Документация Deckhouse Development Platform

Дата генерации: 14.07.2026

Документация

Руководство администратора

Обзор

Данный документ представляет собой руководство администратора Deckhouse Development Platform (DDP) и является частью эксплуатационной документации продукта.

Установка

Deckhouse Development Platform (DDP) можно установить двумя способами: с внешними инстансами PostgreSQL и Redis (подключение к уже развёрнутым базам данных вне кластера) или с внутренними инстансами (развёртывание PostgreSQL и Redis внутри кластера). Внешние инстансы рекомендуются для production, внутренние подходят для тестов и пилотной эксплуатации.

Установка с внутренними инстансами

Для установки DDP включите модуль `development-platform` в вашем кластере Kubernetes под управлением Deckhouse Kubernetes Platform. Для этого можно использовать ModuleConfig с минимальным количеством настроек:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io # Email супер-администратора, который будет
иметь полный доступ к конфигурации платформы. Может быть изменен в любой момент.
    security:
      secretKey: "16charssecretkey" # Секретный ключ для шифрования приватных данных.
При изменении потребуется перегенерация токенов доступа к API платформы и повторное
заполнение учётных данных пользователями.
```

После установки веб-интерфейс DDP будет доступен по адресу `https://ddp.<ваш домен>`.

При развёртывании без указания секций `postgres` и `redis` платформа разворачивает внутренние инстансы PostgreSQL и Redis внутри кластера. Такой сценарий не рекомендуется для production и подходит только для тестов и пилотной эксплуатации; для промышленной эксплуатации используйте внешние ресурсы.

Настройка внутренних инстансов (опционально)

Если вы используете внутренние инстансы, можно явно указать `mode: internal` и задать образы из приватного Docker registry:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: internal
      image: registry.example.com/postgres:16.3  # Образ PostgreSQL из приватного
registry.
    redis:
      mode: internal
      image: registry.example.com/redis:7.4.0    # Образ Redis из приватного registry.
    additionalImagePullSecrets:
      - "custom-registry-secret"                 # (опционально) дополнительные
секреты для доступа к приватному registry.
```

Установка с внешними инстансами

Этот вариант установки рекомендуется для production: платформа подключается к уже развёрнутым PostgreSQL и Redis вне кластера, что обеспечивает отказоустойчивость и упрощает резервное копирование и масштабирование баз данных.

Подключение внешнего PostgreSQL

Для использования внешнего инстанса PostgreSQL необходимо указать параметры подключения в секции `postgres`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: external
      host: postgres.example.com  # Имя хоста или IP-адрес сервера PostgreSQL.
      port: 5432                  # Порт PostgreSQL (по умолчанию 5432).
      database: ddp               # Название базы данных.
      username: ddp_user          # Имя пользователя для подключения.
      password: secure_password   # Пароль для подключения.
```

Расширение pg_trgm

Платформе требуется расширение PostgreSQL `pg_trgm`. Если используется внешний PostgreSQL, включите его до запуска DDP — подключитесь к базе данных под пользователем с правами на создание расширений и выполните:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

При установке с внутренними инстансами расширение создаётся автоматически.

Подключение внешнего Redis

Для использования внешнего инстанса Redis необходимо указать параметры подключения в секции `redis`:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    redis:
      mode: external
      host: redis.example.com       # Имя хоста или IP-адрес сервера Redis.
      port: 6379                    # Порт Redis (по умолчанию 6379).
      database: "0"                 # Индекс базы данных Redis (по умолчанию "0").
      password: redis_password      # Пароль для подключения (необязательно; если
Redis без пароля — оставить пустым).
```

Полный пример с внешними инстансами

Пример конфигурации с подключением к внешним инстансам PostgreSQL и Redis:

```
apiVersion: deckhouse.io/v1alpha1
kind: ModuleConfig
metadata:
  name: development-platform
spec:
  enabled: true
  version: 1
  settings:
    rbac:
      superAdminEmail: admin@deckhouse.io
    security:
      secretKey: "16charssecretkey"
    postgres:
      mode: external
      host: postgres.production.example.com
      port: 5432
      database: ddp
      username: ddp_user
      password: secure_postgres_password
    redis:
      mode: external
      host: redis.production.example.com
      port: 6379
      database: "0"
      password: secure_redis_password
```

Архитектура

Верхнеуровневая архитектура

Цель и назначение системы

Deckhouse Development Platform (DDP) — это IDP (Internal Developer Platform), внутренняя платформа разработки, которая объединяет в себе инструменты, инфраструктурные сервисы и процессы разработки.

Платформа предоставляет единый слой взаимодействия с инфраструктурными и инженерными сервисами и используется для автоматизации операций, связанных с жизненным циклом продуктов. DDP обеспечивает единый доступ к инструментам и ресурсам, формализует и автоматизирует процессы разработки, а также упрощает подключение новых пользователей и команд.

Основные задачи платформы:

- Обеспечение единых стандартов разработки для команд через шаблоны и типовые конфигурации.

- Управление динамическими окружениями с автоматическим созданием, обновлением и удалением сред.
- Интеграция с внешними инфраструктурными сервисами через API.
- Упрощённый онбординг разработчиков и команд за счёт шаблонных решений и документации.
- Оцифровка процессов и сценариев разработки через BPMN-подобный механизм описания процессов.
- Встроенная CMDB для автоматического сбора данных из инфраструктурных сервисов и визуализации зависимостей.
- Контроль параметров сущностей через механизм проверок статуса.

Пользователи системы

Команды разработки

Команды разработки используют DDP для работы с сущностями и ресурсами платформы. Эти команды создают и управляют сущностями, запускают действия, просматривают данные из источников данных, используют виджеты и работают с процессами.

Менеджмент

Менеджмент получает статистику и данные для принятия решений. Пользователи управленческого уровня могут просматривать дашборды с метриками, анализировать эффективность разработки, отслеживать ключевые показатели и получать отчеты о работе команд.

Команды инфраструктуры и платформенные команды

Эти команды отвечают за предоставление сервисов и ресурсов через DDP. Они настраивают интеграции с внешними сервисами, конфигурируют ресурсы, создают шаблоны и автоматизации, управляют окружениями и обеспечивают работу платформы.

Команды безопасности

Команды безопасности используют DDP для предоставления и управления средствами защиты, проведения аудита и контроля соблюдения требований. В рамках платформы они настраивают политики безопасности и права доступа, просматривают аудит-логи и управляют механизмами защиты.

Администраторы

Администраторы — пользователи, обладающие полными правами на настройку и сопровождение платформы. Они управляют учётными записями пользователей, настраивают роли и права доступа через систему RBAC, конфигурируют интеграции с внешними сервисами, управляют учётными данными, ресурсами, автоматизациями и параметрами платформы, а также просматривают аудит-логи.

Права доступа для всех групп пользователей настраиваются через систему ролевого доступа (RBAC), которая обеспечивает гибкое управление разрешениями на уровне глобальных ролей, ролей ресурсов и ролей сущностей.

Доступ к системе

Веб-интерфейс

Все группы пользователей получают доступ к платформе через веб-интерфейс DDP Frontend, предназначенный для работы с платформой в веб-браузере.

Протокол доступа: HTTPS (TCP/443)

Аутентификация: Пользователи проходят аутентификацию через компонент Dex, который интегрируется с внешними провайдерами аутентификации (Keycloak, LDAP и др.) через стандартные протоколы (OIDC, LDAP). После успешной аутентификации пользователь получает JWT токен доступа, который используется для авторизации запросов к платформе.

REST API

Администраторы и разработчики могут напрямую обращаться к DDP Backend через REST API для программного взаимодействия с платформой.

Протокол доступа: HTTPS (TCP/443)

Аутентификация:

- Браузер: сессия по файлу cookie с флагом `HttpOnly` (JWT от Dex).
- Программный доступ: API-токен из раздела «Профиль», заголовок `Authorization: Bearer <api_token>`.

REST API предоставляет полный набор функций для управления всеми объектами платформы: сущностями, ресурсами, действиями, источниками данных, виджетами, процессами и другими компонентами.

Доступ снаружи/внутри периметра компании

DDP развёртывается в кластере Kubernetes и может быть доступна как внутри корпоративного периметра, так и снаружи в зависимости от конфигурации сетевой инфраструктуры.

Доступ внутри периметра

Типичный сценарий: Пользователи получают доступ к платформе через внутреннюю сеть компании.

- Веб-интерфейс и REST API доступны через внутренний Ingress-контроллер.
- Доступ осуществляется по внутренним доменным именам (например, `ddp.internal.company.com`).
- Аутентификация может быть интегрирована с корпоративным LDAP или Active Directory через Dex.

- Дополнительная защита может обеспечиваться на уровне сетевых политик Kubernetes.

Доступ снаружи периметра

Типичный сценарий: Пользователи получают доступ к платформе извне корпоративной сети.

- Веб-интерфейс и REST API доступны через публичный Ingress-контроллер.
- Доступ осуществляется по публичным доменным именам (например, `ddp.company.com`).
- Аутентификация может быть интегрирована с внешними провайдерами аутентификации через Dex.

Гибридный доступ

Платформа может быть настроена для поддержки обоих сценариев одновременно:

- Внутренние пользователи получают доступ через внутренний Ingress.
- Внешние пользователи (например, удаленные сотрудники) получают доступ через публичный Ingress или VPN.
- Различные группы пользователей могут использовать разные провайдеры аутентификации через Dex.

Конфигурация доступа настраивается администратором при развёртывании платформы через параметры Ingress-контроллера и сетевые политики Kubernetes.

Функциональная архитектура

Описание компонентов

В состав Deckhouse Development Platform (DDP) входят следующие компоненты.

DDP Frontend

DDP Frontend — это веб-интерфейс платформы, построенный на Vue.js 3 с TypeScript. Компонент предоставляет пользователям графический интерфейс для работы с платформой через веб-браузер.

Основные функции:

- Отображение пользовательского интерфейса для управления сервисами, окружениями и ресурсами.
- Обработка пользовательских действий и передача запросов в DDP Backend через REST API.

Технические характеристики:

- Принимает входящие соединения от пользователей.
- Взаимодействует с DDP Backend.

DDP Backend

DDP Backend — это серверная часть платформы, реализованная на языке Go. Компонент предоставляет REST API для управления всеми механизмами платформы.

Основные функции:

- Предоставление REST API для программного взаимодействия с платформой.
- Обработка бизнес-логики платформы.
- Реализация системы контроля доступа (RBAC) и управления правами пользователей.
- Интеграция с внешними инфраструктурными сервисами для выполнения различных операций.
- Управление данными через взаимодействие с PostgreSQL и Redis.

Технические характеристики:

- Принимает API-запросы от пользователей и DDP Frontend.
- Выполняет валидацию токенов доступа локально, используя кэшированные ключи подписи.
- Подключается к Redis для координации очереди задач.
- Подключается к PostgreSQL для хранения и управления данными.

DDP Worker

DDP Worker — это компонент для выполнения фоновых задач и асинхронных операций, реализованный на языке Go. Worker обрабатывает задачи из очередей Redis Streams и выполняет операции по запуску действий и синхронизации источников данных.

Основные функции:

- Синхронизация данных из внешних источников.
- Выполнение действий в асинхронном режиме.
- Выполнение проверок статуса сущностей.

Технические характеристики:

- Может быть развернут в нескольких экземплярах для обеспечения масштабируемости.
- Использует Redis Streams для получения задач из очередей.
- Подключается к PostgreSQL для чтения и обновления данных.
- Интегрируется с внешними инфраструктурными сервисами для выполнения операций.

Dex

Dex — это компонент системы аутентификации и авторизации, выполняющий роль identity provider. Dex обеспечивает единую точку входа для пользователей и интегрируется с различными внешними провайдерами аутентификации.

> ⚠ Dex не входит в состав модуля DDP и является частью Deckhouse Kubernetes Platform (DKP).

Основные функции:

- Управление процессом аутентификации пользователей.
- Выдача токенов доступа.
- Интеграция с внешними провайдерами аутентификации (Keycloak, LDAP и другими).
- Предоставление ключей подписи для проверки токенов.

Технические характеристики:

- Выписывает токены для аутентификации в DDP Backend.
- Периодически предоставляет ключи подписи токенов для проверки.
- Интегрируется с внешними провайдерами аутентификации через стандартные протоколы (OIDC, LDAP и др.).

Redis

Redis — это хранилище данных в памяти, используемое платформой для работы с очередями задач.

> Redis может быть установлен в составе модуля DDP для тестовых и демонстрационных целей. В промышленной эксплуатации рекомендуется использование выделенных инстансов Redis.

Основные функции:

- Реализация очередей задач через Redis Streams для асинхронной обработки.
- Поддержка распределенных блокировок для координации работы нескольких экземпляров Worker.
- Хранение событий, генерируемых при изменениях сущностей.

