
- Статус диска отображается слева вверху, под именем диска.

6.5.3.3. Изменение размера диска

Размер дисков можно увеличивать, даже если они уже подключены к работающей виртуальной машине. Для этого отредактируйте поле spec.persistentVolumeClaim.size:

Проверьте размер до изменения:

```
d8 k get vd linux-vm-root
```

Пример вывода:

```
NAME          PHASE   CAPACITY   AGE
linux-vm-root   Ready   10Gi       10m
```

Примените изменения:

```
d8 k patch vd linux-vm-root --type merge -p '{"spec":{"persistentVolumeClaim":{"size":"11Gi"}}}'
```

Проверьте размер после изменения:

```
d8 k get vd linux-vm-root
```

Пример вывода:

```
NAME          PHASE   CAPACITY   AGE
linux-vm-root   Ready   11Gi       12m
```

Как изменить размер диска в веб-интерфейсе:

Способ 1:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Диски ВМ».
- Выберите нужный диск и нажмите на символ карандаша в колонке «Размер».
- Во всплывающем окне можете изменить размер на больший.
- Нажмите на кнопку «Применить».
- Статус диска отображается в колонке «Статус».

Способ 2:

- Перейдите на вкладку «Проекты» и выберите нужный проект.

- Перейдите в раздел «Виртуализация» -> «Диски ВМ».
- Выберите нужный диск и нажмите на его имя.
- В открывшейся форме на вкладке «Конфигурация» в поле «Размер» можете изменить размер на больший.
- Нажмите на появившуюся кнопку «Сохранить».
- Статус диска отображается слева вверху, под его именем.

6.5.4. Виртуальные машины

Для создания виртуальной машины используется ресурс VirtualMachine. Его параметры позволяют сконфигурировать:

- класс виртуальной машины
- ресурсы, требуемые для работы виртуальной машины (процессор, память, диски и образы);
- правила размещения виртуальной машины на узлах кластера;
- настройки загрузчика и оптимальные параметры для гостевой ОС;
- политику запуска виртуальной машины и политику применения изменений;
- сценарии начальной конфигурации (cloud-init);
- перечень блочных устройств.

6.5.4.1. Создание виртуальной машины

Ниже представлен пример конфигурации виртуальной машины. В примере используется сценарий первичной инициализации виртуальной машины (cloud-init), который устанавливает гостевого агента qemu-guest-agent и сервис nginx, а также создает пользователя cloud с паролем cloud:

Пароль в примере был сгенерирован с использованием команды `mkpasswd --method=SHA-512 --rounds=4096 -S saltsalt` и при необходимости вы можете его поменять на свой:

Создайте виртуальную машину с диском созданным ранее:

```
d8 k apply -f - <<"EOF"
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachine
metadata:
  name: linux-vm
spec:
  # Название класса ВМ.
  virtualMachineClassName: host
  # Блок скриптов первичной инициализации ВМ.
  provisioning:
    type: UserData
```

Пример cloud-init-сценария для создания пользователя cloud с паролем cloud и установки сервиса агента qemu-guest-agent и сервиса nginx.

```
    userData: |
      #cloud-config
      package_update: true
      packages:
        - nginx
        - qemu-guest-agent
      run_cmd:
        - systemctl daemon-reload
        - systemctl enable --now nginx.service
        - systemctl enable --now qemu-guest-agent.service
      ssh_pwauth: True
      users:
      - name: cloud
        passwd:
'$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo
1RquzS0R6GgdIDlvS.kbUwDablGZKZcTP/'
        shell: /bin/bash
        sudo: ALL=(ALL) NOPASSWD:ALL
        lock_passwd: False
      final_message: "The system is finally up, after $UPTIME seconds"
  # Настройки ресурсов ВМ.
  cpu:
    # Количество ядер ЦП.
    cores: 1
    # Запросить 10% процессорного времени одного физического ядра.
    coreFraction: 10%
  memory:
    # Объем оперативной памяти.
    size: 1Gi
  # Список дисков и образов, используемых в ВМ.
  blockDeviceRefs:
    # Порядок дисков и образов в данном блоке определяет приоритет загрузки.
    - kind: VirtualDisk
      name: linux-vm-root
EOF
```

Проверьте состояние виртуальной машины после создания:

```
d8 k get vm linux-vm
```

Пример вывода:

```
NAME        PHASE     NODE           IPADDRESS     AGE
linux-vm   Running   virtlab-pt-2   10.66.10.12   11m
```

После создания виртуальная машина автоматически получит IP-адрес из диапазона, указанного в настройках модуля (блок virtualMachineCIDRs).

Как создать виртуальную машину в веб-интерфейсе:

1. Перейдите на вкладку «Проекты» и выберите нужный проект.
2. Перейдите в раздел «Виртуализация» -> «Виртуальные машины».
3. Нажмите «Создать».
4. В открывшейся форме в поле «Имя» введите linux-vm.
5. В разделе «Параметры машины» в поле «Ядер» задайте 1.
6. В разделе «Параметры машины» в поле «Доля ЦП» задайте 10%.
7. В разделе «Параметры машины» в поле «Размер» задайте 1Gi.

8. В разделе «Диски и образы» в подразделе «Загрузочные диски» нажмите «Добавить».
9. В открывшейся форме нажмите «Выбрать из существующих».
10. В списке выберите диск linux-vm-root.
11. Прокрутите страницу вниз до раздела «Дополнительные параметры».
12. Включите переключатель «Cloud-init».
13. В появившееся поле вставьте ваши данные:

```
#cloud-config
package_update: true
packages:
  - nginx
  - qemu-guest-agent
run_cmd:
  - systemctl daemon-reload
  - systemctl enable --now nginx.service
  - systemctl enable --now qemu-guest-agent.service
ssh_pwauth: True
users:
- name: cloud
  passwd:
'$6$rounds=4096$saltsalt$fPmUsbjAuA7mnQNTajQM6ClhesyG0.yyQhvahas02ejfMAq1ykBo
1RquzS0R6GgdIDlvS.kbUwDablGZKZcTP/'
  shell: /bin/bash
  sudo: ALL=(ALL) NOPASSWD:ALL
  lock_passwd: False
final_message: "The system is finally up, after $UPTIME seconds"
```

14. Нажмите кнопку «Создать».
15. Статус ВМ отображается слева вверху, под ее именем.

6.5.4.2. Жизненный цикл ВМ

Виртуальная машина проходит через несколько этапов своего существования — от создания до удаления. Эти этапы называются фазами и отражают текущее состояние ВМ. Чтобы понять, что происходит с ВМ, нужно проверить ее статус (поле .status.phase), а для более детальной информации — блок .status.conditions. Ниже описаны все основные фазы жизненного цикла ВМ, их значение и особенности.

Pending - ожидание готовности ресурсов. ВМ только что создана, перезапущена или запущена после остановки и ожидает готовности необходимых ресурсов (дисков, образов, ip-адресов и т.д.).

Возможные проблемы:

- не готовы зависимые ресурсы: диски, образы, классы ВМ, секрет со сценарием начальной конфигурации и пр.

Диагностика: В .status.conditions стоит обратить внимание на условия *Ready. По ним можно определить, что блокирует переход к следующей фазе, например, ожидание готовности дисков (BlockDevicesReady) или класса ВМ (VirtualMachineClassReady).

d8 k get vm <vm-name> -o json | jq '.status.conditions[] | select(.type | test(".*Ready"))'

Starting - запуск виртуальной машины. Все зависимые ресурсы ВМ - готовы, и система пытается запустить ВМ на одном из узлов кластера.

Возможные проблемы:

- нет подходящего узла для запуска;
- на подходящих узлах недостаточно CPU или памяти;
- превышены квоты неймспейса или проекта.

Диагностика:

Если запуск затягивается, проверьте .status.conditions, условие type: Running

```
d8 k get vm <vm-name> -o json | jq '.status.conditions[] | select(.type=="Running")'
```

Running - виртуальная машина запущена. ВМ успешно запущена и работает.

Особенности:

- при установленном в гостевой системе qemu-guest-agent, условие AgentReady будет истинно,а в .status.guestOSInfo будет отображена информация о запущенной гостевой ОС.
- условие type: FirmwareUpToDate, status: False информирует о том, что прошивку ВМ требуется обновить.
- условие type: ConfigurationApplied, status: False информирует о том, что конфигурация ВМ не применена для запущенной ВМ.
- условие type: SizingPolicyMatched, status: False означает, что конфигурация ресурсов виртуальной машины не соответствует политике сайзинга, заданной в связанном объекте VirtualMachineClass. Чтобы сохранить изменения в конфигурации ВМ, необходимо сначала привести ее параметры в соответствие с требованиями этой политики.
- условие type: AwaitingRestartToApplyConfiguration, status: True отображает информацию о необходимости выполнить вручную перезагрузку ВМ, т.к. некоторые изменения конфигурации невозможно применить без перезагрузки ВМ.

Возможные проблемы:

- Внутренний сбой в работе ВМ или гипервизора.

Диагностика:

Проверьте .status.conditions условие type: Migrating, а также блок .status.migrationState

d8 k get vm <vm-name> -o json | jq '.status | {condition: .conditions[] | select(.type=="Migrating"), migrationState}'

Условие type: SizingPolicyMatched, status: False отображает несоответствие конфигурации ресурсов политике сайзинга используемого VirtualMachineClass. При нарушении политики сохранить параметры ВМ без приведения ресурсов в соответствие политике - невозможно.

Условия отображают информацию о состоянии ВМ, а также на возникающие проблемы. Понять, что не так с ВМ можно путем их анализа:

d8 k get vm fedora -o json | jq '.status.conditions[] | select(.message != "")'

6.5.4.3. Настройки CPU и coreFraction

При создании виртуальной машины вы можете настроить количество процессорных ресурсов которые она будет использовать, с помощью параметров cores и coreFraction. Параметр cores задает количество виртуальных ядер процессора выделенных для ВМ. Параметр coreFraction задает гарантированную минимальную долю вычислительной мощности, выделяемой на каждое ядро.

Доступные значения coreFraction могут быть определены в ресурсе VirtualMachineClass для заданного диапазона ядер (cores), допускается использовать только эти значения.

Например, если указать cores: 2, для ВМ будет выделено два виртуальных ядра соответствующих двум физическим ядрам гипервизора. При coreFraction: 20% ВМ гарантировано получит не менее 20% процесорной мощности каждого ядра, независимо от загрузки узла гипервизора. При этом, если на узле есть свободные ресурсы, ВМ может использовать до 100% мощности каждого ядра, что позволяет достичь максимальной производительности. Таким образом ВМ гарантировано получает 0.2 CPU каждого физического ядра и может задействовать до 100% мощности двух ядер (2 CPU), если на узле есть незадействованные ресурсы.

Если параметр coreFraction не определен, каждому виртуальному ядру ВМ выделяется 100% ядра физического процессора гипервизора.

Пример конфигурации:

```
spec:
  cpu:
    cores: 2
    coreFraction: 20%
```

Такой подход позволяет обеспечить стабильную работу ВМ даже при высокой нагрузке в условиях переподписки процессорных ресурсов, когда виртуальным машинам выделено больше ядер, чем доступно на гипервизоре.

Параметры cores и coreFraction учитываются при планировании размещения ВМ на узлах. Гарантированная мощность (минимальная доля каждого ядра) учитывается при выборе узла, чтобы он мог обеспечить необходимую производительность для всех ВМ. Если узел не располагает достаточными ресурсами для выполнения гарантий, ВМ не будет запущена на этом узле.

Визуализация на примере виртуальных машин со следующими конфигурациями CPU, при размещении их на одном узле:

VM1:

```
spec:
  cpu:
    cores: 1
    coreFraction: 20%
```

VM2:

```
spec:
  cpu:
    cores: 1
    coreFraction: 80%
```

6.5.4.4. Настройка ресурсов и политика сайзинга

Политика сайзинга в VirtualMachineClass, заданная в разделе .spec.sizingPolicies, определяет правила настройки ресурсов виртуальных машин, включая количество ядер, объем памяти и долю использования ядер (coreFraction). Эта политика не является обязательной. Если она отсутствует для ВМ, можно указывать произвольные значения для ресурсов без строгих требований. Однако, если политика сайзинга присутствует, конфигурация виртуальной машины должна строго ей соответствовать. В противном случае сохранение конфигурации будет невозможно.

Политика делит количество ядер (cores) на диапазоны, например, 1–4 ядра или 5–8 ядер. Для каждого диапазона указывается, сколько памяти можно выделить (memory) на одно ядро и/или какие значения coreFraction разрешены.

Если конфигурация ВМ (ядра, память или coreFraction) не соответствует политике, в статусе появится условие type: SizingPolicyMatched, status: False.

Если изменить политику в VirtualMachineClass, конфигурацию существующих ВМ может потребоваться изменить в соответствии с новой политикой. Виртуальные машины, не соответствующие условиям новой политики продолжат работать, но любые изменения их

конфигурации нельзя будет сохранить до тех пор, пока они не будут соответствовать новым условиям.

Например:

```
spec:
  sizingPolicies:
    - cores:
        min: 1
        max: 4
      memory:
        min: 1Gi
        max: 8Gi
      coreFractions: [5, 10, 20, 50, 100]
    - cores:
        min: 5
        max: 8
      memory:
        min: 5Gi
        max: 16Gi
      coreFractions: [20, 50, 100]
```

Если ВМ использует 2 ядра, она попадает в диапазон 1–4 ядра. В этом случае объем памяти можно выбрать от 1 ГБ до 8 ГБ, а coreFraction — только из значений 5%, 10%, 20%, 50% или 100%. Для 6 ядер — диапазон 5–8 ядер, где объем памяти должен составлять от 5 ГБ до 16 ГБ, а coreFraction — 20%, 50% или 100%.

Помимо сайзинга виртуальных машин, политика также позволяет реализовать желаемую максимальную переподписку для ВМ. Например, указав в политике значение coreFraction: 20%, вы гарантируете любой ВМ не менее 20% вычислительных ресурсов процессора, что фактически определит максимально возможную переподписку в размере 5:1.

6.5.4.5. Топологии CPU

Топология CPU виртуальной машины (ВМ) определяет, как ядра процессора распределяются по сокетам. Это важно для обеспечения оптимальной производительности и совместимости с приложениями, которые могут зависеть от конфигурации процессора. В конфигурации ВМ вы задаете только общее количество ядер процессора, а топология (количество сокетов и ядер в каждом сокете) рассчитывается автоматически на основе этого значения.

Количество ядер процессора указывается в конфигурации ВМ следующим образом:

```
spec:
  cpu:
    cores: 1
```

Далее система автоматически определяет топологию в зависимости от заданного числа ядер. Правила расчета зависят от диапазона количества ядер и описаны ниже.

Если количество ядер от 1 до 16 (1 ≤ .spec.cpu.cores ≤ 16):

- используется 1 сокет;
- количество ядер в сокете равно заданному значению;
- шаг изменения: 1 (можно увеличивать или уменьшать количество ядер по одному);
- допустимые значения: любое целое число от 1 до 16 включительно.

Пример: Если задано .spec.cpu.cores = 8, то топология: 1 сокет с 8 ядрами.

Если количество ядер от 17 до 32 (16 < .spec.cpu.cores ≤ 32):

- используется 2 сокета;
- ядра равномерно распределяются между сокетами (количество ядер в каждом сокете одинаковое);
- шаг изменения: 2 (общее количество ядер должно быть четным);
- допустимые значения: 18, 20, 22, 24, 26, 28, 30, 32;
- ограничения: минимум 9 ядер в сокете, максимум 16 ядер в сокете.

Пример: Если задано .spec.cpu.cores = 20, то топология: 2 сокета по 10 ядер каждый.

Если количество ядер от 33 до 64 (32 < .spec.cpu.cores ≤ 64):

- используется 4 сокета;
- ядра равномерно распределяются между сокетами;
- шаг изменения: 4 (общее количество ядер должно быть кратно 4);
- допустимые значения: 36, 40, 44, 48, 52, 56, 60, 64;
- ограничения: минимум 9 ядер в сокете, максимум 16 ядер в сокете.

Пример: Если задано .spec.cpu.cores = 40, то топология: 4 сокета по 10 ядер - каждый.

Если количество ядер больше 64 (.spec.cpu.cores > 64):

- используется 8 сокетов;
- ядра равномерно распределяются между сокетами;
- шаг изменения: 8 (общее количество ядер должно быть кратно 8);
- допустимые значения: 72, 80, 88, 96 и так далее до 248;
- ограничения: минимум 9 ядер в сокете.

Пример: Если задано .spec.cpu.cores = 80, то топология: 8 сокетов по 10 ядер каждый.

Шаг изменения указывает, на сколько можно увеличивать или уменьшать общее количество ядер, чтобы они равномерно распределялись по сокетам.

Максимально возможное количество ядер - 248.

Текущая топология ВМ (количество сокетов и ядер в каждом сокете) отображается в статусе ВМ в следующем формате:

```
status:
  resources:
    cpu:
      coreFraction: 10%
      cores: 1
      requestedCores: "1"
      runtimeOverhead: "0"
      topology:
        sockets: 1
        coresPerSocket: 1
```

6.5.4.6. Агент гостевой ОС

Для повышения эффективности управления ВМ рекомендуется установить QEMU Guest Agent (агент) — инструмент, который обеспечивает взаимодействие между гипервизором и операционной системой внутри ВМ.

Агент обеспечивает создание консистентных снимков дисков и ВМ. Он позволяет получать информацию о работающей ОС, которая будет отражена в статусе ВМ. Пример:

```
status:
  guestOSInfo:
    id: fedora
    kernelRelease: 6.11.4-301.fc41.x86_64
    kernelVersion: "#1 SMP PREEMPT_DYNAMIC Sun Oct 20 15:02:33 UTC 2024"
    machine: x86_64
    name: Fedora Linux
    prettyName: Fedora Linux 41 (Cloud Edition)
    version: 41 (Cloud Edition)
    versionId: "41"
```

Позволяет отслеживать, что ОС действительно загрузилась:

```
d8 k get vm -o wide
```

Пример вывода (колонка AGENT):

```
NAME    PHASE    CORES   COREFRACTION   MEMORY   NEED RESTART   AGENT   MIGRATABLE
NODE           IPADDRESS     AGE
fedora   Running   6       5%             8000Mi   False          True    True
virtlab-pt-1   10.66.10.1   5d21h
```

Установка QEMU Guest Agent:

Для Debian-based ОС:

```
sudo apt install qemu-guest-agent
```

Для Centos-based ОС:

```
sudo yum install qemu-guest-agent
```

Запуск службы агента:

```
sudo systemctl enable --now qemu-guest-agent
```

6.5.4.7. Подключение к виртуальной машине

Для подключения к виртуальной машине доступны следующие способы:

- протокол удаленного управления (например SSH), который должен быть предварительно настроен на виртуальной машине.

- серийная консоль (serial console).
- протокол VNC.

Пример подключения к виртуальной машине с использованием серийной консоли:

```
d8 v console linux-vm
```

Пример вывода:

```
Successfully connected to linux-vm console. The escape sequence is ^]
linux-vm login: cloud
Password: cloud
```

Нажмите Ctrl+] для завершения работы с серийной консолью.

Пример команды для подключения по VNC:

```
d8 v vnc linux-vm
```

Пример команды для подключения по SSH:

```
d8 v ssh cloud@linux-vm --local-ssh
```

Как подключиться к виртуальной машине в веб-интерфейсе:

- перейдите на вкладку «Проекты» и выберите нужный проект.
- перейдите в раздел «Виртуализация» -> «Виртуальные машины».
- из списка выберите необходимую ВМ и нажмите на ее имя.
- в открывшейся форме перейдите на вкладку «TTY» для работы с серийной консолью.
- в открывшейся форме перейдите на вкладку «VNC» для подключения по VNC.
- перейдите в открывшееся окно. Здесь можно подключиться к ВМ.

6.5.4.8. Политика запуска и управление состоянием ВМ

Политика запуска виртуальной машины предназначена для автоматизированного управления состоянием виртуальной машины. Определяется она в виде параметра .spec.runPolicy в спецификации виртуальной машины. Поддерживаются следующие политики:

AlwaysOnUnlessStoppedManually - (по умолчанию) после создания ВМ всегда находится в запущенном состоянии. В случае сбоев работа ВМ восстанавливается автоматически. Остановка ВМ возможно только путем вызова команды d8 v stop или создания соответствующей операции.

AlwaysOn - после создания ВМ всегда находится в работающем состоянии, даже в случае ее выключения средствами ОС. В случае сбоев работа ВМ восстанавливается автоматически.

Manual - после создания состоянием ВМ управляет пользователь вручную с использованием команд или операций.

AlwaysOff - после создания ВМ всегда находится в выключенном состоянии. Возможность включения ВМ через команды\операции - отсутствует.

Состоянием виртуальной машины можно управлять с помощью следующих методов:

- Создание ресурса VirtualMachineOperation (vmop).

- Использование утилиты d8 с соответствующей подкомандой.

Ресурс VirtualMachineOperation декларативно определяет императивное действие, которое должно быть выполнено на виртуальной машине. Это действие применяется к виртуальной машине сразу после создания соответствующего vmop. Действие применяется к виртуальной машине один раз.

Пример операции для выполнения перезагрузки виртуальной машины с именем linux-vm:

```
d8 k create -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineOperation
metadata:
  generateName: restart-linux-vm-
spec:
  virtualMachineName: linux-vm
  # Тип применяемой операции = применяемая операция.
  type: Restart
EOF
```

Посмотреть результат действия можно с использованием команды:

```
d8 k get virtualmachineoperation
# или
d8 k get vmop
```

Аналогичное действие можно выполнить с использованием утилиты d8:

```
d8 v restart linux-vm
```

Перечень возможных операций приведен в таблице ниже:

d8	**vmop type**	**Действие**
d8 v stop	Stop	Остановить ВМ
d8 v start	Start	Запустить ВМ
d8 v restart	Restart	Перезапустить ВМ
d8 v evict	Evict	Мигрировать ВМ на другой узел

Чтобы выполнить операцию в веб-интерфейсе:

Перейдите на вкладку «Проекты» и выберите нужный проект.

Перейдите в раздел «Виртуализация» -> «Виртуальные машины».

Из списка выберите нужную виртуальную машину и нажмите кнопку с многоточием.

Во всплывающем меню можете выбрать возможные операции для ВМ.

6.5.4.9. Изменение конфигурации ВМ

Конфигурацию виртуальной машины можно изменять в любое время после создания ресурса VirtualMachine. Однако то, как эти изменения будут применены, зависит от текущей фазы виртуальной машины и характера внесенных изменений.

Изменения в конфигурацию виртуальной машины можно внести с использованием следующей команды:

```
d8 k edit vm linux-vm
```

Если виртуальная машина находится в выключенном состоянии (.status.phase: Stopped), внесенные изменения вступят в силу сразу после ее запуска.

Если виртуальная машина работает (.status.phase: Running), то способ применения изменений зависит от их типа:

Блок конфигурации	**Особенность применения**
.metadata.labels	Сразу
.metadata.annotations	Сразу
.spec.liveMigrationPolicy	Сразу
.spec.runPolicy	Сразу
.spec.disruptions.restartApprovalMode	Сразу
.spec.affinity	Сразу
.spec.nodeSelector	Сразу
.spec.*	Требуется перезапуск ВМ

Как изменить конфигурацию ВМ в веб-интерфейсе:

- перейдите на вкладку «Проекты» и выберите нужный проект.
- перейдите в раздел «Виртуализация» -> «Виртуальные машины».
- из списка выберите необходимую ВМ и нажмите на ее имя.
- вы находитесь на вкладке «Конфигурация», где можете вносить изменения.
- список измененных параметров и предупреждение, если необходимо перезапустить ВМ, отображаются вверху страницы.

Рассмотрим пример изменения конфигурации виртуальной машины:

Предположим, мы хотим изменить количество ядер процессора. В данный момент виртуальная машина запущена и использует одно ядро, что можно подтвердить, подключившись к ней через серийную консоль и выполнив команду nproc.

```
d8 v ssh cloud@linux-vm --local-ssh --command "nproc"
```

Пример вывода:

```
1
```

Примените следующий патч к виртуальной машине, чтобы изменить количество ядер с 1 на 2.

```
d8 k patch vm linux-vm --type merge -p '{"spec":{"cpu":{"cores":2}}}'
```

Пример вывода:

```
# virtualmachine.virtualization.deckhouse.io/linux-vm patched
```

Изменения в конфигурации внесены, но еще не применены к виртуальной машине. Проверьте это, повторно выполнив:

```
d8 v ssh cloud@linux-vm --local-ssh --command "nproc"
```

Пример вывода:

```
1
```

Для применения этого изменения необходим перезапуск виртуальной машины. Выполните следующую команду, чтобы увидеть изменения, ожидающие применения (требующие перезапуска):

```
d8 k get vm linux-vm -o jsonpath="{.status.restartAwaitingChanges}" | jq .
```

Пример вывода:

```
[
  {
    "currentValue": 1,
    "desiredValue": 2,
    "operation": "replace",
    "path": "cpu.cores"
  }
]
```

Выполните команду:

```
d8 k get vm linux-vm -o wide
```

Пример вывода:

```
NAME         PHASE      CORES    COREFRACTION    MEMORY    NEED RESTART    AGENT
MIGRATABLE   NODE           IPADDRESS     AGE
linux-vm     Running    2        100%            1Gi       True            True
True         virtlab-pt-1   10.66.10.13   5m16s
```

В колонке NEED RESTART мы видим значение True, а это значит что для применения изменений требуется перезагрузка.

Выполните перезагрузку виртуальной машины:

```
d8 v restart linux-vm
```

После перезагрузки изменения будут применены и блок .status.restartAwaitingChanges будет пустой.

Выполните команду для проверки:

```
d8 v ssh cloud@linux-vm --local-ssh --command "nproc"
```

Пример вывода:

```
2
```

Порядок применения изменений виртуальной машины через «ручной» рестарт является поведением по умолчанию. Если есть необходимость применять внесенные изменения сразу и автоматически, для этого нужно изменить политику применения изменений:

```
spec:
  disruptions:
    restartApprovalMode: Automatic
```

Чтобы выполнить операцию в веб-интерфейсе:

- перейдите на вкладку «Проекты» и выберите нужный проект.
- перейдите в раздел «Виртуализация" -> «Виртуальные машины».
- из списка выберите необходимую ВМ и нажмите на ее имя.
- на вкладке«“Конфигурация» прокрутите страницу вниз до раздела «Дополнительные параметры».
- включите переключатель «Автоприменение изменений».
- нажмите на появившуюся кнопку «Сохранить».

6.5.4.10. Сценарии начальной инициализации

Сценарии начальной инициализации предназначены для первичной конфигурации виртуальной машины при ее запуске.

В качестве сценариев начальной инициализации поддерживаются:

```
CloudInit.
Sysprep.
```

Сценарий CloudInit можно встраивать непосредственно в спецификацию виртуальной машины, но этот сценарий ограничен максимальной длиной в 2048 байт:

```
spec:
  provisioning:
    type: UserData
    userData: |
      #cloud-config
      package_update: true
      ...
```

При более длинных сценариях и/или наличия приватных данных, сценарий начальной инициализации виртуальной машины может быть создан в ресурсе Secret. Пример ресурса Secret со сценарием CloudInit приведен ниже:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: cloud-init-example
data:
  userData: <base64 data>
type: provisioning.virtualization.deckhouse.io/cloud-init
```

Фрагмент конфигурации виртуальной машины с при использовании скрипта начальной инициализации CloudInit хранящегося в ресурсе Secret:

```
spec:
  provisioning:
    type: UserDataRef
    userDataRef:
      kind: Secret
      name: cloud-init-example
```

Примечание: Значение поля .data.userData должно быть закодировано в формате Base64.

Для конфигурирования виртуальных машин под управлением ОС Windows с использованием Sysprep, поддерживается только вариант с ресурсом Secret.

Пример ресурса Secret с сценарием Sysprep приведен ниже:

```
apiVersion: v1
kind: Secret
metadata:
  name: sysprep-example
data:
  unattend.xml: <base64 data>
type: provisioning.virtualization.deckhouse.io/sysprep
```

Примечание: Значение поля .data.unattend.xml должно быть закодировано в формате Base64.

Фрагмент конфигурации виртуальной машины с использованием скрипта начальной инициализации Sysprep в ресурсе Secret:

```
spec:
  provisioning:
    type: SysprepRef
    sysprepRef:
      kind: Secret
      name: sysprep-example
```

6.5.4.11. Размещение ВМ по узлам

Для управления размещением виртуальных машин (параметров размещения) по узлам можно использовать следующие подходы:

- простое связывание по меткам (nodeSelector) — базовый способ выбора узлов с заданными метками.
- предпочтительное связывание (Affinity):
  - nodeAffinity — определяет приоритетные узлы для размещения.

- virtualMachineAndPodAffinity — обеспечивает совместное размещение ВМ и контейнеров.
- избежание совместного размещения (AntiAffinity):
  - virtualMachineAndPodAntiAffinity — предотвращает размещение ВМ и контейнеров на одном узле.

Все указанные параметры (включая параметр .spec.nodeSelector из VirtualMachineClass) применяются комплексно при планировании ВМ. Если хотя бы одно условие не может быть выполнено, запуск ВМ не будет выполнен. Для минимизации рисков рекомендуется:

- создавать непротиворечивые правила размещения;
- проверять совместимость правил до их применения;
- учитывать следующие типы условий:
  - жесткие (requiredDuringSchedulingIgnoredDuringExecution) — требуют строгого соблюдения;
  - мягкие (preferredDuringSchedulingIgnoredDuringExecution) — допускают частичное выполнение.

Использовать комбинации меток вместо одиночных ограничений. Например, вместо required для одного лейбла (например, env=prod) используйте несколько preferred условий.

Учитывать порядок запуска взаимозависимых ВМ. При использовании Affinity между ВМ (например, бэкенд зависит от базы данных) запускайте сначала ВМ, на которые ссылаются правила, чтобы избежать блокировок.

Планировать резервные узлы для критических нагрузок. Для ВМ с жесткими требованиями (например, AntiAffinity) предусмотрите дополнительные узлы, чтобы избежать простоев при сбое или выводе узла в режим обслуживания.

Узлы, на которых запускаются виртуальные машины, не должны иметь каких-либо ограничений по размещению подов (taints).

При изменении параметров размещения:

Если текущее расположение ВМ соответствует новым требованиям, она остается на текущем узле.

Если требования нарушаются:

ВМ автоматически перемещается на подходящий узел с помощью живой миграции.

Простое связывание по меткам (nodeSelector):

nodeSelector — это простейший способ контролировать размещение виртуальных машин, используя набор меток. Он позволяет задать, на каких узлах могут запускаться виртуальные машины, выбирая узлы с необходимыми метками.

```
spec:
  nodeSelector:
    disktype: ssd
```

В этом примере в кластере три узла: два с быстрыми дисками (disktype=ssd) и один с медленными (disktype=hdd). Виртуальная машина будет размещена только на узлах, которые имеют метку disktype со значением ssd.

Предпочтительное связывание (Affinity):

Affinity предоставляет более гибкие и мощные инструменты по сравнению с nodeSelector. Он позволяет задавать «предпочтения» и «обязательности» для размещения виртуальных машин. Affinity поддерживает два вида: nodeAffinity и virtualMachineAndPodAffinity.

Требования к размещению могут быть:

Жесткие (requiredDuringSchedulingIgnoredDuringExecution) — ВМ размещается только на узлах, удовлетворяющих условию.

Мягкие (preferredDuringSchedulingIgnoredDuringExecution) — ВМ размещается на подходящих узлах, если это возможно.

nodeAffinity — определяет узлы для запуска ВМ с помощью выражений селекторов меток.

Пример использования nodeAffinity с жестким правилом:

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
```

В этом примере в кластере три узла: два с быстрыми дисками (disktype=ssd) и один с медленными (disktype=hdd). Виртуальная машина будет размещена только на узлах, которые имеют метку disktype со значением ssd.