Технические характеристики:

- Используется для хранения очередей задач, которые обрабатываются DDP Worker.

PostgreSQL

PostgreSQL — это основная реляционная база данных платформы, используемая для хранения всей постоянной информации.

> PostgreSQL может быть установлен в составе модуля DDP для тестовых и демонстрационных целей. В промышленной эксплуатации рекомендуется использование выделенных инстансов PostgreSQL.

Описание сетевых взаимодействий

traffic-flows

Входящие соединения

- Пользователи подключаются к DDP Frontend по протоколу HTTPS (TCP/443) для работы с веб-интерфейсом.
- Пользователи могут напрямую обращаться к DDP Backend по протоколу HTTPS (TCP/443) для использования REST API.

Взаимодействие между компонентами платформы

- DDP Frontend → DDP Backend (TCP/8080, опционально mTLS): Frontend передает запросы пользователей в Backend для обработки.
- DDP Backend ↔ Dex (TCP/443): Backend получает ключи подписи от Dex для проверки токенов.
- DDP Backend ↔ Redis (TCP/6379): Backend использует Redis для координации очередей задач и хранения событий.
- DDP Backend ↔ PostgreSQL (TCP/5432): Backend выполняет операции чтения и записи постоянных данных.
- DDP Worker ↔ Redis (TCP/6379): Worker получает задачи из очередей Redis Streams и использует распределенные блокировки.
- DDP Worker ↔ PostgreSQL (TCP/5432): Worker обновляет данные в базе после выполнения задач.

Интеграция с внешними сервисами

- Dex ↔ Провайдеры аутентификации (Keycloak, LDAP и др.): Dex интегрируется с внешними системами управления пользователями.
- DDP Backend ↔ Инфраструктурные сервисы (TCP/443): Backend взаимодействует с внешними сервисами для получения информации.
- DDP Worker ↔ Инфраструктурные сервисы (TCP/443): Worker выполняет операции с внешними сервисами при синхронизации данных и выполнении действий.

Воркеры

Воркеры — это компоненты платформы, которые обрабатывают задачи из очереди в фоновом режиме. Они обеспечивают асинхронное выполнение различных операций, таких как:

- Синхронизация данных из источников данных.
- Запуск действий и процессов.
- Проверки статуса сущностей.

Воркеры работают независимо от основного приложения и позволяют масштабировать нагрузку на систему, перенося ресурсоёмкие операции на выделенные реплики.

Конфигурация

Воркеры настраиваются через конфигурацию платформы. Доступны следующие параметры:

Количество реплик

Параметр `workers.replicas` определяет количество одновременно работающих воркеров. Каждый воркер может обрабатывать задачи независимо от других.

Максимальное количество параллельных задач

Параметр `workers.maxTasks` определяет, сколько задач может обрабатываться параллельно каждым воркером.

> Общее количество задач, обрабатываемых одновременно всей системой, рассчитывается как `workers.replicas × workers.maxTasks`. Например, при 2 воркерах и 10 задачах на каждый воркер, система может обрабатывать до 20 задач одновременно.

Мониторинг

Для мониторинга работы воркеров и очереди задач доступен виджет «Очередь задач», который отображает:

- Размер очереди (общее количество задач).
- Количество ожидающих задач.
- Количество активных воркеров (консьюмеров).
- Детальную информацию о задачах в очереди.
- Статус каждой задачи (новая, в обработке).

Настройка через переменные окружения

Параметры воркеров также можно настроить через переменные окружения:

- `WORKER_MAX_TASKS` — максимальное количество параллельных задач на воркер (по умолчанию: 10).

Пример применения настроек:

```
export WORKER_MAX_TASKS=15
```

Приоритет настроек:

1. Значение из конфигурационного файла.
2. Значение из переменной окружения.
3. Значение по умолчанию.

Типы обрабатываемых задач

Воркеры обрабатывают следующие типы задач:

Задачи проверки статуса сущностей

Задачи проверки статуса выполняются воркерами, которые определяют состояние сущностей на основе настроенных правил. Такой подход позволяет:

- Снизить нагрузку на основной сервер.
- Обеспечить стабильность работы при большом количестве сущностей.
- Гарантировать выполнение проверок даже при высокой нагрузке на основное приложение.

Задачи синхронизации источников данных

Воркеры обрабатывают задачи синхронизации данных из внешних источников, что позволяет:

- Выполнять синхронизацию в фоновом режиме.
- Не блокировать интерфейс пользователя.
- Обрабатывать большие объемы данных.

Задачи выполнения действий

Действия, требующие длительного выполнения, обрабатываются воркерами, что обеспечивает:

- Асинхронное выполнение действий.
- Возможность отслеживания прогресса выполнения.
- Стабильность работы интерфейса.

Масштабирование

При необходимости увеличения производительности системы можно:

1. Увеличить количество реплик воркеров — это позволит обрабатывать больше задач одновременно
2. Увеличить количество параллельных задач на воркер — это повысит утилизацию каждого воркера

> При увеличении нагрузки на воркеры необходимо убедиться, что у кластера достаточно ресурсов (CPU, память) для обработки всех задач.

Устранение неполадок

Воркеры не обрабатывают задачи

Если задачи накапливаются в очереди и не обрабатываются:

1. Проверьте, что воркеры развернуты и работают (через виджет «Очередь задач»).
2. Убедитесь, что количество реплик воркеров больше 0.
3. Проверьте логи воркеров на наличие ошибок.
4. Убедитесь, что у воркеров достаточно ресурсов (CPU, память).

Медленная обработка задач

Если задачи обрабатываются медленно:

1. Увеличьте количество реплик воркеров.
2. Увеличьте количество параллельных задач на воркер (если позволяет ресурсная база).
3. Проверьте производительность внешних систем, с которыми работают воркеры.
4. Убедитесь, что нет узких мест в сети или базе данных.

Безопасность

Обзор

Deckhouse Development Platform (DDP) предоставляет набор встроенных механизмов безопасности, обеспечивающих контроль доступа, защиту учётных данных и аудит действий пользователей при работе с платформой.

Аутентификация

Аутентификация пользователей в DDP выполняется с использованием внешнего сервиса Dex, развернутого в составе Deckhouse Kubernetes Platform.

Dex обеспечивает интеграцию с внешними провайдерами аутентификации (Keycloak, LDAP и другими) через стандартные протоколы OIDC и LDAP. DDP Backend использует ключи подписи, получаемые от Dex, для проверки JWT-токенов и аутентификации запросов к API. Настройка выполняется на стороне DKP, дополнительная настройка в DDP не требуется.

Ролевая модель (RBAC)

В DDP реализована ролевая модель доступа, обеспечивающая разграничение прав пользователей и контроль операций, выполняемых в платформе.

Модель поддерживает иерархию ролей и проверку разрешений на уровне супер-администратора, глобальных ролей, ролей ресурсов и ролей сущностей. Подробнее — в документации по ролевой модели.

Имперсонация

DDP позволяет пользователю с разрешением `impersonate:users` временно работать в интерфейсе от имени другого пользователя для проверки прав доступа и диагностики. Сессия ограничена по времени; операции с учётными данными и секретами целевого пользователя в этом режиме недоступны.

Подробнее — в документации по имперсонации.

Учётные данные

DDP обеспечивает централизованное и безопасное хранение учётных данных, используемых для интеграции с внешними системами и сервисами. Хранение и обработка учётных данных выполняются на стороне DDP Backend с использованием базы данных PostgreSQL.

Перед сохранением в базе данных учетные данные шифруются с применением алгоритма AES-GCM. Расшифровка выполняется DDP Backend только в момент использования учетных данных в действиях, источниках данных или виджетах. Подробнее — в документации по учетным данным.

Ротация ключа шифрования

При смене ключа шифрования (`security.secretKey`) DDP позволяет перешифровать все связанные данные из веб-интерфейса, не теряя учётные данные пользователей, секреты Vault и другие зашифрованные значения. Операция доступна пользователям с разрешением `rotate:encryption-key` . Подробнее — в документации по ротации ключа шифрования.

Доверенные сертификаты

DDP позволяет добавлять в хранилище корневые и промежуточные сертификаты (CA) или сертификаты серверов. Они используются при проверке TLS в исходящих HTTPS-запросах к внешним сервисам (в действиях, виджетах, источниках данных и др.).

Управление доверенными сертификатами осуществляется через веб-интерфейс в разделе «Администрирование» → «Доверенные сертификаты». Подробнее — в документации по доверенным сертификатам.

Аудит-логи

Аудит-логи предназначены для регистрации действий пользователей и контроля операций, выполняемых через API платформы. Формирование аудит-логов выполняется на стороне DDP Backend.

Такие логи создаются автоматически для всех HTTP-запросов, за исключением GET. Каждая запись содержит информацию о пользователе, IP-адресе клиента, эндпоинте, HTTP-методе и статусе ответа.

Аудит-логи сохраняются в базе данных PostgreSQL и доступны для просмотра через веб-интерфейс платформы. Также возможна выгрузка аудит-логов в CSV из интерфейса платформы. Подробнее — в документации по аудит-логам.

HTTP-заголовки безопасности

Для защиты веб-интерфейса и API DDP используются стандартные HTTP-заголовки безопасности. Их формирование и применение выполняется на стороне DDP Frontend и DDP Backend. Заголовки направлены на снижение рисков XSS-атак, clickjacking, предотвращения MIME-sniffing, контроля передачи информации о реферере, а также на принудительное использование защищённого соединения HTTPS.

В платформе применяются следующие HTTP-заголовки безопасности:

- Content Security Policy (CSP).
- X-Frame-Options.
- X-Content-Type-Options.
- X-XSS-Protection.
- Referrer-Policy.
- Permissions-Policy.
- Strict-Transport-Security (HSTS).

Настройка осуществляется в конфигурации DDP в секции `security.headers`. Подробнее — в документации по HTTP-заголовкам безопасности.

Ролевая модель

Ролевая модель Deckhouse Development Platform (DDP) определяет, какие действия пользователи могут выполнять в платформе и к каким объектам они имеют доступ. Модель используется для разграничения прав при работе с API и веб-интерфейсом и применяется на уровне платформы в целом и отдельных объектов.

Ролевая модель реализована в DDP Backend и использует базу данных PostgreSQL для хранения ролей, разрешений и их связей.

Компоненты ролевой модели

Ролевая модель состоит из следующих компонентов:

- «Разрешение» — соответствует конкретному действию в DDP.
- «Роль» — объединяет в себе набор тех или иных разрешений.
- «Привязка роли» — связывает роль и пользователей, либо команды.

Для каждого объекта DDP существует свой набор разрешений, ролей и привязок ролей. Также есть глобальные роли, разрешения и привязки ролей, которые разрешают операции на глобальном уровне.

> Если у пользователя нет соответствующих разрешений, он не сможет выполнять никакие операции на платформе. Система проверяет разрешения для каждой операции.

Типы объектов и разрешения

Глобальные разрешения

Глобальные разрешения действуют на уровне всей платформы и предоставляют доступ ко всем объектам определенного типа:

Ресурсы:

- `create:resources` — создание ресурсов.
- `read:resources` — просмотр ресурсов.
- `update:resources` — редактирование ресурсов.
- `update:resources-order` — изменение порядка и группировки ресурсов в каталоге.
- `delete:resources` — удаление ресурсов.

Сущности:

- `create:entities` — создание сущностей.
- `read:entities` — просмотр сущностей.
- `update:entities` — редактирование сущностей.
- `delete:entities` — удаление сущностей.

Источники данных:

- `create:datasources` — создание источников данных.
- `read:datasources` — просмотр источников данных.
- `update:datasources` — редактирование источников данных.
- `sync:datasources` — синхронизация источников данных.
- `delete:datasources` — удаление источников данных.

Действия:

- `create:actions` — создание действий.
- `read:actions` — просмотр действий.
- `update:actions` — редактирование действий.
- `run:actions` — выполнение действий.
- `delete:actions` — удаление действий.

Автоматизации:

- `create:automations` — создание автоматизаций.
- `read:automations` — просмотр автоматизаций.
- `update:automations` — редактирование автоматизаций.
- `delete:automations` — удаление автоматизаций.

Сценарии:

- `create:workflows` — создание сценариев.
- `read:workflows` — просмотр сценариев.
- `update:workflows` — редактирование сценариев.
- `delete:workflows` — удаление сценариев.

Вебхуки:

- `create:webhooks` — создание вебхуков.

- `read:webhooks` — просмотр вебхуков.
- `update:webhooks` — редактирование вебхуков.
- `delete:webhooks` — удаление вебхуков.

Виджеты:

- `create:widgets` — создание виджетов.
- `read:widgets` — просмотр виджетов.
- `update:widgets` — редактирование виджетов.
- `run:widget-actions` — выполнение действий виджетов.
- `delete:widgets` — удаление виджетов.