Если использовать мягкое требование (preferredDuringSchedulingIgnoredDuringExecution), то при отсутствии ресурсов для запуска ВМ на узлах с дисками disktype=ssd она будет запланирована на узле с дисками disktype=hdd.

virtualMachineAndPodAffinity управляет размещением виртуальных машин относительно других виртуальных машин. Он позволяет задавать предпочтение размещения виртуальных машин на тех же узлах, где уже запущены определенные виртуальные машины.

Пример мягкого правила:

```
spec:
  affinity:
    virtualMachineAndPodAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          podAffinityTerm:
            labelSelector:
              matchLabels:
                server: database
            topologyKey: "kubernetes.io/hostname"
```

В этом примере виртуальная машина будет размещена, если будет такая возможность (так как используется preferred) только на узлах на которых присутствует виртуальная машина с меткой server и значением database.

Избежание совместного размещения (AntiAffinity) используется для предотвращения совместного размещения ВМ на узлах. Полезно для обеспечения отказоустойчивости или балансировки нагрузки.

Требования к размещению могут быть:

Жесткие (requiredDuringSchedulingIgnoredDuringExecution) — ВМ размещается только на узлах, удовлетворяющих условию.

Мягкие (preferredDuringSchedulingIgnoredDuringExecution) — ВМ размещается на подходящих узлах, если это возможно.

Будьте осторожны при использовании жестких требований в небольших кластерах, где мало узлов для запуска виртуальных машин (ВМ). Если используется параметр virtualMachineAndPodAntiAffinity с типом requiredDuringSchedulingIgnoredDuringExecution для виртуальных машин, это означает, что каждая копия ВМ должна размещаться на отдельном узле. В условиях ограниченного количества узлов в кластере это может привести к ситуации, когда некоторые ВМ не смогут быть запущены из-за недостатка доступных узлов.

Термины Affinity и AntiAffinity применимы только к отношению между виртуальными машинами. Для узлов используемые привязки называются nodeAffinity. В nodeAffinity нет отдельного антитеза, как в случае с virtualMachineAndPodAffinity, но можно создать противоположные условия, задав отрицательные операторы в выражениях меток: чтобы акцентировать внимание на исключении определенных узлов, можно воспользоваться nodeAffinity с оператором, таким как NotIn.

Пример использования virtualMachineAndPodAntiAffinity:

```
spec:
  affinity:
    virtualMachineAndPodAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

```
        - labelSelector:
            matchLabels:
              server: database
          topologyKey: "kubernetes.io/hostname"
```

В данном примере создаваемая виртуальная машина не будет размещена на одном узле с виртуальной машиной с меткой server: database.

6.5.4.12. Статические и динамические блочные устройства

Блочные устройства можно разделить на два типа по способу их подключения: статические и динамические (hotplug).

Блочные устройства и их особенности:

Тип блочного устройства	Комментарий
VirtualImage	Подключается в режиме для чтения, или как cdrom для iso-образов
ClusterVirtualImage	Подключается в режиме для чтения, или как cdrom для iso-образов
VirtualDisk	Подключается в режиме для чтения и записи

Статические блочные устройства указываются в спецификации виртуальной машины в блоке .spec.blockDeviceRefs в виде списка. Порядок устройств в этом списке определяет последовательность их загрузки. Таким образом, если диск или образ указан первым, загрузчик сначала попробует загрузиться с него. Если это не удастся, система перейдет к следующему устройству в списке и попытается загрузиться с него. И так далее до момента обнаружения первого загрузчика.

Изменение состава и порядка устройств в блоке .spec.blockDeviceRefs возможно только с перезагрузкой виртуальной машины.

Фрагмент конфигурации VirtualMachine со статически подключенными диском и проектным образом:

```
spec:
  blockDeviceRefs:
    - kind: VirtualDisk
      name: <virtual-disk-name>
    - kind: VirtualImage
      name: <virtual-image-name>
```

Динамические блочные устройства можно подключать и отключать от виртуальной машины, находящейся в запущенном состоянии, без необходимости ее перезагрузки.

Для подключения динамических блочных устройств используется ресурс VirtualMachineBlockDeviceAttachment (vmbda).

Создайте ресурс, который подключит пустой диск blank-disk к виртуальной машине linux-vm:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineBlockDeviceAttachment
metadata:
  name: attach-blank-disk
spec:
  blockDeviceRef:
    kind: VirtualDisk
    name: blank-disk
  virtualMachineName: linux-vm
EOF
```

После создания VirtualMachineBlockDeviceAttachment может находиться в следующих состояниях (фазах):

Pending - ожидание готовности всех зависимых ресурсов.

InProgress - идет процесс подключения устройства.

Attached - устройство подключено.

Диагностика проблем с ресурсом осуществляется путем анализа информации в блоке .status.conditions.

Проверьте состояние вашего ресурса:

```
d8 k get vmbda attach-blank-disk
```

Пример вывода:

```
NAME                PHASE      VIRTUAL MACHINE NAME   AGE
attach-blank-disk   Attached   linux-vm               3m7s
Подключитесь к виртуальной машине и удостоверитесь, что диск подключен:
d8 v ssh cloud@linux-vm --local-ssh --command "lsblk"
```

Пример вывода:

```
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda       8:0    0   10G  0 disk <--- статично подключенный диск linux-vm-root
|-sda1    8:1    0  9.9G  0 part /
|-sda14   8:14   0    4M  0 part
`-sda15   8:15   0  106M  0 part /boot/efi
sdb       8:16   0    1M  0 disk <--- cloudinit
sdc      8:32   0 95.9M  0 disk <--- динамически подключенный диск blank-disk
```

Для отключения диска от виртуальной машины удалите ранее созданный ресурс:

```
d8 k delete vmbda attach-blank-disk
```

Подключение образов, осуществляется по аналогии. Для этого в качестве kind необходимо указать VirtualImage или ClusterVirtualImage и имя образа:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineBlockDeviceAttachment
metadata:
  name: attach-vm-iso
spec:
  blockDeviceRef:
```

```
    kind: VirtualImage # или ClusterVirtualImage
    name: myimage-iso
  virtualMachineName: linux-vm
EOF
```

6.5.4.13. Организация взаимодействия с ВМ

К виртуальным машинам можно обращаться напрямую по их фиксированным IP-адресам. Однако такой подход имеет ограничения: прямое использование IP-адресов требует ручного управления, усложняет масштабирование и делает инфраструктуру менее гибкой. Альтернативой служат сервисы — механизм, который абстрагирует доступ к ВМ, предоставляя логические точки входа вместо привязки к физическим адресам.

Сервисы упрощают взаимодействие как с отдельными ВМ, так и с группами подобных ВМ. Например, тип сервиса ClusterIP создает фиксированный внутренний адрес, через который можно обращаться как к одной, так и к группе ВМ, независимо от их реальных IP-адресов. Это позволяет другим компонентам системы взаимодействовать с ресурсами через стабильное имя или IP, автоматически направляя трафик к нужным машинам.

Сервисы также служат инструментом балансировки нагрузки: они равномерно распределяют запросы между всеми связанными машинами, обеспечивая отказоустойчивость и простоту расширения без необходимости перенастройки клиентов.

Для сценариев, где важен прямой доступ внутри кластера к конкретным ВМ (например, для диагностики или настройки кластеров), можно использовать headless-сервисы. Headless-сервисы не назначают общий IP, а вместо этого связывают DNS-имя с реальными адресами всех связанных машин. Запрос к такому имени возвращает список IP, что позволяет выбирать нужную ВМ вручную, сохраняя при этом удобство использования предсказуемых DNS-записей.

Для внешнего доступа сервисы дополняются механизмами вроде NodePort, который открывает порт на узле кластера, LoadBalancer, автоматически создающим облачный балансировщик нагрузки, или Ingress, управляющим маршрутизацией HTTP/HTTPS-трафика.

Все эти подходы объединяет способность скрывать сложность инфраструктуры за простыми интерфейсами: клиенты работают с конкретным адресом, а система сама решает, как направить запрос к нужной ВМ, даже если их количество или состояние меняется.

Имя сервиса формируется как <service-name>.<namespace or project name>.svc.<clustername>, или более коротко: <service-name>.<namespace or project name>.svc. Например, если имя вашего сервиса — http, а пространство имен — default, то полное DNS-имя будет http.default.svc.cluster.local.

Принадлежность ВМ к сервису определяется набором лейблов. Чтобы установить лейблы на ВМ в контексте управления инфраструктурой, используйте следующую команду:

```
d8 k label vm <vm-name> label-name=label-value
Пример команды:
d8 k label vm linux-vm app=nginx
```

Пример вывода команды:

```
virtualmachine.virtualization.deckhouse.io/linux-vm labeled
```

Headless сервис.

Headless-сервис позволяет легко направлять запросы внутри кластера без необходимости в балансировке нагрузки. Вместо этого он просто возвращает все IP-адреса виртуальных машин, подключенных к этому сервису.

Даже если вы используете headless-сервис только для одной виртуальной машины, это все равно полезно. Благодаря использованию DNS-имени, вы можете обращаться к машине, не завися от ее текущего IP-адреса. Это упрощает управление и настройку, потому что другие приложения внутри кластера могут использовать это DNS-имя для подключения вместо использования конкретного IP-адреса, который может измениться.

Пример создания headless-сервиса:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: http
  namespace: default
spec:
  clusterIP: None
  selector:
   # Лейбл по которому сервис определяет на какую виртуальную машину направлять
трафик.
    app: nginx
EOF
```

После создания, к ВМ или группе ВМ можно будет обратиться по имени: http.default.svc

Сервис с типом ClusterIP.

ClusterIP — это стандартный тип сервиса, который предоставляет внутренний IP-адрес для доступа к сервису внутри кластера. Этот IP-адрес используется для маршрутизации трафика между различными компонентами системы. ClusterIP позволяет виртуальным машинам взаимодействовать друг с другом через предсказуемый и стабильный IP-адрес, что упрощает внутреннюю коммуникацию в кластере.

Пример конфигурации ClusterIP:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
```

```
  name: http
spec:
  selector:
   # Лейбл по которому сервис определяет на какую виртуальную машину направлять
трафик.
    app: nginx
EOF
```

Сервис с типом NodePort.

NodePort — это расширение сервиса ClusterIP, которое обеспечивает доступ к сервису через заданный порт на всех узлах кластера. Это делает сервис доступным извне кластера через комбинацию IP адреса узла и порта.

NodePort подходит для сценариев, когда требуется непосредственный доступ к сервису извне кластера без использования внешнего балансировщика.

Создайте следующий сервис:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: linux-vm-nginx-nodeport
spec:
  type: NodePort
  selector:
   # Лейбл по которому сервис определяет на какую виртуальную машину направлять
трафик.
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 31880
EOF
```

В данном примере будет создан сервис с типом NodePort, который открывает внешний порт 31880 на всех узлах вашего кластера. Этот порт будет направлять входящий трафик на внутренний порт 80 виртуальной машины, где запущено приложение Nginx.

Если не указывать значение nodePort явно, для сервиса будет назначен произвольный порт, который можно посмотреть в статусе сервиса, сразу после его создания.

Сервис с типом LoadBalancer.

LoadBalancer — это тип сервиса, который автоматически создает внешний балансировщик нагрузки с постоянным IP-адресом. Этот балансировщик распределяет входящий трафик среди виртуальных машин, обеспечивая доступность сервиса из интернета.

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: linux-vm-nginx-lb
spec:
```

```
  type: LoadBalancer
  selector:
   # Лейбл по которому сервис определяет на какую виртуальную машину направлять
трафик.
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

Публикация сервисов виртуальной машины с использованием Ingress.

Ingress позволяет управлять входящими HTTP/HTTPS запросами и маршрутизировать их к различным серверам в рамках вашего кластера. Это наиболее подходящий метод, если вы хотите использовать доменные имена и SSL-терминацию для доступа к вашим виртуальным машинам.

Для публикации сервиса виртуальной машины через Ingress необходимо создать следующие ресурсы:

Внутренний сервис для связки с Ingress. Пример:

```
d8 k apply -f - <<EOF
apiVersion: v1
kind: Service
metadata:
  name: linux-vm-nginx
spec:
  selector:
   # Лейбл по которому сервис определяет на какую виртуальную машину направлять
трафик.
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

И ресурс Ingress для публикации. Пример:

```
d8 k apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: linux-vm
spec:
  rules:
    - host: linux-vm.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: linux-vm-nginx
                port:
                  number: 80
EOF
```

6.5.4.14. Живая миграция ВМ

Живая миграция виртуальных машин (ВМ) — это процесс перемещения работающей ВМ с одного физического узла на другой без ее отключения. Эта функция играет ключевую роль в управлении виртуализованной инфраструктурой, обеспечивая непрерывность работы приложений во время технического обслуживания, балансировки нагрузки или обновлений.

Процесс живой миграции включает несколько этапов:

1) Создание нового экземпляра ВМ.

На целевом узле создается новая ВМ в приостановленном состоянии. Ее конфигурация (процессор, диски, сеть) копируется с исходного узла.

2) Первичная передача памяти.

Вся оперативная память ВМ копируется на целевой узел по сети. Это называется первичной передачей.

3) Отслеживание изменений (Dirty Pages).

Пока память передается, ВМ продолжает работать на исходном узле и может изменять некоторые страницы памяти. Такие страницы называются «грязными» (dirty pages), и гипервизор их помечает.

4) Итеративная синхронизация.

После первичной передачи начинается повторная отправка только измененных страниц. Этот процесс повторяется в несколько циклов:

- Чем выше нагрузка на ВМ, тем больше «грязных» страниц появляется, и тем дольше длится миграция.
- При хорошей пропускной способности сети объем несинхронизированных данных постепенно уменьшается.

5) Финальная синхронизация и переключение.

Когда количество «грязных» страниц становится минимальным, ВМ на исходном узле приостанавливается (обычно на 100 миллисекунд):

- Оставшиеся изменения памяти передаются на целевой узел.
- Состояние процессора, устройств и открытых соединений синхронизируется.
- ВМ запускается на новом узле, а исходная копия удаляется.

Скорость сети играет важную роль. Если пропускная способность низкая, итераций становится больше, а время простоя ВМ может увеличиться. В худшем случае миграция может вообще не завершиться.

Механизм AutoConverge.

Если сеть не справляется с передачей данных, а количество «грязных» страниц продолжает расти, будет полезен механизм AutoConverge. Он помогает завершить миграцию даже при низкой пропускной способности сети.

Принципы работы механизма AutoConverge:

1) Замедление процессора ВМ.

Гипервизор постепенно снижает частоту процессора исходной ВМ. Это уменьшает скорость появления новых «грязных» страниц. Чем выше нагрузка на ВМ, тем сильнее замедление.

2) Ускорение синхронизации.

Как только скорость передачи данных превышает скорость изменения памяти, запускается финальная синхронизация, и ВМ переключается на новый узел.

3) Автоматическое завершение.

Финальная синхронизация запускается, когда скорость передачи данных превышает скорость изменения памяти.

AutoConverge гарантирует, что миграция завершится, даже если сеть не справляется с передачей данных. Однако замедление процессора может повлиять на производительность приложений, работающих на ВМ, поэтому его использование нужно контролировать.

Настройка политики миграции.

Для настройки поведения миграции используйте параметр .spec.liveMigrationPolicy в конфигурации ВМ. Допустимые значения параметра:

AlwaysSafe — Миграция всегда выполняется без замедления процессора (AutoConverge не используется). Подходит для случаев, когда важна максимальная производительность ВМ, но требует высокой пропускной способности сети.

PreferSafe — (используется в качестве политики по умолчанию) Миграция выполняется без замедления процессора (AutoConverge не используется). Однако можно запустить миграцию с замедлением процессора, используя ресурс VirtualMachineOperation с параметрами type=Evict и force=true.

AlwaysForced — Миграция всегда использует AutoConverge, то есть процессор замедляется при необходимости. Это гарантирует завершение миграции даже при плохой сети, но может снизить производительность ВМ.

PreferForced — Миграция использует AutoConverge, то есть процессор замедляется при необходимости. Однако можно запустить миграцию без замедления процессора, используя ресурс VirtualMachineOperation с параметрами type=Evict и force=false.

Виды миграции.

Миграция может осуществляться пользователем вручную, либо автоматически при следующих системных событиях:

- Обновлении «прошивки» виртуальной машины.
- Перераспределение нагрузки в кластере.
- Перевод узла в режим технического обслуживания (Drain узла)
- При изменении параметров размещения ВМ.

Триггером к живой миграции является появление ресурса VirtualMachineOperations с типом Evict.

Префиксы названия ресурса VirtualMachineOperations с типом Evict, создаваемые для живых миграций вызванных системными событиями:

Вид системного события	Префикс имени ресурса
Обновлении «прошивки»	firmware-update-*
Перераспределение нагрузки	evacuation-*
Drain узла	evacuation-*
Изменение параметров размещения	nodeplacement-update-*

Данный ресурс может находится в следующих состояниях:

Pending - ожидается выполнение операции.

InProgress - живая миграция выполняется.

Completed - живая миграция виртуальной машины завершилась успешно.

Failed - живая миграция виртуальной машины завершилась неуспешно.

Посмотреть активные операции можно с использованием команды:

```
d8 k get vmop
```

Пример вывода:

```
NAME                     PHASE       TYPE    VIRTUALMACHINE      AGE
firmware-update-fnbk2    Completed   Evict   linux-vm            1m
```

Прервать любую живую миграцию пока она находится в фазе Pending, InProgress можно удалив соответствующий ресурс VirtualMachineOperations.

Выполнение живой миграции ВМ с использованием VirtualMachineOperations.

Рассмотрим пример. Перед запуском миграции посмотрите текущий статус виртуальной машины:

```
d8 k get vm
```

```
Пример вывода:
NAME                                        PHASE      NODE            IPADDRESS
AGE
linux-vm                                    Running    virtlab-pt-1    10.66.10.14
79m
```

В выводе показано, что на данный момент она запущена на узле virtlab-pt-1.

Для осуществления миграции виртуальной машины с одного узла на другой, с учетом требований к размещению виртуальной машины используется команда:

```
d8 v evict -n <namespace> <vm-name>
```

Выполнение данной команды приводит к созданию ресурса VirtualMachineOperations.

Запустить миграцию можно также создав ресурс VirtualMachineOperations (vmop) с типом Evict вручную:

```
d8 k create -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineOperation
metadata:
  generateName: evict-linux-vm-
spec:
  # Имя виртуальной машины.
  virtualMachineName: linux-vm
  # Операция для миграции.
  type: Evict
  # Разрешить замедление процессора механизмом AutoConverge, для гарантии, что миграция выполнится.
  force: true
EOF
```

Для отслеживания миграции виртуальной машины сразу после создания ресурса vmop, выполните команду:

```
d8 k get vm -w
Пример вывода:
NAME                                         PHASE        NODE            IPADDRESS
AGE
linux-vm                                    Running      virtlab-pt-1    10.66.10.14
79m
linux-vm                                    Migrating    virtlab-pt-1    10.66.10.14
79m
linux-vm                                    Migrating    virtlab-pt-1    10.66.10.14
79m
linux-vm                                    Running      virtlab-pt-2    10.66.10.14
79m
```

Живая миграция ВМ при изменении параметров размещения

Рассмотрим механизм миграции на примере кластера с двумя группами узлов (NodeGroups): green и blue . Допустим, виртуальная машина (ВМ) изначально запущена на узле группы green , а ее конфигурация не содержит ограничений на размещение.

Шаг 1. Добавление параметра размещения Укажем в спецификации ВМ требование к размещению в группе green :

```
spec:
```

```
  nodeSelector:
    node.deckhouse.io/group: green
```

После сохранения изменений ВМ продолжит работать на текущем узле, так как условие nodeSelector уже выполняется.

Шаг 2. Изменение группы размещения Изменим требование на размещение в группе blue :

```
spec:
  nodeSelector:
    node.deckhouse.io/group: blue
```

Теперь текущий узел (группы green) не соответствует новым условиям. Система автоматически создаст объект VirtualMachineOperations типа Evict, что инициирует живую миграцию ВМ на доступный узел группы blue.

Пример вывода ресурса

```
NAME                          PHASE       TYPE    VIRTUALMACHINE      AGE
nodeplacement-update-dabk4    Completed   Evict   linux-vm            1m
```

6.5.5. IP-адреса ВМ

Блок .spec.settings.virtualMachineCIDRs в конфигурации модуля virtualization задает список подсетей для назначения ip-адресов виртуальным машинам (общий пул ip-адресов). Все адреса в этих подсетях доступны для использования, за исключением первого (адрес сети) и последнего (широковещательный адрес).

Ресурс VirtualMachineIPAddressLease (vmipl): кластерный ресурс, который управляет арендой IP-адресов из общего пула, указанного в virtualMachineCIDRs.

Чтобы посмотреть список аренд IP-адресов (vmipl), используйте команду:

```
d8 k get vmipl
```

Пример вывода:

```
NAME              VIRTUALMACHINEIPADDRESS                                    STATUS
AGE
ip-10-66-10-14     {"name":"linux-vm-7prpx","namespace":"default"}        Bound
12h
```

Ресурс VirtualMachineIPAddress (vmip): проектный/неймспейсный ресурс, который отвечает за резервирование арендованных IP-адресов и их привязку к виртуальным машинам. IP-адреса могут выделяться автоматически или по явному запросу.

По умолчанию IP-адрес виртуальной машине назначается автоматически из подсетей, определенных в модуле и закрепляется за ней до ее удаления. Проверить назначенный IP-адрес можно с помощью команды:

```
d8 k get vmip
Пример вывода:
NAME             ADDRESS        STATUS     VM          AGE
linux-vm-7prpx   10.66.10.14    Attached   linux-vm    12h
```

Алгоритм автоматического присвоения IP-адреса виртуальной машине выглядит следующим образом:

- Пользователь создает виртуальную машину с именем <vmname>.
- Контроллер модуля автоматически создает ресурс vmip с именем <vmname>-<hash>, чтобы запросить IP-адрес и связать его с виртуальной машиной.
- Для этого vmip создается ресурс аренды vmipl, который выбирает случайный IP-адрес из общего пула.
- Как только ресурс vmip создан, виртуальная машина получает назначенный IP-адрес.

IP-адрес виртуальной машине назначается автоматически из подсетей, определенных в модуле, и остается закрепленным за машиной до ее удаления. После удаления виртуальной машины ресурс vmip также удаляется, но IP-адрес временно остается закрепленным за проектом/неймспейсом и может быть повторно запрошен явно.

6.5.5.1. Запрос требуемого IP-адреса

Создайте ресурс vmip:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineIPAddress
metadata:
  name: linux-vm-custom-ip
spec:
  staticIP: 10.66.20.77
  type: Static
EOF
```

Создайте новую или измените существующую виртуальную машину и в спецификации укажите требуемый ресурс vmip явно:

spec:

```
  virtualMachineIPAddressName: linux-vm-custom-ip
```

6.5.5.2. Сохранение присвоенного IP-адреса

Чтобы автоматически выданный ip-адрес виртуальной машины не удалился вместе с самой виртуальной машиной выполните следующие действия.

Получите название ресурса vmip для заданной виртуальной машины:

```
d8 k get vm linux-vm -o jsonpath="{.status.virtualMachineIPAddressName}"
```

Пример вывода:

```
linux-vm-7prpx
```

Удалите блоки .metadata.ownerReferences из найденного ресурса:

```
d8 k patch vmip linux-vm-7prpx --type=merge --patch
'{"metadata":{"ownerReferences":null}}'
```

После удаления виртуальной машины, ресурс vmip сохранится и его можно будет переиспользовать снова во вновь созданной виртуальной машине:

```
spec:
```

```
  virtualMachineIPAddressName: linux-vm-7prpx
```

Даже если ресурс vmip будет удален. Он остается арендованным для текущего проекта/неймспейса еще 10 минут. Поэтому существует возможность вновь его занять по запросу:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineIPAddress
metadata:
  name: linux-vm-custom-ip
spec:
  staticIP: 10.66.20.77
  type: Static
EOF
```

6.5.6. Снимки

Снимки предназначены для сохранения состояния ресурса в конкретный момент времени. На данный момент времени поддерживаются снимки дисков и снимки виртуальных машин.

6.5.6.1. Создание снимков дисков

Для создания снимков виртуальных дисков используется ресурс VirtualDiskSnapshot . Эти снимки могут служить источником данных при создании новых дисков, например, для клонирования или восстановления информации.

Чтобы гарантировать целостность данных, снимок диска можно создать в следующих случаях:

- Диск не подключен ни к одной виртуальной машине.
- ВМ выключена.
- ВМ запущена, но установлен qemu-guest-agent в гостевой ОС. Файловая система успешно “заморожена” (операция fsfreeze).

Если консистентность данных не требуется (например, для тестовых сценариев), снимок можно создать:

- На работающей ВМ без “заморозки” файловой системы.
- Даже если диск подключен к активной ВМ.

Для этого в манифесте VirtualDiskSnapshot укажите:

```
spec:
  requiredConsistency: false
```

При создании снимка требуется указать названия класса снимка томов VolumeSnapshotClasses, который будет использоваться для создания снимка.

Для получения списка поддерживаемых ресурсов VolumeSnapshotClasses выполните команду:

```
d8 k get volumesnapshotclasses
```

```
Пример вывода:
NAME                      DRIVER                                      DELETIONPOLICY
AGE
csi-nfs-snapshot-class    nfs.csi.k8s.io                              Delete
34d
sds-replicated-volume         replicated.csi.storage.deckhouse.io     Delete
39d
Пример манифеста для создания снимка диска:

d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDiskSnapshot
metadata:
  name: linux-vm-root-snapshot
spec:
  requiredConsistency: true
  virtualDiskName: linux-vm-root
  volumeSnapshotClassName: sds-replicated-volume
EOF
Для просмотра списка снимков дисков, выполните следующую команду:
d k get vdsnapshot
Пример вывода:
NAME                   PHASE     CONSISTENT   AGE
linux-vm-root-snapshot Ready     true         3m2s
```

После создания VirtualDiskSnapshot может находиться в следующих состояниях (фазах):

Pending - ожидание готовности всех зависимых ресурсов, требующихся для создания снимка.

InProgress — идет процесс создания снимка виртуального диска.

Ready — создание снимка успешно завершено, и снимок виртуального диска доступен для использования.

Failed — произошла ошибка во время процесса создания снимка виртуального диска.

Terminating — ресурс находится в процессе удаления.

Диагностика проблем с ресурсом осуществляется путем анализа информации в блоке .status.conditions.

Как создать снимок диска в веб-интерфейсе:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Снимки дисков».
- Нажмите «Создать снимок диска».
- В поле «Имя снимка диска» введите имя для снимка.
- На вкладке «Конфигурация» в поле «Имя диска» выберите диск, с которого будет создан снимок.
- Включите переключатель «Гарантия целостности».
- Нажмите кнопку «Создать».
- Статус образа отображается слева вверху, под именем снимка.

6.5.6.2. Восстановление дисков из снимков

Для того чтобы восстановить диск из ранее созданного снимка диска, необходимо в качестве dataSource указать соответствующий объект:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualDisk
metadata:
  name: linux-vm-root
spec:
  # Настройки параметров хранения диска.
  persistentVolumeClaim:
    # Укажем размер больше чем значение.
    size: 10Gi
    # Подставьте ваше название StorageClass.
    storageClassName: i-sds-replicated-thin-r2
  # Источник из которого создается диск.
  dataSource:
    type: ObjectRef
    objectRef:
      kind: VirtualDiskSnapshot
      name: linux-vm-root-snapshot
EOF
```

Как восстановить диск из ранее созданного снимка в веб-интерфейсе:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Диски ВМ».
- Нажмите «Создать диск».
- В открывшейся форме в поле «Имя диска» введите имя для диска.
- В поле «Источник» убедитесь, что установлен чек-бокс «Снимки».
- Из выпадающего списка выберите снимок диска из которого хотите восстановиться.
- В поле «Размер» установите размер такой же или больше, чем размер оригинального диска.
- В поле «Имя StorageClass» введите «StorageClass» оригинального диска.
- Нажмите кнопку «Создать».
- Статус диска отображается слева вверху, под именем диска.

6.5.6.3. Создание снимков ВМ

Снимок виртуальной машины — это сохраненное состояние виртуальной машины в определенный момент времени. Для создания снимков виртуальных машин используется ресурс VirtualMachineSnapshot.

Рекомендуется отключить все образы (VirtualImage/ClusterVirtualImage) от виртуальной машины перед созданием ее снимка. Образы дисков не сохраняются вместе со снимком ВМ, и их

отсутствие в кластере при восстановлении может привести к тому, что виртуальная машина не сможет запуститься и будет находиться в состоянии Pending, ожидая доступности образа.

6.5.6.3.1. Типы снимков

Снимки могут быть консистентными и неконсистентными, за это отвечает параметр requiredConsistency, по умолчанию его значение равно true, что означает требование консистентного снимка.

Консистентный снимок гарантирует согласованное и целостное состояние дисков виртуальной машины. Такой снимок можно создать при выполнении одного из следующих условий:

- Виртуальная машина выключена.
- В гостевой системе установлен qemu-guest-agent, который на момент создания снимка временно приостанавливает работу файловой системы для обеспечения ее согласованности.

Неконсистентный снимок может не отражать согласованное состояние дисков виртуальной машины и ее компонентов. Такой снимок создается в следующих случаях:

- ВМ запущена, и в гостевой ОС не установлен или не запущен qemu-guest-agent.
- ВМ запущена, и в гостевой ОС не установлен qemu-guest-agent, но в манифесте снимка указан параметр requiredConsistency: false, и хочется избежать приостановки работы файловой системы.

Существует риск потери данных или нарушения их целостности при восстановлении из такого снимка.

6.5.6.3.2. Сценарии использования снимков

Снимки виртуальных машин можно использовать для реализации следующих сценариев:

- Восстановление ВМ на момент создания снимка.
- Создание клона ВМ / Использование снимка как шаблона для создания ВМ.

Если снимок планируется использовать как шаблон для создания других машин или клонов, перед его созданием выполните в гостевой ОС:

- Удаление персональных данных (файлы, пароли, история команд).
- Установку критических обновлений ОС.
- Очистку системных журналов.
- Сброс сетевых настроек.
- Удаление уникальных идентификаторов (например, через sysprep для Windows).

- Оптимизацию дискового пространства.
- Сброс конфигураций инициализации (cloud-init clean).
- Создавайте снимок с явным указанием не сохранять IP-адрес: keepIPAddress: Never.

При создании снимка следуйте следующим рекомендациям:

- Отключите все образы, если они были подключены к виртуальной машине.
- Не используйте статический IP-адрес в VirtualMachineIPAddress. Если использовался статический адрес, сконвертируйте его в автоматический.
- Создавайте снимок с явным указанием не сохранять IP-адрес: keepIPAddress: Never.

6.5.6.3.3. Создание снимков

При создании снимка необходимо указать названия классов снимков томов VolumeSnapshotClass, которые будут использованы для создания снимков дисков, подключенных к виртуальной машине.