Дашборды:

- `create:dashboards` — создание дашбордов.
- `read:dashboards` — просмотр дашбордов.
- `update:dashboards` — редактирование дашбордов.
- `delete:dashboards` — удаление дашбордов.

MCP:

- `read:mcp-servers` — просмотр подключённых MCP-серверов.
- `edit:mcp-servers` — подключение, редактирование, синхронизация каталога и удаление MCP-серверов.
- `read:mcp-collections` — просмотр MCP-коллекций.
- `edit:mcp-collections` — создание, редактирование и удаление MCP-коллекций.
- `read:mcp-tools` — просмотр пользовательских MCP-инструментов.
- `edit:mcp-tools` — создание, редактирование и удаление пользовательских MCP-инструментов.

Внешние сервисы:

- `create:external-services` — создание внешних сервисов.
- `read:external-services` — просмотр внешних сервисов.
- `update:external-services` — редактирование внешних сервисов.
- `delete:external-services` — удаление внешних сервисов.

Доверенные сертификаты:

- `create:trusted-certificates` — добавление доверенных сертификатов.
- `read:trusted-certificates` — просмотр доверенных сертификатов.
- `update:trusted-certificates` — редактирование доверенных сертификатов.
- `delete:trusted-certificates` — удаление доверенных сертификатов.

Процессы:

- `create:processes` — создание процессов.
- `read:processes` — просмотр процессов.

- `update:processes` — редактирование процессов.
- `delete:processes` — удаление процессов.

Команды:

- `create:teams` — создание команд.
- `update:teams` — редактирование команд.
- `delete:teams` — удаление команд.
- `update:team-variables` — редактирование переменных команд.
- `edit:team-filter-rules` — настройка правил фильтрации групп при синхронизации из Dex.

Иконки:

- `create:icons` — создание иконок.
- `delete:icons` — удаление иконок.

Типы учётных данных:

- `edit:user-access-credentials-types` — создание, редактирование и удаление типов учётных данных; настройка интеграции с Vault (просмотр и изменение конфигурации, проверка подключения).
- `rotate:encryption-key` — ротация ключа шифрования. Подробнее — в документации по ротации ключа шифрования.

Пользователи:

- `edit:users` — создание, блокировка, разблокировка и удаление пользователей.
- `impersonate:users` — запуск и завершение имперсонации пользователей. Подробнее — в документации по имперсонации.

> Разрешение `update:team-variables` позволяет редактировать переменные только тех команд, участником которых является пользователь. Даже супер-администратор не сможет изменить переменные команд, в которых не состоит.

Разрешения на просмотр страниц

Разрешения `view:` контролируют доступ к различным разделам интерфейса платформы:

- `view:admin-page` — доступ к разделу «Администрирование».
- `view:self-service-page` — доступ к разделу «Самообслуживание».
- `view:ai-page` — доступ к разделу «AI».

> Без соответствующих `view:` разрешений пользователь не сможет увидеть разделы в навигационном меню, даже если у него есть другие разрешения для работы с объектами в этих разделах.

Разрешения на уровне объектов

Для каждого типа объекта существуют разрешения, которые действуют только на конкретный объект:

Для ресурсов:

- `read:resource` — просмотр конкретного ресурса.
- `update:resource` — редактирование конкретного ресурса.
- `delete:resource` — удаление конкретного ресурса.
- `create:entities` — создание сущностей ресурса.
- `read:entities` — просмотр сущностей ресурса.
- `update:entities` — редактирование сущностей ресурса.
- `delete:entities` — удаление сущностей ресурса.
- `run:actions` — выполнение действий для сущностей ресурса.
- `control:processes` — управление процессами для сущностей ресурса.
- `edit:role-bindings` — редактирование привязок ролей для ресурса.

Для сущностей:

- `read:entity` — просмотр конкретной сущности.
- `update:entity` — редактирование конкретной сущности.
- `delete:entity` — удаление конкретной сущности.
- `run:actions` — выполнение действий для сущности.
- `control:processes` — управление процессами для сущности.
- `edit:role-bindings` — редактирование привязок ролей для сущности.

MCP-коллекции

Для MCP-коллекций:

- `use:mcp-collections` — вызов инструментов коллекции в AI-ассистенте и через MCP-сервер платформы.
- `edit:role-bindings` — редактирование привязок ролей для коллекции.

Для других объектов (действия, автоматизации, процессы, вебхуки, виджеты, дашборды, внешние сервисы и т. д.):

- `read:[object-type]` — просмотр объекта.
- `update:[object-type]` — редактирование объекта.
- `delete:[object-type]` — удаление объекта.
- `edit:role-bindings` — редактирование привязок ролей для объекта.

Иерархия проверки разрешений

Ролевая модель иерархическая, при обработке запроса DDP Backend проверяет права доступа в определенном порядке. Например для того, чтобы проверить, можно ли изменять пользователю конкретную сущность, будет выполняться следующая последовательность действий:

1. Супер-администратор

- Проверка, является ли пользователь супер-администратором. Если пользователь является супер-администратором, то дальнейшие проверки не производятся.

2. Глобальные роли

- Проверка по JWT-токену пользователя в каких группах он состоит
- Поиск глобальных ролей для пользователя и его групп по привязкам глобальных ролей.
- Проверка разрешений в глобальных ролях: если в одной из глобальных ролей, привязанных к пользователю, либо его команде, присутствует разрешение `update:entities`, то пользователь будет обладать правами на редактирование любых сущностей в DDP, дальнейшие проверки не производятся.

3. Роли ресурса

- Поиск ролей ресурса для пользователя и его групп по привязкам ролей конкретного ресурса.
- Проверка разрешений в ролях ресурса: если в одной из ролей ресурса, привязанных к пользователю, либо его команде, присутствует разрешение `update:entities`, то пользователь будет обладать правами на редактирование любых сущностей данного ресурса, дальнейшие проверки не производятся.

4. Роли сущностей

- Поиск ролей сущностей для пользователя и его групп по привязкам ролей конкретной сущности.
- Проверка разрешений в ролях сущности: если в одной из ролей сущности, привязанных к пользователю, либо его команде, присутствует разрешение `update:entity`, то пользователь будет обладать правами на редактирование данной сущности.

5. Владение объектом (при включенной опции ownerIsAdmin)

- Проверка, является ли пользователь владельцем объекта.
- Проверка, состоит ли пользователь в команде-владельце объекта.
- Если пользователь является владельцем объекта или участником команды-владельца, ему предоставляются все права администратора на объект.

Если ни на одном уровне не было найдено соответствующего разрешения, то действие будет запрещено.

> Проверка владения включается только если в конфигурации платформы включена опция `ownerIsAdmin`. Подробнее о владении объектами — в разделе «Владение объектами».

Владение объектами

Владение объектами — это механизм, который позволяет пользователям и командам становиться владельцами определённых объектов в системе и автоматически получать все права администратора на эти объекты.

Владелец объекта — это пользователь или команда, которая была назначена владельцем конкретного объекта.

Влияние владения на права доступа

При включённой опции `ownerIsAdmin` владелец объекта получает все права администратора для этого объекта. В этом случае владелец может:

- читать, изменять и удалять объект;
- управлять привязками ролей и доступом других пользователей;
- выполнять любые действия, доступные администратору объекта.

Опция OwnerAsAdmin

Опция `ownerIsAdmin` задаёт поведение прав доступа для владельцев объектов.

- «Включена (true)» — владельцы объектов получают полные права администратора на свои объекты.
- «Отключена (false)» — владельцы объектов не получают автоматических прав; доступ контролируется только через роли.

> Опция `ownerIsAdmin` должна быть настроена администратором платформы в конфигурационном файле. По умолчанию опция выключена.

Автоматическое назначение владельца при создании

При создании нового объекта (действия, источника данных, виджета, ресурса и других объектов) платформа автоматически устанавливает текущего пользователя в качестве владельца объекта. При необходимости можно переназначить владельца, либо создать объект без владельца.

Примеры использования владения

Владение ресурсом

Если пользователь создаёт ресурс, он автоматически становится его владельцем. При включенной опции `ownerIsAdmin` он получает все права администратора на этот ресурс, включая:

- Управление сущностями ресурса.
- Настройку привязок ролей.
- Полное редактирование и удаление ресурса.

Владение сущностью

Если пользователь создаёт сущность, он становится её владельцем и может:

- Просматривать и редактировать сущность.
- Управлять привязками ролей для сущности.
- Удалять сущность.

Команда-владелец

Если владельцем объекта назначена команда, то при включенной опции `ownerIsAdmin` все участники этой команды получают права администратора на объект.

Роль по умолчанию

Платформа позволяет назначить одну глобальную роль по умолчанию, разрешения к которой применяются ко всем авторизованным пользователям. Настройка выполняется в разделе «Администрирование → Управление доступом» с помощью переключателя в таблице «Роли».

Ролью по умолчанию может быть назначена только глобальная роль.

Команды

Команды используются для группировки пользователей и коллективного назначения ролей. Пользователь может состоять в нескольких командах, в этом случае его разрешения будут объединены из всех команд, в которых он состоит.

> Команды и их состав синхронизируются из внешней системы аутентификации (Dex). Управление командами через интерфейс DDP недоступно.

Настройка ролей и управление доступом

Создание роли

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Роли».
3. Нажмите кнопку «Создать роль».
4. Заполните форму:
    - Название — уникальное название роли.
    - Описание — описание назначения роли.
    - Тип объекта - выберите уровень действия роли:
        - `Глобальные` — для глобальных разрешений.
        - `Ресурсы` — для разрешений на уровне ресурсов.
        - `Сущности` — для разрешений на уровне сущностей.
        - `Действия` — для разрешений на уровне действий.
        - и другие типы объектов.

    - Разрешения - выберите необходимые разрешения из списка.

5. Нажмите «Сохранить».

Назначение роли пользователям

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Привязки ролей».
3. Нажмите кнопку «Создать привязку роли».
4. Заполните форму:
    - Название — название привязки роли.
    - Описание - описание назначения привязки.
    - Роль — выберите созданную роль.
    - Объект — выберите конкретный объект (если роль не глобальная).
    - Пользователи — добавьте пользователей, которым назначается роль.
    - Команды — добавьте команды, которым назначается роль.
5. Нажмите «Сохранить».

Настройка роли по умолчанию

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Роли».
3. Найдите глобальную роль, которую хотите сделать ролью по умолчанию.
4. Включите переключатель «Роль по умолчанию».
5. Подтвердите действие.

> Ролью по умолчанию может быть только глобальная роль.

Редактирование существующих ролей

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Роли».
3. Найдите нужную роль и нажмите «Редактировать».
4. Внесите необходимые изменения:
    - Измените название или описание.
    - Добавьте или удалите разрешения.
5. Нажмите «Сохранить».

Редактирование привязок ролей

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Привязки ролей».
3. Найдите нужную привязку и нажмите «Редактировать».
4. Внесите необходимые изменения:
    - Измените название или описание.

    - Добавьте или удалите пользователей.
    - Добавьте или удалите команды.

5. Нажмите «Сохранить».

Удаление ролей и привязок

1. Перейдите в соответствующий раздел.
2. Найдите нужную роль или привязку.
3. Нажмите кнопку «Удалить».
4. Подтвердите удаление.

> Удаление роли приведет к удалению всех связанных привязок ролей.

Просмотр разрешений пользователя или команды

Для пользователя

1. Перейдите в раздел «Администрирование -> Пользователи».
2. Откройте профиль нужного пользователя.
3. Перейдите на вкладку «Разрешения».
4. Просмотрите:
    - Глобальные разрешения.
    - Разрешения по объектам.
    - Роли, назначенные пользователю,
    - Команды, в которых состоит пользователь.

Для команды

1. Перейдите в раздел «Администрирование -> Команды».
2. Откройте профиль нужной команды.
3. Перейдите на вкладку «Разрешения».
4. Просмотрите:
    - Глобальные разрешения команды.
    - Разрешения по объектам.
    - Роли, назначенные команде.
    - Пользователи, состоящие в команде.

> Состав команды синхронизируется из внешней системы аутентификации (Dex).

Пресеты ролей

Платформа предоставляет готовые пресеты ролей для быстрой настройки типовых сценариев доступа:

Глобальные пресеты

Пресет «Admin»

- Тип: `глобальные`.
- Разрешения: все глобальные разрешения.
- Назначение: администраторы системы.

Пресет «Viewer»

- Тип: `глобальные`.
- Разрешения:
  - `read:actions`, `read:automations`, `read:dashboards`.
  - `read:datasources`, `read:entities`, `read:external-services`.
  - `read:processes`, `read:resource-relations`, `read:resources`.
  - `read:seeds`, `read:system-alerts`, `read:trusted-certificates`, `read:webhooks`.
  - `read:widgets`, `read:workflows`.
  - `read:audit-logs`, `view:admin-page`, `view:self-service-page`.
- Назначение: пользователи с правами только на просмотр.

Пресет «Developer»

- Тип: `глобальные`.
- Разрешения:
  - `read:actions`, `read:dashboards`, `read:external-services`.
  - `read:processes`, `read:widgets`, `read:workflows`.
  - `run:actions`, `run:widget-actions`, `control:processes`.
  - `update:team-variables`.
- Назначение: разработчики, которым нужен доступ к просмотру информации и выполнению действий, но не требуется доступ к созданию, изменению или удалению объектов. Разработчики видят только те сущности, к которым им предоставлен доступ через привязки ролей на уровне ресурсов или сущностей.

Пресет «Platform engineer»

- Тип: `глобальные`.
- Разрешения: полный доступ к каталогу и к странице «Самообслуживание».
- Назначение: инженеры, которые настраивают платформу (процессы, источники данных, дашборды и т.д.).

Пресеты для процессов

Пресет «Process admin»

- Тип: `процессы`.
- Разрешения: `delete:process`, `edit:role-bindings`, `read:process`, `update:process`.
- Назначение: администраторы процессов.

Пресет «Process viewer»

- Тип: `процессы`.
- Разрешения: `read:process`.
- Назначение: пользователи с правами на просмотр процессов.

Пресеты для ресурсов

Пресет «Resource admin»

- Тип: `ресурсы`.
- Разрешения: все разрешения для ресурсов.
- Назначение: администраторы конкретных ресурсов.

Использование пресетов

1. Перейдите в раздел «Администрирование -> Управление доступом».
2. Выберите вкладку «Роли».
3. Нажмите «Создать роль».
4. Выберите тип объекта.
5. В разделе «Пресеты» нажмите на нужный пресет.
6. Настройте параметры роли:
   - Измените название и описание при необходимости.
   - Добавьте или удалите разрешения.
7. Нажмите «Сохранить».

Примеры настройки ролей

Роль «Администратор системы»

- Тип: `глобальные`.
- Разрешения: все глобальные разрешения (используйте пресет «Admin»).
- Назначение: супер-администраторы.

Роль «Администратор ресурса»

- Тип: `ресурсы`.
- Разрешения: все разрешения для ресурсов (используйте пресет «Resource admin»).
- Назначение: конкретный ресурс, команда «Администраторы».

Роль «Оператор процессов»

- Тип: `процессы`.
- Разрешения: `delete:process`, `edit:role-bindings`, `read:process`, `update:process` (используйте пресет «Process admin»).
- Назначение: команда «Операторы процессов».

Практические рекомендации

Создание ролей

1. Определите уровень доступа: решите, нужна ли глобальная роль или роль для конкретного объекта.
2. Выберите необходимые разрешения: используйте принцип минимальных привилегий.
3. Создайте роль в соответствующем разделе администрирования.
4. Назначьте роль пользователям или командам через привязки ролей.

Управление доступом

1. Используйте команды для группового управления доступом.
2. Применяйте роли по умолчанию для базовых разрешений всех пользователей.
3. Используйте иерархию - глобальные роли для широких прав, роли объектов для специфических прав.
4. Регулярно пересматривайте назначенные роли и привязки.

Безопасность

1. Принцип минимальных привилегий: предоставляйте только необходимые разрешения.
2. Регулярный аудит: проверяйте назначенные роли и их использование.
3. Используйте команды вместо индивидуальных назначений для упрощения управления.
4. Документируйте роли - используйте описания для понимания назначения ролей.

Имперсонация

DDP поддерживает режим имперсонации: пользователь с глобальным разрешением `impersonate:users` может временно работать в веб-интерфейсе от имени другого пользователя.

> Имперсонация доступна только при входе через браузер (сессия Dex). Запуск через API-токены не поддерживается.

Запуск и завершение

1. Перейдите в раздел «Администрирование» → «Пользователи».
2. В строке нужного пользователя нажмите «Войти как этот пользователь».
3. Платформа переключит сессию на выбранного пользователя; внизу экрана появится баннер «Вы вошли как: …» с таймером до автоматического завершения.
4. Чтобы завершить имперсонацию досрочно, нажмите «Завершить сессию» в баннере.

Кнопка «Войти как этот пользователь» недоступна для заблокированных пользователей и для вашей собственной учётной записи.

Ограничения безопасности при запуске

Имперсонация не будет запущена, если:

- выбранный пользователь заблокирован;
- выбран тот же пользователь, под которым уже выполнен вход;
- выбран супер-администратор, а оператор не является супер-администратором;
- выбран пользователь, у которого уже есть глобальное разрешение `impersonate:users` (исключение: оператор — супер-администратор).

Срок действия сессии

Сессия имперсонации имеет ограниченное время жизни:

- длительность сессии: 30 минут;
- по истечении времени сессия завершается автоматически;
- в интерфейсе отображается таймер обратного отсчёта до автоматического завершения.

Если пользователь не завершил имперсонацию вручную, платформа завершит её автоматически после истечения срока действия.

Аудит

Для HTTP-запросов, попадающих в аудит-логи (POST, PUT, DELETE, PATCH), в поле «Email» указывается оператор и целевой пользователь в формате `оператор@example.com [as целевой@example.com]`.

Это позволяет в записи аудит-лога определить:

- кто инициировал запрос;
- от чьего имени выполнялась операция в сессии;
- HTTP-метод, путь и статус ответа.

Запросы GET в аудит-логи не записываются. Подробнее — в документации по аудит-логам.

Учётные данные

Механизм учётных данных позволяет безопасно хранить и использовать реквизиты доступа к внешним сервисам в действиях, источниках данных, виджетах и других компонентах Deckhouse Development Platform (DDP). Все учётные данные шифруются перед сохранением в базе данных и расшифровываются только при использовании.

Принцип работы

Типы учётных данных

Администратор платформы создаёт типы учётных данных в разделе «Администрирование» → «Учётные данные». Каждый тип имеет:

- «Название» — отображаемое название типа.
- «Идентификатор» — уникальный идентификатор для использования в конфигурациях.
- «Описание» — подсказка для пользователей.

Учётные данные пользователей

Пользователи заполняют свои персональные учётные данные в разделе «Профиль» → «Учётные данные». Для каждого типа учётных данных пользователь может указать свое значение (например, токен доступа, пароль, ключ API).

После сохранения пользователь не может просмотреть значение учётных данных, только обновить его. В профиле отображается информация о том, заполнены ли те или иные учётные данные.

Выбор хранилища для типа учётных данных

При создании или редактировании типа учётных данных указывается хранилище:

- «База данных» — значения хранятся в PostgreSQL (по умолчанию).
- Vault — значения хранятся в HashiCorp Vault или в Deckhouse Stronghold. Доступно только если в платформе включена и настроена интеграция с Vault.

Один и тот же тип учётных данных всегда использует одно выбранное хранилище. Изменить хранилище для существующего типа можно; при этом уже сохранённые учётные данные в старом хранилище не переносятся автоматически.

Хранение в базе данных

Учётные данные с типом хранилища «База данных» сохраняются в PostgreSQL. Перед сохранением значения шифруются с использованием алгоритма AES-GCM.

Параметры шифрования:

- Алгоритм: AES-GCM.
- Размер ключа: 16, 24 или 32 байта (128, 192 или 256 бит).
- Размер nonce: 12 байт.
- Ключ шифрования: задается в конфигурации DDP в поле `security.secretKey`.

Зашифрованные значения сохраняются в базе данных с префиксом `GCM:`, что позволяет системе определить метод шифрования при расшифровке.

⚠ При изменении ключа шифрования (`security.secretKey`) в конфигурации без предварительного перешифрования все текущие учётные данные станут недоступны. Используйте ротацию ключа шифрования в веб-интерфейсе.

Хранение в HashiCorp Vault

Если для типа учётных данных выбрано хранилище Vault, то реквизиты пользователей сохраняются платформой в HashiCorp Vault или в Deckhouse Stronghold.

Как это устроено:

- В Vault используется движок KV Secrets Engine v2. Путь к секретам задаётся в настройках Vault в DDP (например, `secrets/data/ddp`).
- Для каждого пользователя создаётся отдельный секрет по пути `{path}/credentials/{uuid_типа}/users/{uuid_пользователя}` . Внутри секрета один ключ `value` , значение — зашифрованные учётные данные. Шифрование выполняется ключом DDP (`security.secretKey`).
- Подключение к Vault выполняется по `AppRole` (AppRole Role ID и AppRole Secret ID). AppRole Role ID и AppRole Secret ID сохраняются в БД DDP в зашифрованном виде. Токен Vault продлевается автоматически; при сбое продления выполняется повторный вход по AppRole.

Настройка Vault в DDP:

1. В разделе «Администрирование» → «Учётные данные» нажмите «Настройки Vault».
2. Включите использование Vault (Включено).
3. Заполните параметры:
   - `URL` — адрес сервера Vault (например, `https://vault.example.com`).
   - «AppRole Role ID» — идентификатор AppRole, полученный при создании роли в Vault.
   - «AppRole Secret ID» — секрет AppRole, используется вместе с Role ID для аутентификации в Vault.
   - «Путь» — путь к секретам в KV v2. Для KV v2 путь должен содержать сегмент `/data/` (например, `secrets/data/ddp` или `secrets/data/ddp/credentials`). Mount (например, `secrets`) должен быть заранее создан в Vault; подкаталоги после `data/` создаются платформой автоматически при сохранении учётных данных.
4. Нажмите «Проверить подключение», чтобы убедиться, что DDP может аутентифицироваться и обращаться к Vault.
5. Сохраните конфигурацию.

После сохранения конфигурации при создании или редактировании типа учётных данных в списке хранилищ будет доступен вариант Vault. Если Vault выключен или не настроен, типы с хранилищем Vault останутся в списке, но учётные данные такого типа заполнять и использовать будет нельзя до включения и настройки Vault.

Права доступа

Для просмотра, изменения конфигурации Vault и проверки подключения требуется глобальное разрешение на редактирование типов учётных данных: `edit:user-access-credentials-types`.

Особенности конфигурации

- Поддерживается только KV Secrets Engine v2. Путь в настройках должен включать `/data/` (например, `mount/data/подкаталог`).
- Одна конфигурация Vault на инстанс DDP: один URL, одна AppRole, один базовый путь. Все типы учётных данных с хранилищем Vault используют этот путь (подкаталоги формируются автоматически по UUID типа).
- Очистить все сохранённые значения типа из текущего хранилища (база данных или Vault) в любой момент можно через действие «Очистить хранилище» в таблице типов учётных данных.
- Смена ключа шифрования DDP (`security.secretKey`) без предварительного перешифрования делает нечитаемыми и учётные данные в Vault, так как значения в Vault хранятся в зашифрованном виде. Используйте ротацию ключа шифрования.

Использование учётных данных

Учётные данные расшифровываются только при необходимости их использования, непосредственно перед обращением к внешнему сервису. После использования расшифрованные значения не сохраняются в памяти дольше необходимого.

Интеграция в компоненты платформы

Действия

В конфигурации действия можно указать список требуемых учётных данных через поле «Учётные данные». Каждое учётное данное определяется:

- «Идентификатор» — идентификатор учётного данного (соответствует идентификатору типа учётных данных).
- «Тип учётных данных» — выбор типа учётных данных из списка.

При выполнении действия система:

1. Определяет пользователя, от имени которого выполняется действие (инициатор или учётная запись, выбранная в поле «Учётная запись для запуска», если включена опция «Выбрать учётную запись для запуска»).
2. Извлекает учётные данные этого пользователя из базы данных.
3. Расшифровывает значения учётных данных.
4. Подставляет расшифрованные значения в шаблоны конфигурации действия (URL, заголовки, тело запроса) через плейсхолдеры вида `{{.credentials.идентификатор}}`.

Источники данных

В конфигурации источника данных можно указать список требуемых учётных данных аналогично действиям. При синхронизации данных система:

1. Использует учётные данные системного пользователя, указанного в конфигурации источника данных (поле «Системная учётная запись»).
2. Извлекает и расшифровывает учётные данные системного пользователя.
3. Подставляет значения в шаблоны запросов источника данных.

> ⚠ Источники данных всегда используют учётные данные системного пользователя, а не инициатора запуска синхронизации.

Виджеты

Виджеты могут использовать учётные данные для получения данных из внешних сервисов. Механизм работы аналогичен действиям:

1. Определяется пользователь (инициатор запроса или учётная запись, выбранная в поле «Учётная запись для виджета», если включена опция «Выбрать учётную запись для виджета»).
2. Извлекаются и расшифровываются учётные данные.
3. Значения подставляются в конфигурацию виджета.