Чтобы получить список поддерживаемых ресурсов VolumeSnapshotClasses, выполните команду:

```
d8 k get volumesnapshotclasses
Пример вывода:
NAME                     DRIVER                                      DELETIONPOLICY
AGE
csi-nfs-snapshot-class    nfs.csi.k8s.io                                    Delete
34d
sds-replicated-volume         replicated.csi.storage.deckhouse.io       Delete
39d
```

Создание снимка виртуальной машины будет неудачным, если выполнится хотя бы одно из следующих условий:

- не все зависимые устройства виртуальной машины готовы;
- есть изменения, ожидающие перезапуска виртуальной машины;
- среди зависимых устройств есть диск, находящийся в процессе изменения размера.
- При создании снимка динамический IP-адрес ВМ автоматически преобразуется в статический и сохраняется для восстановления.

Если не требуется преобразование и использование старого IP-адреса виртуальной машины, можно установить соответствующую политику в значение Never. В этом случае будет использован тип адреса без преобразования (Auto или Static).

```
spec:
  keepIPAddress: Never
Пример манифеста для создания снимка виртуальной машины:
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineSnapshot
metadata:
  name: linux-vm-snapshot
```

```
spec:
  virtualMachineName: linux-vm
  volumeSnapshotClasses:
    - storageClassName: i-sds-replicated-thin-r2 # Подставьте ваше название
StorageClass.
    volumeSnapshotClassName: sds-replicated-volume # Подставьте ваше название
VolumeSnapshotClass.
  requiredConsistency: true
  keepIPAddress: Never
EOF
```

После успешного создания снимка, в его статусе будет отражен перечень ресурсов, которые были сохранены в снимке.

Пример вывода:

```
status:
  ...
  resources:
  - apiVersion: virtualization.deckhouse.io/v1alpha2
    kind: VirtualMachine
    name: linux-vm
  - apiVersion: v1
    kind: Secret
    name: cloud-init
  - apiVersion: virtualization.deckhouse.io/v1alpha2
    kind: VirtualDisk
    name: linux-vm-root
```

Как создать снимок ВМ в веб-интерфейсе:

- Перейдите на вкладку «Проекты» и выберите нужный проект.
- Перейдите в раздел «Виртуализация» -> «Виртуальные машины».
- Из списка выберите необходимую ВМ и нажмите на ее имя.
- Перейдите на вкладку «Снимки».
- Нажмите кнопку «Создать».
- В открывшейся форме в поле «Имя снимка» введите linux-vm-snapshot.
- На вкладке «Конфигурация" в поле «Политика преобразования IP-адреса» выберите значение Never.
- Включите переключатель «Гарантия целостности».
- В поле «Класс хранилища снимка» выберите класс для снимка диска.
- Нажмите кнопку «Создать».
- Статус снимка отображается слева вверху, под именем снимка.

6.5.6.4. Восстановление из снимков

Для восстановления виртуальной машины из снимка используется ресурс VirtualMachineRestore . В процессе восстановления в кластере автоматически создаются следующие объекты:

VirtualMachine — основной ресурс ВМ с конфигурацией из снимка.

VirtualDisk — диски, подключенные к ВМ на момент создания снимка.

VirtualBlockDeviceAttachment — связи дисков с ВМ (если они существовали в исходной конфигурации).

VirtualMachineIPAddress — IP адрес виртуальной машины(если при создании снимка был указан параметр keepIPAddress: Always).

Secret — секреты с настройками cloud-init или sysprep (если они были задействованы в оригинальной ВМ).

Важно: ресурсы создаются только в том случае , если они присутствовали в конфигурации ВМ на момент создания снимка. Это гарантирует восстановление точной копии среды, включая все зависимости и настройки.

6.5.6.4.1. Восстановление ВМ

Для восстановления виртуальной машины используются два режима: Safe и Forced. Они определяются параметром restoreMode ресурса VirtualMachineRestore:

```
spec:
  restoreMode: Safe | Forced
Safe используется по умолчанию.
```

Чтобы восстановить виртуальную машину в режиме Safe, необходимо удалить ее текущую конфигурацию и все связанные диски. Это связано с тем, что процесс восстановления возвращает виртуальную машину и ее диски к состоянию, зафиксированному в момент создания резервного снимка.

Forced режим используется для того, чтобы привести уже существующую виртуальную машину к состоянию на момент создания снимка.

Forced может нарушить работу существующей виртуальной машины, так как во время восстановления она будет остановлена, ресурсы VirtualDisks и VirtualMachineBlockDeviceAttachments будут удалены для последующего восстановления.

Пример манифеста для восстановления виртуальной машины из снимка в режиме Safe:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineRestore
metadata:
  name: <restore name>
spec:
  restoreMode: Safe
  virtualMachineSnapshotName: <virtual machine snapshot name>
EOF
```

6.5.6.4.2. Создание клона ВМ / Использование снимка как шаблона для создания ВМ

Снимок виртуальной машины может использоваться как для создания ее точной копии (клона), так и в качестве шаблона для развертывания новых ВМ с аналогичной конфигурацией.

Для этого требуется создать ресурс VirtualMachineRestore и задать параметры переименования в блоке .spec.nameReplacements, чтобы избежать конфликтов имен.

Перечень ресурсов и их имена доступны в статусе снимка ВМ в блоке status.resources.

Пример манифеста для восстановления ВМ из снимка:

```
d8 k apply -f - <<EOF
apiVersion: virtualization.deckhouse.io/v1alpha2
kind: VirtualMachineRestore
metadata:
  name: <name>
spec:
  virtualMachineSnapshotName: <virtual machine snapshot name>
  nameReplacements:
    - from:
        kind: VirtualMachine
        name: <old vm name>
      to: <new vm name>
    - from:
        kind: VirtualDisk
        name: <old disk name>
      to: <new disk name>
    - from:
        kind: VirtualDisk
        name: <old secondary disk name>
      to: <new secondary disk name>
    - from:
        kind: VirtualMachineBlockDeviceAttachment
        name: <old attachment name>
      to: <new attachment name>
EOF
```

При восстановлении виртуальной машины из снимка важно учитывать следующие условия:

- Если ресурс VirtualMachineIPAddress уже существует в кластере, он не должен быть назначен другой ВМ .
- Для статических IP-адресов (type: Static) значение должно полностью совпадать с тем, что было зафиксировано в снимке.
- Секреты, связанные с автоматизацией (например, конфигурация cloud-init или sysprep), должны точно соответствовать восстанавливаемой конфигурации.

Несоблюдение этих требований приведет к ошибке восстановления, и ресурс VirtualMachineRestore перейдет в состояние Failed. Это связано с тем, что система проверяет целостность конфигурации и уникальность ресурсов для предотвращения конфликтов в кластере.

При восстановлении или клонировании виртуальной машины операция может быть выполнена успешно, но ВМ останется в статусе Pending. Это происходит, если ВМ зависит от ресурсов (например, образов дисков или классов виртуальных машин) или их конфигураций, которые были изменены или удалены на момент восстановления.

Проверьте блок условий ВМ с помощью команды:

```
d8 k vm get <vmname> -o json | jq '.status.conditions'
```

Проверьте вывод на наличие ошибок, связанных с отсутствующими или измененными ресурсами. Вручную обновите конфигурацию ВМ, чтобы устранить зависимости, которые больше не доступны в кластере.

7. Доставка приложений

7.1. Описание функциональных характеристик

ПО «Deckhouse Platform» организовывает:

- Сборку образов;
- Дистрибуцию образов в реестр контейнеров;
- Очистку реестра контейнеров от собранных образов;
- Дистрибуцию Helm-чартов;
- Дистрибуция бандлов — Helm-чартов и связанных с ними образов как единого целого;
- Развертывание Helm-чартов и бандлов в кластеры Kubernetes.

Основные функциональные характеристики ПО «Deckhouse Platform» включают:

Название функции	**Результат**
Сборка образов с Docker и Dockerfile и их дистрибуция: с распределенным кешированием слоев в реестре контейнерове, безопасными параллельными сборками, автоматическим тегированием образов на основе их содержимого и ранее собранных слоев и, как следствие, высокой воспроизводимостью сборки	Быстрые, безопасные, воспроизводимые сборки образов и хранение образов в реестре контейнеров
Очистка реестра контейнеров от собранных образов: умная очистка более ненужных образов на основе Git-политик и информации об используемых образах из Kubernetes-кластеров	Эффективное использование реестра контейнеров
Дистрибуция Helm-чартов: загрузка и скачивание Helm-чартов в OCI и HTTP репозитории Helm-чартов	Helm-чарты, готовые к развертыванию в Kubernetes
Дистрибуция бандлов: загрузка, скачивание, перенос бандлов между OCI-репозиториями, с возможностью доставлять бандлы в изолированные окружения	Бандлы, готовые к развертыванию в Kubernetes
Развертывание Helm-чартов и бандлов в Kubernetes-кластеры: установка, обновление и удаление Helm-чартов/бандлов в Kubernetes-кластерах, с тщательным отслеживанием состояния развертывания и гибко	Helm-чарты/бандлы, развернутые в Kubernetes

Название функции	Результат
настраиваемым порядком развертывания ресурсов	

7.2. Конфигурация проекта

7.2.1. Основы

ПО «Deckhouse Platform» следует принципам подхода IaC (Infrastructure as Code) и стимулирует пользователя хранить конфигурацию доставки проекта вместе с кодом приложения в Git и осознанно использовать внешние зависимости. Отвечает за это механизм под названием гитерминизм.

Типовая конфигурация проекта состоит из нескольких файлов:

- `werf.yaml`;
- одного или нескольких Dockerfile-файлов;
- Helm-чарта.

`werf.yaml` – главный конфигурационный файл проекта в ПО «Deckhouse Platform». Его основное предназначение — связывание инструкций для сборки и развертывания.

Инструкции для сборки – определяются для каждого компонента приложения. Представлены в формате Dockerfile-файлов.

Инструкции развертывания – определяются для всего приложения (и всех окружений развертывания) и должны быть представлены в виде Helm-чарта.

Пример типовой конфигурации проекта:

```
# werf.yaml
project: app
configVersion: 1
---
image: backend
context: backend
dockerfile: Dockerfile
---
image: frontend
context: frontend
dockerfile: Dockerfile
```

```
$ tree -a
.
├── .helm
│   ├── templates
│   │   ├── NOTES.txt
│   │   ├── _helpers.tpl
│   │   ├── deployment.yaml
│   │   ├── hpa.yaml
│   │   ├── ingress.yaml
│   │   ├── service.yaml
```

```
│   │   └── serviceaccount.yaml
│   └── values.yaml
├── backend
│   ├── Dockerfile
│   └── ...
├── frontend
│   ├── Dockerfile
│   └── ...
└── werf.yaml
```

7.2.2. Гитерминизм

7.2.2.1. О гитерминизме

Для обеспечения согласованности и гарантии воспроизводимости ПО «Deckhouse Platform» вводит механизм гитерминизма. Его название происходит от совмещения слов `git` и `determinism`, что можно понимать как режим «детерминированный Git». Конфигурацию и сборочный контекст ПО «Deckhouse Platform» читает из текущего коммита репозитория проекта, а также по умолчанию не позволяет использовать внешние зависимости.

Любое отступление пользователя от гитерминизма должно фиксироваться в специальном файле `werf-giterminism.yaml`, чтобы процесс управления конфигурацией был осмысленным, а использование прозрачным для всех участников доставки.

7.2.2.2. Исключение неиспользуемых файлов

ПО «Deckhouse Platform» не позволяет работать с незакоммиченными и неотслеживаемыми файлами в Git. Если файлы не требуются, то их следует явно исключать с помощью файлов `.gitignore` и `.helmignore`.

7.2.2.3. Использование незакоммиченных и неотслеживаемых файлов при отладке и разработке

При отладке и разработке изменение файлов проекта может доставлять неудобства за счет необходимости создания промежуточных коммитов. Мы работаем над режимом разработки, чтобы упростить этот процесс и в то же время оставить всю логику работы неизменной.

В текущих версиях режим разработки (активируется опцией `--dev`) работает с состоянием worktree Git-репозитория проекта, с отслеживаемыми (tracked) и неотслеживаемыми (untracked) файлами. ПО «Deckhouse Platform» игнорирует изменения с учетом правил, описанных в `.gitignore`, а также правил, заданных пользователем опцией `--dev-ignore=<glob>` (может использоваться несколько раз).

7.2.2.4. Выборочное разрешение недетерминированной функциональности

Использование функции Go-шаблонизатора `env` при шаблонизации werf.yaml усложняет совместное использование и воспроизводимость конфигурации в заданиях CI и среди разработчиков, поскольку значение переменной среды влияет на окончательный дайджест собираемых образов. Значение должно быть идентичным на всех этапах CI-пайплайна и во время локальной разработки при воспроизведении.

Для активации функции `env` необходимо использовать `werf-giterminism.yaml`, но мы рекомендуем еще раз подумать о возможных последствиях.

Использование директивы `contextAddFiles` при сборке Dockerfile-образа усложняет совместное использование и воспроизводимость конфигурации в заданиях CI и среди разработчиков, поскольку данные файла влияют на окончательный дайджест собираемых образов и должны быть идентичными на всех этапах CI-пайплайна и во время локальной разработки при воспроизведении.

Для активации директивы `contextAddFiles` необходимо использовать werf-giterminism.yaml, но мы рекомендуем еще раз подумать о возможных последствиях.

Использование алиасов тегов с опцией `--use-custom-tag` при развертывании с неизменяемыми значениями (например, `%image%-master`) делает предыдущие выкаты невоспроизводимыми и требует указания политики `imagePullPolicy: Always` для каждого образа при конфигурации контейнеров приложения в Helm-чарте.

Для активации опции `--use-custom-tag` необходимо использовать `werf-giterminism.yaml`, но мы рекомендуем еще раз подумать о возможных последствиях.

7.3. Сборка

7.3.1. Основы

Доставка приложения в Kubernetes предполагает его контейнеризацию (сборку одного или нескольких образов) для последующего развертывания в кластере.

Для сборки пользователю необходимо описать сборочные инструкции в виде Dockerfile’a, а остальное ПО «Deckhouse Platform» возьмет на себя:

- оркестрация одновременной/параллельной сборки образов приложения;
- кроссплатформенная и мультиплатформенная сборка образов;
- общий кеш промежуточных слоев и образов в container registry, доступный с любых раннеров;

- оптимальная схема тегирования, основанная на содержимом образа, предотвращающая лишние пересборки и время простоя приложения при выкате;
- система обеспечения воспроизводимости и неизменности образов для коммита: однажды собранные образы для коммита более не будут пересобраны.

Парадигма сборки и публикации образов в ПО «Deckhouse Platform» отличается от парадигмы, предлагаемой сборщиком Docker, в котором есть несколько команд: `build`, `tag` и `push`. ПО «Deckhouse Platform» собирает, тегирует и публикует образы в один шаг. Данная особенность связана с тем, что ПО «Deckhouse Platform» по сути не собирает образы, а синхронизирует текущее состояние приложения (для текущего коммита) с container registry, дособирая недостающие слои образов и синхронизируя работу параллельных сборщиков.

В общем случае сборка образов с ПО «Deckhouse Platform» предполагает наличие container registry (`--repo`), поскольку образ не только собирается, но и сразу публикуется. При этом ручной вызов команды `d8 dk build` не требуется, так как сборка выполняется автоматически при запуске всех верхнеуровневых команд ПО «Deckhouse Platform», в которых требуются образы (например, `d8 dk converge`).

Однако ручной запуск команды `d8 dk build` может быть полезным во время локальной разработки. В этом случае сборку можно запускать отдельно и без участия container registry.

7.3.2. Образы и зависимости

7.3.2.1. Добавление образов

Для сборки с ПО «Deckhouse Platform» необходимо добавить описание образов в `werf.yaml` проекта. Каждый образ добавляется директивой `image` с указанием имени образа:

```
project: example
configVersion: 1
---
image: frontend
# ...
---
image: backend
# ...
---
image: database
# ...
```

Имя образа — это уникальный внутренний идентификатор образа, который позволяет ссылаться на него при конфигурации и при вызове команд ПО «Deckhouse Platform».

Далее для каждого образа в `werf.yaml` необходимо определить сборочные инструкции с помощью Dockerfile.

7.3.2.1.1. Использование Dockerfile

Конфигурация сборки Dockerfile может выглядеть следующим образом:

```
# Dockerfile
FROM node
WORKDIR /app
COPY package*.json /app/
RUN npm ci
COPY . .
CMD ["node", "server.js"]
```

```
# werf.yaml
project: example
configVersion: 1
---
image: backend
dockerfile: Dockerfile
```

7.3.2.1.2. Использование определенной Dockerfile-стадии

Также вы можете описывать несколько целевых образов из разных стадий одного и того же Dockerfile:

```
# Dockerfile

FROM node as backend
WORKDIR /app
COPY package*.json /app/
RUN npm ci
COPY . .
CMD ["node", "server.js"]

FROM python as frontend
WORKDIR /app
COPY requirements.txt /app/
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "app:app", "-b", "0.0.0.0:80", "--log-file", "-"]
```

```
# werf.yaml
project: example
configVersion: 1
---
image: backend
dockerfile: Dockerfile
target: backend
---
image: frontend
dockerfile: Dockerfile
target: frontend
```

И конечно вы можете описывать образы, основанные на разных Dockerfile:

```
# werf.yaml
```

```
project: example
configVersion: 1
---
image: backend
dockerfile: dockerfiles/Dockerfile.backend
---
image: frontend
dockerfile: dockerfiles/Dockerfile.frontend
```

7.3.2.1.3. Выбор директории сборочного контекста

Чтобы указать сборочный контекст используется директива `context`. **Важно:** в этом случае путь до Dockerfile указывается относительно директории контекста:

```
project: example
configVersion: 1
---
image: docs
context: docs
dockerfile: Dockerfile
---
image: service
context: service
dockerfile: Dockerfile
```

Для образа `docs` будет использоваться Dockerfile по пути `docs/Dockerfile`, а для `service` — `service/Dockerfile`.

7.3.2.1.4. Добавление произвольных файлов в сборочный контекст

По умолчанию контекст сборки Dockerfile-образа включает только файлы из текущего коммита репозитория проекта. Файлы, не добавленные в Git, или некоммитнутые изменения не попадают в сборочный контекст. Такая логика действует в соответствии) по умолчанию.

Чтобы добавить в сборочный контекст файлы, которые не хранятся в Git, нужна директива `contextAddFiles` в `werf.yaml`, а также нужно разрешить использование директивы `contextAddFiles` в `werf-giterminism.yaml`:

```
# werf.yaml
project: example
configVersion: 1
---
image: app
context: app
contextAddFiles:
- file1
- dir1/
- dir2/file2.out
```

```
# werf-giterminism.yaml
giterminismConfigVersion: 1
config:
```

```
  dockerfile:
    allowContextAddFiles:
    - app/file1
    - app/dir1/
    - app/dir2/file2.out
```

В данной конфигурации контекст сборки будет состоять из следующих файлов:

- `app/**/*` из текущего коммита репозитория проекта;
- файлы `app/file1`, `app/dir2/file2.out` и директория `dir1`, которые находятся в директории проекта.

7.3.2.2. Взаимодействие между образами

7.3.2.2.1. Наследование и импортирование файлов

При написании одного Dockerfile в нашем распоряжении имеется механизм multi-stage. Он позволяет объявить в Dockerfile отдельный образ-стадию и использовать ее в качестве базового для другого образа, либо скопировать из нее отдельные файлы.

ПО «Deckhouse Platform» позволяет реализовать это не только в рамках одного Dockerfile, но и между произвольными образами, определяемыми в `werf.yaml`, в том числе собираемыми из разных Dockerfile’ов. Всю оркестрацию и выстраивание зависимостей ПО «Deckhouse Platform» возьмет на себя и произведет сборку за один шаг (вызов `d8 dk build`).

Пример использования образа собранного из `base.Dockerfile` в качестве базового для образа из Dockerfile:

```
# base.Dockerfile
FROM myimage:myversion
RUN apt update -q && apt install -y gcc g++ build-essential make curl python3
```

```
# Dockerfile
ARG BASE_IMAGE
FROM ${BASE_IMAGE}
WORKDIR /app
COPY . .
CMD [ "/app/server", "start" ]
```

```
# werf.yaml
project: example
configVersion: 1
---
image: base
dockerfile: base.Dockerfile
---
image: app
dockerfile: Dockerfile
dependencies:
- image: base
```

```
  imports:
  - type: ImageName
    targetBuildArg: BASE_IMAGE
```

7.3.2.2.2. Передача информации о собранном образе в другой образ

ПО «Deckhouse Platform» позволяет получить информацию о собранном образе при сборке другого образа. Например, если в сборочных инструкциях образа app требуются имена и digest’ы образов auth и controlplane, опубликованных в container registry, то конфигурация могла бы выглядеть так:

```
# modules/auth/Dockerfile
FROM alpine
WORKDIR /app
COPY . .
RUN ./build.sh
```

```
# modules/controlplane/Dockerfile
FROM alpine
WORKDIR /app
COPY . .
RUN ./build.sh
```

```
# Dockerfile
FROM alpine
WORKDIR /app
COPY . .

ARG AUTH_IMAGE_NAME
ARG AUTH_IMAGE_DIGEST
ARG CONTROLPLANE_IMAGE_NAME
ARG CONTROLPLANE_IMAGE_DIGEST

RUN  echo  AUTH_IMAGE_NAME=${AUTH_IMAGE_NAME}                                  >>
modules_images.env
RUN  echo  AUTH_IMAGE_DIGEST=${AUTH_IMAGE_DIGEST}                              >>
modules_images.env
RUN   echo   CONTROLPLANE_IMAGE_NAME=${CONTROLPLANE_IMAGE_NAME}                >>
modules_images.env
RUN     echo     CONTROLPLANE_IMAGE_DIGEST=${CONTROLPLANE_IMAGE_DIGEST}      >>
modules_images.env
```

```
# werf.yaml
project: example
configVersion: 1
---
image: auth
dockerfile: Dockerfile
context: modules/auth/
---
image: controlplane
dockerfile: Dockerfile
```

```
context: modules/controlplane/
---
image: app
dockerfile: Dockerfile
dependencies:
- image: auth
  imports:
  - type: ImageName
    targetBuildArg: AUTH_IMAGE_NAME
  - type: ImageDigest
    targetBuildArg: AUTH_IMAGE_DIGEST
- image: controlplane
  imports:
  - type: ImageName
    targetBuildArg: CONTROLPLANE_IMAGE_NAME
  - type: ImageDigest
    targetBuildArg: CONTROLPLANE_IMAGE_DIGEST
```

В процессе сборки ПО «Deckhouse Platform» автоматически подставит в указанные build-arguments соответствующие имена и идентификаторы. Всю оркестрацию и выстраивание зависимостей ПО «Deckhouse Platform» возьмет на себя и произведет сборку за один шаг (вызов `d8 dk build`).

7.3.2.3. Мультиплатформенная и кроссплатформенная сборка

ПО «Deckhouse Platform» позволяет собирать образы как для родной архитектуры хоста, где запущен ПО «Deckhouse Platform», так и в кроссплатформенном режиме с помощью эмуляции целевой архитектуры, которая может быть отлична от архитектуры хоста. Также ПО «Deckhouse Platform» позволяет собрать образ сразу для множества целевых платформ.

7.3.2.3.1. Сборка образов под одну целевую платформу

По умолчанию в качестве целевой используется платформа хоста, где запущен ПО «Deckhouse Platform». Выбор другой целевой платформы для собираемых образов осуществляется с помощью параметра `--platform`:

```
d8 dk build --platform linux/arm64
```

— все конечные образы, указанные в `werf.yaml`, будут собраны для указанной платформы с использованием эмуляции.

Целевую платформу можно также указать директивой конфигурации `build.platform`:

```
# werf.yaml
project: example
configVersion: 1
build:
  platform:
  - linux/arm64
---
image: frontend
```

```
dockerfile: frontend/Dockerfile
---
image: backend
dockerfile: backend/Dockerfile
```

В этом случае запуск `d8 dk build` без параметров вызовет сборку образов для указанной платформы (при этом явно указанный параметр `--platform` переопределяет значение из `werf.yaml`).

7.3.2.3.2. Сборка образов под множество целевых платформ

Поддерживается и сборка образов сразу для набора архитектур. В этом случае в container registry публикуется манифест включающий в себя собранные образы под каждую из указанных платформ (во время скачивания такого образа автоматически будет выбираться образ под требуемую архитектуру).

Можно определить общий список платформ для всех образов в `werf.yaml` с помощью конфигурации:

```
# werf.yaml
project: example
configVersion: 1
build:
  platform:
  - linux/arm64
  - linux/amd64
  - linux/arm/v7
```

Можно определить список целевых платформ отдельно для каждого собираемого образа (такая настройка будет иметь приоритет над общим списком определенным в `werf.yaml`):

```
# werf.yaml
project: example
configVersion: 1
---
image: mysql
dockerfile: ./Dockerfile.mysql
platform:
- linux/amd64
---
image: backend
dockerfile: ./Dockerfile.backend
platform:
- linux/amd64
- linux/arm64
```

Общий список можно также переопределить параметром `--platform` непосредственно в момент вызова сборки:

```
d8 dk build --platform=linux/amd64,linux/i386
```

— такой параметр переопределяет список целевых платформ указанных в `werf.yaml` (как общих, так и для отдельных образов).

7.3.3. Сборочный процесс

7.3.3.1. Аутентификация в container registry

Перед работой с образами необходимо аутентифицироваться в container registry:

```
d8 dk cr login <registry url>
```

Например:

```
# Login with username and password from command line
d8 dk cr login -u username -p password registry.example.com

# Login with token from command line
d8 dk cr login -p token registry.example.com

# Login into insecure registry (over http)
d8 dk cr login --insecure-registry registry.example.com
```

В случае использования команды `d8 dk ci-env` с поддерживаемыми CI/CD-системами аутентификация во встроенные container registry выполняется в рамках команды, поэтому использование команды `d8 dk cr login` в этом случае не требуется.

7.3.3.2. Тегирование образов

Тегирование образов с ПО «Deckhouse Platform» выполняется автоматически в рамках сборочного процесса. Используется оптимальная схема тегирования, основанная на содержимом образа, которая предотвращает лишние пересборки и время простоя приложения при выкате.

7.3.3.2.1. Получение тегов

Для получения тегов образов может использоваться опция `--save-build-report` для команд `d8 dk build, d8 dk converge` и пр.:

```
# По умолчанию формат JSON.
d8 dk build --save-build-report --repo REPO

# Поддерживается формат envfile.
d8 dk converge --save-build-report --build-report-path .werf-build-report.env
--repo REPO

# В команде рендера финальные теги будут доступны только с параметром --repo.
d8 dk render --save-build-report --repo REPO
```

Получить теги заранее, не вызывая сборочный процесс, на данный момент невозможно, можно получить лишь теги уже собранных ранее образов.

7.3.3.2.2. Добавление произвольных тегов

Пользователь может добавить произвольное количество дополнительных тегов с опцией `--add-custom-tag`:

```
d8 dk build --repo REPO --add-custom-tag main

# Можно добавить несколько тегов-алиасов.
d8 dk build --repo REPO --add-custom-tag main --add-custom-tag latest --add-custom-tag prerelease
```

Шаблон тега может включать следующие параметры:

- `%image%`, `%image_slug%` или `%image_safe_slug%` для использования имени образа из `werf.yaml` (обязательно при сборке нескольких образов);
- `%image_content_based_tag%` для использования content-based тега.

```
d8 dk build --repo REPO --add-custom-tag "%image%-latest"
```

При использовании опций создаются **дополнительные теги-алиасы**, ссылающиеся на автоматические теги-хэши. Полное отключение создания автоматических тегов не предусматривается.

7.3.3.3. Послойное кэширование образов

Послойное кэширование образов является неотъемлемой частью сборочного процесса ПО «Deckhouse Platform». ПО «Deckhouse Platform» сохраняет и переиспользует сборочный кэш в container registry, а также синхронизирует работу параллельных сборщиков.

Предполагается, что репозиторий образов для проекта не будет удален или очищен сторонними средствами без негативных последствий для пользователей CI/CD, построенного на основе `d8 dk cleanup`.

7.3.3.3.1. Dockerfile

По умолчанию Dockerfile-образы кешируются одним образом в container registry.

Для включения послойного кеширования Dockerfile-инструкций в container registry необходимо использовать директиву `staged` в `werf.yaml`:

```
# werf.yaml
image: example
dockerfile: ./Dockerfile
staged: true
```

7.3.3.3.2. Параллельность и порядок сборки образов

Все образы, описанные в `werf.yaml`, собираются параллельно на одном сборочном хосте. При наличии зависимостей между образами сборка разбивается на этапы, где каждый этап содержит набор независимых образов и может собираться параллельно.

При использовании Dockerfile-стадий параллельность их сборки также определяется на основе дерева зависимостей. Также, если Dockerfile-стадия используется разными образами, объявленными в `werf.yaml`, ПО «Deckhouse Platform» обеспечит однократную сборку этой общей стадии без лишних пересборок

Параллельная сборка в ПО «Deckhouse Platform» регулируется двумя параметрами `--parallel` и `--parallel-tasks-limit`. По умолчанию параллельная сборка включена и собирается не более 5 образов одновременно.

Рассмотрим следующий пример:

```
# backend/Dockerfile
FROM node as backend
WORKDIR /app
COPY package*.json /app/
RUN npm ci
COPY . .
CMD ["node", "server.js"]
```

```
# frontend/Dockerfile

FROM ruby as application
WORKDIR /app
COPY Gemfile* /app
RUN bundle install
COPY . .
RUN bundle exec rake assets:precompile
CMD ["rails", "server", "-b", "0.0.0.0"]

FROM nginx as assets
WORKDIR /usr/share/nginx/html
COPY configs/nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=application /app/public/assets .
COPY --from=application /app/vendor .
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

```
image: backend
dockerfile: Dockerfile
context: backend
---
image: frontend
dockerfile: Dockerfile
context: frontend
target: application
---
image: frontend-assets
```

```
dockerfile: Dockerfile
context: frontend
target: assets
```

Имеется 3 образа `backend`, `frontend` и `frontend-assets`. Образ `frontend-assets` зависит от `frontend`, потому что он импортирует скомпилированные ассеты из `frontend`.

Формируются следующие наборы для сборки:

```
┌ Concurrent builds plan (no more than 5 images at the same time)
│ Set #0:
│ - ⛴ image backend
│ - ⛴ image frontend
│
│ Set #1:
│ - ⛴ frontend-assets
└ Concurrent builds plan (no more than 5 images at the same time)
```

7.3.3.3.3. Использование container registry

При использовании ПО «Deckhouse Platform» container registry используется не только для хранения конечных образов, но также для сборочного кэша и служебных данных, необходимых для работы ПО «Deckhouse Platform» (например, метаданные для очистки container registry на основе истории Git). Репозиторий container registry задается параметром `--repo`:

```
d8 dk converge --repo registry.mycompany.org/project
```

В дополнение к основному репозиторию существует ряд дополнительных:

- `--final-repo` для сохранения конечных образов в отдельном репозитории;
- `--secondary-repo` для использования репозитория в режиме `read-only` (например, для использования container registry CI, в который нельзя пушить, но можно переиспользовать сборочный кэш);
- `--cache-repo` для поднятия репозитория со сборочным кэшом рядом со сборщиками.

Важно: Для корректной работы ПО «Deckhouse Platform» container registry должен быть надежным (persistent), а очистка должна выполняться только с помощью специальной команды `d8 dk cleanup`.

7.3.3.3.4. Дополнительный репозиторий для конечных образов

С помощью параметра `--cache-repo` можно указать один или несколько т.н. **кеширующих** репозиториев.

```
# Дополнительный кэширующий репозиторий в локальной сети.
d8 dk build --repo registry.mycompany.org/project --cache-repo
localhost:5000/project
```

Кеширующий репозиторий может помочь сократить время загрузки сборочного кэша, но для этого скорость загрузки из него должна быть значительно выше по сравнению с основным репозиторием — как правило, это достигается за счет поднятия container registry в локальной сети, но это необязательно.

При загрузке сборочного кэша кеширующие репозитории имеют больший приоритет, чем основной репозиторий. При использовании кеширующих репозиториев сборочный кэш продолжает сохраняться и в основном репозитории.

Очистка кеширующего репозитория может осуществляться путем его полного удаления без каких-либо рисков.

7.3.3.3.5. Синхронизация сборщиков

Для обеспечения согласованности в работе параллельных сборщиков, а также гарантии воспроизводимости образов и промежуточных слоев, ПО «Deckhouse Platform» берет на себя ответственность за синхронизацию сборщиков.

Сервис синхронизации — это компонент ПО «Deckhouse Platform», который предназначен для координации нескольких процессов ПО «Deckhouse Platform» и выполняет роль менеджера блокировок. Блокировки требуются для корректной публикации новых образов в container registry и реализации алгоритма сборки.

На сервис синхронизации отправляются только обезличенные данные в виде хэш-сумм тегов, публикуемых в container registry.

В качестве сервиса синхронизации может выступать:

- HTTP-сервер синхронизации, реализованный в команде `d8 dk synchronization`.
- Ресурс ConfigMap в кластере Kubernetes. В качестве механизма используется библиотека `lockgate`, реализующая распределенные блокировки через хранение аннотаций в выбранном ресурсе.
- Локальные файловые блокировки, предоставляемые операционной системой.

7.3.3.3.5.1. HTTP-сервер

Сервер синхронизации можно запустить командой d8 dk synchronization, например для использования порта 55581 (по умолчанию):

```
d8 dk synchronization --host 0.0.0.0 --port 55581
```

— данный сервер поддерживает только работу в режиме HTTP, для использования HTTPS необходима настройка дополнительной SSL-терминации сторонними средствами (например через Ingress в Kubernetes).

Далее во всех командах ПО «Deckhouse Platform», которые используют параметр `--repo` дополнительно указывается параметр `--synchronization=http[s]://DOMAIN`, например:

```
d8 dk build --repo registry.mydomain.org/repo --synchronization https://synchronization.domain.org
d8 dk converge --repo registry.mydomain.org/repo --synchronization https://synchronization.domain.org
```

7.3.3.3.5.2. Специальный ресурс в Kubernetes

Требуется лишь предоставить рабочий кластер Kubernetes, и выбрать namespace, в котором будет хранится сервисный `ConfigMap/werf`, через аннотации которого будет происходить распределенная блокировка.

Далее во всех командах ПО «Deckhouse Platform», которые используют параметр `--repo` дополнительно указывается параметр `--synchronization=kubernetes://NAMESPACE[:CONTEXT][@(base64:CONFIG_DATA)|CONFIG_PATH]`, например:

```
# Используем стандартный ~/.kube/config или KUBECONFIG.
d8 dk build --repo registry.mydomain.org/repo --synchronization kubernetes://mynamespace
d8 dk converge --repo registry.mydomain.org/repo --synchronization kubernetes://mynamespace

# Явно указываем содержимое kubeconfig через base64.
d8 dk build --repo registry.mydomain.org/repo --synchronization kubernetes://mynamespace@base64:YXBpVmVyc2lvbjogdjEKa2luZDogQ29uZmlnCnByZWZlcmVuY2VzOiB7fQoKY2x1c3RlcnM6Ci0gY2x1c3RlcjoKICBuYW1lOiBkZXZlbG9wbWVudAotIGNsdXN0ZXI6CiAgbmFtZTogc2NyYXRjaAoKdXNlcnM6Ci0gbmFtZTogZGV2ZWxvcGVyCi0gbmFtZTogZXhwZXJpbWVudGVyCgpjb250ZXh0czoKLSBjb250ZXh0OgogIG5hbWU6IGRldi1mcm9udGVuZAotIGNvbnRleHQ6CiAgbmFtZTogZGV2LXN0b3JhZ2UKLSBjb250ZXh0OgogIG5hbWU6IGV4cC1zY3JhdGNoCg==

# Используем контекст mycontext в конфиге /etc/kubeconfig.
d8 dk build --repo registry.mydomain.org/repo --synchronization kubernetes://mynamespace:mycontext@/etc/kubeconfig
```

Данный способ неудобен при доставке проекта в разные кластера Kubernetes из одного Git-репозитория из-за сложности корректной настройки. В этом случае для всех команд ПО «Deckhouse Platform» требуется указывать один и тот же адрес кластера и ресурс, даже если деплой происходит в разные контура, чтобы обеспечить консистивность данных в container registry.

Поэтому для такого случая рекомендуется запустить отдельный общий сервис синхронизации, чтобы исключить вероятность некорректной конфигурации.

7.3.3.3.5.3. Локальная синхронизация

Включается опцией `--synchronization=:local`. Локальный менеджер блокировок использует файловые блокировки, предоставляемые операционной системой.

```
d8 dk build --repo registry.mydomain.org/repo --synchronization :local
d8 dk converge --repo registry.mydomain.org/repo --synchronization :local
```

Данный способ подходит лишь в том случае, если в вашей CI/CD системе все запуски ПО «Deckhouse Platform» происходят с одного и того же раннера.

7.4. Развертывание

7.4.1. Основы

При организации доставки приложения в Kubernetes необходимо определиться с тем, какой формат выбрать для управления конфигурацией развертывания (параметризации, управления зависимостями, конфигурации под различные окружения и т.д.), а также способом применения этой конфигурации – непосредственно механизмом развертывания.

В ПО «Deckhouse Platform» встроен Helm, и именно он используется для решения перечисленных задач. Разработка и сопровождение конфигурации реализуется с помощью Helm-чарта, а для процесса развертывания предлагается Helm с дополнительными возможностями:

- отслеживание состояния выкатываемых ресурсов (с возможностью изменения поведения для каждого ресурса):
  - умное ожидание готовности ресурсов;
  - мгновенное завершение проблемного развертывания без необходимости ожидания таймаута;
  - прогресс развертывания, логи, системные события и ошибки приложения.
- использование произвольного порядка развертывания для любых ресурсов, а не только для хуков;
- ожидание создания и готовности ресурсов, не принадлежащих релизу;
- интеграция сборки и развертывания и многое другое.

ПО «Deckhouse Platform» стремится сделать работу с Helm более простой, удобной и гибкой, при этом не ломая обратную совместимость с Helm-чартами, Helm-шаблонами и Helm-релизами.

7.4.1.1. Простой пример развертывания

Для развертывания простого приложения достаточно двух файлов и команды `d8 dk converge`, запущенной в Git-репозитории приложения:

```
# .helm/templates/hello.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - image: nginxdemos/hello:plain-text
```

```
# werf.yaml:
configVersion: 1
project: hello
```

```
d8 dk converge --repo registry.example.org/repo --env production
```

Результат:

```
Deployment hello развернут в Namespace’е hello-production.
```

7.4.1.2. Расширенный пример развертывания

Более сложный пример развертывания со сборкой образов и внешними Helm-чартами:

```
# Dockerfile:
FROM node
WORKDIR /app
COPY . .
RUN npm ci
CMD ["node", "server.js"]
```


```
# Dockerfile:
FROM node
WORKDIR /app
COPY . .
RUN npm ci
CMD ["node", "server.js"]
```


```
# .helm/Chart.yaml:
dependencies:
- name: postgresql
  version: "~12.1.9"
```

```
  repository: https://charts.bitnami.com/bitnami
```

```
# .helm/values.yaml:
backend:
  replicas: 1
```

```
# .helm/templates/backend.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: {{ $.Values.backend.replicas }}
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - image: {{ $.Values.werf.image.backend }}
```

```
d8 dk converge --repo registry.example.org/repo --env production
```

Результат: собран образ `backend`, а затем Deployment `backend` и ресурсы чарта `postgresql` развернуты в Namespace'е `myapp-production`.

7.4.2. Чарты и зависимости

7.4.2.1. О чартах

Чарты в ПО «Deckhouse Platform» – это Helm-чарты с некоторыми дополнительными возможностями. А Helm-чарты – это распространяемые пакеты с Helm-шаблонами, values-файлами и некоторыми метаданными. Из чартов формируются конечные Kubernetes-манифесты для дальнейшего развертывания.

7.4.2.2. Создание нового чарта

При развертывании с `d8 dk converge` или публикации с `d8 dk bundle publish` используется чарт, лежащий в директории `.helm` в корне Git-репозитория. Этот чарт называется основным. Директорию основного чарта можно изменить директивой `deploy.helmChartDir` файла `werf.yaml`.

Для обычного чарта требуется создание файла `Chart.yaml` и указание в нем имени и версии чарта:

```
# Chart.yaml:
```

```
apiVersion: v2
name: mychart
version: 1.2.3
```

А вот для основного чарта это не обязательно, т. к. при отсутствии файла `Chart.yaml` или отсутствии в нем имени или версии чарта будет использована следующая конфигурация:

```
# .helm/Chart.yaml:
apiVersion: v2
name: <имя проекта ПО «Deckhouse Platform»>
version: 1.0.0
```

Если такая конфигурация основного чарта вас не устраивает, то создайте файл `.helm/Chart.yaml` самостоятельно и переопределите вышеупомянутые директивы.

В случае, если ваш чарт будет содержать только именованные шаблоны для использования в других чартах, добавьте в `Chart.yaml` директиву `type: library`:

```
# Chart.yaml:
type: library
```

Если ваш чарт совместим только с частью версий Kubernetes, то ограничьте версии кластера Kubernetes, в которые чарт можно развернуть, директивой `kubeVersion`, например:

```
# Chart.yaml:
kubeVersion: "~1.20.3"
```

При желании можно добавить следующие информационные директивы:

```
# Chart.yaml:
appVersion: "1.0"
deprecated: false
icon: https://example.org/mychart-icon.svg
description: This is My Chart
home: https://example.org
sources:
  - https://github.com/my/chart
keywords:
  - apps
annotations:
  anyAdditionalInfo: here
maintainters:
  - name: John Doe
    email: john@example.org
    url: https://john.example.org
```

Полученный чарт уже можно развернуть или опубликовать, хотя в таком виде от него мало пользы. Теперь вам понадобится, по меньшей мере, либо добавить шаблоны в директорию `templates`, либо подключить зависимые чарты.

7.4.2.3. Добавление файлов в чарт

По мере необходимости добавьте в чарт шаблоны, параметры, зависимые чарты и прочее. Содержимое основного чарта может выглядеть так:

```
.helm/
  charts/
    dependent-chart/
      # ...
  templates/
    deployment.yaml
    _helpers.tpl
    NOTES.txt
  crds/
    crd.yaml
  secret/
    some-secret-file
  values.yaml
  values.schema.json
  secret-values.yaml
  Chart.yaml
  Chart.lock
  README.md
  LICENSE
  .helmignore
```

Подробнее:

- `charts/*` — зависимые чарты, чьи Helm-шаблоны/values-файлы используются для формирования манифестов наравне с Helm-шаблонами/values-файлами родительского чарта;
- `templates/*.yaml` — Helm-шаблоны, из которых формируются Kubernetes-манифесты;
- `templates/*.tpl` — файлы с Helm-шаблонами для использования в других Helm-шаблонах. Результат шаблонизации этих файлов игнорируется;
- `templates/NOTES.txt` — ПО «Deckhouse Platform» отображает содержимое этого файла в терминале в конце каждого удачного развертывания;
- `crds/*.yaml` — Custom Resource Definitions, которые развертываются до развертывания манифестов в `templates`;
- `secret/*` — зашифрованные файлы, расшифрованное содержимое которых можно подставлять в Helm-шаблоны;
- `values.yaml` — файлы с декларативной конфигурацией для использования в Helm-шаблонах. Конфигурация в них может переопределяться переменными окружения, аргументами командной строки или другими values-файлами;
- `values.schema.json` — JSON-схема для валидации `values.yaml`;

- `secret-values.yaml` — зашифрованный файл с декларативной конфигурацией, аналогичный `values.yaml`. Его расшифрованное содержимое объединяется с values.yaml во время формирования манифестов;
- `Chart.yaml` — основная конфигурация и метаданные чарта;
- `Chart.lock` — lock-файл, защищающий от нежелательного изменения/обновления зависимых чартов;
- `README.md` — документация чарта;
- `LICENSE` — лицензия чарта;
- `.helmignore` — список файлов в директории чарта, которые не нужно включать в чарт при его публикации.

7.4.2.4. Подключение дополнительных чартов

7.4.2.4.1. Подключение зависимых локальных чартов

Подключить зависимые локальные чарты можно, положив их в директорию `charts` родительского чарта. В таком случае манифесты сформируются и для родительского, и для зависимых чартов, после чего объединятся вместе для дальнейшего развертывания. Дополнительная конфигурации не обязательна.

Пример содержимого основного чарта, имеющего локальные зависимые чарты:

```
.helm/
  charts/
    postgresql/
      templates/
        postgresql.yaml
      Chart.yaml
    redis/
      templates/
        redis.yaml
      Chart.yaml
  templates/
    backend.yaml
```

Обратите внимание, что у локальных зависимых чартов имя должно обязательно совпадать с именем их директории.

Если локальному зависимому чарту требуется дополнительная конфигурация, то в файле `Chart.yaml` родительского чарта укажите имя зависимого чарта без указания `dependencies[].repository` и добавьте интересующие директивы таким образом:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: redis
  condition: redis.enabled
```

При необходимости подключить локальный чарт не из директории `charts`, а из другого места, используйте директиву `dependencies[].repository` так:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: redis
  repository: file://../redis
```

7.4.2.4.2. Подключение зависимых чартов из репозитория

Подключить дополнительные чарты из OCI/HTTP-репозитория можно, указав их как зависимые в директиве `dependencies` файла `Chart.yaml` родительского чарта. В таком случае манифесты сформируются и для родительского, и для зависимых чартов, после чего объединятся вместе для дальнейшего развертывания.

Пример конфигурации чарта из репозитория, зависимого от основного чарта:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: database
  version: "~1.2.3"
  repository: https://example.com/charts
```

После каждого добавления/обновления удаленных зависимых чартов или изменения их конфигурации требуется:

1. (Если используется приватный OCI или HTTP-репозиторий с зависимым чартом) Добавить OCI или HTTP-репозиторий вручную с `d8 dk helm repo add`, указав нужные опции для доступа к репозиторию.
2. Вызвать `d8 dk helm dependency update`, который обновит `Chart.lock`.
3. Закоммитить обновленные `Chart.yaml` и `Chart.lock` в Git.

Также при использовании чартов из репозитория рекомендуется добавить `.helm/charts/**.tgz` в `.gitignore`.

7.4.2.4.3. Указание имени подключаемого чарта

В директиве `dependencies[].name` родительского чарта указывается оригинальное имя зависимого чарта, установленное его разработчиком, например:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: backend
```

Если нужно подключить несколько зависимых чартов с одинаковым именем или подключить один и тот же зависимый чарт несколько раз, то используйте директиву `dependencies[].alias` родительского чарта, чтобы поменять имена подключаемых чартов, например:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: backend
  alias: main-backend
- name: backend
  alias: secondary-backend
```

7.4.2.4.4. Указание версии подключаемого чарта

Ограничить подходящие версии зависимого чарта, из которых будет выбрана наиболее свежая, можно директивой `dependencies[].version` родительского чарта, например:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: backend
  version: "~1.2.3"
```

Результат: будет использована самая свежая версия 1.2.x, но как минимум 1.2.3.

7.4.2.4.5. Указание источника подключаемого чарта

Указать путь к источнику чартов, в котором можно найти указанный зависимый чарт, можно директивой `dependencies[].repository` родительского чарта, например:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: mychart
  repository: oci://example.org/myrepo
```

Результат: будет использован чарт `mychart` из OCI-репозитория `example.org/myrepo`.

7.4.2.4.6. Включение/отключение зависимых чартов

По умолчанию все зависимые чарты включены. Для произвольного включения/отключения зависимых чартов можно использовать директиву `dependencies[].condition` родительского чарта, например:

```
# .helm/Chart.yaml:
apiVersion: v2
dependencies:
- name: backend
  condition: backend.enabled
```

Результат: зависимый чарт backend будет включен, только если параметр $.Values.backend.enabled имеет значение true (по умолчанию — true).

Также можно использовать директиву dependencies[].tags родительского чарта для включения/отключения целых групп зависимых чартов сразу, например:

```
# .helm/Chart.yaml:
dependencies:
- name: backend
  tags: ["app"]
- name: frontend
  tags: ["app"]
```

Результат: зависимые чарты backend и frontend будут включены, только если параметр $.Values.tags.app имеет значение true (по умолчанию — true).

7.4.3. Шаблоны

7.4.3.1. Шаблонизация

Механизм шаблонизации в ПО «Deckhouse Platform» ничем не отличается от Helm. Используется движок шаблонов Go text/template, расширенный готовым набором функций Sprig и Helm.

7.4.3.2. Файлы шаблонов

В директории templates чарта находятся файлы шаблонов.

Файлы шаблонов templates/*.yaml формируют конечные Kubernetes-манифесты для развертывания. Каждый из этих файлов может формировать сразу несколько манифестов Kubernetes-ресурсов. Для этого манифесты должны быть разделены строкой ---.

Файлы шаблонов templates/_*.tpl содержат только именованные шаблоны для использования в других файлах. Файлы *.tpl не формируют Kubernetes-манифесты сами по себе.

7.4.3.3. Действия

Главный элемент шаблонизации — действие. Действие может возвращать только строки. Действие заключается в двойные фигурные скобки:

```
{{ print “hello”}}
```

Результат:

```
hello
```

7.4.3.4. Переменные

Переменные используются для хранения или указания на данные любого типа.

Объявление и присваивание переменной:

```
{{ $myvar := "hello" }}
```

Присваивание нового значения существующей переменной:

```
{{ $myvar = "helloworld" }}
```

Использование переменной:

```
{{ $myvar }}
```

Результат:

```
helloworld
```

Использование предопределенных переменных:

```
{{ $.Values.werf.env }}
```

Данные можно подставлять и без объявления переменных:

```
labels:
  app: {{ "myapp" }}
```

Результат:

```
labels:
  app: myapp
```

Также в переменные можно сохранять результат выполнения функций или конвейеров:

```
{{ $myvar := 1 | add 1 1 }}
{{ $myvar }}
```

Результат:

```
3
```

7.4.3.5. Области видимости переменных

Область видимости ограничивает видимость переменных. По умолчанию область видимости переменных ограничена файлом-шаблоном.

Область видимости может меняться при использовании некоторых блоков и функций. К примеру, блок `if` создает новую область видимости, а переменные, объявленные в блоке `if`, будут недоступны снаружи:

```
{{ if true }}
  {{ $myvar := "hello" }}
{{ end }}
{{ $myvar }}
```

Результат:

```
Error: ... undefined variable “$myvar”
```

Чтобы обойти это ограничение, объявите переменную за пределами блока, а значение присвойте ей внутри блока:

```
{{ $myvar := "" }}
{{ if true }}
```

```
  {{ $myvar = "hello" }}
{{ end }}
{{ $myvar }}
```

Результат:

```
hello
```

7.4.3.6. Типы данных

Доступные типы данных:

Тип данных	Пример
Логический	`{{ true }}`
Строка	`{{ "hello" }}`
Целое число	`{{ 1 }}`
Число с плавающей точкой	`{{ 1.1 }}`
Список с элементами любого типа, упорядоченный	`{{ list 1 2 3 }}`
Словарь с ключами-строками и значениями любого типа, неупорядоченный	`{{ dict "key1" 1 "key2" 2 }}`
Специальные объекты	`{{ $.Files }}`
Нуль	`{{ nil }}`

7.4.3.7. Функции

В ПО «Deckhouse Platform» встроена обширная библиотека функций для использования в шаблонах. Основная их часть — функции Helm.

Функции можно использовать только в действиях. Функции могут иметь аргументы и могут возвращать данные любого типа. Например, приведенная ниже функция сложения принимает три аргумента-числа и возвращает число:

```
{{ add 3 2 1 }}
```

Результат:

```
6
```

Обратите внимание, что результат выполнения действия всегда конвертируется в строку независимо от возвращаемого функцией типа данных.

Аргументами функций могут быть:

- простые значения: `1`;
- вызовы других функций: `add 1 1`;
- конвейеры: `1 | add 1`;
- комбинации вышеперечисленных типов: 1 | add (add 1 1).

Если аргумент — не простое значение, а вызов другой функции или конвейер, заключите его в круглые скобки `()`:

```
{{add 3 (add 1 1) (1 | add 1} }}
```

Чтобы игнорировать возвращаемый функцией результат, просто сохраните его в переменную `$_`:

```
{{ $_ := set $myDict "mykey" "myvalue"}}
```

7.4.3.8. Конвейеры

Конвейеры позволяют передать результат выполнения первой функции как последний аргумент во вторую функцию, а результат второй функции — как последний аргумент в третью и так далее:

```
{{ now | unixEpoch | quote }}
```

Здесь результат выполнения функции `now` (получить текущую дату) передается как аргумент в функцию `unixEpoch` (преобразует дату в Unix time), после чего полученное значение передается в функцию `quote` (оборачивает в кавычки).

Итоговый результат:

```
"1671466310"
```

Использование конвейеров не обязательно, и при желании их можно переписать следующим образом:

```
{{ quote (unixEpoch (now)) }}
```

… однако рекомендуется использовать именно конвейеры.

7.4.3.9. Логические операции и сравнения

Логические операции реализуются следующими функциями:

Операция	Функция	Пример
НЕ	`not <arg>`	`{{ not false }}`
И	`and <arg> <arg> [<arg>, ...]`	`{{ and true true }}`
ИЛИ	`or <arg> <arg> [<arg>, ...]`	`{{ or false true }}`

Сравнения реализуются следующими функциями:

Сравнение	Функция	Пример
Эквивалентно	`eq <arg> <arg> [<arg>, ...]`	`{{ eq "hello" "hello" }}`
Не эквивалентно	`neq <arg> <arg> [<arg>, ...]`	`{{ neq "hello" "world" }}`
Меньше	`lt <arg> <arg>`	`{{ lt 1 2 }}`
Больше	`gt <arg> <arg>`	`{{ gt 2 1 }}`
Меньше или эквивалентно	`le <arg> <arg>`	`{{ le 1 2 }}`
Больше или эквивалентно	`ge <arg> <arg>`	`{{ ge 2 1 }}`

Пример комбинирования:

```
{{ and (eq true true) (neq true false) (not (empty "hello")) }}
```

7.4.3.10. Ветвления

Ветвления `if/else` позволяют выполнять шаблонизацию только при выполнении/невыполнении определенных условий. Пример:

```
{{ if $.Values.app.enabled }}
# ...
{{ end }}
```

Условие считается невыполненным, если результатом его вычисления является:

- логическое `false`;
- число `0`;
- пустая строка `""`;
- пустой список `[]`;
- пустой словарь `{}`;
- нуль: `nil`.

В остальных случаях условие считается выполненным. Условием могут быть данные, переменная, функция или конвейер.

Полный пример:

```
{{ if eq $appName "backend" }}
app: mybackend
{{ else if eq $appName "frontend" }}
app: myfrontend
{{ else }}
app: {{ $appName }}
{{ end }}
```

Простые ветвления можно реализовывать не только с `if/else`, но и с функцией `ternary`. Например, следующее выражение с `ternary`:

```
{{ ternary "mybackend" $appName (eq $appName "backend") }}
… аналогично приведенной ниже конструкции if/else:
{{ if eq $appName "backend" }}
app: mybackend
{{ else }}
app: {{ $appName }}
{{ end }}
```

7.4.3.11. Циклы

7.4.3.11.1. Циклы по спискам

Циклы `range` позволяют перебирать элементы списка и выполнять нужную шаблонизацию на каждой итерации:

```
{{ range $urls }}
{{ . }}
{{ end }}
```

Результат:

```
https://example.org
https://sub.example.org
```

Относительный контекст `.` всегда указывает на элемент списка, соответствующий текущей итерации, хотя указатель можно сохранить и в произвольную переменную:

```
{{ range $elem := $urls }}
{{ $elem }}
{{ end }}
```

Результат будет таким же:

```
https://example.org
https://sub.example.org
Получить индекс элемента в списке можно следующим образом:
{{ range $i, $elem := $urls }}
{{ $elem }} имеет индекс {{ $i }}
{{ end }}
```

Результат:

```
https://example.org имеет индекс 0
https://sub.example.org имеет индекс 1
```

7.4.3.11.2. Циклы по словарям

Циклы `range` позволяют перебирать ключи и значения словарей и выполнять нужную шаблонизацию на каждой итерации:

```
# values.yaml:
apps:
  backend:
    image: openjdk
  frontend:
    image: node
```

```
# templates/app.yaml:
{{ range $.Values.apps }}
{{ .image }}
{{ end }}
```

Результат:

```
openjdk
node
```

Относительный контекст . всегда указывает на значение элемента словаря, соответствующего текущей итерации, при этом указатель можно сохранить и в произвольную переменную:

```
{{ range $app := $.Values.apps }}
{{ $app.image }}
{{ end }}
```

Результат будет таким же:

```
openjdk
node
```

Получить ключ элемента словаря можно так:

```
{{ range $appName, $app := $.Values.apps }}
{{ $appName }}: {{ $app.image }}
{{ end }}
```

Результат:

```
backend: openjdk
frontend: node
```

7.4.3.11.3. Контроль выполнения цикла

Специальное действие `continue` позволяет пропустить текущую итерацию цикла. В качестве примера пропустим итерацию для элемента `https://example.org`:

```
{{ range $url := $urls }}
{{ if eq $url "https://example.org" }}{{ continue }}{{ end }}
{{ $url }}
{{ end }}
```

Специальное действие `break` позволяет не только пропустить текущую итерацию, но и прервать весь цикл:

```
{{ range $url := $urls }}
{{ if eq $url "https://example.org" }}{{ break }}{{ end }}
{{ $url }}
{{ end }}
```

7.4.3.12. Контекст

7.4.3.12.1. Корневой контекст ($)

Корневой контекст — словарь, на который ссылается переменная `$`. Через него доступны values и некоторые специальные объекты. Корневой контекст имеет глобальную видимость в пределах файла-шаблона (исключение — блок `define` и некоторые функции).

Пример использования:

```
{{ $.Values.mykey }}
```

Результат:

```
myvalue
```

К корневому контексту можно добавлять произвольные ключи/значения, которые также станут доступны из любого места файла-шаблона:

```
{{ $_ := set $ "mykey" "myvalue"}}
{{ $.mykey }}
```

Результат:

```
myvalue
```

Корневой контекст остается неизменным даже в блоках, изменяющих относительный контекст (исключение — `define`):

```
{{ with $.Values.backend }}
- command: {{ .command }}
  image: {{ $.Values.werf.image.backend }}
{{ end }}
```

Некоторые функции вроде `tpl` или `include` могут терять корневой контекст. Для сохранения доступа к корневому контексту многим из них можно передать корневой контекст аргументом:

```
{{ tpl "{{ .Values.mykey }}" $ }}
```

Результат:

```
myvalue
```

7.4.3.12.2. Относительный контекст (.)

Относительный контекст — данные любого типа, на которые ссылается переменная `.`. По умолчанию относительный контекст указывает на корневой контекст.

Некоторые блоки и функции могут менять относительный контекст. В примере ниже в первой строке относительный контекст указывает на корневой контекст `$`, а во второй строке — уже на `$.Values.containers`:

```
{{ range .Values.containers }}
{{ . }}
{{ end }}
```

Для смены относительного контекста можно использовать блок `with`:

```
{{ with $.Values.app }}
```

```
image: {{ .image }}
{{ end }}
```

7.4.3.13. Переиспользование шаблонов

7.4.3.13.1. Именованные шаблоны

Для переиспользования шаблонизации объявите именованные шаблоны в блоках `define` в файлах `templates/_*.tpl`:

```
# templates/_helpers.tpl:
{{ define "labels" }}
app: myapp
team: alpha
{{ end }}
```

Далее подставляйте именованные шаблоны в файлы `templates/*.(yaml|tpl)` функцией `include`:

```
# templates/deployment.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  selector:
    matchLabels: {{ include "labels" nil | nindent 6 }}
  template:
    metadata:
      labels: {{ include "labels" nil | nindent 8 }}
```

Результат:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  selector:
    matchLabels:
      app: myapp
      team: alpha
  template:
    metadata:
      labels:
        app: myapp
        team: alpha
```

Имя именованного шаблона для функции `include` может быть динамическим:

```
{{ include (printf "%s.labels" $prefix) nil }}
```

Именованные шаблоны обладают глобальной видимостью — единожды объявленный в родительском или любом дочернем чарте именованный шаблон становится доступен сразу во всех чартах — и в родительском, и в дочерних. Убедитесь, что в подключенных родительском и дочерних чартах нет именованных шаблонов с одинаковыми именами.

7.4.3.13.2. Параметризация именованных шаблонов

Функция `include`, подставляющая именованные шаблоны, принимает один произвольный аргумент. Этот аргумент можно использовать для параметризации именованного шаблона, где этот аргумент станет относительным контекстом `.`:

```
{{ include "labels" "myapp" }}
{{ define "labels" }}
app: {{ . }}
{{ end }}
```

Результат:

```
app: myapp
```

Для передачи сразу нескольких аргументов используйте список с несколькими аргументами:

```
{{ include "labels" (list "myapp" "alpha") }}
{{ define "labels" }}
app: {{ index . 0 }}
team: {{ index . 1 }}
{{ end }}
… или словарь:
{{ include "labels" (dict "app" "myapp" "team" "alpha") }}
{{ define "labels" }}
app: {{ .app }}
team: {{ .team }}
{{ end }}
```

Необязательные позиционные аргументы можно реализовать так:

```
{{ include "labels" (list "myapp") }}
{{ include "labels" (list "myapp" "alpha") }}
{{ define "labels" }}
app: {{ index . 0 }}
{{ if gt (len .) 1 }}
team: {{ index . 1 }}
{{ end }}
{{ end }}
```

А необязательные непозиционные аргументы — так:

```
{{ include "labels" (dict "app" "myapp") }}
{{ include "labels" (dict "team" "alpha" "app" "myapp") }}
{{ define "labels" }}
app: {{ .app }}
{{ if hasKey . "team" }}
team: {{ .team }}
{{ end }}
{{ end }}
```

Именованному шаблону, не требующему параметризации, просто передайте `nil`:

```
{{ include "labels" nil }}
```

7.4.3.13.3. Результат выполнения include

Функция `include`, подставляющая именованный шаблон, **всегда возвращает только текст**. Для возврата структурированных данных нужно десериализовать результат выполнения include с помощью функции `fromYaml`:

```
{{ define "commonLabels" }}
app: myapp
{{ end }}
{{ $labels := include "commonLabels" nil | fromYaml }}
{{ $labels.app }}
```

Результат:

```
myapp
```

Обратите внимание, что `fromYaml` не работает для списков. Специально для них (и только для них) предназначена функция `fromYamlArray`.

Для явной сериализации данных можно воспользоваться функциями `toYaml` и `toJson`, для десериализации — функциями `fromYaml/fromYamlArray` и `fromJson/fromJsonArray`.