Проверки статуса

Проверки статуса используют учётные данные для выполнения проверок состояния сущностей. В конфигурации правил проверки статуса можно указать список требуемых учётных данных. При выполнении проверки система:

1. Использует учётные данные системного пользователя, указанного в конфигурации правила проверки статуса (поле «Системная учётная запись»).
2. Извлекает и расшифровывает учётные данные системного пользователя.
3. Подставляет значения в шаблоны конфигурации правила (URL, заголовки, параметры запросов) через плейсхолдеры вида `{{.credentials.идентификатор}}`.

> ⚠ Проверки статуса всегда используют учётные данные системного пользователя, указанного в конфигурации правила. Если системный пользователь не указан, правило пропускается при выполнении проверки.

Внешние сервисы

Внешние сервисы могут иметь собственный список учётных данных. Если действие, источник данных или виджет использует внешний сервис (включена опция «Использовать внешний сервис»), то:

1. Учётные данные из конфигурации внешнего сервиса наследуются компонентом, если в его конфигурации не указаны собственные учётные данные.
2. Приоритет имеют учётные данные, указанные непосредственно в конфигурации компонента.

Настройка

Ключ шифрования

Ключ шифрования настраивается в конфигурации DDP в секции `security.secretKey`:

```
security:
  secretKey: "your-32-byte-secret-key-here"
```

Требования к ключу:

- Длина: 16, 24 или 32 байта (128, 192 или 256 бит).
- Только печатаемые ASCII символы.
- Не должен быть слабым или часто используемым ключом.
- Не должен состоять из одного повторяющегося символа.

Интеграция с Vault

Подключение к HashiCorp Vault (URL, AppRole, путь к KV v2) настраивается в веб-интерфейсе в разделе «Администрирование» → «Учётные данные» → «Настройки Vault». В конфигурационном файле DDP задаётся только ключ шифрования (`security.secretKey`), который используется в том числе для шифрования значений, хранящихся в Vault.

Безопасность

- Учётные данные никогда не передаются пользователю в расшифрованном виде.
- Расшифровка происходит только непосредственно перед отправкой запроса.
- Расшифрованные значения не логируются.

Ротация ключа шифрования

Deckhouse Data Platform (DDP) позволяет безопасно сменить ключ шифрования (`security.secretKey`) без потери учётных данных и других зашифрованных значений. Операция выполняется из веб-интерфейса и перешифровывает все данные, которые хранятся с использованием текущего ключа.

Основания для ротации ключа

Выполните ротацию ключа шифрования, если необходимо заменить `security.secretKey` в конфигурации DDP Backend — например, по политике безопасности или после компрометации ключа.

> ⚠ Не меняйте `security.secretKey` в конфигурации и не перезапускайте бэкенд до завершения перешифрования в веб-интерфейсе, иначе сохранённые значения станут недоступны.

Права доступа

Для запуска ротации требуется глобальное разрешение `rotate:encryption-key`.

Перешифровываемые данные

Операция затрагивает следующие категории данных:

- «Учётные данные пользователей» — значения в PostgreSQL для типов с хранилищем «База данных».
- «Учётные данные AI-провайдеров» — персональные реквизиты для AI-интеграций.
- «Временные ответы действий» — зашифрованные временные ответы в записях выполнения действий.
- «Секреты Vault» — значения учётных данных, хранящихся в HashiCorp Vault или Deckhouse Stronghold.
- «Конфигурация Vault» — AppRole Role ID и AppRole Secret ID в настройках интеграции с Vault.

Порядок выполнения

1. Перейдите в раздел «Администрирование» → «Учётные данные».
2. Нажмите «Ротация ключа шифрования».
3. В поле «Старый ключ шифрования» введите текущий ключ из конфигурации DDP (`security.secretKey`).
4. В поле «Новый ключ шифрования» введите новый ключ. Требования к ключу те же, что при первоначальной настройке:
   - длина — 16, 24 или 32 байта;
   - только печатаемые ASCII-символы;
   - ключ не должен состоять из одного повторяющегося символа.
5. Нажмите «Перешифровать».
6. Дождитесь завершения операции и проверьте таблицу результатов. Для каждой категории отображается количество успешно перешифрованных, пропущенных и неуспешных записей.
7. Обновите `security.secretKey` в конфигурации DDP Backend на новый ключ.
8. Перезапустите DDP Backend.

> После перешифрования учётные данные остаются недоступны для использования, пока в конфигурации бэкенда не указан новый ключ и сервис не перезапущен.

Результаты операции

В таблице результатов для каждой категории данных указываются три счётчика:

- «Успешно» — записи перешифрованы новым ключом.
- «Пропущено» — запись уже доступна с новым ключом (например, была перешифрована ранее).
- «Неуспешно» — запись не удалось перешифровать (чаще всего из-за неверного старого ключа или повреждённых данных).

Строка «Итого» содержит суммарные значения по всем категориям.

Если в столбце «Неуспешно» есть ненулевые значения, не обновляйте конфигурацию бэкенда до устранения причины ошибок. Проверьте корректность старого ключа и повторите операцию.

Аудит-логи

Аудит-логи — это механизм записи всех операций, выполняемых пользователями через API Deckhouse Development Platform (DDP). Логи сохраняются в базе данных PostgreSQL и используются для аудита действий пользователей.

Компоненты

В формировании и хранении аудит-логов участвуют следующие компоненты:

- DDP Backend — формирует записи аудит-логов.
- PostgreSQL — обеспечивает хранение данных.

Содержимое аудит-логов

Для каждого HTTP-запроса (кроме GET) создаётся запись аудит-лога, содержащая:

- «Email» — адрес электронной почты пользователя, выполнившего запрос.
- «IP» — IP-адрес клиента.
- «Тело запроса» — тело HTTP-запроса.
- «Путь» — путь API, к которому был выполнён запрос.
- «Метод» — HTTP-метод запроса (POST, PUT, DELETE, PATCH).
- «Статус» — HTTP-код ответа.
- «Дата» — дата выполнения запроса.

Принцип работы

Аудит-логи формируются автоматически в DDP Backend с использованием промежуточного слоя обработки запросов (middleware) для всех HTTP-запросов, кроме GET.

Для снижения нагрузки на базу данных используется буферизация записей со следующими параметрами:

- Размер батча — 40 записей.
- Интервал записи — 20 секунд.
- Размер буфера в памяти — 4 записи.

Хранение

Аудит-логи хранятся в таблице `audit_logs` базы данных PostgreSQL. Таблица партиционирована на основе поля `timestamp` с разделением по дням.

Структура таблицы поддерживается автоматически: ежедневно в 00:00 по серверному времени подготавливается пространство для записи логов на следующие 7 дней.

Логи хранятся неограниченно долго. Автоматическая очистка старых записей не выполняется. Для управления временем хранения необходимо настроить автоматическое удаление старых групп логов через внешние инструменты (например, cron-задачи или скрипты очистки базы данных).

Просмотр логов

Аудит-логи доступны через веб-интерфейс в разделе «Администрирование» → «Аудит». Логи можно фильтровать по следующим параметрам:

- Период (дата и время начала и окончания);
- Email пользователя;
- IP-адрес;
- Путь;
- Метод запроса;
- Статус ответа.

Выгрузка в CSV

На странице просмотра аудит-логов доступна кнопка «Скачать .csv». Выгрузка выполняется только при наличии разрешения на чтение аудит-логов.

В файл попадают все записи, соответствующие текущим фильтрам и сортировке. Если число подходящих событий превышает лимит в 100 000 записей, выгрузка не выполняется.

Настройка

Механизм аудит-логирования включён по умолчанию и не требует дополнительной настройки. Для управления временем хранения логов необходимо настроить автоматическое удаление старых партиций базы данных.

HTTP-заголовки безопасности

Deckhouse Development Platform (DDP) поддерживает следующие HTTP-заголовки для повышения безопасности взаимодействия:

1. `Content-Security-Policy`
    - Настраиваемая политика безопасности контента.
    - Подробное описание приведено в разделе Content Security Policy.
2. `X-Frame-Options`
    - Защита от clickjacking-атак.
    - Значение по умолчанию: `SAMEORIGIN`.
    - Настраивается в конфигурации DDP в секции `security.headers.xFrameOptions`.
    - Возможные значения: `DENY`, `SAMEORIGIN`.
3. `X-Content-Type-Options`
    - Предотвращение MIME-sniffing.
    - Значение: `nosniff`.
4. `X-XSS-Protection`
    - Защита от XSS-атак (для устаревших браузеров).
    - Значение: `1; mode=block`.
5. `Referrer-Policy`
    - Контроль передачи информации о реферере.
    - Значение: `strict-origin-when-cross-origin`.
6. `Permissions-Policy` (ранее Feature-Policy)
    - Ограничение использования функций браузера.
    - Значение: `geolocation=(), microphone=(), camera=(), payment=(), usb=(), magnetometer=(), gyroscope=(), accelerometer=()`.
7. `Strict-Transport-Security (HSTS)`
    - Принудительное использование HTTPS.
    - Значение: `max-age=31536000; includeSubDomains; preload`.

Конфигурация

Все заголовки настраиваются в конфигурации DDP в секции `security.headers`:

```
security:
  headers:
    # Включить/выключить все заголовки безопасности.
    enabled: true
    csp:
      # Разрешить загрузку изображений из внешних источников.
      allowExternalImages: false
      # Разрешить использование iframe виджетов.
      allowIframe: false
      # Дополнительные источники, добавляемые к директивам CSP: img-src, connect-src,
frame-src, frame-ancestors. Например: https://example.com.
      additionalSources: ""
    # Значение заголовка X-Frame-Options (DENY или SAMEORIGIN).
    xFrameOptions: "SAMEORIGIN"
```

Опция enabled определяет включение или отключение всех заголовков безопасности. При значении true заголовки добавляются к HTTP-ответам, при значении false — заголовки не добавляются.

Применение

- Frontend: Заголовки добавляются на уровне веб-сервера для раздачи статических файлов и HTML страниц в компоненте DDP Frontend.
- Backend API: Заголовки добавляются через middleware для всех API ответов в компоненте DDP Backend.

> Все заголовки безопасности включены по умолчанию (`enabled: true`). При необходимости их можно отключить или настроить в конфигурации DDP.

Content Security Policy

Content Security Policy (CSP) — это механизм безопасности, который помогает предотвратить XSS атаки и другие инъекции кода, ограничивая источники, из которых могут загружаться ресурсы (скрипты, стили, изображения и т.д.).

CSP по умолчанию

```
default-src 'self';
script-src 'self';
style-src 'self' 'unsafe-inline';
font-src 'self';
img-src 'self' data:;
connect-src 'self' wss: ws:;
frame-src 'self';
frame-ancestors 'self';
worker-src 'self' blob:;
object-src 'none';
base-uri 'self';
form-action 'self';
upgrade-insecure-requests
```

Отключение или ослабление политик может потребоваться в следующих случаях:

- Использование iframe виджетов (требует настройки `allowIframe: true`).
- Добавление иконок из внешних источников для объектов DDP (требует настройки `allowExternalImages: true`).

Примеры настройки

Автоматическое обновление сервисов

Инструкция описывает настройку автоматического механизма обновления всех сервисов, созданных из шаблона, при выпуске новой версии этого шаблона. Новая версия шаблона определяется по новому тегу, добавленному в Git-репозиторий шаблона.

Компоненты системы и их назначение

Для реализации автоматического обновления сервисов используются следующие компоненты Deckhouse Development Platform (DDP):

- Ресурсы — это элементы модели данных каталога DDP, которые определяют структуру хранения информации. Каждый ресурс может содержать параметры для хранения метаданных сущностей, а также связи с другими ресурсами для отображения зависимостей.
    - В рамках инструкции используются два ресурса (названия могут отличаться, в зависимости от ваших настроек):
        - «Шаблоны» — хранит информацию о шаблонных репозиториях (URL и ID репозитория, последний тег версии).

        - «Сервисы» — хранит информацию о созданных сервисах (URL и ID репозитория, переменные шаблонизации, ветка по умолчанию).

    > Названия могут отличаться, в зависимости от ваших настроек.

- Связь:
    - Устанавливает зависимость между ресурсами и обеспечивают возможность автоматического нахождения всех сервисов, связанных с конкретным шаблоном.
- Источник данных — механизм DDP, который позволяет синхронизировать информацию из внешних инфраструктурных систем (например, GitLab) в каталог платформы. Источники данных периодически опрашивают внешние системы, получают актуальную информацию и обновляют соответствующие сущности в каталоге:
    - Синхронизирует информацию о шаблонных репозиториях из GitLab.
    - Обновляет список тегов для каждого шаблона.
    - Автоматически обновляет параметр «Последний тег» в сущностях ресурса «Шаблоны».
- Действие — механизм DDP, который позволяет пользователям платформы взаимодействовать с внешними инфраструктурными сервисами через API. Действия могут создавать проекты в GitLab, создавать секреты в системах управления секретами, развёртывать приложения и выполнять другие операции.
    - В данной настройке используются три действия (названия могут отличаться, в зависимости от ваших настроек):
        - создаёт новый проект в GitLab для размещения сервиса.
        - клонирует код из шаблонного репозитория в новый репозиторий сервиса с применением переменных шаблонизации.
        - создаёт Merge Request в репозитории сервиса с изменениями из новой версии шаблона.