7.4.3.13.4. Контекст именованных шаблонов

Объявленные в `templates/_*.tpl` именованные шаблоны теряют доступ к корневому и относительному контекстам файла, в который они включаются функцией `include`. Исправить это можно, передав корневой и/или относительный контекст в виде аргументов `include`:

```
{{ include "labels" $ }}
{{ include "labels" . }}
{{ include "labels" (list $ .) }}
{{ include "labels" (list $ . "myapp") }}
```

7.4.3.13.5. include в include

В блоках `define` тоже можно использовать функцию `include` для включения именованных шаблонов:

```
{{ define "doSomething" }}
{{ include "doSomethingElse" . }}
{{ end }}
```

Через `include` можно вызвать даже тот именованный шаблон, из которого и происходит вызов, т. е. вызвать его рекурсивно:

```
{{ define "doRecursively" }}
{{ if ... }}
{{ include "doRecursively" . }}
{{ end }}
{{ end }}
```

7.4.3.14. Шаблонизация с tpl

Функция `tpl` позволяет выполнить шаблонизацию любой строки и тут же получить результат. Она принимает один аргумент, который должен быть корневым контекстом.

Пример шаблонизации `values`:

```
# values.yaml:
appName: "myapp"
deploymentName: "{{ .Values.appName }}-deployment"
```

```
# templates/app.yaml:
{{ tpl $.Values.deploymentName $ }}
```

Результат:

```
myapp-deployment
```

Пример шаблонизации произвольных файлов, которые сами по себе не поддерживают Helm-шаблонизацию:

```
{{ tpl ($.Files.Get "nginx.conf") $ }}
```

Для передачи дополнительных аргументов в функцию `tpl` можно добавить аргументы как новые ключи корневого контекста:

```
{{ $_ := set $ "myarg" "myvalue"}}
{{ tpl "{{ $.myarg }}" $ }}
```

7.4.3.15. Контроль отступов

Используйте функцию `nindent` для выставления отступов:

```
       containers: {{ .Values.app.containers | nindent 6 }}
```

Результат:

```
      containers:
      - name: backend
        image: openjdk
```

Пример комбинации с другими данными:

```
       containers:
       {{ .Values.app.containers | nindent 6 }}
       - name: frontend
         image: node
```

Результат:

```
      containers:
      - name: backend
        image: openjdk
      - name: frontend
        image: node
```

Используйте `-` после `{{` и/или до `}}` для удаления лишних пробелов до и/или после результата выполнения действия, например:

```
  {{- "hello" -}} {{ "world" }}
```

Результат:

```
helloworld
```

7.4.3.16. Комментарии

Поддерживаются два типа комментариев — комментарии шаблонизации `{{ /* */ }}` и комментарии манифестов #.

7.4.3.16.1. Комментарии шаблонизации

Комментарии шаблонизации скрываются при формировании манифестов:

```
{{ /* Этот комментарий пропадет */ }}
app: myApp
```

Комментарии могут быть многострочными:

```
{{ /*
Hello
World
/* }}
```

Шаблоны в них игнорируются:

```
{{ /*
{{ print "Эта шаблонизация игнорируется" }}
/* }}
```

7.4.3.16.2. Комментарии манифестов

Комментарии манифестов сохраняются при формировании манифестов:

```
# Этот комментарий сохранится
app: myApp
```

Комментарии могут быть только однострочнными:

```
# Для многострочных комментариев используйте
# несколько однострочных комментариев подряд
```

Шаблоны в них выполняются:

```
# {{ print "Эта шаблонизация выполняется" }}
```

7.4.3.17. Отладка

Используйте `d8 dk render`, чтобы полностью сформировать и отобразить конечные Kubernetes-манифесты. Укажите опцию `--debug`, чтобы увидеть манифесты, даже если они не являются корректным YAML.

Отобразить содержимое переменной:

```
output: {{ $appName | toYaml }}
```

Отобразить содержимое переменной-списка или словаря:

```
output: {{ $dictOrList | toYaml | nindent 2 }}
```

Отобразить тип данных у переменной:

```
output: {{ kindOf $myvar }}
```

Отобразить произвольную строку, остановив дальнейшее формирование шаблонов:

```
{{ fail (printf "Тип данных: %s" (kindOf $myvar)) }}
```

7.4.4. Параметризация шаблонов

7.4.4.1. Основы параметризации

Содержимое словаря `$.Values` можно использовать для параметризации шаблонов. Каждый чарт имеет свой словарь `$.Values`. Словарь формируется слиянием параметров, полученных из файлов параметров, опций командной строки и других источников.

Простой пример параметризации через `values.yaml`:

```
# values.yaml:
myparam: myvalue

# templates/example.yaml:
{{ $.Values.myparam }}
```

Результат:

```
myvalue
```

Более сложный пример:

```
# values.yaml:
myparams:
- value: original
```

```
# templates/example.yaml:
{{           (index           $.Values.myparams           0).value           }}

d8 dk render --set myparams[0].value=overriden
```

Результат:

```
overriden
```

7.4.4.2. Источники параметров и их приоритет

Словарь `$.Values` формируется объединением параметров из источников параметров в указанном порядке:

1. `values.yaml` текущего чарта.
2. `secret-values.yaml` текущего чарта.
3. Словарь в `values.yaml` родительского чарта, у которого ключ — алиас или имя текущего чарта.
4. Словарь в `secret-values.yaml` родительского чарта, у которого ключ — алиас или имя текущего чарта.
5. Файлы параметров из переменной `WERF_VALUES_*`.

6. Файлы параметров из опции `--values`.
7. Файлы секретных параметров из переменной `WERF_SECRET_VALUES_*`.
8. Файлы секретных параметров из опции `--secret-values`.
9. Параметры в set-файлах из переменной `WERF_SET_FILE_*`.
10. Параметры в set-файлах из опции `--set-file`.
11. Параметры из переменной `WERF_SET_STRING_*`.
12. Параметры из опции `--set-string`.
13. Параметры из переменной `WERF_SET_*`.
14. Параметры из опции `--set`.
15. Служебные параметры ПО «Deckhouse Platform».
16. Параметры из директивы `export-values` родительского чарта.
17. Параметры из директивы `import-values` дочерних чартов.

Правила объединения параметров:

- простые типы данных перезаписываются;
- списки перезаписываются;
- словари объединяются;
- при конфликтах параметры из источников выше по списку перезаписываются параметрами из источников ниже по списку.

7.4.4.3. Параметризация чарта

Чарт можно параметризовать через его файл параметров:

```
# values.yaml:
myparam: myvalue

# templates/example.yaml:
{{ $.Values.myparam }}
```

Результат:

```
myvalue
```

Также добавить/переопределить параметры чарта можно и аргументами командной строки:

```
d8           dk           render           --set           myparam=overriden
# или WERF_SET_MYPARAM=myparam=overriden d8 dk render
```

```
# .helm/values-production.yaml:
myparam: overriden
```

… или дополнительными файлами параметров:

```
# .helm/values-production.yaml:
myparam: overriden
```

```
d8        dk        render        --values        .helm/values-production.yaml
# или WERF_VALUES_PROD=.helm/values-production.yaml d8 dk render
```

… или файлом секретных параметров основного чарта:

```
# .helm/secret-values.yaml:
myparam: <encrypted>
```

```
d8 dk render
```

… или дополнительными файлами секретных параметров основного чарта:

```
# .helm/secret-values-production.yaml:
myparam: <encrypted>
```

```
d8    dk    render    --secret-values    .helm/secret-values-production.yaml
# или WERF_SECRET_VALUES_PROD=.helm/secret-values-production.yaml d8 dk render
```

… или set-файлами:

```
# myparam.txt:
overriden
d8          dk          render          --set-file          myparam=myparam.txt
# или WERF_SET_FILE_PROD=myparam=myparam.txt d8 dk render
```

Результат везде тот же:

```
overriden
```

7.4.4.4. Параметризация зависимых чартов

Зависимый чарт можно параметризовать как через его собственный файл параметров, так и через файл параметров родительского чарта.

К примеру, здесь параметры из словаря `mychild` в файле `values.yaml` чарта `myparent` перезаписывают параметры в файле `values.yaml` чарта `mychild`:

```
# Chart.yaml:
name: myparent
dependencies:
- name: mychild
```

```
# values.yaml:
mychild:
  myparam: overriden
```

```
# charts/mychild/values.yaml:
myparam: original
```

```
# charts/mychild/templates/example.yaml:
{{ $.Values.myparam }}
```

Результат:

```
overriden
```

Обратите внимание, что словарь, находящийся в `values.yaml` родительского чарта и содержащий параметры для зависимого чарта, должен иметь в качестве имени `alias` (если есть) или `name` зависимого чарта.

Также добавить/переопределить параметры зависимого чарта можно и аргументами командной строки:

```
# Chart.yaml:
name: myparent
dependencies:
- name: mychild
```

```
# values.yaml:
mychild:
  myparam: overriden
```

… или дополнительными файлами параметров:

```
# .helm/values-production.yaml:
mychild:
  myparam: overriden
```

```
d8 dk render --values .helm/values-production.yaml
# или WERF_VALUES_PROD=.helm/values-production.yaml d8 dk render
```

… или файлом секретных параметров основного чарта:

```
# .helm/secret-values.yaml:
mychild:
  myparam: <encrypted>
```

```
d8 dk render
```

… или дополнительными файлами секретных параметров основного чарта:

```
# .helm/secret-values-production.yaml:
mychild:
  myparam: <encrypted>
```

```
d8 dk render --secret-values .helm/secret-values-production.yaml
# или WERF_SECRET_VALUES_PROD=.helm/secret-values-production.yaml d8 dk render
```

… или set-файлами:

```
# mychild-myparam.txt:
overriden
```

… или директивой `export-values`:

```
# Chart.yaml:
name: myparent
dependencies:
- name: mychild
  export-values:
  - parent: myparam
    child: myparam
```

```
# values.yaml:
myparam: overriden
```

```
d8 dk render
```

Результат везде тот же:

```
overriden
```

7.4.4.5. Использование параметров зависимого чарта в родительском

Для передачи параметров зависимого чарта в родительский можно использовать директиву `import-values` в родительском чарте:

```
# Chart.yaml:
name: myparent
dependencies:
- name: mychild
  import-values:
  - child: myparam
    parent: myparam
```

```
# values.yaml:
myparam: original
```

```
# charts/mychild/values.yaml:
myparam: overriden
```

```
# templates/example.yaml:
{{ $.Values.myparam }}
```

Результат:

```
overriden
```

7.4.4.6. Глобальные параметры

Параметры чарта доступны только в этом же чарте (и ограниченно доступны в зависимых от него). Один из простых способов получить доступ к параметрам одного чарта в других подключенных чартах — использование глобальных параметров.

Глобальный параметр имеет глобальную область видимости — параметр, объявленный в родительском, дочернем или другом подключенном чарте становится доступен во всех подключенных чартах по одному и тому же пути:

```
# Chart.yaml:
name: myparent
dependencies:
- name: mychild1
- name: mychild2
```

```
# charts/mychild1/values.yaml:
global:
  myparam: myvalue
```

```
# templates/example.yaml:
myparent: {{ $.Values.global.myparam }}
# charts/mychild1/templates/example.yaml:
mychild1: {{ $.Values.global.myparam }}
# charts/mychild2/templates/example.yaml:
mychild2: {{ $.Values.global.myparam }}
```

Результат:

```
myparent: myvalue
---
mychild1: myvalue
---
mychild2: myvalue
```

7.4.4.7. Секретные параметры

Для хранения секретных параметров можно использовать файлы секретных параметров, хранящиеся в зашифрованном виде в Git-репозитории.

По умолчанию ПО «Deckhouse Platform» пытается найти файл .helm/secret-values.yaml, содержащий зашифрованные параметры, и при нахождении файла расшифровывает его и объединяет расшифрованные параметры с остальными:

```
# .helm/values.yaml:
plainParam: plainValue
```

```
# .helm/secret-values.yaml:
secretParam:
1000625c4f1d874f0ab853bf1db4e438ad6f054526e5dcf4fc8c10e551174904e6d0
```

```
{{ $.Values.plainParam }}
{{ $.Values.secretParam }}
```

Результат:

```
plainValue
secretValue
```

7.4.4.7.1. Работа с файлами секретных параметров

Порядок работы с файлами секретных параметров:

1. Возьмите существующий секретный ключ или создайте новый командой `d8 dk helm secret generate-secret-key`.
2. Сохраните секретный ключ в переменную окружения `WERF_SECRET_KEY`, либо в файлы `<корень Git-репозитория>/.werf_secret_key` или `<домашняя директория>/.werf/global_secret_key`.
3. Командой `d8 dk helm secret values edit .helm/secret-values.yaml` откройте файл секретных параметров и добавьте/измените в нем расшифрованные параметры.
4. Сохраните файл — файл зашифруется и сохранится в зашифрованном виде.
5. Закоммитите в Git добавленный/измененный файл `.helm/secret-values.yaml`;
6. При дальнейших вызовах ПО «Deckhouse Platform» секретный ключ должен быть установлен в вышеупомянутых переменной окружения или файлах, иначе файл секретных параметров не сможет быть расшифрован.

Имеющий доступ к секретному ключу может расшифровать содержимое файла секретных параметров, поэтому держите секретный ключ в безопасном месте!

При использовании файла `<корень Git-репозитория>/.werf_secret_key` обязательно добавьте его в `.gitignore`, чтобы случайно не сохранить его в Git-репозитории.

Многие команды ПО «Deckhouse Platform» можно запускать и без указания секретного ключа благодаря опции `--ignore-secret-key`, но в таком случае параметры будут доступны для использования не в расшифрованной форме, а в зашифрованной.

7.4.4.7.2. Дополнительные файлы секретных параметров

В дополнение к файлу `.helm/secret-values.yaml` можно создавать и использовать дополнительные секретные файлы:

```
# .helm/secret-values-production.yaml:
secret: 1000625c4f1d874f0ab853bf1db4e438ad6f054526e5dcf4fc8c10e551174904e6d0
```

```
d8 dk --secret-values .helm/secret-values-production.yaml
```

7.4.4.8. Информация о собранных образах

ПО «Deckhouse Platform» хранит информацию о собранных образах в параметрах `$.Values.werf` основного чарта:

```
werf:
  image:
   # Полный путь к собранному Docker-образу для ПО «Deckhouse Platform»-образа
"backend":
    backend:
example.org/apps/myapp:a243949601ddc3d4133c4d5269ba23ed58cb8b18bf2b64047f35ab
d2-1598024377816
  # Адрес container registry для собранных образов:
  repo: example.org/apps/myapp
  tag:
   # Тег собранного Docker-образа для ПО «Deckhouse Platform»-образа "backend":
          backend: a243949601ddc3d4133c4d5269ba23ed58cb8b18bf2b64047f35abd2-
1598024377816
```

Пример использования:

```
image: {{ $.Values.werf.image.backend }}
```

Результат:

```
image:
example.org/apps/myapp:a243949601ddc3d4133c4d5269ba23ed58cb8b18bf2b64047f35ab
d2-1598024377816
```

Для использования `$.Values.werf` в зависимых чартах воспользуйтесь директивой `export-values`:

```
# .helm/Chart.yaml:
dependencies:
- name: backend
  export-values:
  - parent: werf
    child: werf
```

```
# .helm/charts/backend/templates/example.yaml:
image: {{ $.Values.werf.image.backend }}
```

Результат:

```
image:
example.org/apps/myapp:a243949601ddc3d4133c4d5269ba23ed58cb8b18bf2b64047f35ab
d2-1598024377816
```

7.4.4.9. Информация о релизе

ПО «Deckhouse Platform» хранит информацию о релизе в свойствах объекта `$.Release`:

```
# Устанавливается ли релиз в первый раз:
IsInstall: true
# Обновляется ли уже существующий релиз:
IsUpgrade: false
# Имя релиза:
```

```
Name: myapp-production
# Имя Kubernetes Namespace:
Namespace: myapp-production
# Номер ревизии релиза:
Revision: 1
```

… и в параметрах `$.Values.werf` основного чарта:

```
werf:
  # Имя ПО «Deckhouse Platform»-проекта:
  name: myapp
  # Окружение:
  env: production
```

Пример использования:

```
{{ $.Release.Namespace }}
{{ $.Values.werf.env }}
```

Результат:

```
myapp-production
production
```

Для использования `$.Values.werf` в зависимых чартах воспользуйтесь директивой `export-values`:

```
# .helm/Chart.yaml:
dependencies:
- name: backend
  export-values:
  - parent: werf
    child: werf
```

```
# .helm/charts/backend/templates/example.yaml:
{{ $.Values.werf.env }}
```

Результат:

```
production
```

7.4.4.10. Информация о чарте

ПО «Deckhouse Platform» хранит информацию о текущем чарте в объекте `$.Chart`:

```
# Является ли чарт основным:
IsRoot: true

# Содержимое Chart.yaml:
Name: mychart
Version: 1.0.0
Type: library
KubeVersion: "~1.20.3"
AppVersion: "1.0"
Deprecated: false
Icon: https://example.org/mychart-icon.svg
Description: This is My Chart
```

```
Home: https://example.org
Sources:
  - https://github.com/my/chart
Keywords:
  - apps
Annotations:
  anyAdditionalInfo: here
Dependencies:
- Name: redis
  Condition: redis.enabled
```

Пример использования:

```
{{ $.Chart.Name }}
```

Результат:

```
mychart
```

7.4.4.11. Информация о шаблоне

ПО «Deckhouse Platform» хранит информацию о текущем шаблоне в свойствах объекта `$.Template`:

```
# Относительный путь к директории templates чарта:
BasePath: mychart/templates
# Относительный путь к текущему файлу шаблона:
Name: mychart/templates/example.yaml
```

Пример использования:

```
{{ $.Template.Name }}
```

Результат:

```
mychart/templates/example.yaml
```

7.4.4.12. Информация о Git-коммите

ПО «Deckhouse Platform» хранит информацию о Git-коммите, на котором он был запущен, в параметрах `$.Values.werf.commit` основного чарта:

```
werf:
  commit:
    date:
      # Дата Git-коммита, на котором был запущен ПО «Deckhouse Platform» (человекочитаемая форма):
      human: 2022-01-21 18:51:39 +0300 +0300
     # Дата Git-коммита, на котором был запущен ПО «Deckhouse Platform» (Unix time):
      unix: 1642780299
    # Хэш Git-коммита, на котором был запущен ПО «Deckhouse Platform»:
    hash: 1b28e6843a963c5bdb3579f6fc93317cc028051c
```

Пример использования:

```
{{ $.Values.werf.commit.hash }}
```

Результат:

```
1b28e6843a963c5bdb3579f6fc93317cc028051c
```

Для использования `$.Values.werf.commit` в зависимых чартах воспользуйтесь директивой `export-values`:

```
# .helm/Chart.yaml:
dependencies:
- name: backend
  export-values:
  - parent: werf
    child: werf
```

```
# .helm/charts/backend/templates/example.yaml:
{{ $.Values.werf.commit.hash }}
```

Результат:

```
1b28e6843a963c5bdb3579f6fc93317cc028051c
```

7.4.4.13. Информация о возможностях кластера Kubernetes

ПО «Deckhouse Platform» предоставляет информацию о возможностях кластера Kubernetes, в который ПО «Deckhouse Platform» стал бы применять Kubernetes-манифесты, через свойства объекта `$.Capabilities`:

```
KubeVersion:
  # Полная версия кластера Kubernetes:
  Version: v1.20.0
  # Мажорная версия кластера Kubernetes:
  Major: "1"
  # Минорная версия кластера Kubernetes:
  Minor: "20"
# API, поддерживаемые кластером Kubernetes:
APIVersions:
- apps/v1
- batch/v1
- # ...
```

… и методы объекта `$.Capabilities`:

- `APIVersions.Has <arg>` — поддерживается ли кластером Kubernetes указанное аргументом API (например, `apps/v1`) или ресурс (например, `apps/v1/Deployment`).

Пример использования:

```
{{ $.Capabilities.KubeVersion.Version }}
{{ $.Capabilities.APIVersions.Has "apps/v1" }}
```

Результат:

```
v1.20.0
true
```

7.4.5. Разные окружения

7.4.5.1. Параметризация шаблонов в зависимости от окружения

Окружение ПО «Deckhouse Platform» указывается опцией `--env ($WERF_ENV)`, либо автоматически выставляется командой `d8 dk ci-env`. Текущее окружение доступно в параметре `$.Values.werf.env` основного чарта.

Окружение ПО «Deckhouse Platform» используется при формировании имени релиза и имени Namespace’а, а также может использоваться для параметризации шаблонов:

```
# .helm/values.yaml:
memory:
  staging: 1G
  production: 2G
```

```
# .helm/templates/example.yaml:
memory: {{ index $.Values.memory $.Values.werf.env }}
d8 dk render --env production
```

Результат:

```
memory: 2G
```

Для использования `$.Values.werf.env` в зависимых чартах воспользуйтесь директивой `export-values`:

```
# .helm/Chart.yaml:
dependencies:
- name: child
  export-values:
  - parent: werf
    child: werf

# .helm/charts/child/templates/example.yaml:
{{ $.Values.werf.env }}
```

Результат:

```
production
```

7.4.5.2. Развертывание в разные Kubernetes Namespace

Имя Kubernetes Namespace для развертываемых ресурсов формируется автоматически по специальному шаблону `[[ project ]]-[[ env ]]`, где `[[ project ]]` — имя проекта ПО «Deckhouse Platform», а `[[ env ]]` — имя окружения.

Достаточно изменить окружение ПО «Deckhouse Platform» и вместе с ним изменится и Namespace:

```
# werf.yaml:
project: myapp
```

```
d8 dk converge --env staging
d8 dk converge --env production
```

Результат: один экземпляр приложения развернут в Namespace `myapp-staging`, а второй — в `myapp-production`.

Обратите внимание, что если в манифесте Kubernetes-ресурса явно указан Namespace, то для этого ресурса будет использован именно указанный в нем Namespace.

7.4.5.2.1. Изменение шаблона имени

Если вас не устраивает специальный шаблон, из которого формируется имя Namespace, вы можете его изменить:

```
# werf.yaml:
project: myapp
deploy:
  namespace: "backend-[[ env ]]"
```

```
d8 dk converge --env production
```

Результат: приложение развернуто в Namespace `backend-production`.

7.4.5.2.2. Прямое указание имени Namespace

Вместо формирования имени Namespace по специальному шаблону можно указывать Namespace явно для каждой команды (рекомендуется также изменять и имя релиза):

```
d8 dk converge --namespace backend-production --release backend-production
```

Результат: приложение развернуто в Namespace `backend-production`.

7.4.5.2.3. Форматирование имени Namespace

Namespace, сформированный по специальному шаблону или указанный опцией `--namespace`, приводится к формату `RFC 1123 Label Names` автоматически. Отключить автоматическое форматирование можно директивой `deploy.namespaceSlug` файла `werf.yaml`.

Вручную отформатировать любую строку согласно формату `RFC 1123 Label Names` можно командой `d8 dk slugify -f kubernetes-namespace`.

7.4.5.3. Развертывание в разные кластеры Kubernetes

По умолчанию ПО «Deckhouse Platform» развертывает Kubernetes-ресурсы в кластер, на который настроена команда `d8 k`. Для развертывания в разные кластеры можно использовать разные kube-контексты единого kube-config файла (по умолчанию — `$HOME/.kube/config`):

```
d8 dk converge --namespace backend-production --release backend-production
```

… или использовать разные kube-config файлы:

```
d8 dk converge --kube-config "$HOME/.kube/staging.config"
# или $WERF_KUBE_CONFIG=...
d8 dk converge --kube-config-base64 "$KUBE_PRODUCTION_CONFIG_IN_BASE64"  # или
$WERF_KUBE_CONFIG_BASE64=...
```

7.4.5.4. Развертывание из-под разных пользователей Kubernetes

По умолчанию ПО «Deckhouse Platform» для развертывания использует пользователя Kubernetes, через которого работает команда `d8 k`. Для развертывания из-под разных пользователей используйте разные kube-контексты:

```
d8 dk converge --kube-context admin
# или $WERF_KUBE_CONTEXT=...
d8 dk converge --kube-context regular-user
```

7.4.6. Порядок развертывания

7.4.6.1. Стадии развертывания

Развертывание Kubernetes-ресурсов происходит в следующей последовательности:

1. Развертывание CustomResourceDefinitions из директорий crds подключенных чартов.
2. Развертывание хуков pre-install, pre-upgrade или pre-rollback по одному хуку за раз, от хуков с меньшим весом к большему. Если хук имеет зависимость от внешнего ресурса, то он развернется только после его готовности.
3. Развертывание основных ресурсов: объединение ресурсов с одинаковым весом в группы (ресурсы без указанного веса имеют вес 0) и развертывание по одной группе за раз, от групп с ресурсами меньшего веса к группам с ресурсами большего веса. Если ресурс в группе имеет зависимость от внешнего ресурса, то она начнет развертывание только после его готовности.

4. Развертывание хуков post-install, post-upgrade или post-rollback по одному хуку за раз, от хуков с меньшим весом к большему. Если хук имеет зависимость от внешнего ресурса, то он развернется только после его готовности.

7.4.6.2. Развертывание CustomResourceDefinitions

Для развертывания CustomResourceDefinitions поместите CRD-манифесты в нешаблонизируемые файлы `crds/*.yaml` в любом из подключенных чартов. При следующем развертывании эти CRD будут развернуты первыми, а хуки и основные ресурсы будут развернуты только после них.

Пример:

```
# .helm/crds/crontab.yaml:
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
# ...
spec:
  names:
    kind: CronTab
```

```
# .helm/templates/crontab.yaml:
apiVersion: example.org/v1
kind: CronTab
# ...
d8 dk converge
```

Результат: сначала развернут CRD для CronTab-ресурса, а затем развернут сам CronTab-ресурс.

7.4.6.3. Изменение порядка развертывания ресурсов

По умолчанию ПО «Deckhouse Platform» объединяет все основные ресурсы (основные — не являющиеся хуками или CRDs из `crds/*.yaml`) в одну группу, создает ресурсы этой группы, а затем отслеживает их готовность.

Создание ресурсов группы происходит в следующем порядке:

- Namespace;
- NetworkPolicy;
- ResourceQuota;
- LimitRange;
- PodSecurityPolicy;
- PodDisruptionBudget;
- ServiceAccount;

- Secret;
- SecretList;
- ConfigMap;
- StorageClass;
- PersistentVolume;
- PersistentVolumeClaim;
- CustomResourceDefinition;
- ClusterRole;
- ClusterRoleList;
- ClusterRoleBinding;
- ClusterRoleBindingList;
- Role;
- RoleList;
- RoleBinding;
- RoleBindingList;
- Service;
- DaemonSet;
- Pod;
- ReplicationController;
- ReplicaSet;
- Deployment;
- HorizontalPodAutoscaler;
- StatefulSet;
- Job;
- CronJob;
- Ingress;
- APIService.

Отслеживание готовности включается для всех ресурсов группы одновременно сразу после создания всех ресурсов группы.

Для изменения порядка развертывания ресурсов можно создать новые группы ресурсов через задание ресурсам веса, отличного от веса по умолчанию 0. Все ресурсы с одинаковым весом объединяются в группы, а затем группы ресурсов развертываются по очереди, от группы с меньшим весом к большему, например:

```
# .helm/templates/example.yaml:
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: database
  annotations:
    werf.io/weight: "-1"
# ...
---
apiVersion: batch/v1
kind: Job
metadata:
  name: database-migrations
# ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app1
  annotations:
    werf.io/weight: "1"
# ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app2
  annotations:
    werf.io/weight: "1"
# ...
d8 dk converge
```

Результат: сначала был развернут ресурс `database`, затем — `database-migrations`, а затем параллельно развернулись `app1` и `app2`.

7.4.6.4. Запуск задач перед/после установки, обновления, отката или удаления релиза

Для развертывания определенных ресурсов только перед или после установки, обновления, отката или удаления релиза преобразуйте ресурс в хук аннотацией `helm.sh/hook`, например:

```
# .helm/templates/example.yaml:
apiVersion: batch/v1
kind: Job
metadata:
  name: database-initialization
  annotations:
    helm.sh/hook: pre-install
# ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
d8 dk converge
```

Результат: ресурс `database-initialization` будет развернут только при первой установке релиза, а ресурс `myapp` будет развертываться и при установке, и при обновлении, и при откате релиза.

Аннотация `helm.sh/hook` объявляет ресурс-хуком и указывает, при каких условиях этот ресурс должен развертываться (можно указать несколько условий через запятую). Возможные условия для развертывания хука:

- `pre-install` — при установке релиза до установки основных ресурсов;
- `pre-upgrade` — при обновлении релиза до обновления основных ресурсов;
- `pre-rollback` — при откате релиза до отката основных ресурсов;
- `pre-delete` — при удалении релиза до удаления основных ресурсов;
- `post-install` — при установке релиза после установки основных ресурсов;
- `post-upgrade` — при обновлении релиза после обновления основных ресурсов;
- `post-rollback` — при откате релиза после отката основных ресурсов;
- `post-delete` — при удалении релиза после удаления основных ресурсов.

Для задания хукам порядка развертывания присвойте им разные веса (по умолчанию — 0), чтобы хуки развертывались по очереди, от хука с меньшим весом к большему, например:

```
# .helm/templates/example.yaml:
apiVersion: batch/v1
kind: Job
metadata:
  name: first
  annotations:
    helm.sh/hook: pre-install
    helm.sh/hook-weight: "-1"
# ...
---
apiVersion: batch/v1
kind: Job
metadata:
  name: second
  annotations:
    helm.sh/hook: pre-install
# ...
---
apiVersion: batch/v1
kind: Job
metadata:
  name: third
  annotations:
    helm.sh/hook: pre-install
    helm.sh/hook-weight: "1"
# ...
d8 dk converge
```

Результат: сначала будет развернут хук `first`, затем хук `second`, затем хук `third`.

По умолчанию при повторных развертываниях того же самого хука старый хук в кластере удаляется прямо перед развертыванием нового хука. Этап удаления старого хука можно изменить аннотацией `helm.sh/hook-delete-policy`, которая принимает следующие значения:

- `hook-succeeded` — удалять новый хук сразу после его удачного развертывания, при неудачном развертывании не удалять совсем;
- `hook-failed` — удалять новый хук сразу после его неудачного развертывания, при удачном развертывании не удалять совсем;
- `before-hook-creation` — (по умолчанию) удалять старый хук сразу перед созданием нового.

7.4.6.5. Ожидание готовности ресурсов, не принадлежащих релизу

Развертываемым в текущем релизе ресурсам могут требоваться ресурсы, которые не принадлежат текущему релизу. ПО «Deckhouse Platform» может дожидаться готовности этих внешних ресурсов благодаря аннотации `<name>.external-dependency.werf.io/resource`, например:

```
# .helm/templates/example.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
   secret.external-dependency.werf.io/resource: secret/my-dynamic-vault-secret
# ...
d8 dk converge
```

Результат: Deployment `myapp` начнет развертывание только после того, как Secret `my-dynamic-vault-secret`, создаваемый автоматически оператором в кластере, будет создан и готов.

А так можно ожидать готовности сразу нескольких внешних ресурсов:

```
# .helm/templates/example.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
secret.external-dependency.werf.io/resource: secret/my-dynamic-vault-secret
    database.external-dependency.werf.io/resource: statefulset/my-database
# ...
```

По умолчанию ПО «Deckhouse Platform» ищет внешний ресурс в Namespace релиза (если, конечно, ресурс не кластерный). Namespace внешнего ресурса можно изменить аннотацией `<name>.external-dependency.werf.io/namespace`:

```
# .helm/templates/example.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
   secret.external-dependency.werf.io/resource: secret/my-dynamic-vault-secret
    secret.external-dependency.werf.io/namespace: my-namespace
```

Обратите внимание, что ожидать готовность внешнего ресурса будут и все другие ресурсы релиза с тем же весом, так как ресурсы объединяются по весу в группы и развертываются именно группами.