    > Названия могут отличаться, в зависимости от ваших настроек.

- Процесс — механизм автоматизации сложных бизнес-процессов, который позволяет создавать визуальные схемы выполнения действий с поддержкой последовательного или параллельного выполнения. Процессы объединяют несколько действий в единую цепочку выполнения.
    - В данной настройке процесс объединяет два действия:
        - Последовательно запускает создание проекта в GitLab, а затем создание репозитория из шаблона.
        - Обеспечивает интерфейс для пользователя при создании сервиса из шаблона.

- Автоматизация — механизм реагирования платформы на изменения спецификации сущностей. Автоматизации подписываются на события обновления сущностей ресурсов по определённым правилам (триггерам) и запускают выполнение действий, процессов или синхронизацию источников данных.
    - Отслеживает изменения параметра «Последний тег» в сущностях ресурса «Шаблоны».
    - При обнаружении нового тега автоматически запускает действие «Обновление сервиса из шаблона» (название зависит от ваших настроек) для всех связанных сервисов.
    - Создаёт Merge Request в каждом репозитории сервиса с изменениями из новой версии шаблона.

workflow

Последовательность работы:

1. «Источник данных» периодически синхронизирует информацию о шаблонах из GitLab и обновляет параметр с идентификатором `repository_last_tag` (может отличаться в зависимости от ваших настроек) в сущностях ресурса «Шаблоны».
2. «Автоматизация» отслеживает изменения параметра с идентификатором `repository_last_tag` (может отличаться в зависимости от ваших настроек) через триггеры.
3. При обнаружении нового тега автоматизация запускает действие «Обновление сервиса из шаблона» (название может отличаться в зависимости от ваших настроек) для всех сервисов, связанных с обновлённым шаблоном.
4. «Действие» создаёт Merge Request в репозитории каждого сервиса, включающий изменения из новой версии шаблона.

Таким образом, при добавлении нового тега в шаблонный репозиторий все связанные сервисы автоматически получают предложение обновиться через Merge Request.

Настройка компонентов

В следующих разделах описывается конфигурация каждого компонента системы. При настройке учитывайте, что названия ресурсов и параметров могут отличаться от приведённых в инструкции. В этом случае необходимо обеспечить согласованность названий между всеми компонентами: если вы используете другие названия параметров в ресурсах, их нужно использовать во всех действиях, автоматизациях и источниках данных, которые ссылаются на эти параметры.

Будет описано:

1. Требуемые ресурсы в DDP и их параметры.
2. Требуемые связи между ресурсами.
3. Требуемые источники данных для получения информации о новых версиях шаблонов.
4. Требуемые действия для создания сервиса из шаблона.
5. Требуемый процесс для реализации последовательного запуска цепочки действий.

6. Требуемая автоматизация для реализации следующей функциональности:
    - автоматическая проверка DDP наличия новой версии шаблона;
    - автоматическое создание в сервисах, созданных из шаблона, Merge Request с изменениями из новой версии.

Создание ресурсов и связи

«Ресурсы» — это элементы модели данных каталога DDP, которые определяют структуру хранения информации. Каждый ресурс может содержать параметры для хранения метаданных сущностей, а также связи с другими ресурсами для отображения зависимостей.

Ресурс «Шаблоны»

У вас должен быть создан ресурс для хранения информации о шаблонных репозиториях. В рамках данной инструкции он назван «Шаблоны», однако вы можете использовать любое удобное для вас название. Если ресурса ещё нет, создайте его.

Ресурс должен содержать следующие параметры для хранения данных о шаблонных репозиториях:

- Параметр «URL» типа URL — используется для хранения URL-адреса репозитория шаблона в GitLab. Этот параметр необходим для действий, которые работают с репозиторием шаблона.
- Параметр «ID репозитория» типа Number — хранит внутренний ID репозитория в GitLab. Используется для действий, которые требуют указания ID проекта GitLab.
- Параметр «Последний тег» типа String — содержит название последнего тега (версии) шаблона. Этот параметр используется автоматизацией для отслеживания появления новых версий шаблона. При каждом обновлении этого параметра источником данных автоматизация проверяет изменение и запускает обновление связанных сервисов.

> ⚠ Убедитесь, что идентификаторы параметров корректно указаны во всех действиях, автоматизациях и источниках данных, которые ссылаются на эти параметры.

Если параметры отсутствуют, откройте ресурс на редактирование и добавьте их на вкладке «Параметры» ресурса, нажав кнопку «Добавить параметр».

Ресурс «Сервисы»

У вас должен быть создан второй ресурс для хранения информации о созданных сервисах. В рамках инструкции он называется «Сервисы», но вы можете выбрать любое удобное для вас название. Если ресурса ещё нет, создайте его.

Ресурс должен содержать следующие параметры для хранения метаданных о созданных сервисах:

- Параметр «URL» типа URL — ссылка на репозиторий сервиса в GitLab. Используется действиями для работы с репозиторием сервиса.
- Параметр «ID репозитория» типа Number — внутренний ID репозитория в GitLab. Необходим для действий, которые требуют указания ID проекта GitLab.
- Параметр «Переменные использованные при шаблонизации» типа YAML — содержит переменные, которые были применены при клонировании кода из шаблона в сервис. Эти переменные необходимы при обновлении сервиса из шаблона, так как они должны быть применены повторно для корректного рендеринга новых изменений из шаблона.
- Параметр «Ветка по умолчанию» типа String — название основной ветки репозитория (например, `main` или `master`). Используется действием обновления сервиса для определения целевой ветки Merge Request.

> ⚠ Убедитесь, что идентификаторы параметров корректно указаны во всех действиях, автоматизациях и источниках данных, которые ссылаются на эти параметры.

Если параметры отсутствуют, откройте ресурс на редактирование и добавьте их на вкладке «Параметры» ресурса, нажав кнопку «Добавить параметр».

Создание ресурса

Для создания ресурсов перейдите в раздел «Каталог». Создайте новый ресурс. Для этого в нижней части сайдбара каталога нажмите кнопку «Создать ресурс».

Откроется окно по созданию нового ресурса с различными вкладками.

Основная информация:

Заполните следующие поля на вкладке «Основная информация»:

- «Название»: [Укажите название ресурса].
- «Владелец»: [Укажите владельца].
- «Команда владелец»: [Укажите команду владелец].

Параметры:

Перейдите на вкладку «Параметры» и добавьте требуемые параметры ресурса. Для добавления нового параметра нажмите кнопку «Добавить параметр».

Пример требуемых параметров:

Шаблоны:

Название	Идентификатор	Описание	Тип
URL	web_url	Ссылка на репозиторий сервиса	URL
ID репозитория	project_id	Внутренний ID репозитория в GitLab	Number
Последний тег	repository_last_tag	Последний созданный тег в git репозитории	String

Сервисы:

Название	Идентификатор	Описание	Тип
URL	web_url	Ссылка на репозиторий сервиса	URL
ID репозитория	project_id	Внутренний ID репозитория в GitLab	Number
Переменные использованные при шаблонизации	template_values	Переменные, которые были применены при клонировании из шаблона	YAML
Ветка по умолчанию	default_branch	Название основной ветки репозитория	String

Связь

У вас должна быть настроена связь, которая отражает зависимость между шаблонами и созданными из них сервисами. Если связи ещё нет, создайте её.

Связь должна быть настроена следующим образом:

- «Родительский ресурс»: выбран ресурс, хранящий информацию о шаблонах.
- «Дочерний ресурс»: выбран ресурс, хранящий информацию о сервисах.

Создание связи

Для создания связи нажмите кнопку «Создать связь» в нижней части сайдбара раздела «Каталог».

Откроется окно по созданию новой связи. Заполните данные согласно таблице ниже:

Параметр	Значение
Название	[Введите название связи, например `Шаблоны-Сервисы`]
Родительский ресурс	[Выберите название ресурса с шаблонами]
Дочерний ресурс	[Выберите название ресурса с сервисами]

Источник данных

У вас должен быть настроен источник данных, который автоматически получает информацию об актуальном состоянии шаблонов из GitLab. Если источника данных ещё нет, создайте его.

Проверьте следующие настройки:

Основная информация:

- «Ресурс»: выбран ресурс «Шаблоны» (или соответствующий ресурс, если он назван иначе).

Бэкенд:

На вкладке «Бэкенд» проверьте подключение к GitLab, а именно что происходит автоматическая синхронизация:

- «Периодическая синхронизация»: должна быть включена, для автоматического обновления информации о шаблонах при их изменении в GitLab.
- «Периодичность запуска»: должно быть указано расписание в формате CRON (например, `1 * * * *` для запуска каждый час). Расписание определяет, как часто источник данных будет проверять наличие новых тегов в репозиториях шаблонов.

Параметры бэкенда:

Должны быть заданы следующие ключи:

Ключ	Значение	Описание
group_ids	[ID группы с шаблонами]	Указано ID группы GitLab, в которой находятся шаблонные репозитории
tags	true	Включено получение информации о тегах для каждого репозитория

Подробнее о доступных параметрах — в разделе «GitlabProjects».

Правила:

Правила обновления:

Настройте правила обновления параметров ресурса «Шаблоны» данными, полученными из GitLab:

- Параметр «URL»: необходимо хранить URL репозитория шаблона в GitLab. В качестве источника должен использоваться `{{ .web_url }}`.
- Параметр «ID репозитория»: необходимо хранить внутренний ID репозитория в GitLab. В качестве источника должен использоваться `{{ .id }}`.

- Параметр «Последний тег»: необходимо хранить последний тег git-репозитория (в рамках данной инструкции используется как версия шаблона). Например, можно использовать источник `{{ ( index ( .ddp_repository_tags ) 0 ).name }}` — данное выражение получает последний тег, созданный в git-репозитории, беря первый элемент из массива тегов. При необходимости способ получения тега можно настроить иначе. Этот параметр критически важен для работы автоматизации, так как изменение его значения является триггером для обновления сервисов.

Идентификаторы источников берутся из спецификации Projects API GitLab.

Создание источника данных

Для создания источников данных перейдите в раздел «Самообслуживание» в пункт «Источники данных». В правом верхнем углу нажмите кнопку «Создать».

Откроется окно по созданию нового источника данных с различными вкладками.

Основная информация:

- «Название»: укажите название источника данных (например, `Шаблоны сервисов`).
- «Ресурс»: выберите ресурс, который содержит информацию о шаблонах.
- «Владелец»: назначьте владельца источника данных.
- «Команда владелец»: укажите команду, ответственную за источник данных.

Бэкенд:

На вкладке «Бэкенд» настройте подключение к GitLab:

- «Бэкенд»: выберите встроенный тип источника данных `GitlabProjects` для работы с проектами GitLab.
- «Периодическая синхронизация»: включите автоматический запуск синхронизации, для автоматического обновления информации о шаблонах при их изменении в GitLab.
- «Периодичность запуска»: укажите расписание в формате CRON (например, `1 * * * *` для запуска каждый час). Расписание определяет, как часто источник данных будет проверять наличие новых тегов в репозиториях шаблонов.
- «URL»: укажите URL вашего экземпляра GitLab (например, `https://gitlab.example.com`).

Параметры бэкенда:

Настройте параметры запроса к GitLab API:

Ключ	Значение	Описание
group_ids	[ID группы с шаблонами]	Укажите ID группы GitLab, в которой находятся шаблонные репозитории
tags	true	Включает получение информации о тегах для каждого репозитория

Подробнее о доступных параметрах — в разделе «GitlabProjects».

Правила:

На вкладке «Правила» настройте логику создания и обновления сущностей:

- «Создание новых сущностей»: включите создание новых сущностей. DDP автоматически будет создавать записи о новых шаблонах, если шаблонный репозиторий обнаружен в GitLab, но ещё не существует в каталоге.
- «Владелец создаваемых сущностей»: назначьте владельца для новых сущностей.
- «Команда владелец создаваемых сущностей»: укажите команду-владельца для новых сущностей.

Правила создания:

Настройте правила для создания идентификатора и названия новой сущности в платформе:

- «Идентификатор»: идентификатор создаваемой сущности. Например, можно использовать выражение `{{ replaceChar .path_with_namespace "/" "-" }}` — оно преобразует путь с неймспейсом, заменяя слеши на дефисы для получения уникального ID.
- «Название»: название создаваемой сущности. Например, можно использовать выражение `{{ .name }}` — оно использует название проекта из GitLab.