7.4.7. Сценарии развертывания

7.4.7.1. Обычное развертывание

Обычно развертывание осуществляется командой `d8 dk converge`, которая собирает образы и развертывает приложение, но требует запуска из Git-репозитория приложения. Пример:

```
d8 dk converge --repo example.org/mycompany/myapp
```

Если требуется разделить шаги сборки и развертывания, то это можно сделать так:

```
d8 dk build --repo example.org/mycompany/myapp
```

```
d8 dk converge --require-built-images --repo example.org/mycompany/myapp
```

7.4.7.2. Развертывание с использованием произвольных тегов образов

По умолчанию собранные образы получают тег на основе их содержимого, который становится доступен в Values для их дальнейшего использования в шаблонах при развертывании. Но если возникает необходимость тегировать образы иным тегом, то можно использовать параметр `--use-custom-tag`, например:

```
d8 dk converge --use-custom-tag '%image%-v1.0.0' --repo
example.org/mycompany/myapp
```

Результат: образы были собраны и опубликованы с тегами `<имя image>-v1.0.0`, после чего теги этих образов стали доступны в Values, на основе которых были сформированы и применены конечные манифесты Kubernetes.

В имени тега, указываемом в параметре `--use-custom-tag`, можно использовать шаблоны `%image%`, `%image_slug%` и `%image_safe_slug%` для подставления имени образа и `%image_content_based_tag%` для подставления оригинального тега на основе содержимого.

Обратите внимание, что при указании произвольного тега публикуется также и образ с тегом на основе содержимого. В дальнейшем при вызове `d8 dk cleanup` образ с тегом на основе содержимого и образы с произвольными тегами удаляются вместе.

Если требуется разделить шаги сборки и развертывания, то это можно сделать так:

```
d8 dk build --add-custom-tag '%image%-v1.0.0' --repo example.org/mycompany/myapp
```

```
d8 dk converge --require-built-images --use-custom-tag '%image%-v1.0.0' --repo example.org/mycompany/myapp
```

7.4.7.3. Развертывание без доступа к Git-репозиторию приложения

Если нужно развернуть приложение без доступа к Git-репозиторию приложения, то необходимо выполнить три шага:

1. Сборка образов и их публикация в container registry.
2. Добавление переданных параметров и публикация основного чарта в OCI-репозиторий. Чарт содержит указатели на опубликованные в первом шаге образы.
3. Применение опубликованного бандла в кластер.

Первые два шага выполняются командой `d8 dk bundle publish`, находясь в Git-репозитории приложения, например:

```
d8 dk bundle publish --tag latest --repo example.org/mycompany/myapp
```

А третий шаг выполняется командой `d8 dk bundle apply` уже без необходимости находиться в Git-репозитории приложения, например:

```
d8 dk bundle apply --tag latest --release myapp --namespace myapp-production --repo example.org/mycompany/myapp
```

Конечный результат будет тот же самый, что и при использовании `d8 dk converge`.

Если требуется разделить первый и второй шаг, то это можно сделать так:

```
d8 dk build --repo example.org/mycompany/myapp
```

```
d8 dk bundle publish --require-built-images --tag latest --repo example.org/mycompany/myapp
```

7.4.7.4. Развертывание без доступа к Git-репозиторию и container registry приложения

Если нужно развернуть приложение без доступа к Git-репозиторию приложения и без доступа к container registry приложения, то необходимо выполнить пять шагов:

1. Сборка образов и их публикация в container registry приложения.
2. Добавление переданных параметров и публикация основного чарта в OCI-репозиторий. Чарт содержит указатели на опубликованные в первом шаге образы.
3. Экспорт бандла и связанных с ним образов в локальный архив.
4. Импорт заархивированного бандла и его образов в container registry, доступный из Kubernetes-кластера, используемого для развертывания.
5. Применение в кластер бандла, опубликованного в новом container registry.

Первые два шага выполняются командой `d8 dk bundle publish`, находясь в Git-репозитории приложения, например:

```
d8 dk bundle publish --tag latest --repo example.org/mycompany/myapp
```

Третий шаг выполняется командой `d8 dk bundle copy` уже без необходимости находиться в Git-репозитории приложения, например:

```
d8 dk bundle copy --from example.org/mycompany/myapp:latest --to archive:myapp-latest.tar.gz
```

Теперь полученный локальный архив `myapp-latest.tar.gz` переносится удобным способом туда, откуда имеется доступ в container registry, используемый для развертывания в Kubernetes-кластер, и снова выполняется команда `d8 dk bundle copy`, например:

```
d8 dk bundle copy --from archive:myapp-latest.tar.gz --to registry.internal/mycompany/myapp:latest
```

В результате чарт и связанные с ним образы опубликуются в новый container registry, к которому из Kubernetes-кластера уже есть доступ. Осталось только развернуть опубликованный бандл в кластер командой `d8 dk bundle apply`, например:

```
d8 dk bundle apply --tag latest --release myapp --namespace myapp-production --repo registry.internal/mycompany/myapp
```

На этом шаге уже не требуется доступ ни в Git-репозиторий приложения, ни в его первоначальный container registry. Конечный результат развертывания бандла будет тот же самый, что и при использовании `d8 dk converge`.

Если требуется разделить первый и второй шаг, то это можно сделать так:

```
d8 dk build --repo example.org/mycompany/myapp
d8 dk bundle publish --require-built-images --tag latest --repo example.org/mycompany/myapp
```

7.4.7.5. Развертывание сторонним инструментом

Если нужно выполнить применение конечных манифестов приложения не с ПО «Deckhouse Platform», а с использованием другого инструмента (d8 k, kubectl, Helm, …), то необходимо выполнить три шага:

1. Сборка образов и их публикация в container registry.
2. Формирование конечных манифестов.
3. Развертывание получившихся манифестов в кластер, используя сторонний инструмент.

Первые два шага выполняются командой `d8 dk render`, находясь в Git-репозитории приложения:

```
d8 dk render --output manifests.yaml --repo example.org/mycompany/myapp
```

Теперь полученные манифесты можно передать в сторонний инструмент для дальнейшего развертывания, например:

```
d8 k apply -f manifests.yaml
```

Обратите внимание, что некоторые специальные возможности ПО «Deckhouse Platform» вроде возможности изменения порядка развертывания ресурсов на основании их веса (аннотация `werf.io/weight`) скорее всего не будут поддерживаться при применении манифестов сторонним инструментом.

Если требуется разделить первый и второй шаг, то это можно сделать так:

```
d8 dk build --repo example.org/mycompany/myapp
```

```
d8 dk render --require-built-images --output manifests.yaml --repo example.org/mycompany/myapp
```

7.4.7.6. Развертывание сторонним инструментом без доступа к Git-репозиторию приложения

Если нужно выполнить применение конечных манифестов приложения не с ПО «Deckhouse Platform», а с использованием другого инструмента (d8, Helm, …), при этом не имея доступа к Git-репозиторию приложения, то необходимо выполнить четыре шага:

1) Сборка образов и их публикация в container registry.

Добавление переданных параметров и публикация основного чарта в OCI-репозиторий. Чарт содержит указатели на опубликованные в первом шаге образы.

2) Формирование из бандла конечных манифестов.

3) Развертывание получившихся манифестов в кластер используя сторонний инструмент.

Первые два шага выполняются командой `d8 dk bundle publish`, находясь в Git-репозитории приложения:

```
d8 dk bundle publish --tag latest --repo example.org/mycompany/myapp
```

А третий шаг выполняется командой `d8 dk bundle render` уже без необходимости находиться в Git-репозитории приложения, например

```
d8 dk bundle render --output manifests.yaml --tag latest --release myapp --
namespace myapp-production --repo example.org/mycompany/myapp
```

4) Теперь полученные манифесты можно передать в сторонний инструмент для дальнейшего развертывания, например:

```
d8 k apply -f manifests.yaml
```

Обратите внимание, что некоторые специальные возможности ПО «Deckhouse Platform», вроде возможности изменения порядка развертывания ресурсов на основании их веса (аннотация `werf.io/weight`), скорее всего не будут поддерживаться при применении манифестов сторонним инструментом.

Если требуется разделить первый и второй шаг, то это можно сделать так:

```
d8 dk build --repo example.org/mycompany/myapp
```

`d8 dk bundle publish --require-built-images --tag latest --repo example.org/mycompany/myapp`

7.4.7.7. Сохранение отчета о развертывании

Команды `d8 dk converge` и `d8 dk bundle apply` имеют параметр `--save-deploy-report`, который позволяет сохранить отчет о последнем развертывании в файл. Отчет содержит имя релиза, Namespace, статус развертывания и ряд других данных. Пример:

```
d8 dk converge --save-deploy-report
```

Результат: после развертывания появится файл `.werf-deploy-report.json`, содержащий информацию о последнем релизе.

Путь к отчету о развертывании можно изменить параметром `--deploy-report-path`.

7.4.7.8. Удаление развернутого приложения

Удалить развернутое приложение можно командой `d8 dk dismiss`, запущенной из Git-репозитория приложения, например:

```
d8 dk dismiss --env staging
```

При отсутствии доступа к Git-репозиторию приложения можно явно указать имя релиза и Namespace:

```
d8 dk dismiss --release myapp-staging --namespace myapp-staging
```

… или использовать отчет о предыдущем развертывании, включаемый опцией `--save-deploy-report` у `d8 dk converge` и `d8 dk bundle apply`, который содержит имя релиза и Namespace:

```
d8 dk converge --save-deploy-report
cp .werf-deploy-report.json /anywhere
cd /anywhere
d8 dk dismiss --use-deploy-report
```

Путь к отчету о развертывании можно изменить параметром `--deploy-report-path`.

7.4.8. Отслеживание ресурсов

7.4.8.1. Отслеживание состояния ресурсов

Развертывание ресурсов делится на две стадии: применение ресурсов в кластер и отслеживание состояния этих ресурсов. ПО «Deckhouse Platform» реализует продвинутое отслеживание состояния ресурсов благодаря библиотеке `kubedog`.

Отслеживание ресурсов включено по умолчанию для всех поддерживаемых ресурсов, а именно для:

- всех ресурсов релиза;
- некоторых ресурсов, опосредованно создаваемых ресурсами релиза;
- ресурсов вне релиза, указанных в аннотациях `<name>.external-dependency.werf.io/resource`.

Для ресурсов Deployment, StatefulSet, DaemonSet, Job и Flagger Canary задействуются специальные отслеживатели состояния, которые не только точно определяют, удачно или неудачно ресурс был развернут, но и отслеживают состояние дочерних ресурсов, таких как Pod’ы, создаваемые Deployment’ом.

Для остальных ресурсов, не имеющих специальных отслеживателей состояния, задействуется универсальный отслеживатель, который предполагает удачность развертывания ресурса на основании доступной в кластере информации о ресурсе. В редких случаях, если универсальный отслеживатель ошибается в своих предположениях, отслеживание для этого ресурса можно отключить.

7.4.8.2. Изменение критериев неудачного развертывания ресурса

По умолчанию ПО «Deckhouse Platform» прерывает развертывание и помечает его как неудачное, если произошло более двух ошибок при развертывании одного из ресурсов.

Изменить максимальное количество ошибок развертывания для ресурса можно аннотацией `werf.io/failures-allowed-per-replica`, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/failures-allowed-per-replica: "5"
```

Если ресурс имеет аннотацию `werf.io/fail-mode: HopeUntilEndOfDeployProcess`, то ошибки его развертывания будут учитываться только после того, как все остальные ресурсы удачно развернутся.

А помеченный аннотацией `werf.io/track-termination-mode: NonBlocking` ресурс будет отслеживаться только пока все остальные ресурсы не будут развернуты, после чего этот ресурс автоматически посчитается развернутым, даже если это не так.

Универсальный отслеживатель отсутствие активности у ресурса в течение 4 минут считает ошибкой развертывания. Изменить этот период можно аннотацией `werf.io/no-activity-timeout`, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/no-activity-timeout: 10m
```

7.4.8.3. Отключение отслеживания состояния и игнорирование ошибок ресурса

Для отключения отслеживания состояния ресурса и игнорирования ошибок его развертывания пометьте ресурс аннотациями `werf.io/fail-mode:`

IgnoreAndContinueDeployProcess и werf.io/track-termination-mode: NonBlocking, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/fail-mode: IgnoreAndContinueDeployProcess
    werf.io/track-termination-mode: NonBlocking
```

7.4.8.4. Отображение логов контейнеров

Благодаря библиотеке kubedog отображаются логи контейнеров, создаваемых при развертывании Deployment, StatefulSet, DaemonSet и Job.

Выключить отображение логов для ресурса можно аннотацией werf.io/skip-logs: "true", например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/skip-logs: "true"
```

А в аннотации werf.io/show-logs-only-for-containers можно явно перечислить контейнеры, логи которых следует отображать, в то же время скрыв логи всех остальных контейнеров, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/show-logs-only-for-containers: "backend,frontend"
```

… или наоборот — в аннотации werf.io/skip-logs-for-containers перечислить контейнеры, логи которых не следует отображать, в то же время отображая логи всех остальных контейнеров, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/skip-logs-for-containers: "sidecar"
```

Для отображения только тех строк лога, которые соответствуют регулярному выражению, используйте аннотацию `werf.io/log-regex`, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/log-regex: ".*ERROR.*"
```

Возможно отфильтровать строки лога согласно регулярному выражению не для всех контейнеров сразу, а только для определенного контейнера, если использовать аннотацию `werf.io/log-regex-for-<имя контейнера>`, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/log-regex-for-backend: ".*ERROR.*"
```

7.4.8.5. Отображение Events ресурсов

Благодаря библиотеке `kubedog` отображаются Events отслеживаемых ресурсов, если ресурс имеет аннотацию `werf.io/show-service-messages: "true"`, например:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  annotations:
    werf.io/show-service-messages: "true"
```

7.4.9. Управление релизами

7.4.9.1. О релизах

Результатом развертывания является релиз — совокупность развернутых в кластере ресурсов и служебной информации.

Технически релизы ПО «Deckhouse Platform» являются релизами Helm 3 и полностью с ними совместимы. Служебная информация по умолчанию хранится в специальном Secret-ресурсе.

7.4.9.2. Автоматическое формирование имени релиза

По умолчанию имя релиза формируется автоматически по специальному шаблону `[[ project ]]-[[ env ]]`, где `project` — имя проекта ПО «Deckhouse Platform», а `env` — имя окружения, например:

```
# werf.yaml:
project: myapp
```

```
d8 dk converge --env staging
d8 dk converge --env production
```

Результат: созданы релизы `myapp-staging` и `myapp-production`.

7.4.9.3. Изменение шаблона имени релиза

Если вас не устраивает специальный шаблон, из которого формируется имя релиза, вы можете его изменить:

```
# werf.yaml:
project: myapp
deploy:
  helmRelease: "backend-[[ env ]]"
```

```
d8 dk converge --env production
```

Результат: создан релиз `backend-production`.

7.4.9.4. Прямое указание имени релиза

Вместо формирования имени релиза по специальному шаблону можно указывать имя релиза явно для каждой команды:

```
d8 dk converge --release backend-production  # или $WERF_RELEASE=...
```

Результат: создан релиз `backend-production`.

7.4.9.5. Форматирование имени релиза

Имя релиза, сформированное по специальному шаблону или указанное опцией `--release`, приводится к формату `RFC 1123 Label Names` автоматически. Отключить автоматическое форматирование можно директивой `deploy.helmReleaseSlug` файла `werf.yaml`.

Вручную отформатировать любую строку согласно формату `RFC 1123 Label Names` можно командой d8 dk slugify -f helm-release.

7.4.9.6. Добавление в релиз уже существующих в кластере ресурсов

ПО «Deckhouse Platform» не позволяет развернуть новый ресурс релиза поверх уже существующего в кластере ресурса, если ресурс в кластере не является частью текущего релиза. Такое поведение предотвращает случайные обновления ресурсов, принадлежащих другому релизу или развернутых без ПО «Deckhouse Platform». Если все же попытаться это сделать, то отобразится следующая ошибка:

```
Error: helm upgrade have failed: UPGRADE FAILED: rendered manifests contain a
resource that already exists...
```

Чтобы добавить ресурс в кластере в текущий релиз и разрешить его обновление, выставьте ресурсу в кластере аннотации `meta.helm.sh/release-name: <имя текущего релиза>`, meta.helm.sh/release-namespace: `<Namespace текущего релиза>` и лейбл `app.kubernetes.io/managed-by: Helm`, например:

```
d8 k annotate deploy/myapp meta.helm.sh/release-name=myapp-production
d8 k annotate deploy/myapp meta.helm.sh/release-namespace=myapp-production
d8 k label deploy/myapp app.kubernetes.io/managed-by=Helm
```

… после чего перезапустите развертывание:

```
d8 dk converge
```

Результат: ресурс релиза myapp успешно обновил ресурс `myapp` в кластере и теперь ресурс в кластере является частью текущего релиза.

7.4.9.7. Автоматическое аннотирование выкатываемых ресурсов релиза

ПО «Deckhouse Platform» автоматически выставляет следующие аннотации всем ресурсам чарта в процессе развертывания:

- `"werf.io/version": FULL_WERF_VERSION` — версия ПО «Deckhouse Platform», использованная в процессе запуска команды `d8 dk converge`;
- `"project.werf.io/name": PROJECT_NAME` — имя проекта, указанное в файле конфигурации `werf.yaml`;
- `"project.werf.io/env": ENV` — имя окружения, указанное с помощью параметра `--env` или переменной окружения `WERF_ENV` (аннотация не устанавливается, если окружение не было указано при запуске).

При использовании команды `d8 dk ci-env` с поддерживаемыми CI/CD системами добавляются аннотации, которые позволяют пользователю перейти в связанный пайплайн, задание и коммит при необходимости.

7.4.9.8. Добавление произвольных аннотаций и лейблов для выкатываемых ресурсов релиза

Пользователь может устанавливать произвольные аннотации и лейблы используя CLI-параметры при развертывании `--add-annotation annoName=annoValue` (может быть указан несколько раз) и `--add-label labelName=labelValue` (может быть указан несколько раз). Аннотации и лейблы так же могут быть заданы с помощью соответствующих переменных `WERF_ADD_LABEL*` и `WERF_ADD_ANNOTATION*` (к примеру, `WERF_ADD_ANNOTATION_1=annoName1=annoValue1` и `WERF_ADD_LABEL_1=labelName1=labelValue1`).

Например, для установки аннотаций и лейблов `commit-sha=9aeee03d607c1eed133166159fbea3bad5365c57`, `gitlab-user-email=vasya@myproject.com` всем ресурсам Kubernetes в чарте, можно использовать следующий вызов команды деплоя:

```
d8 dk converge \
  --add-annotation "commit-sha=9aeee03d607c1eed133166159fbea3bad5365c57" \
  --add-label "commit-sha=9aeee03d607c1eed133166159fbea3bad5365c57" \
  --add-annotation "gitlab-user-email=vasya@myproject.com" \
  --add-label "gitlab-user-email=vasya@myproject.com" \
  --env dev \
  --repo REPO
```

7.5. Дистрибуция

7.5.1. Основы

Обычный цикл доставки приложений с ПО «Deckhouse Platform» выглядит как сборка образов, их публикация и последующее развертывание чартов, для чего бывает достаточно одного вызова команды `d8 dk converge`. Но иногда возникает необходимость разделить дистрибуцию артефактов (образы, чарты) и их развертывание, либо даже реализовать развертывание артефактов вовсе без ПО «Deckhouse Platform», а с использованием стороннего ПО.

В этом разделе рассматриваются способы дистрибуции образов и бандлов (чартов и связанных с ними образов) для их дальнейшего развертывания с ПО «Deckhouse Platform» или без него. Инструкции развертывания опубликованных артефактов можно найти в разделе «Развертывание».

7.5.1.1. Пример дистрибуции образа

Для дистрибуции единственного образа, собираемого через Dockerfile, который потом будет развернут сторонним ПО, достаточно двух файлов и одной команды `d8 dk export`, запущенной в Git-репозитории приложения:

```
# werf.yaml:
project: myproject
configVersion: 1
---
image: myapp
dockerfile: Dockerfile
```

```
# Dockerfile:
FROM node

WORKDIR /app
COPY . .
RUN npm ci

CMD ["node", "server.js"]
```

```
d8 dk export myapp --repo example.org/myproject --tag other.example.org/myproject/myapp:latest
```

Результат: опубликован образ приложения `other.example.org/myproject/myapp:latest`, готовый для развертывания сторонним ПО.

7.5.1.2. Пример дистрибуции бандла

Для дистрибуции бандла для дальнейшего его развертывания с ПО «Deckhouse Platform», подключения его как зависимого чарта или развертывания бандла как чарта сторонним ПО в простейшем случае достаточно трех файлов и одной команды `d8 dk bundle publish`, запущенной в Git-репозитории приложения:

```
# werf.yaml:
project: mybundle
configVersion: 1
---
image: myapp
dockerfile: Dockerfile
```

```
# Dockerfile:
FROM node

WORKDIR /app
COPY . .
RUN npm ci

CMD ["node", "server.js"]
```

```
# .helm/templates/myapp.yaml:
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
```

```
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: {{ $.Values.werf.image.myapp }}
```

```
d8 dk bundle publish --repo example.org/bundles/mybundle
```

Результат: опубликован чарт `example.org/bundles/mybundle:latest` и связанный с ним собранный образ.

7.5.2. Образы

7.5.2.1. О дистрибуции образов

Дистрибуция собираемых ПО «Deckhouse Platform» образов для использования сторонними пользователями и/или ПО осуществляется командой `d8 dk export`. Эта команда соберет и опубликует образы в container registry, при этом убрав все ненужные для стороннего ПО метаданные, чем полностью выведет образы из-под контроля ПО «Deckhouse Platform», позволив организовать их дальнейший жизненный цикл сторонними средствами.

Опубликованные командой `d8 dk export` образы никогда не будут удаляться командой `d8 dk cleanup`, в отличие от образов, опубликованных обычным способом. Очистка экспортированных образов должна быть реализована сторонними средствами.

7.5.2.2. Дистрибуция образа

```
d8 dk bundle publish --repo example.org/bundles/mybundle
```

Результат: образ собран и сначала опубликован с content-based тегом в container registry `example.org/myproject`, а затем опубликован в другой container registry `other.example.org/myproject` как целевой экспортированный образ `other.example.org/myproject/myapp:latest`.

В параметре `--tag` можно указать тот же репозиторий, что и в `--repo`, таким образом используя один и тот же container registry и для сборки, и для экспортированного образа.

7.5.2.3. Дистрибуция нескольких образов

В параметре `--tag` можно использовать шаблоны `%image%`, `%image_slug%` и `%image_safe_slug%` для подставления имени образа из `werf.yaml`, основанном на его содержимом, например:

```
d8 dk export \
    --repo example.org/mycompany/myproject \
    --tag example.org/mycompany/myproject/%image%:latest
```

7.5.2.4. Дистрибуция произвольных образов

Используя позиционные аргументы и имена образов из `werf.yaml` можно выбрать произвольные образы, например:

```
d8 dk export backend frontend \
    --repo example.org/mycompany/myproject \
    --tag example.org/mycompany/myproject/%image%:latest
```

7.5.2.5. Использование content-based-тега при формировании тега

В параметре `--tag` можно использовать шаблон `%image_content_based_tag%` для использования тега образа, основанном на его содержимом, например:

```
d8 dk export \
    --repo example.org/mycompany/myproject \
  --tag example.org/mycompany/myproject/myapp:%image_content_based_tag%
```

7.5.2.6. Использование content-based-тега при формировании тега

Используя параметр `--add-label` можно добавить произвольное количество дополнительных лейблов к экспортируемому образу(ам), например:

```
d8 dk export \
    --repo example.org/mycompany/myproject \
    --tag registry.werf.io/werf/werf:latest \
    --add-label                                    io.artifacthub.package.readme-
url=https://raw.githubusercontent.com/werf/werf/main/README.md \
    --add-label org.opencontainers.image.created=2023-03-13T11:55:24Z \
   --add-label org.opencontainers.image.description="Official image to run ПО
«Deckhouse Platform» in containers"
```

7.5.3. Бандлы и чарты

7.5.3.1. О бандлах и чартах

Бандл — это способ дистрибуции чарта и связанных с ним образов как единого целого.

Командой `d8 dk bundle publish` можно опубликовать чарт и связанные образы для дальнейшего развертывания с ПО «Deckhouse Platform». При этом для развертывания доступ к Git-репозиторию приложения больше не потребуется.

Эта же команда подходит для публикации чарта. Опубликованный в OCI-репозиторий чарт может использоваться в качестве основного или зависимого чарта с ПО «Deckhouse Platform», Helm, Argo CD, Flux и другими решениями.

При упаковке ПО «Deckhouse Platform» автоматически добавляет следующие данные в чарт:

- имена собираемых образов и их динамических тегов в Values чарта;
- значения, переданных через параметры командной строки или переменные окружения, в Values чарта;
- глобальные пользовательские и служебные аннотации и лейблы для добавления в ресурсы чарта при развертывании командой `d8 dk bundle apply`.

Опубликованный бандл (чарт и связанные с ним образы) можно копировать в другой репозиторий container registry или выгружать в/из архива с помощью одной команды `d8 dk bundle copy`.

7.5.3.2. Аутентификация в container registry

Перед работой с образами необходимо аутентифицироваться в container registry. Сделать это можно командой `d8 dk cr login`:

```
d8 dk cr login <registry url>
```

Например:

```
# Login with username and password from command line
d8 dk cr login -u username -p password registry.example.com

# Login with token from command line
d8 dk cr login -p token registry.example.com

# Login into insecure registry (over http)
d8 dk cr login --insecure-registry registry.example.com
```

В случае использования команды `d8 dk ci-env` с поддерживаемыми CI/CD-системами аутентификация во встроенные container registry выполняется в рамках команды, поэтому использование команды `d8 dk cr login` в этом случае не требуется.

7.5.3.3. Публикация бандла

Опубликовать бандл в OCI-репозиторий можно следующим способом:

1. Создайте `werf.yaml`, если его еще нет:

```
# werf.yaml:
project: mybundle
configVersion: 1
```

2. Разместите файлы в директории основного чарта (по умолчанию `.helm` в корне Git-репозитория). Заметьте, что при публикации в чарт будут включены только следующие файлы и директории

```
.helm/
  charts/
  templates/
  crds/
  files/
  Chart.yaml
  values.yaml
  values.schema.json
  LICENSE
  README.md
```

   Для публикации дополнительных файлов/директорий выставьте переменную окружения `WERF_BUNDLE_SCHEMA_NONSTRICT=1`, после чего будут публиковаться все файлы и директории в директории основного чарта, а не только вышеуказанные.

3. Следующей командой опубликуйте бандл. Соберите и опубликуйте описанные в `werf.yaml` образы (если таковые есть), затем опубликуйте содержимое `.helm` в виде OCI-образа `example.org/bundles/mybundle:latest`:

```
d8 dk bundle publish --repo example.org/bundles/mybundle
```

7.5.3.4. Публикация нескольких бандлов из одного Git-репозитория

Разместите `.helm` с содержимым чарта и соответствующий ему `werf.yaml` в отдельную директорию для каждого бандла:

```
bundle1/
  .helm/
    templates/
    # ...
  werf.yaml
bundle2/
  .helm/
    templates/
    # ...
  werf.yaml
```

Теперь опубликуйте каждый бандл по отдельности:

```
cd bundle1
d8 dk bundle publish --repo example.org/bundles/bundle1
```

```
cd ../bundle2
d8 dk bundle publish --repo example.org/bundles/bundle2
```

7.5.3.5. Исключение файлов или директорий из публикуемого чарта

Файл `.helmignore`, находящийся в корне чарта, может содержать фильтры по именам файлов, при соответствии которым файлы или директории не будут добавляться в чарт при публикации. Формат правил такой же, как и в `.gitignore`, за исключением:

`**` не поддерживается;

`!` в начале строки не поддерживается;

`.helmignore` не исключает сам себя по умолчанию.

Также опция `--disable-default-values` для команды `d8 dk bundle publish` позволяет исключить из публикуемого чарта файл `values.yaml`.

7.5.3.6. Указание версии чарта при публикации

По умолчанию чарт публикуется с тегом `latest`. Указать иной тег, например, семантическую версию для публикуемого чарта, можно опцией `--tag`:

```
d8 dk bundle publish --repo example.org/bundles/mybundle --tag v1.0.0
```

Результат: опубликован чарт `example.org/bundles/mybundle:v1.0.0`.

Если при публикации будет обнаружено, что в OCI-репозитории уже существует чарт с таким тегом, то чарт в репозитории будет перезаписан.

7.5.3.7. Изменение версии опубликованного чарта

Для изменения тега уже опубликованного чарта скопируйте бандл с новым тегом с помощью команды `d8 dk bundle copy`, например:

```
d8 dk bundle copy --from example.org/bundles/mybundle:v1.0.0 --to example.org/bundles/renamedbundle:v2.0.0
```

7.5.3.8. Копирование бандла в другой репозиторий

Для удобного копирования бандла в другой репозиторий имеется команда `d8 dk bundle copy`. Кроме непосредственного копирования чарта и связанных с ним образов эта команда также обновит сохраненные в чарте Values, указывающие на путь к образам.

Пример:

```
d8 dk bundle copy --from example.org/bundles/mybundle:v1.0.0 --to example.org/bundles/renamedbundle:v2.0.0
```

7.5.3.9. Экспорт бандла из container registry в архив

После публикации бандл может быть экспортирован из репозитория в локальный архив для дальнейшей дистрибуции иными способами с помощью команды `d8 dk bundle copy`, например:

```
d8 dk bundle copy --from example.org/bundles/mybundle:v1.0.0 --to example.org/bundles/renamedbundle:v2.0.0
```

7.5.3.10. Импорт бандла из архива в репозиторий

Экспортированный в архив бандл можно снова импортировать в тот же или другой OCI-репозиторий командой `d8 dk bundle copy`, например:

```
d8 dk bundle copy --from archive:archive.tar.gz --to other.example.org/bundles/mybundle:v1.0.0
```

После этого вновь опубликованный бандл (чарт и его образы) снова можно использовать привычными способами.

7.5.3.11. Container registries, поддерживающие публикацию бандлов

Для публикации бандлов требуется container registry, поддерживающий спецификацию `OCI (Open Container Initiative)`.

7.6. Очистка

7.6.1. Очистка container registry

7.6.1.1. Обзор

Количество образов может стремительно расти, занимая больше места в container registry и, соответственно, значительно увеличивая его стоимость. Для контроля и поддержания приемлемого роста места, ПО «Deckhouse Platform» предлагает свой подход к очистке, который позволяет учитывать используемые образы в Kubernetes, а также актуальность образов по истории в Git.

Команда `d8 dk cleanup` рассчитана на периодический запуск. Удаление производится в соответствии с политиками очистки и является безопасной процедурой.

Вероятнее всего, политики очистки по умолчанию полностью покроют потребности проекта и дополнительная настройка не потребуется.

Важно отметить, что фактически размер занимаемого образами места в container registry не сокращается после очистки. ПО «Deckhouse Platform» удаляет теги неактуальных образов (манифесты), а для очистки связанных с ними данных необходимо периодически запускать сборщик мусора container registry.

7.6.1.2. Автоматизация очистки container registry

Для того, чтобы автоматизировать очистку неактуальных образов в container registry, необходимо выполнить следующие действия:

- Настроить периодический запуск `d8 dk cleanup` для удаления неактуальных тегов из container registry.
- Настроить периодический запуск сборщика мусора для непосредственного освобождения места в container registry.

7.6.1.3. Игнорирование образов, используемых в Kubernetes

ПО «Deckhouse Platform» подключается **ко всем кластерам** Kubernetes, описанным **во всех kubectl-контекстах**, и собирает имена образов для следующих типов объектов: `pod, deployment, replicaset, statefulset, daemonset, job, cronjob, replicationcontroller`.

Пользователь может регулировать поведение следующими параметрами (и связанными переменными окружения):

```
--kube-config, --kube-config-base64 для определения конфигурации (по умолчанию используется пользовательская конфигурация ~/.kube/config)
--kube-context для выполнения сканирования только в определенном контексте.
--scan-context-namespace-only для сканирования только связанного с контекстом namespace (по умолчанию все).
```

Сканирование Kubernetes отключается с помощью соответствующей директивы в `werf.yaml`:

```
cleanup:
  disableKubernetesBasedPolicy: true
```

Пока в кластере Kubernetes существует объект использующий образ, он никогда не удалится из container registry. Другими словами, если что-то было запущено в вашем кластере Kubernetes, то используемые образы ни при каких условиях не будут удалены при очистке.

7.6.1.4. Игнорирование свежесобранных образов

При удалении ПО «Deckhouse Platform» игнорирует образы, собранные в заданный период времени (по умолчанию за прошедшие 2 часа). При необходимости можно изменить период или совсем отключить политику соответствующими директивами в `werf.yaml`:

```
cleanup:
  disableBuiltWithinLastNHoursPolicy: false
  keepImagesBuiltWithinLastNHours: 2
```

7.6.1.5. Конфигурация политик очистки по истории Git

Конфигурация очистки состоит из набора политик, `keepPolicies`, по которым выполняется выборка значимых образов на основе истории git. Таким образом, в результате очистки **неудовлетворяющие политикам образы удаляются**.

Каждая политика состоит из двух частей:

- `references` определяет множество references, git-тегов или git-веток, которые будут использоваться при сканировании.
- `imagesPerReference` определяет лимит искомых образов для каждого reference из множества.

Любая политика должна быть связана с множеством Git-тегов (`tag`) либо Git-веток (`branch`). Можно указать определенное имя reference или задать специфичную группу, используя синтаксис регулярных выражений golang.

```
cleanup:
  disableBuiltWithinLastNHoursPolicy: false
  keepImagesBuiltWithinLastNHours: 2
```

При сканировании описанный набор git-веток будет искаться среди origin remote references, но при написании конфигурации префикс `origin/` в названии веток опускается.

Заданное множество references можно лимитировать, основываясь на времени создания git-тега или активности в git-ветке. Группа параметров `limit` позволяет писать гибкие и эффективные политики под различные workflow.

```
- references:
    branch: /^features\/.*/
    limit:
      last: 10
      in: 168h
      operator: And
```

В примере описывается выборка из не более чем 10 последних веток с префиксом `features/` в имени, в которых была какая-либо активность за последнюю неделю.

- Параметр `last` позволяет выбирать последние `n` reference’ов из определенного в `branch/tag` множества.
- Параметр `in` (синтаксис доступен в документации) позволяет выбирать Git-теги, которые были созданы в указанный период, или Git-ветки с активностью в рамках периода. Также для определенного множества `branch/tag`.
- Параметр `operator` определяет, какие reference’ы будут результатом политики: удовлетворяющие оба условия или любое из них (`And` по умолчанию).

По умолчанию при сканировании reference количество искомых образов не ограничено, но поведение может настраиваться группой параметров `imagesPerReference`:

```
imagesPerReference:
  last: int
  in: string
  operator: string
```

- Параметр `last` определяет количество искомых образов для каждого reference. По умолчанию количество не ограничено (`-1`).
- Параметр `in` (синтаксис доступен в документации) определяет период, в рамках которого необходимо выполнять поиск образов.
- Параметр `operator` определяет, какие образы сохранятся после применения политики: удовлетворяющие оба условия или любое из них (`And` по умолчанию).

Для Git-тегов проверяется только HEAD-коммит и значение `last >1` не имеет никакого смысла, является невалидным.

7.6.1.5.1. Приоритет политик для конкретного reference

При описании группы политик необходимо идти от общего к частному. Другими словами, `imagesPerReference` для конкретного reference будет соответствовать последней политике, под которую он подпадает:

```
- references:
    branch: /.*/
  imagesPerReference:
    last: 1
- references:
    branch: master
  imagesPerReference:
    last: 5
```

В данном случае, для reference master справедливы обе политики и при сканировании ветки `last` будет равен 5.

7.6.1.5.2. Политики по умолчанию

В случае, если в `werf.yaml` отсутствуют пользовательские политики очистки, используются политики по умолчанию, соответствующие следующей конфигурации:

```
cleanup:
  keepPolicies:
  - references:
      tag: /.*/
      limit:
        last: 10
  - references:
      branch: /.*/
      limit:
        last: 10
```

```
      in: 168h
      operator: And
  imagesPerReference:
    last: 2
    in: 168h
    operator: And
- references:
    branch: /^(main|master|staging|production)$/
  imagesPerReference:
    last: 10
```

Разберем каждую политику по отдельности:

1. Сохранять по одному образу для 10 последних тегов (по дате создания).
2. Сохранять по не более чем два образа, опубликованных за последнюю неделю, для не более 10 веток с активностью за последнюю неделю.
3. Сохранять по 10 образов для веток main, master, staging и production.

7.6.1.5.3. Политики по умолчанию

Если очистка по истории Git не требуются, то ее можно отключить в `werf.yaml` с помощью специальной директивы:

```
cleanup:
  disableGitHistoryBasedPolicy: true
```

7.6.1.6. Особенности работы с container registries

Зона ответственности очистки ПО «Deckhouse Platform» — удаление тегов образов (манифестов), а непосредственное удаление связанных данных выполняется с помощью сборщика мусора container registry (GC).

При вызове сборщика мусора container registry должен быть переведен в режим read-only или полностью отключен. Иначе есть высокая вероятность, что опубликованные во время процедуры образы не будут учтены сборщиком и будут повреждены.

ПО «Deckhouse Platform» пытается автоматически определить используемый container registry, используя заданный адрес репозитория (опция `--repo`). Пользователь может явно задать container registry опцией `--repo-container-registry` или переменной окружения `WERF_REPO_CONTAINER_REGISTRY`.

7.6.1.6.1. Selectel CRaaS

При очистке ПО «Deckhouse Platform» использует Selectel CR API, поэтому при очистке container registry необходимо определить username/password, account and vpc or vpcID.

Для того чтобы задать параметры, следует использовать следующие опции или соответствующие им переменные окружения:

- `--repo-selectel-username`
- `--repo-selectel-password`
- `--repo-selectel-account`
- `--repo-selectel-vpc or`
- `--repo-selectel-vpc-id`

7.6.1.6.2. Известные проблемы

- Иногда Selectel не отдает токен при использовании VPC ID. Попробуйте использовать имя VPC.
- CR API не позволяет удалять теги, которые хранятся в корне registry.
- Небольшой лимит запросов в API. При активной разработке могут быть проблемы с очисткой.

7.6.2. Автоматическая очистка хоста

7.6.2.1. Обзор

Очистка хоста удаляет неактуальных данные и сокращает размер кеша **автоматически** в рамках вызова основных команд ПО «Deckhouse Platform» и сразу для **всех проектов**. При необходимости очистку можно выполнять в ручном режиме с помощью команды `d8 dk host cleanup`.

7.6.2.2. Переопределение директории хранилища Docker

Параметр `--docker-server-storage-path` (или переменная окружения `WERF_DOCKER_SERVER_STORAGE_PATH`) позволяет явно задать директорию хранилища Docker в случае, если ПО «Deckhouse Platform» не может правильно определить ее автоматически.

7.6.2.3. Изменение порога занимаемого места и глубины очистки хранилища Docker

Параметр `--allowed-docker-storage-volume-usage` (`WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE`) позволяет изменить порог занимаемого места на томе, при достижении которого выполняется очистка хранилища Docker (по умолчанию 70%).

Параметр `--allowed-docker-storage-volume-usage-margin` (`WERF_ALLOWED_DOCKER_STORAGE_VOLUME_USAGE_MARGIN`) позволяет установить глубину очистки относительно установленного порога занимаемого места хранилища Docker (по умолчанию 5%).

7.6.2.4. Изменение порога занимаемого места и глубины очистки локального кэша

Параметр `--allowed-local-cache-volume-usage` (`WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE`) позволяет изменить порог занимаемого места на томе, при достижении которого выполняется очистка локального кэша (по умолчанию 70%).

Параметр `--allowed-docker-storage-volume-usage-margin` (`WERF_ALLOWED_LOCAL_CACHE_VOLUME_USAGE_MARGIN`) позволяет установить глубину очистки относительно установленного порога занимаемого места локального кэша (по умолчанию 5%).

7.6.2.5. Выключение автоматической очистки

Пользователь может выключить автоматическую очистку неактуальных данных хоста с помощью параметра `--disable-auto-host-cleanup` (`WERF_DISABLE_AUTO_HOST_CLEANUP`). В этом случае рекомендуется добавить команду `d8 dk host cleanup` в cron, например, следующим образом:

```
# /etc/cron.d/d8-d-host-cleanup
SHELL=/bin/bash
*/30 * * * * gitlab-runner d8 dk host cleanup
```

7.7. Справочник

7.7.1. werf.yaml

```
# Секция мета-информации
---
# Уникальное имя проекта приложения
project: string
# Версия конфигурации. На данный момент поддерживается единственная версия 1
configVersion: int
build: # Общие настройки сборки
  # Общий список целевых платформ для всех образов (например ['linux/amd64',
'linux/arm64',
  # 'linux/arm/v8'])
  platform: [ string, ... ]
deploy: # Настройки выката
  # Путь до директории helm чарта проекта (значение по умолчанию .helm)
  helmChartDir: string
  # Шаблон имени релиза (значение по умолчанию [[ project ]]-[[ env ]])
  helmRelease: string
  # Слагификация имени релиза (значение по умолчанию true)
  helmReleaseSlug: bool
  # Шаблон Kubernetes namespace (значение по умолчанию [[ project ]]-[[ env ]])
  namespace: string
  # Слагификация Kubernetes namespace (значение по умолчанию true)
  namespaceSlug: bool
# Настройка удаления неактульных образов
cleanup:
  # Отключить политику очистки, которая позволяет не удалять запущенные в
Kubernetes образы из
  # container registry
  disableKubernetesBasedPolicy: bool
  # Отключить политику очистки, которая позволяет не удалять образы с учетом
пользовательских
  # политик по истории Git (keepPolicies)
  disableGitHistoryBasedPolicy: bool
  # Отключить политику очистки, которая позволяет не удалять образы, собранные
в рамках заданного
  # периода времени (keepImagesBuiltWithinLastNHours)
  disableBuiltWithinLastNHoursPolicy: bool
```

```
  # Минимальное количество часов, которое должно пройти с момента сборки образа
(значение по
  # умолчанию 2)
  keepImagesBuiltWithinLastNHours: uint
  # Набор политик для выборки актуальных образов, используя историю Git
  keepPolicies:
  # Набор references, который будет использоваться при сканировании
  - references:
      # Множество git origin веток
      branch: string || /REGEXP/
      # Множество git origin тегов
      tag: string || /REGEXP/
    # Набор правил, по которым можно ограничить описанное множество references,
основываясь на
      # времени создания git-тега или активности в git-ветке
      limit:
      # Выборка последних n references из определенного в branch/tag множества
(значение по
        # умолчанию -1)
        last: int
      # Выборка git-тегов, которые были созданы в указанный период, или git-
веток с активностью
        # в рамках периода. Также для определенного множества branch/tag
        in: duration string
        # Определяет какие references будут результатом политики, те которые
удовлетворяют оба
        # условия или любое из них (значение по умолчанию And)
        operator: And || Or
    # Лимит искомых образов для каждого reference из множества
    imagesPerReference:
     # Количество искомых образов для каждого reference (значение по умолчанию
-1)
      last: int
      # Период, в рамках которого необходимо выполнять поиск образов
      in: duration string
    # Определяет какие образы сохранятся после применения политики, те которые
удовлетворяют оба
      # условия или любое из них (значение по умолчанию And)
      operator: And || Or
# Настройки связанные с работой ПО «Deckhouse Platform» с рабочей директорией
git проекта
gitWorktree:
  # Принудительно позволить ПО «Deckhouse Platform» использовать shallow clone
несмотря на ограничения
  # данного подхода
  forceShallowClone: bool
  # Разрешить процессу ПО «Deckhouse Platform» автоматически преобразовать
shallow clone проекта в
  # полный clone в процессе сборки по необходимости (значение по умолчанию true)
  allowUnshallow: bool
  # Разрешить процессу ПО «Deckhouse Platform» автоматически скачать новые ветки
и теги из origin в
  # процессе cleanup по необходимости (значение по умолчанию true)
  allowFetchOriginBranchesAndTags: bool
# Секция Dockerfile image: может использоваться произвольное количество секций
---
# Одно или несколько уникальных имен для образа
image: string || [ string, ... ] || ~
# Путь к Dockerfile относительно директории контекста
dockerfile: string
```

```
# Включить послойное кеширование Dockerfile-инструкций в container registry
staged: bool
# Путь к контексту внутри папки проекта
context: string
#  Список  целевых  платформ  для  данного  образа  (например  ['linux/amd64',
'linux/arm64',
# 'linux/arm/v8'])
platform: [ string, ... ]
# Добавление нехранящихся в git файлов и директорий в сборочный контекст. Пути
должны быть
# относительно директории контекста
contextAddFiles: [ string, ... ]
# Конкретная стадия Dockerfile (по умолчанию — последняя, подобно docker build
--target)
target: string
# Переменные для ARG Dockerfile-инструкций (подобно docker build --build-arg)
args: { name string: value string, ... }
# Установить связь host-to-IP (host:ip) (подобно docker build --add-host)
addHost: [ string, ... ]
# Сетевой режим для инструкций RUN во время сборки (подобно docker build --
network)
network: string
#  Сокет  агента  SSH  или  ключи  для  сборки  определенных  слоев  (только  если
используется BuildKit)
# (подобно docker build --ssh)
ssh: string
# Образы-зависимости для текущего образа
dependencies:
# Имя зависимого образа, который должен быть собран до сборки текущего образа
- image: string
  # Определить аргументы для импорта информации о зависимом образе в текущий
образ используя
  # Dockerfile build-args (опционально)
  imports:
  # Тип импортируемой информации об образе: ImageName, ImageDigest, ImageRepo
или ImageTag
  - type: string
   # Имя аргумента (Dockerfile build-args), который будет содержать указанный
тип информации об
    # образе
    targetBuildArg: string
# Набор директив для изменения манифеста образа.
docker:
  #  Включить  использование  незаэкранированных  символов  (например  кавычки  и
пробелы) в значениях
  # опций.
  exactValues: bool
 # Имя пользователя (или UID) и опционально пользовательская группа (или GID).
  USER: string
  # Рабочая директория.
  WORKDIR: string
  # Точки монтирования.
  VOLUME: [ string, ... ]
  # Переменные окружения.
  ENV: { name string: value string, ... }
  # Метаданные.
  LABEL: { name string: value string, ... }
  #  Описание  сетевых  портов,  которые  будут  прослушиваться  в  запущенном
контейнере.
  EXPOSE: [ string, ... ]
```

```
  # Команда по умолчанию, которая будет выполнена при запуске контейнера (форма
записи shell или
  # exec).
  ENTRYPOINT: string | [ string, ... ]
  # Аргументы по умолчанию для ENTRYPOINT (форма записи shell или exec).
  CMD: string | [ string, ... ]
 # Инструкции, которые Docker может использовать для проверки работоспособности
запущенного
  # контейнера.
  HEALTHCHECK: string
# Точки монтирования.
mount:
# Имя служебной директории
- from: tmp_dir || build_dir
  # Абсолютный или относительный путь до произвольного файла на хосте
  fromPath: string
  # Абсолютный путь в образе
  to: string
# Импортирование из образов и артефактов.
import:
# Имя артефакта, из которого выполнять копирование файлов
- artifact: string
  # Имя образа, из которого выполнять копирование файлов
  image: string
 # Имя стадии, из которой выполнять копирование файлов (по умолчанию последняя)
  stage: string
 # Выбор стадии импортирования файлов при сборке, до стадии install или setup
  before: string
 # Выбор стадии импортирования файлов при сборке, после стадии install или setup
  after: string
  # Абсолютный путь до файла или директории в выбранном образе/артефакте
  add: string
  # Абсолютный путь в конечном образе. По умолчанию соответствует пути add
  to: string
  # Имя или UID владельца
  owner: string
  # Имя или GID группы
  group: string
  # Глобы добавления
  includePaths: [ glob, ... ]
  # Глобы исключения
  excludePaths: [ glob, ... ]
# Образы-зависимости для текущего образа
dependencies:
# Имя зависимого образа, который должен быть собран до сборки текущего образа
- image: string
  # Выбор стадии перед которой должна быть импортирована информация об образе
(требуется указать
  # install или setup). Указанные переменные окружения будут доступны в
пользовательских стадиях
  # после указанной данной директивой стадии.
  before: string
  # Выбор стадии после которой должна быть импортирована информация об образе
(требуется указать
  # install или setup). Указанные переменные окружения будут доступны в
пользовательских стадиях
  # после указанной данной директивой стадии.
  after: string
  # Определить аргументы для импорта информации о зависимом образе в текущий
образ используя
```

```
  # переменные окружения (опционально)
  imports:
  # Тип импортируемой информации об образе: ImageName, ImageDigest, ImageRepo
или ImageTag
  - type: string
   # Имя переменной окружения, которая будет содержать указанный тип информации
об образе
    targetEnv: string
```

7.7.2. werf-giterminism.yaml

```
# Версия конфигурации. На данный момент поддерживается единственная версия 1
giterminismConfigVersion: int
# Правила ослабления гитерминизма для CLI
cli:
  # Разрешить опцию --use-custom-tag
  allowCustomTags: bool
# Правила ослабления гитерминизма для конфигурации ПО «Deckhouse Platform»
(werf.yaml)
config:
  # Читать конфигурационный файл из директории проекта, не сверяя контент с
файлом из текущего
  # коммита и игнорируя исключения в .gitignore
  allowUncommitted: bool
  # Читать определенные шаблоны конфигурационного файла (.werf/**/*.tmpl) из
директории проекта,
  # не сверяя контент с файлами текущего коммита и игнорируя исключения
в .gitignore
  allowUncommittedTemplates: [ glob, ... ]
  # Правила для функций Go-шаблонизатора
  goTemplateRendering:
    # Разрешить определенные переменные окружения (при использовании функции
env).
    allowEnvVariables: [ string || /REGEXP/, ... ]
   # Читать определенные конфигурационные файлы из директории проекта, не сверяя
контент с
    # файлами текущего коммита и игнорируя исключения в .gitignore (используя
функции .Files.Get и
    # .Files.Glob)
    allowUncommittedFiles: [ glob, ... ]
  # Правила для dockerfile-образа
  dockerfile:
   # Читать определенные dockerfiles из директории проекта, не сверяя контент
с файлами текущего
    # коммита и игнорируя исключения в .gitignore
    allowUncommitted: [ glob, ... ]
   # Читать определенные .dockerignore-файлы из директории проекта, не сверяя
контент с файлами
    # текущего коммита и игнорируя исключения в .gitignore
    allowUncommittedDockerignoreFiles: [ glob, ... ]
   # Разрешить использование определенных файлов или директорий из директории
проекта при
    # использовании директивы contextAddFiles.
    allowContextAddFiles: [ string, ... ]
# Правила ослабления гитерминизма для helm-файлов (.helm)
helm:
  # Читать определенные helm-файлы из директории проекта, не сверяя контент с
файлами текущего
  # коммита и игнорируя исключения в .gitignore
```

```
  allowUncommittedFiles: [ glob, ... ]
```

7.7.3. Шаблонизатор werf.yaml

При чтении конфигурации `werf.yaml`, ПО «Deckhouse Platform» использует встроенный движок шаблонов Go (text/template), а также расширяет набор функций с помощью Sprig и функций ПО «Deckhouse Platform».

При организации конфигурации она может быть разбита на отдельные файлы в директории шаблонов.

7.7.4. Встроенные возможности шаблонизатора Go

Для эффективной работы мы рекомендуем изучить все возможности шаблонизатора или хотя бы ознакомиться со следующими разделами:

- Actions, основные возможности шаблонизатора.
- Переменные.
- Работа с текстом и пробелами.
- Функции.

7.7.5. Функции Sprig

Библиотека Sprig предоставляет более 70 функций:

- Строковые функции.
- Математические функции.
- Функции кодирования.
- Работа с данными в формате ключ/значение (dict).
- Работа с файловыми путями.
- Остальное.

ПО «Deckhouse Platform» не поддерживает функцию expandenv и имеет свою собственную реализацию для функции env.

7.7.6. Дополнительные функции

7.7.6.1. Различные окружения

.*Env*

Переменная `.Env` позволяет организовывать конфигурацию для нескольких сред (testing, staging, production и т.п.) и переключаться между ними с помощью опции `--env=<environment_name>`.

В helm шаблонах таким же образом может использоваться переменная `.Values.werf.env`.

7.7.6.2. Информация о текущем коммите

.Commit.Hash

`{{ .Commit.Hash }}` возвращает SHA текущего коммита. Если есть возможность, лучше избегать использование `.Commit.Hash`, т. к. это может приводить к ненужным пересборкам слоев.

.Commit.Date

`{{ .Commit.Date.Human }}` возвращает дату текущего коммита в человекопонятной форме.
`{{ .Commit.Date.Unix }}` возвращает дату текущего коммита в формате Unix epoch.

7.7.6.3. Шаблонизация

include

Функция include позволяет переиспользовать общие части конфигурации, а также разбивать конфигурацию на несколько файлов.

Синтаксис:

```
{{ include "<TEMPLATE_NAME>" <VALUES> }}
```

tpl

Функция `tpl` позволяет обрабатывать строки как Go-шаблоны.

Синтаксис:

```
{{ tpl "<STRING>" <VALUES> }}
```

`<STRING>` — контент файла проекта, значение переменной окружения либо произвольная строка.

`<VALUES>` — значения шаблона. При использовании текущего контекста, ., в шаблоне можно использовать все шаблоны и значения (в том числе, описанные в файлах директории шаблонов).

7.7.6.4. Переменные окружения

env

Функция `env` читает переменную окружения.

Синтаксис:

```
{{ env "<ENV_NAME>" }}
{{ env "<ENV_NAME>" "default_value" }}
```

По умолчанию, использование функции `env` запрещено гитерминизмом

7.7.6.5. Файлы проекта

.Files.Get

Функция `.Files.Get` получает содержимое определенного файла проекта.

Синтаксис:

```
{{ .Files.Get "<FILE_PATH>" }}
```

По умолчанию, использование файлов, которые имеют незакоммиченные изменения, запрещено гитерминизмом

.Files.Glob

Функция `.Files.Glob` позволяет работать с файлами проекта и их содержимым по глобу.

Функция поддерживает `shell pattern matching` и `**`. Результаты вызова функции можно объединить, используя sprig-функцию `merge`, к примеру:

```
{{ $filesDict := merge (.Files.Glob "glob1") (.Files.Glob "glob2") }}
```

Синтаксис:

```
{{ .Files.Glob "<GLOB>" }}
```

По умолчанию, использование файлов, которые имеют незакоммиченные изменения, запрещено гитерминизмом

7.7.6.6. Другие

required

Функция `required` проверяет наличие значения у определенной переменной. Если значение пусто, то рендеринг шаблона завершится ошибкой с заданным пользователем сообщением.

Синтаксис:

```
value: {{ required "<ERROR_MSG>" <VALUE> }}
```

fromYaml

Функция `fromYaml` декодирует YAML-документ в структуру.

Синтаксис:

```
value: {{ fromYaml "<STRING>" }}
```

7.7.7. Директория шаблонов

Файлы шаблонов могут храниться в зарезервированной директории (по умолчанию `.werf`) с расширением `.tmpl` (поддерживается произвольная вложенность `.werf/**/*.tmpl`).

Все файлы шаблонов и основная конфигурация определяют общий контекст:

- Файл шаблонов является полноценным шаблоном и может использоваться функцией `include` по относительному пути (`{{ include "directory/partial.tmpl" . }}`).
- Шаблон определенный с помощью `define` в одном файле шаблонов доступен в любом другом, включая основную конфигурацию.

7.7.8. Аннотации для деплоя

Данный раздел содержит описание аннотаций, которые меняют поведение механизма отслеживания ресурсов в процессе выката с помощью ПО «Deckhouse Platform». Все аннотации должны быть объявлены в шаблонах чарта.

- `werf.io/weight` — задает вес ресурса, который определяет порядок развертывания ресурсов.
- `<any-name>.external-dependency.werf.io/resource` — дождаться, пока указанная внешняя зависимость будет запущена, и только после этого приступить к развертыванию аннотированного ресурса.
- `<any-name>.external-dependency.werf.io/namespace` — задать пространство имен для внешней зависимости.
- `werf.io/replicas-on-creation` — задает количество реплик, которое должно быть установлено при первичном создании ресурса (полезно при использовании HPA).
- `werf.io/track-termination-mode` — определяет условие при котором ПО «Deckhouse Platform» остановит отслеживание ресурса.
- `werf.io/fail-mode` — определяет как ПО «Deckhouse Platform» обработает ресурс в состоянии ошибки. Ресурс в свою очередь перейдет в состояние ошибки после превышения порога допустимых ошибок, обнаруженных при отслеживании этого ресурса в процессе выката.
- `werf.io/failures-allowed-per-replica` — определяет порог ошибок, обнаруживаемых при отслеживании этого ресурса в процессе выката, после превышения которого ресурс перейдет в состояние ошибки. ПО «Deckhouse Platform» обработает это состояние в соответствии с настройкой fail mode.

- `werf.io/ignore-readiness-probe-fails-for-CONTAINER_NAME` — переопределить высчитываемый автоматически период, в течение которого неуспешные readiness-пробы будут игнорироваться и не будут переводить ресурс в состояние ошибки.
- `werf.io/no-activity-timeout` — переопределить период неактивности, по истечении которого ресурс перейдет в состояние ошибки.
- `werf.io/log-regex` — показывать в логах только те строки вывода ресурса, которые подходят под указанный шаблон.
- `werf.io/log-regex-for-CONTAINER_NAME` — показывать в логах только те строки вывода для указанного контейнера, которые подходят под указанный шаблон.
- `werf.io/skip-logs` — выключить логирование вывода для ресурса.
- `werf.io/skip-logs-for-containers` — выключить логирование вывода для указанного контейнера.
- `werf.io/show-logs-only-for-containers` — включить логирование вывода только для указанных контейнеров ресурса.
- `werf.io/show-service-messages` — включить вывод сервисных сообщений и событий Kubernetes для данного ресурса.

Resource weight

```
werf.io/weight: "NUM"
```

Пример:

```
werf.io/weight: "10"
werf.io/weight: "-10"
```

Может быть положительным числом, отрицательным числом или нулем. Значение передается в виде строки. По умолчанию `weight` имеет значение 0. Работает только для ресурсов, не относящихся к хукам. Для хуков используйте `helm.sh/hook-weight`, логика работы которого почти такая же.

Этот параметр задает вес ресурсов, определяя порядок их развертывания. Сначала ПО «Deckhouse Platform» группирует ресурсы в соответствии с их весом, а затем последовательно развертывает их, начиная с группы с наименьшим весом. В этом случае ПО «Deckhouse Platform» не будет приступать к развертыванию следующей группы ресурсов, пока развертывание предыдущей не завершено успешно.

External dependency resource

```
<any-name>.external-dependency.werf.io/resource: type[.version.group]/name
```

Пример:

```
secret.external-dependency.werf.io/resource: secret/config
someapp.external-dependency.werf.io/resource: deployments.v1.apps/app
```

Задает внешнюю зависимость для ресурса. Ресурс с аннотацией будет развернут только после создания и готовности внешней зависимости.

External dependency namespace

```
<any-name>.external-dependency.werf.io/namespace: name
```

Указывает пространство имен для внешней зависимости, заданной соответствующей аннотацией. Префикс <any-name> должен быть таким же, как у аннотации, определяющей внешнюю зависимость.

Replicas on creation

Когда для ресурса включен HPA, использование `spec.replicas` может привести к непредсказуемому поведению, потому что каждый раз когда происходит converge для ПО «Deckhouse Platform» chart через CI/CD количество реплик ресурса будет сброшено к статически заданному в шаблонах чарта значению `spec.replicas`, даже если это значение изменил HPA в рантайме.

Одно из рекомендованных решений — совсем удалить spec.replicas из шаблонов чарта. Однако если необходимо установить начальное значение реплик при создании ресурса, можно воспользоваться аннотацией: "`werf.io/replicas-on-creation`".