При необходимости вы можете задать свои собственные правила на основе Go template.

Правила сопоставления:

Настройте правила сопоставления уже существующих сущностей при синхронизации:

- «Идентификатор»: идентификатор сущности, которую необходимо обновить. Например, можно использовать то же выражение `{{ replaceChar .path_with_namespace "/" "-" }}`, что и в правилах создания.
- «Название»: название сущности, которую необходимо обновить. Например, можно использовать то же выражение `{{ .name }}`, что и в правилах создания.

При необходимости вы можете задать свои собственные правила на основе Go template. Если не планируется вручную создавать сущности ресурса, рекомендуется использовать одинаковые правила для сопоставления и создания.

Правила обновления:

Настройте правила обновления параметров ресурса «Шаблоны» данными, полученными из GitLab:

- Параметр «URL»: в качестве значения используется URL репозитория шаблона в GitLab. В качестве источника используйте `{{ .web_url }}`.
- Параметр «ID репозитория»: в качестве значения используется внутренний ID репозитория в GitLab. В качестве источника используйте `{{ .id }}`.
- Параметр «Последний тег»: в качестве значения используется тег git-репозитория. Например, можно использовать источник `{{ ( index ( .ddp_repository_tags ) 0 ).name }}` — данное выражение получает последний тег, созданный в git-репозитории, беря первый элемент из массива тегов. При необходимости способ получения тега можно настроить иначе. Этот параметр критически важен для работы автоматизации, так как изменение его значения является триггером для обновления сервисов.

Идентификаторы источников берутся из спецификации Projects API GitLab.

Авторизация:

На вкладке «Авторизация» настройте заголовки и учётные данные, необходимые для подключения к GitLab:

Настройте учётные данные:

- «Идентификатор»: `token`.
- «Реквизиты»: укажите реквизиты, которые содержат токен GitLab с необходимыми правами доступа к репозиториям. Токен должен иметь права на чтение проектов и информации о тегах в указанной группе GitLab.

Так же необходимо настроить заголовок:

- «Ключ»: `Private-Token`.
- «Значение»: `{{ .credentials.token }}` — используется значение, которое было настроено для учётных данных. Если в качестве идентификатора учётных данных используется не `token`, необходимо заменить `token` на заданное вами.

Подробнее о доступных заголовках — в разделе «GitlabProjects» и «Системная учётная запись».

Создание действий

Необходимо настроить три действия, которые будут отвечать за создание сервиса из шаблона и их обновление в случае выпуска новых версий шаблона.

Действие «Создать проект в GitLab»

У вас должно быть создано действие, которое отвечает за инициализацию нового проекта в GitLab. Оно запускается первым при создании нового сервиса и создаёт проект в GitLab, в который будет помещён сервис, созданный из шаблона. Если действия ещё нет, создайте его.

Проверьте, что действие содержит следующие настройки:

Основная информация:

- «Ресурс»: выбран ресурс, который содержит информацию о сервисах.

Обновление:

На вкладке «Обновление» активирована опция «Обновление параметров сущности» и настроены правила обновления, которые записывают ID, URL и ветку по умолчанию только что созданного проекта в сущность, для которой запускалось действие.

> Ниже указаны примеры названий параметров сущности. В поле `параметр сущности` необходимо указывать идентификатор соответствующего параметра ресурса, который хранит информацию о сервисах.

- Параметр «ID репозитория» используется источник `{{ .response.id }}` — сохраняет ID созданного проекта из ответа GitLab API в параметр сущности.
- Параметр «URL»: используется источник `{{ .response.web_url }}` — сохраняет URL проекта в GitLab созданного проекта в параметр сущности.
- Параметр «Ветка по умолчанию»: используется источник `{{ .response.default_branch }}` — сохраняет название ветки по умолчанию в параметр сущности.

> Идентификаторы источников (`id` , `web_url` , `default_branch`) берутся из спецификации Projects API GitLab.

Действие «Создать репозиторий из шаблона»

У вас должно быть создано действие для клонирования кода из шаблонного репозитория в новый репозиторий сервиса с применением переменных шаблонизации. Это действие будет запускаться после создания проекта в GitLab и заполнять его кодом из шаблона. Если действия ещё нет, создайте его.

Проверьте, что действие содержит следующие настройки:

Основная информация:

- «Ресурс»: выбран ресурс, который содержит информацию о сервисах (действие запускается для сущности сервиса, который нужно обновить).

Пользовательская форма:

Проверьте наличие следующих полей в пользовательской форме:

- Параметр «Идентификатор шаблонного репозитория» с идентификатором `template_ddp_slug` (или другим, если это необходимо) типа Entities — будет автоматически подставляться идентификатор сущности шаблона внутри DDP, из которого будет клонироваться код. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Идентификатор». Это позволяет выбрать нужный шаблон из связанных сущностей. Данный параметр можно сделать скрытым.

Обновление:

На вкладке «Обновление» активирована опция «Обновление параметров сущности» и настроено сохранение переменных, использованных при шаблонизации, обратно в сущность, для которой запускалось действие.

> Ниже указан пример названия параметра сущности. В поле `параметр сущности` необходимо указывать идентификатор соответствующего параметра ресурса, который хранит информацию о сервисах.

- Параметр «Переменные использованные при шаблонизации»: используйте источник `{{ toYAML .response.values.merged }}` — данное выражение сохраняет переменные в формате YAML. Эти переменные критически важны для будущего обновления сервиса из шаблона, так как они должны быть применены повторно. Данные получены в качестве ответа на успешное выполнение действия.

Активирована опция «Создание связей сущности», чтобы действие автоматически устанавливало связь между созданным сервисом и его шаблоном:

- «Ресурс»: выбран ресурс «Сервисы» (или соответствующий ресурс, если он назван иначе).
- «Связь»: выбрана связь «Шаблоны-Сервисы» (или название вашей связи, если оно отличается).
- «Идентификатор родительской сущности»: используйте `{{ .property.template_ddp_slug }}` — идентификатор шаблона из параметра действия.

  > Если идентификатор параметра не `template_ddp_slug`, замените на корректный.

- «Идентификатор дочерней сущности»: оставьте пустым — будет использована текущая сущность, для которой запускается действие.

Действие «Обновление сервиса из шаблона»

У вас должно быть создано действие, которое будет использоваться в автоматизации для создания нового Merge Request в репозиторий сервиса. Этот MR будет содержать изменения из новой версии (нового тега) связанного шаблона. Если действия ещё нет, создайте его.

Проверьте, что действие содержит следующие настройки:

Основная информация:

- «Ресурс»: выбран ресурс, который содержит информацию о сервисах.

Пользовательская форма:

Проверьте наличие следующих полей в пользовательской форме:

- Параметр «Переменные используемые при шаблонизации» с идентификатором `template_values` (или другим, если это необходимо) типа String — он хранит переменные, которые были использованы при изначальной шаблонизации (хранятся в параметре сущности). Значение по умолчанию: `{{ .entity.properties.template_values }}` — использует сохранённые переменные из параметра сущности. Важно: если параметр называется иначе (например, `template_vars`), используйте соответствующий идентификатор.
- Параметр «Версия» с идентификатором `version` (или другим, если это необходимо) типа Entities — получает значение нового тега (версии) из сущности шаблона. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс, если он назван иначе), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Параметр», Параметр: «Последний тег» (или аналогичный).
- Параметр «Обновляемый репозиторий» с идентификатором `target_project_id` (или другим, если это необходимо) типа String — это ID репозитория сервиса в GitLab, в который создаётся MR. Значение по умолчанию: `{{ .entity.properties.project_id }}` — использует ID проекта из параметра сущности. Если идентификатор (`project_id`) параметра, который хранит внутренний ID проекта в GitLab в ресурсе, называется иначе, используйте соответствующий идентификатор.
- Параметр «Шаблонный репозиторий» с идентификатором `source_project_id` (или другим, если это необходимо) типа Entities — это ID репозитория шаблона в GitLab. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс, если он назван иначе), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Параметр», Параметр: «ID репозитория» (или аналогичный).
- Параметр «Ветка обновляемого репозитория» с идентификатором `target_repo_branch` (или другим, если это необходимо) типа String — это основная ветка репозитория сервиса (например, `master` или `main`), куда будет направлен MR. При необходимости можете указать любую другую. Значение по умолчанию: `{{ .entity.properties.default_branch }}`

— использует ветку по умолчанию из параметра сущности. Если идентификатор параметра (`default_branch`) называется иначе, используйте соответствующий идентификатор.

Конфигурация запроса:

Тело запроса:

> В качестве идентификаторов используются те, которые предлагались выше. Если были заданы другие, необходимо также изменить их ниже в теле запроса.

Тело запроса должно быть настроено примерно таким образом:

```
merge_request_spec:
  source_branch: update-to-{{ .version }}
  target_branch: {{ .target_repo_branch }}
  title: update-to-{{ .version }}
source_project_tag: {{ .version }}
source_project_id: "{{ .source_project_id }}"
target_project_id: "{{ .target_project_id }}"
values: {{ .template_values | nindent 4 }}
```

В теле запроса:

- `source_branch` — название ветки, содержащей изменения. В качестве примера используется ветка с названием, включающим версию тега.
- `target_branch` — целевая ветка сервиса, куда будет направлен MR.
- `title` — заголовок MR с указанием версии.
- `source_project_tag` — тег версии шаблона, из которого берутся изменения.
- `source_project_id` и `target_project_id` — ID репозиториев шаблона и сервиса.
- `values` — переменные шаблонизации применяются повторно для корректного рендеринга изменений.

Подробнее о настройках встроенного бэкенда — в разделе «CreateGitlabMergeRequest».

Создание действия

Для создания действия перейдите в раздел «Самообслуживание», пункт «Действия». Создайте новое действие. Для этого нажмите справа вверху кнопку «Создать». Откроется окно по созданию нового действия с различными вкладками.

Вкладка «Основная информация»:

Заполните следующие поля на вкладке «Основная информация»:

- «Название»: [Укажите название действия].
- «Ресурс»: выберите ресурс «Сервисы» (или соответствующий ресурс, если он назван иначе).

- «Владелец»: [Укажите владельца].
- «Команда владелец»: [Укажите команду владелец].

Пользовательская форма:

На вкладке «Пользовательская форма» настройте поля для заполнения при запуске действия. Параметры, которые необходимо настроить, зависят от действия, которое вы создаёте.

Действие «Создать проект в GitLab»:

- Параметр «Название» с идентификатором `name` (или иным при необходимости) типа String — это название нового проекта. Значение по умолчанию: `{{ .entity.name }}` — использует название сущности в DDP.
- Параметр «Путь» с идентификатором `path` (или иным при необходимости) типа String — это URL-путь до репозитория в GitLab. Значение по умолчанию: `{{ .entity.slug }}` — использует идентификатор сущности в DDP.
- Параметр «Группа репозиториев» с идентификатором `group` (или иным при необходимости) типа String — ID группы GitLab, где будет создан сервис. Значение по умолчанию: укажите ID группы в GitLab, в которой необходимо создать проект.

Действие «Создать репозиторий из шаблона»:

- Параметр «Идентификатор шаблонного репозитория» с идентификатором `template_ddp_slug` (или иным при необходимости) типа Entities — это идентификатор сущности шаблона внутри DDP, из которого будет клонироваться код. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Идентификатор». Это позволяет выбрать нужный шаблон из связанных сущностей.
- Параметр «URL сервисного репозитория» с идентификатором `service_git_url` (или иным при необходимости) типа URL — это URL репозитория, в который будет помещён отрендеренный шаблон. Значение по умолчанию: `{{ .entity.properties.web_url }}` — использует URL сущности сервиса из каталога. Если параметр URL в ресурсе называется иначе, используйте соответствующий идентификатор (например, `{{ .entity.properties.service_url }}`).
- Параметр «URL шаблонного репозитория» с идентификатором `template_git_url` (или иным при необходимости) типа Entities — это URL репозитория шаблона, который используется бэкендом для клонирования. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Параметр», Параметр: «URL» (или иной аналогичный, в зависимости от ваших настроек).

> Дополнительные параметры зависят от вашего шаблона и должны быть настроены в соответствии с переменными, используемыми в шаблоне.