```
"werf.io/replicas-on-creation": "NUM"
```

Задает число реплик, которые должны быть установлены для ресурса при его первичном создании.

`"NUM"` должно быть указано строкой (в двойных кавычках), потому что аннотации не поддерживают передачу других типов данных кроме строк, аннотации с другим типом данных будут проигнорированы.

Track termination mode

```
"werf.io/track-termination-mode": WaitUntilResourceReady|NonBlocking
```

Определяет условие остановки отслеживания ресурса в процессе деплоя:

- `WaitUntilResourceReady` (по умолчанию) — весь процесс деплоя будет отслеживать и ожидать готовности ресурса с данной аннотацией. Т.к. данный режим

включен по умолчанию, то, по умолчанию, процесс деплоя ждет готовности всех ресурсов.

- `NonBlocking` — ресурс с данной аннотацией отслеживается только пока есть другие ресурсы, готовности которых ожидает процесс деплоя.

Используйте аннотации — `"werf.io/track-termination-mode": NonBlocking` и `"werf.io/fail-mode": IgnoreAndContinueDeployProcess`, когда описываете в релизе объект Job, который должен быть запущен в фоне и не влияет на процесс деплоя.

Используйте аннотацию `"werf.io/track-termination-mode": NonBlocking`, когда описываете в релизе объект `StatefulSet` с ручной стратегией выката (параметр `OnDelete`) и не хотите блокировать весь процесс деплоя из-за этого объекта, дожидаясь его обновления.

Fail mode

```
"werf.io/fail-mode": FailWholeDeployProcessImmediately|HopeUntilEndOfDeployProcess|IgnoreAndContinueDeployProcess
```

Определяет как ПО «Deckhouse Platform» будет обрабатывать ресурс в состоянии ошибки, которое возникает после превышения порога ошибок, возникающих во время отслеживания данного ресурса в процессе деплоя:

- `FailWholeDeployProcessImmediately` (по умолчанию) — в случае ошибки при деплое ресурса с данной аннотацией, весь процесс деплоя будет завершен с ошибкой.
- `HopeUntilEndOfDeployProcess` — в случае ошибки при деплое ресурса с данной аннотацией его отслеживание будет продолжаться, пока есть другие ресурсы, готовности которых ожидает процесс деплоя, либо все оставшиеся ресурсы имеют такую-же аннотацию. Если с ошибкой остался только этот ресурс или несколько ресурсов с такой-же аннотацией, то в случае сохранения ошибки весь процесс деплоя завершается с ошибкой.
- `IgnoreAndContinueDeployProcess` — ошибка при деплое ресурса с данной аннотацией не влияет на весь процесс деплоя.

Failures allowed per replica

```
"werf.io/failures-allowed-per-replica": "NUMBER"
```

По умолчанию, при отслеживании статуса ресурса допускается срабатывание ошибки 1 раз, прежде чем весь процесс деплоя считается ошибочным. Этот параметр влияет на поведение

настройки Fail mode: определяет порог срабатывания, после которого начинает работать режим реакции на ошибки.

Ignore readiness probe failures for container

```
"werf.io/ignore-readiness-probe-fails-for-CONTAINER_NAME": "TIME"
```

Эта аннотация позволяет переопределить высчитываемый автоматически период, в течение которого неуспешные readiness-пробы не станут переводить ресурс в состояние ошибки, т. е. будут проигнорированы. По умолчанию период игнорирования неудачных readiness-проб автоматически вычисляется на основе конфигурации readiness-пробы. Заметим, что если в конфигурации readiness-пробы указано `failureThreshold: 1`, тогда первая же неудачная readiness-проба переведет ресурс в состояние ошибки, независимо от периода игнорирования.

Пример:

```
"werf.io/ignore-readiness-probe-fails-for-backend": "20s"
```

Вывод:

No activity timeout

```
werf.io/no-activity-timeout: "TIME"
```

По умолчанию: `4m`

Пример:

```
werf.io/no-activity-timeout: "8m30s"
werf.io/no-activity-timeout: "90s"
```

При отсутствии новых событий и обновлений ресурса в течение `TIME` ресурс перейдет в состояние ошибки.

Log regex

```
"werf.io/log-regex": RE2_REGEX
```

Определяет `Re2 regex` шаблон, применяемый ко всем логам всех контейнеров всех подов ресурса с этой аннотацией. ПО «Deckhouse Platform» будет выводить только те строки лога, которые удовлетворяют regex-шаблону. По умолчанию ПО «Deckhouse Platform» выводит все строки лога.

Log regex for container

```
"werf.io/log-regex-for-CONTAINER_NAME": RE2_REGEX
```

Определяет `Re2 regex` шаблон, применяемый к логам контейнера с именем `CONTAINER_NAME` всех подов с данной аннотацией. ПО «Deckhouse Platform» будет выводить только

те строки лога, которые удовлетворяют regex-шаблону. По умолчанию ПО «Deckhouse Platform» выводит все строки лога.

Skip logs

```
"werf.io/skip-logs": "true"|"false"
```

Если установлена в "true", то логи всех контейнеров пода с данной аннотацией не выводятся при отслеживании. Отключено по умолчанию.

Skip logs for containers

```
"werf.io/skip-logs-for-containers": CONTAINER_NAME1,CONTAINER_NAME2,CONTAINER_NAME3...
```

Список (через запятую) контейнеров пода с данной аннотацией, для которых логи не выводятся при отслеживании.

Show logs only for containers

```
"werf.io/show-logs-only-for-containers": CONTAINER_NAME1,CONTAINER_NAME2,CONTAINER_NAME3...
```

Список (через запятую) контейнеров пода с данной аннотацией, для которых выводятся логи при отслеживании. Для контейнеров, чьи имена отсутствуют в списке, логи не выводятся. По умолчанию выводятся логи для всех контейнеров всех подов ресурса.

Show service messages

```
"werf.io/show-service-messages": "true"|"false"
```

Если установлена в "true", то при отслеживании для ресурсов будет выводиться дополнительная отладочная информация, такая как события Kubernetes. По умолчанию, ПО «Deckhouse Platform» выводит такую отладочную информацию только в случае если ошибка ресурса приводит к ошибке всего процесса деплоя.

8. Принципы безопасной работы средства

При эксплуатации ПО «Deckhouse Platform» должно быть обеспечено выполнение следующих условий:

- наличие администраторов безопасности, обеспечивающих правильную эксплуатацию ПО «Deckhouse Platform», в том числе:
  - предотвращение несанкционированного доступа к идентификаторам и паролям привилегированных пользователей (администраторов безопасности);
  - предотвращение реализации некорректных методов управления доступом, типов (чтение, запись, выполнение или иной тип) и правил разграничения доступа;
  - обеспечение физической сохранности оборудования, на которое установлено изделие, и исключение возможности доступа к ним посторонних лиц;
- периодический контроль целостности изделия;
- ежедневная проверка рабочих мест администратором безопасности на наличие вредоносного ПО;
- ежемесячный поиск актуальных уязвимостей и сведений об уязвимостях изделия и среды функционирования, анализ идентифицированных уязвимостей на предмет возможности их использования для нарушения безопасности.

В ПО «Deckhouse Platform» реализованы функции безопасности. Перечень функций безопасности в соответствии с исполнениями представлены в таблице ниже.

№ п/п	Функции безопасности	Исполнения
1.	Идентификация и аутентификация пользователей в средстве контейнеризации	Kubernetes+Virtualization; Kubernetes
2.	Изоляция контейнеров средством контейнеризации	Kubernetes+Virtualization; Kubernetes
3.	Выявление уязвимостей в образах контейнеров	Kubernetes+Virtualization; Kubernetes
4.	Проверка корректности конфигурации контейнеров	Kubernetes+Virtualization; Kubernetes

№ п/п	Функции безопасности	Исполнения
5.	Контроль целостности контейнеров и их образов в средстве контейнеризации	Kubernetes+Virtualization; Kubernetes
6.	Регистрация событий безопасности в средстве контейнеризации	Kubernetes+Virtualization; Kubernetes
7.	Управление доступом в средстве контейнеризации	Kubernetes+Virtualization; Kubernetes
8.	Доверенная загрузка виртуальных машин	Kubernetes+Virtualization; Virtualization
9.	Контроль целостности в средстве виртуализации	Kubernetes+Virtualization; Virtualization
10.	Регистрация событий безопасности в средстве виртуализации	Kubernetes+Virtualization; Virtualization
11.	Управление доступом в средстве виртуализации	Kubernetes+Virtualization; Virtualization
12.	Резервное копирование виртуальных машин	Kubernetes+Virtualization; Virtualization
13.	Управление потоками информации в средстве виртуализации	Kubernetes+Virtualization; Virtualization
14.	Защита памяти	Kubernetes+Virtualization; Virtualization
15.	Ограничение программной среды	Kubernetes+Virtualization; Virtualization
16.	Идентификация и аутентификация пользователей в средстве виртуализации	Kubernetes+Virtualization; Virtualization

№ п/п	Функции безопасности	Исполнения
17.	Централизованное управление образами виртуальных машин и виртуальными машинами	Kubernetes+Virtualization; Virtualization
18.	Управление доступом субъектов доступа к объектам доступа (УПД)	Kubernetes+Virtualization; Kubernetes; Virtualization

9. Типы событий безопасности, связанные с доступными пользователю функциями средства

В ПО «Deckhouse Platform» регистрируются следующие события безопасности, связанные с доступными пользователю функциями ПО «Deckhouse Platform»:

- получение доступа к образам контейнеров;
- запуск и остановка контейнеров с указанием причины остановки;
- изменение ролевой модели;
- модификация запускаемых контейнеров;
- изменение конфигурации ПО «Deckhouse Platform»;
- выявление известных уязвимостей в образах контейнеров и некорректности конфигурации;
- факты нарушения целостности объектов контроля;
- успешные и неуспешные попытки аутентификации пользователей ПО «Deckhouse Platform»;
- доступ пользователей к виртуальным машинам;
- создание и удаление виртуальных машин;
- запуск и остановка виртуальных машин с указанием причины остановки;
- изменение конфигураций виртуальных машин.

Аварийные ситуации

9.1. Действия после сбоев и ошибок эксплуатации ПО «Deckhouse Platform»

В случае несоблюдения условий выполнения технологического процесса, в том числе возникновении сбоев и ошибок эксплуатации ПО «Deckhouse Platform», необходимо обратиться к техническому персоналу и представителям эксплуатирующих подразделений.

9.2. Несанкционированное вмешательство в данные

В случаях обнаружения несанкционированного вмешательства в данные необходимо обратиться к техническому персоналу и представителям эксплуатирующих подразделений.

Приложение А

Компонент	**Интерфейс обновления требуемого объекта ClusterRole (Кластерная роль)**	**Функция интерфейса**	**SuperAdmin**	**ClusterAdmin**	**ClusterEditor**	**Admin**	**Editor**	**PrivilegedUser**	**User**
dex	Интерфейс реализующий взаимодействие по протоколу OpenID Connect и OAuth 2.0	Веб-интерфейс для ввода логина и пароля	+	+	+	+	+	+	+
kube-apiserver	Интерфейс замены ClusterRole (Кластерная роль)	Создать роль кластера (Кластерную роль)	+	+	-	-	-	-	-
	Интерфейс удаления роли кластера	Частичное обновление требуемого объекта ClusterRole (Кластерная роль)	+	+	-	-	-	-	-

	Интерфейс удаления подборки кластерных ролей	Замена требуемого объекта ClusterRole (Кластерная роль)	+	+	-	-	-	-	-
	Интерфейс просмотра кластерной роли	Удалить роль кластера	+	+	-	-	-	-	-
	Интерфейс просмотра (или наблюдения) списка кластерных ролей	Удалить подборку объекта ClusterRole (Кластерная роль)	+	+	-	-	-	-	-
	Интерфейс просмотра изменений в объекте ClusterRole (Кластерная роль)	Считывание требуемого объекта ClusterRole (Кластерная роль)	+	+	-	-	-	-	-
	Интерфейс просмотра изменений в объекте ClusterRole (Кластерная роль)	Список или просмотр объекта ClusterRole (Кластерная роль)	+	+	-	-	-	-	-

	Интерфейс создания clusterrolebinding	Просмотр изменений в объекте ClusterRole (Кластерная роль). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	-	-	-	-	-
	Интерфейс частичного обновления ClusterRoleBinding	Просмотр изменений в объекте ClusterRole (Кластерная роль). Устарело: вместо этого следует	+	+	-	-	-	-	-

		использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).							
	Интерфейс замены clusterrolebinding	Создать объект ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс замены clusterrolebinding	Частичное обновление объекта ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс удаления набора clusterrolebinding	Замена требуемого объекта ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс просмотра	Удалить объект	+	+	-	-	-	-	-

	clusterrolebinding	ClusterRoleBinding (Привязка роли кластера)							
	Интерфейс просмотра (или наблюдения) списка clusterrolebinding	Удалить подборку объекта ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс просмотра изменений в объекте ClusterRoleBinding	Считывание требуемого объекта ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке ClusterRoleBinding.	Список или просмотр объекта ClusterRoleBinding (Привязка роли кластера)	+	+	-	-	-	-	-
	Интерфейс создания роли	Просмотр изменений в объекте ClusterRoleBinding (Привязка роли кластера). Устарело: вместо этого	+	+	+	-	-	-	-

		следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).							
	Интерфейс частичного обновления требуемой Роли	Просмотр отдельных изменений в списке объекта ClusterRoleBinding (Привязка роли кластера). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс замены	Создать Роль	+	+	+	-	-	-	-

	требуемой Роли								
	Интерфейс удаления роли	Частичное обновление требуемой Роли	+	+	+	-	-	-	-
	Интерфейс удаления набора ролей	Замена требуемой Роли	+	+	+	-	-	-	-
	Интерфейс просмотра требумой Роли	Удалить Роль	+	+	+	-	-	-	-
	Интерфейс просмотра (или наблюдения) списка ролей пространства имен	Удалить подборку Ролей	+	+	+	-	-	-	-
	Интерфейс просмотра (или наблюдения) списка ролей	Считывание требуемой Роли	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в роли.	Список или просмотр требуемой Роли	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных	Список или просмотр требуемой	+	+	+	-	-	-	-

	изменений в списке ролей пространства имен.	Роли							
	Интерфейс просмотра отдельных изменений в списке ролей.	Просмотр изменений в объекте Роль. Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-
	Интерфейс создания RoleBinding	Просмотр отдельных изменений в списке Ролей. Устарело: вместо этого используйте параметр	+	+	+	-	-	-	-

		'watch' (просмотр) в списке операций.							
	Интерфейс частичного обновления требуемого RoleBinding	Просмотр отдельных изменений в списке Ролей. Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс замены требуемого RoleBinding	Создать RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс удаления RoleBinding	Частичное обновление требуемого RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс удаления набора RoleBinding	Замена требуемого RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс просмотра	Удалить RoleBinding	+	+	+	-	-	-	-

	требуемого RoleBinding	(Связка ролей)							
	Интерфейс просмотра (или наблюдения) списка RoleBinding пространства имен	Удалить подборку RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс просмотра (или наблюдения) списка RoleBinding	Считывание требуемого RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс наблюдения изменений RoleBinding	Список или просмотр объекта RoleBinding (Связка ролей)	+	+	+	-	-	-	-
	Интерфейс наблюдения отдельных изменений в списке RoleBinding пространства имен	Список или просмотр объекта RoleBinding (Связка ролей)	+	+	+	-	-	-	-

	Интерфейс наблюдения отдельных изменений в списке RoleBinding	Просмотр изменений в объекте RoleBinding (Связка ролей). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-
	Интерфейс считывания лога требуемого Пода	Просмотр отдельных изменений в списке RoleBinding (Связка ролей). Устарело: вместо этого используйте параметр	+	+	+	+	+	-	-

		'watch' (просмотр) в списке операций.							
	Интерфейс создания пода	Просмотр отдельных изменений в списке RoleBinding (Связка ролей). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	+	+	-	-
	Интерфейс создания выселения пода	Считывание лога требуемого Пода	+	+	+	+	+	-	-
	Интефейс частичного обновления требуемого Пода	Создать Под	+	+	+	+	+	-	-
	Интерфейс замены требуемого Пода	Создать выселение Пода	+	+	+	+	+	-	-

	Интерфейс удаления пода	Частичное обновление требуемого Пода	+	+	+	+	+	-	-
	Интерфейс удаления подборки подов	Замена требуемого Пода	+		+	+	+	-	-
	Интерфейс считывания требуемого пода	Удалить Под	+	+	+	+	+	-	-
	Интерфейс просмотра (или наблюдения) списка подов пространства имен	Удалить подборку Пода	+	+	+	+	+	-	-
	Интерфейс просмотра (или наблюдения) списка подов	Считывание требуемого Пода	+	+	+	+	+	-	-
	Интерфейс наблюдения изменений требуемого Пода	Список или просмотр объектов Пода	+	-	+	+	+	-	-
	Интерфейс наблюдения	Список или просмотр	+	-	+	+	+	-	-

	изменений в списке Подов пространства имен	объектов Пода							
	Интерфейс наблюдения изменений в списке Подов	Просмотр изменений в объекте Под. Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	-	+	+	+	-	-
	Интерфейс частичного обновления статуса требуемого Пода	Просмотр отдельных изменений в списке Подов. Устарело: вместо этого используйте параметр	+	+	+	+	-	-	-

		'watch' (просмотр) в списке операций.							
	Интерфейс считывания статуса требуемого Под	Просмотр отдельных изменений в списке Подов. Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	+	-	-	-
	Интефейс замены статуса требуемого Пода	Частичное обновление статуса требуемого Пода	+	+	+	+	-	-	-
	Интерфейс частичного обновление эфемерных контейнеров требуемого Пода	Считывание статуса требуемого Под	+	+	+	+	-	-	-
	Интерфейс считывания эфемерных	Замена статуса требуемого Пода	+	+	+	+	-	-	-

	контейнеров требуемого Пода								
	Интерфейс замены эфемерных контейнеров требуемого Пода	Частичное обновление эфемерных контейнеров требуемого Пода	+	+	+	+	-	-	-
	Интерфейс подключения POST-запросов к переадресации портов Пода	Считывание эфемерных контейнеров требуемого Пода	+	+	+	-	-	-	-
	Интерфейс подключения POST-запросов к прокси-серверу Пода	Замена эфемерных контейнеров требуемого Пода	+	+	+	-	-	-	-
	Интерфейс подключения POST-запросов к прокси-серверу Пода	Подключить POST-запросы к переадресации портов Пода	+	+	+	-	-	-	-
	Интерфейс подключения DELETE-запросов к прокси-серверу Пода	Подключить POST-запросы к прокси-серверу Пода	+	+	+	-	-	-	-

	Интерфейс подключения GET-запросов к переадресации портов Пода	Подключить DELETE-запрос к прокси-серверу Пода	+	+	+	-	-	-	-
	Интерфейс подключения GET-запросов к прокси-серверу Пода	Подключить DELETE-запрос к прокси-серверу Пода	+	+	+	-	-	-	-
	Интерфейс подключения GET-запросов к прокси-серверу Пода	Подключить GET-запросы к переадресации портов Пода	+	+	+	-	-	-	-
	Интерфейс подключения HEAD-запросов к прокси-серверу Пода	Подключить GET-запросы к прокси-серверу Пода	+	+	+	-	-	-	-
	Интерфейс подключения PUT-запросов к прокси-серверу Пода	Подключить HEAD-запросы к прокси-серверу Пода	+	+	+	-	-	-	-
	Интерфейс создания HorizontalPodAutoscale	Подключить PUT-запросы к прокси-серверу Пода	+	+	+	-	-	-	-

	Интерфейс частичного обновления требуемого HorizontalPodAutoscale	Подключить PUT-запросы к прокси-серверу Пода	+	+	+	-	-	-	-
	Интерфейс заменаы требуемого HorizontalPodAutoscale	Создать HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления HorizontalPodAutoscale	Частичное обновление требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора HorizontalPodAutoscale	Замена требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-

	Интерфейс считывания требуемого HorizontalPodAutoscale	Удалить HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	+	-	-	-
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale пространства имен	Удалить побдорку HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	+	-	-	-
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale	Считывание требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	+	-	-	-
	Интерфейс просмотра изменений HorizontalPodAutoscale	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальное автомасштабир	+	+	+	+	-	-	-

		ование подов)							
	Интерфейс просмотра отдельных изменений в списке HorizontalPodAutoscale	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	+	-	-	-
	Интерфейс просмотра отдельных изменений в списке HorizontalPodAutoscale	Просмотр изменений в объекте HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного	+	+	+	+	-	-	-

		элемента с помощью параметра 'fieldselector' (выбор поля).							
	Интерфейс частичного обновление статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	+	-	-	-
	Интерфейс считывания статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale	+	+	+	+	-	-	-

		(Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замены статуса требуемого HorizontalPodAutoscale	Частичное обновление статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	+	-	-	-
	Интерфейс оздания PodTemplate	Считывание статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-

	Интерфейс частичного обновления требуемого PodTemplate	Замена статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс замены требуемого PodTemplate	Создать PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс удаления PodTemplate	Частичное обновление требуемого PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс удаления подборку PodTemplate	Замена требуемого PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого PodTemplate	Удалить PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения объекта PodTemplate пространства	Удалить подборку PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-

	имен								
	Интерфейс просмотра или наблюдения объекта PodTemplate	Считывание требуемого PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в PodTemplate	Список или просмотр объекта PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке PodTemplate пространства имен	Список или просмотр объекта PodTemplate (Шаблон Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке PodTemplate	Просмотр изменений в объекте PodTemplate (Шаблон Пода). Устарело: вместо этого следует использовать параметр 'watch'	+	+	+	-	-	-	-

		(просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).							
	Интерфейс создания PodDisruptionBudget	Просмотр отдельных изменений в списке PodTemplate (Шаблон Пода). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс частичного обновления PodDisruptionBudget	Просмотр отдельных изменений в списке PodTemplate (Шаблон	+	+	+	-	-	-	-

		Пода). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замены требуемого PodDisruptionBudget	Создать объект PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс удаления PodDisruptionBudget	Частичное обновление объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора PodDisruptionBudget	Замена требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-

	Интерфейс считывания требуемого PodDisruptionBudget	Удалить объект PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения PodDisruptionBudget пространства имен	Удалить подборку объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения PodDisruptionBudget	Считывание требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в PodDisruptionBudget	Список или просмотр объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-

	Интерфейс просмотр отдельных изменений в списке PodDisruptionBudget	Список или просмотр объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке PodDisruptionBudget	Просмотр изменений в объекте PodDisruptionBudget (Квота количества неработающих подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector'	+	+	+	-	-	-	-

		(выбор поля).							
	Интерфейс частичного обновления статуса требуемого PodDisruptionBudget	Просмотр отдельных изменений в списке объекта PodDisruptionBudget (Квота количества неработающих подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс считывания статуса требуемого PodDisruptionBudget	Просмотр отдельных изменений в списке объекта PodDisruptionBudget (Квота количества	+	+	+	-	-	-	-

		неработающих подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замены статуса требуемого PodDisruptionBudget	Частичное обновление статуса требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс создания PodSecurityPolicy	Считывание статуса требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс частичного	Замена статуса требуемого	+	+	+	-	-	-	-

	обновления требуемого PodSecurityPolicy	объекта PodDisruptionBudget (Квота количества неработающих подов)							
	Интерфейс замены требуемого PodSecurityPolicy	Создать ресурс PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс удаления PodSecurityPolicy	Частичное обновление требуемого ресурса PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс удаления набора PodSecurityPolicy	Замена требуемого ресурса PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого PodSecurityPolicy	Удалить ресурс PodSecurityPolicy (Политика безопасности	+	+	+	-	-	-	-

		Пода)							
	Интерфейс просмотра или наблюдения PodSecurityPolicy	Удалить подборку ресурса PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в PodSecurityPolicy	Считывание требуемого ресурса PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке PodSecurityPolicy.	Список или просмотр объекта PodSecurityPolicy (Политика безопасности Пода)	+	+	+	-	-	-	-
	Интерфейс создания HorizontalPodAutoscale	Просмотр изменений в объекте PodSecurityPolicy (Политика безопасности Пода). Устарело:	+	+	+	-	-	-	-

		вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).							
	Интерфейс частичного обновления требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке PodSecurityPolicy (Политика безопасности Пода). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-

	Интерфейс замены требуемого HorizontalPodAutoscale	Создать HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления HorizontalPodAutoscale	Частичное обновление требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора HorizontalPodAutoscale	Замена требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого HorizontalPodAutoscale	Удалить HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-

	Интерфейс просмотра или наблюдения HorizontalPodAutoscale пространства имен	Удалить побдорку HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale	Считывание требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в HorizontalPodAutoscale	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке HorizontalPodA	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальн	+	+	+	-	-	-	-

	utoscale пространства имен	ое автомасштабирование подов)							
	Интерфейс просмотра отдельных изменений в списке HorizontalPodAutoscale	Просмотр изменений в объекте HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-

	Интерфейс частичного обновления статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс считывания статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте	+	+	+	-	-	-	-

		параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замены статуса требуемого HorizontalPodAutoscale	Частичное обновление статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс создания HorizontalPodAutoscale	Считывание статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс частичного обновления требуемого HorizontalPodAutoscale	Замена статуса требуемого HorizontalPodAutoscale (Горизонтальное	+	+	+	-	-	-	-

		автомасштабирование подов)							
	Интерфейс замены требуемого HorizontalPodAutoscale	Создать HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления HorizontalPodAutoscale	Частичное обновление требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора HorizontalPodAutoscale	Замена требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого HorizontalPodAutoscale	Удалить HorizontalPodAutoscale (Горизонтальное	+	+	+	-	-	-	-

		автомасштабирование подов)							
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale пространства имен	Удалить побдорку HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale	Считывание требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в объекте HorizontalPodAutoscale	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в	Список или просмотр объекта HorizontalPodA	+	+	+	-	-	-	-

	списке HorizontalPodAutoscale	utoscale (Горизонтальное автомасштабирование подов)							
	Интерфейс просмотра отдельных изменений в списке HorizontalPodAutoscale	Просмотр изменений в объекте HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-

	Интерфейс частичного обновление статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс считывания статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте	+	+	+	-	-	-	-

		параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замены статуса требуемого HorizontalPodAutoscale	Частичное обновление статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс создания HorizontalPodAutoscale	Считывание статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс частичного обновленя требуемого HorizontalPodAutoscale	Замена статуса требуемого HorizontalPodAutoscale (Горизонтальное	+	+	+	-	-	-	-

		автомасштабирование подов)							
	Интерфейс замены требуемого HorizontalPodAutoscale	Создать HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления HorizontalPodAutoscale	Частичное обновление требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора HorizontalPodAutoscale	Замена требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого HorizontalPodAutoscale	Удалить HorizontalPodAutoscale (Горизонтальное	+	+	+	-	-	-	-

		автомасштабирование подов)							
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale пространства имен	Удалить побдорку HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения HorizontalPodAutoscale	Считывание требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в объекте HorizontalPodAutoscale	Список или просмотр объекта HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в	Список или просмотр объекта HorizontalPodA	+	+	+	-	-	-	-

	списке HorizontalPodAutoscale	utoscale (Горизонтальное автомасштабирование подов)							
	Интерфейс просмотр отдельных изменений в списке HorizontalPodAutoscale	Просмотр изменений в объекте HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-

	Интерфейс частичного обновления статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-
	Интерфейс считывания статуса требуемого HorizontalPodAutoscale	Просмотр отдельных изменений в списке HorizontalPodAutoscale (Горизонтальное автомасштабирование подов). Устарело: вместо этого используйте	+	+	+	-	-	-	-

		параметр 'watch' (просмотр) в списке операций.							
	Интерфейс замеыа статуса требуемого HorizontalPodAutoscale	Частичное обновление статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс создания PodDisruptionBudget	Считывание статуса требуемого HorizontalPodAutoscale (Горизонтальное автомасштабирование подов)	+	+	+	-	-	-	-
	Интерфейс частичного обновления PodDisruptionBudget	Замена статуса требуемого HorizontalPodAutoscale (Горизонтальное	+	+	+	-	-	-	-

		автомасштабирование подов)							
	Интерфейс замены требуемого PodDisruptionBudget	Создать объект PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс удаления PodDisruptionBudget	Частичное обновление объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс удаления набора PodDisruptionBudget	Замена требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс считывания требуемого PodDisruptionBudget	Удалить объект PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-

	Интерфейс просмотра или наблюдения PodDisruptionBudget пространства имен	Удалить подборку объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра или наблюдения PodDisruptionBudget	Считывание требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра изменений в PodDisruptionBudget	Список или просмотр объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
	Интерфейс просмотра отдельных изменений в списке PodDisruptionBudget	Список или просмотр объекта PodDisruptionBudget (Квота количества неработающих	+	+	+	-	-	-	-

		подов)							
	Интерфейс просмотра отдельных изменений в списке PodDisruptionBudget	Просмотр изменений в объекте PodDisruptionBudget (Квота количества неработающих подов). Устарело: вместо этого следует использовать параметр 'watch' (просмотр) в списке операций, отфильтрованный до одного элемента с помощью параметра 'fieldselector' (выбор поля).	+	+	+	-	-	-	-
	Интерфейс частичного обновления статуса требуемого	Просмотр отдельных изменений в списке объекта PodDisruptionB	+	+	+	-	-	-	-

	PodDisruptionBudget	udget (Квота количества неработающих подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.							
	Интерфейс считывание статуса требуемого PodDisruptionBudget	Просмотр отдельных изменений в списке объекта PodDisruptionBudget (Квота количества неработающих подов). Устарело: вместо этого используйте параметр 'watch' (просмотр) в списке операций.	+	+	+	-	-	-	-

	Интерфейс замены статуса требуемого PodDisruptionBudget	Частичное обновление статуса требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	+	-	-	-	-
Deckhouse config webhook (Deckhouse)	Интерфейс проверки корректности параметров модуля	Считывание статуса требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	-	-	-	-	-
	Интерфейс проверки наличия прав на редактирование служебных объектов	Замена статуса требуемого объекта PodDisruptionBudget (Квота количества неработающих подов)	+	+	-	-	-	-	-
Runtime audit engine (runtim	Интерфейс регистрации события аудита	Получение метрик статистики работы kubelet	+	-	-	-	-	-	-

e audit engine)	Интерфейс регистрации нескольких событий аудита	Получение метрик статистики работы контейнеров на узле	+	+	-	-	-	-	-
gatekeeper-controller-manager (admission-policy-engine)	Интерфейс проверки объекта на соответствие политики безопасности.	Получение метрик статистики использования ЦПУ и памяти контейнерами на узле	+	-	-	-	-	-	-
	Интерфейс изменения объекта в соответствии с политикой безопасности.	Получение метрик статистики похождения проверок доступности (liveness и readiness - проверок)	+	+	-	-	-	-	-
grafana	searchResult	Просмотр логов	-	+	-	-	-	-	-
virtualization	Интерфейсы получения информации о состоянии объектов VirtualMachine, VirtualMachine	Просмотр информации о состоянии объектов виртулизации	+	+	+	+	+	+	+

	Snapshot,VirtualMachineRestore,ClusterVirtualImage,VirtualDisk,VirtualdiskSnapshot,VirtualImage,VirtualMachineBlockdeviceAttachment,VirtualMachineIPAddress,VirtualMachineClass,VirtualMachineOperation,VirtualMachine								
	Интерфейс создания объекта VirtualMachineOperation	Управление изменением состояния виртуальной машины	+	+	+	+	+	+	-
	Интерфейс удаления, изменения объекта VirtualMachineOperation	Управление изменением состояния виртуальной машины	+	+	+	+	+	-	-
	Интерфейс подключения к консоли	Подключение к консоли виртуальной	+	+	+	+	+	+	-

	виртуальной машины	машины							
	Интерфейс подключения к виртуальной машине по протоколу VNC	Подключение к виртуальной машине по протоколу VNC	+	+	+	+	+	+	-
	Интерфейс проброса портов виртуальной машины (port forward)	Проброс портов виртуальной машины (port forward)	+	+	+	+	+	+	-
	Интерфейс подключения тома к виртуальной машине	Подключение тома к виртуальной машине	+	+	+	+	+	+	-
	Интерфейс удаления тома виртуальной машины	Удаление тома виртуальной машины	+	+	+	+	+	+	-
	Интерфейсы удаления, изменения объектов VirtualMachine, VirtualMachine Snapshot,VirtualMachineRestor	Управление объектами VirtualMachine, VirtualMachine Snapshot,VirtualMachineRestore,VirtualDisk,VirtualdiskSnaps	+	+	+	+	+	-	-

	e,VirtualDisk,VirtualdiskSnapshot,VirtualMachineBlockdeviceAttachment,VirtualMachineIPAddress,VirtualMachineOperation	hot,VirtualMachineBlockdeviceAttachment,VirtualMachineIPAddress,VirtualMachineOperation							
	Интерфейс удаления, изменения объекта VirtualImage	Удаление и изменение образа диска виртуальной машины	+	+	+	+	-	-	-
	Интерфейс просмотра объекта VirtualIPAddressLease	Просмотр информации о выданных виртуальной машине адресах	+	+	+	+	-	-	-
	Интерфейс удаления, изменения объекта ClusterVirtualImage	Управление образ диска виртуальной машины, который может использоваться в качестве источника данных для новых ресурсов VirtualDisks	+	+	+	-	-	-	-

		или установочных образов							
	Интерфейс удаления, изменения объекта VirtualMachine Class	Управление требованиями к виртуальному CPU, размещению виртуальных машин на узлах и политикой определения размера ресурсов ВМ.	+	+	+	-	-	-	-
	Интерфейс удаления, изменения объекта VirtualMachine IPAddressLease	Изменение информации о выданных виртуальной машине адресах	+	+	-	-	-	-	-

Лист регистрации изменений

Номер изменения	Номера страниц				Всего страниц в документе	Наименование и номер документа, вводящего изменения	Подпись, Ф.И.О. внесшего изменения в данный экземпляр	Дата внесения изменения в данный экземпляр	Дата введения изменения
	измененных	замененных	новых	аннулированных					