Действие «Обновление сервиса из шаблона»:

Необходимо настроить поля для сбора данных, которые нужны бэкенду для создания Merge Request. Эти данные будут автоматически заполняться из сущностей ресурса «Сервисы» или связанной с ним сущности из ресурса «Шаблоны»:

- Параметр «Переменные используемые при шаблонизации» с идентификатором `template_values` (или иным при необходимости) типа String — это переменные, которые были использованы при изначальной шаблонизации (хранятся в параметре сущности). Значение по умолчанию: `{{ .entity.properties.template_values }}` — использует сохранённые переменные из параметра сущности. Важно: если параметр называется иначе (например, `template_vars`), используйте соответствующий идентификатор.
- Параметр «Версия» с идентификатором `version` (или иным при необходимости) типа Entities — получает значение нового тега (версии) из сущности шаблона. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Параметр», Параметр: «Последний тег» (или аналогичный).
- Параметр «Обновляемый репозиторий» с идентификатором `target_project_id` (или иным при необходимости) типа String — это внутренний ID репозитория сервиса в GitLab, в который создаётся MR. Значение по умолчанию: `{{ .entity.properties.project_id }}` — использует ID проекта из параметра сущности. Если параметр ID в ресурсе называется иначе, используйте соответствующий идентификатор.
- Параметр «Шаблонный репозиторий» с идентификатором `source_project_id` (или иным при необходимости) типа Entities — это внутренний ID репозитория шаблона в GitLab. В дополнительных настройках укажите: Ресурс: «Шаблоны» (или соответствующий ресурс), Связь: «Шаблоны-Сервисы» (или название вашей связи), Поле: «Параметр», Параметр: «ID репозитория» (или аналогичный).
- Параметр «Ветка обновляемого репозитория» с идентификатором `target_repo_branch` (или иным при необходимости) типа String — основная ветка репозитория сервиса (например, `master` или `main`), куда будет направлен MR. Значение по умолчанию: `{{ .entity.properties.default_branch }}` — использует ветку по умолчанию из параметра сущности. Если идентификатор параметра называется иначе, используйте свой.

Конфигурация запроса:

> В качестве идентификаторов используются те, которые предлагались выше. Если были заданы другие, необходимо также изменить их ниже в теле запроса.

На вкладке «Конфигурация запроса» настройте логику выполнения действия. Логика зависит от настраиваемого действия.

Действие «Создать проект в GitLab»:

- «Тип действия»: выберите встроенный тип действия (значение `BuiltIn`).
- «Встроенный тип действия»: выберите `CreateGitlabProject` для создания проекта в GitLab.
- «URL»: укажите URL вашего экземпляра GitLab (например, `https://gitlab.example.com`).

Тело запроса:

```
initialize_with_readme: false
name: '{{ .name }}'
namespace_id: '{{ .group }}'
path: '{{ .path }}'
```

Это тело запроса передаёт в GitLab API название проекта, ID группы и путь репозитория. Подробнее о настройках — в разделе «CreateGitlabProject».

Действие «Создать репозиторий из шаблона»:

- «Тип действия»: выберите встроенный тип действия (значение `BuiltIn`).
- «Встроенный тип действия»: выберите `CreateRepositoryFromTemplate` для создания репозитория из шаблона.
- «URL»: укажите URL вашего экземпляра GitLab (например, `https://gitlab.example.com`).

Тело запроса:

```
sourceBranch: "master"
templateRepositoryUrl: "{{ .template_git_url }}"
targetRepositoryUrl: "{{ .service_git_url }}"
values:
  зависит от шаблона
```

В теле запроса:

- `sourceBranch` — ветка в репозитории шаблона, из которой будет клонироваться код (обычно `master` или `main`).
- `templateRepositoryUrl` — URL шаблонного репозитория из параметра действия.
- `targetRepositoryUrl` — URL целевого репозитория сервиса.
- `values` — блок должен содержать пары ключ-значение, которые будут использованы для замены переменных в файлах шаблона. Формат и состав этих переменных полностью зависит от того, как настроен ваш шаблон.

Подробнее о настройках — в разделе «CreateRepositoryFromTemplate».

Действие «Обновление сервиса из шаблона»:

- «Тип действия»: выберите встроенный тип действия (значение `BuiltIn`).
- «Встроенный тип действия»: выберите `CreateGitlabMergeRequest` для создания Merge Request в GitLab.
- «URL»: укажите URL вашего экземпляра GitLab (например, `https://gitlab.example.com`).

Тело запроса:

```
merge_request_spec:
  source_branch: update-to-{{ .version }}
  target_branch: {{ .target_repo_branch }}
  title: update-to-{{ .version }}
source_project_tag: {{ .version }}
source_project_id: "{{ .source_project_id }}"
target_project_id: "{{ .target_project_id }}"
values: {{ .template_values | nindent 4 }}
```

В теле запроса:

- `source_branch` — создаётся новая ветка с названием, включающим версию тега.
- `target_branch` — целевая ветка сервиса, куда будет направлен MR.
- `title` — заголовок MR с указанием версии.
- `source_project_tag` — тег версии шаблона, из которого берутся изменения.
- `source_project_id` и `target_project_id` — ID репозиториев шаблона и сервиса.
- `values` — переменные шаблонизации применяются повторно для корректного рендеринга изменений.

Подробнее о настройках — в разделе «CreateGitlabMergeRequest».

Обновление:

Обновление параметров сущности:

Для действий «Создать проект в GitLab» и «Создать репозиторий из шаблона» необходимо активировать опцию «Обновление параметров сущности» и настроить правила обновления, которые запишут некоторую информацию (зависит от действия) в сущность, для которой запускалось действие. Дополнительно необходимо настроить правила обновления сущности. Правила обновления сущности зависят от настраиваемого действия.

Для действия «Создать проект в GitLab» используйте идентификаторы соответствующих параметров ресурса:

- Параметр «ID репозитория»: используйте источник `{{ .response.id }}` — сохраняет ID созданного проекта из ответа GitLab API.
- Параметр «URL»: используйте источник `{{ .response.web_url }}` — сохраняет URL созданного проекта.
- Параметр «Ветка по умолчанию»: используйте источник `{{ .response.default_branch }}` — сохраняет название ветки по умолчанию.

Идентификаторы источников (`id` , `.web_url` , `.default_branch`) берутся из спецификации Projects API GitLab.

Для действия «Создать репозиторий из шаблона» используйте идентификатор соответствующего параметра ресурса:

- Параметр «Переменные использованные при шаблонизации»: используйте источник `{{ toYAML .response.values.merged }}` — данное выражение сохраняет переменные в формате YAML. Эти переменные критически важны для будущего обновления сервиса из шаблона, так как они должны быть применены повторно. Данные получены в качестве ответа на успешное выполнение действия.

Создание связей сущности:

Для действия «Создать репозиторий из шаблона» также должна быть активирована опция «Создание связей сущности», чтобы действие автоматически устанавливало связь между созданным сервисом и его шаблоном:

- «Ресурс»: выберите «Сервисы» (или соответствующий ресурс, если он назван иначе).
- «Связь»: выберите связь «Шаблоны-Сервисы» (или название вашей связи, если оно отличается).
- «Идентификатор родительской сущности»: используйте `{{ .property.template_ddp_slug }}` — идентификатор шаблона из параметра действия.

  > ⚠ Если идентификатор параметра не `template_ddp_slug`, замените на корректный.

- «Идентификатор дочерней сущности»: оставьте пустым — будет использована текущая сущность, для которой запускается действие.

Авторизация:

На вкладке «Авторизация» настройте учётные данные для выполнения действия. Настройки зависят от действия.

Действие «Создать проект в GitLab»:

- «Идентификатор»: `token`.
- «Реквизиты»: укажите реквизиты, которые содержат токен GitLab с необходимыми правами доступа для создания проектов в указанной группе.

Действие «Создать репозиторий из шаблона»:

- «Идентификатор»: `password` — реквизиты, которые содержат пароль (токен) GitLab с необходимыми правами доступа к репозиториям.
- «Идентификатор»: `username` — реквизиты, которые содержат GitLab username.

Действие «Обновление сервиса из шаблона»:

- «Идентификатор»: `password` — реквизиты, которые содержат пароль (токен) GitLab с необходимыми правами доступа к репозиториям.
- «Идентификатор»: `username` — реквизиты, которые содержат GitLab username.

Процесс

У вас должен быть создан процесс, который объединяет действия по созданию проекта в GitLab и созданию сервиса из шаблона в единую последовательную автоматизированную цепочку. Процесс должен последовательно запускать создание проекта в GitLab, а затем создание репозитория из шаблона. Если процесса ещё нет, создайте его.

Проверьте следующие настройки:

Основная информация:

- «Ресурс»: выбран ресурс «Сервисы» (или соответствующий ресурс, если он назван иначе).

Конфигурация:

> Названия действий могут отличаться в зависимости от ваших настроек.

Действия находятся в следующей последовательности:

1. Start.
2. Создать проект в GitLab.
3. Создать репозиторий из шаблона.
4. End.

Создание процесса

Для создания процесса перейдите в раздел «Самообслуживание», пункт «Процессы». Создайте новый процесс. Для этого справа вверху нажмите кнопку «Создать».

Откроется окно по созданию нового процесса с различными вкладками.

Основная информация:

- «Название»: [Укажите название ресурса].
- «Ресурс»: выберите ресурс «Сервисы» (или соответствующий ресурс, если он назван иначе).
- «Владелец»: [Укажите владельца].
- «Команда владелец»: [Укажите команду владелец].

Конфигурация:

> Названия задач и названия действий указываются в качестве примера. При необходимости укажите ваши.

На вкладке «Конфигурация» настройте шаги процесса и их порядок выполнения:

1. Добавьте элемент типа «Задача» с названием «Создать проект в GitLab» и выберите действие «Создать проект в GitLab» для запуска.
2. Добавьте элемент типа «Задача» с названием «Создать репозиторий из шаблона» и выберите действие «Создать репозиторий из шаблона» для запуска.
3. Добавьте элемент типа «Конец» для завершения процесса.

После добавления всех элементов настройте последовательность выполнения:

1. Start.
2. Создать проект в GitLab.
3. Создать репозиторий из шаблона.
4. End.

Автоматизация

У вас должна быть настроена автоматизация, которая запускает действие по созданию Merge Request для обновления сервисов при появлении новой версии шаблона. Автоматизация отслеживает изменения параметра «Последний тег» (или аналогичный) в сущностях ресурса «Шаблоны» (или аналогичный) и при обнаружении нового тега автоматически запускает действие «Обновление сервиса из шаблона» (или аналогичный) для всех связанных сервисов. Если автоматизации ещё нет, создайте её.

Проверьте следующие настройки:

Конфигурация:

Триггеры:

- «Ресурс»: выбран «Шаблоны» (или соответствующий ресурс, если он назван иначе).

- «Условие»: указано `{{ ne .entity.properties.repository_last_tag "v1.0.0" }}` (или аналогичное, если проверка у вас настроена иначе).

> Данное условие срабатывает каждый раз, когда параметр с идентификатором `repository_last_tag` в сущности ресурса «Шаблоны» (или аналогичным) не равен указанному значению (`v1.0.0`). Условие будет проверяться только один раз при изменении параметра `repository_last_tag` . Если параметр «Последний тег» (или аналогичный) имеет другой идентификатор (например, `last_tag`), используйте его в условии: `{{ ne .entity.properties.last_tag "v1.0.0" }}` .

Выполнение:

- «Тип выполнения»: выбрано «Действия».
- «Объект выполнения»: выбрано действие «Обновление сервиса из шаблона» (или соответствующее действие, если оно названо иначе).

Запускать выполнение для связанных ресурсов:

Параметр «Запускать выполнение для связанных ресурсов» включён и настроены связи ресурса:

- «Ресурс»: выбрано «Шаблоны» (или соответствующий ресурс).
- «Связь»: выбрана связь «Шаблоны-Сервисы» (или название вашей связи, если оно отличается).

Создание автоматизации

Для создания автоматизации перейдите в раздел «Самообслуживание», пункт «Автоматизации». Создайте автоматизацию. Для этого нажмите справа вверху кнопку «Создать».

Откроется окно по созданию новой автоматизации с различными вкладками.

Основная информация:

- «Название»: [Укажите название автоматизации].
- «Владелец»: [Укажите владельца].
- «Команда владелец»: [Укажите команду владелец].

Конфигурация:

На вкладке «Конфигурация» настройте логику запуска автоматизации.

Триггеры:

Триггер определяет, когда автоматизация должна запускаться. Добавьте новый триггер:

- «Ресурс»: выберите «Шаблоны» (или соответствующий ресурс, если он назван иначе).

- «Условие»: укажите `{{ ne .entity.properties.repository_last_tag "v1.0.0" }}`.

> ⚠ Используется идентификатор параметра ресурса «Шаблоны» (или аналогичного), который хранит информацию о последнем теге git-репозитория. Если идентификатор отличается, используйте корректный.

Данное условие срабатывает каждый раз, когда параметр `repository_last_tag` в сущности ресурса «Шаблоны» (или аналогичном) не равен указанному значению (`v1.0.0`). Условие будет проверяться только один раз при изменении параметра `repository_last_tag`. Если параметр «Последний тег» имеет другой идентификатор (например, `last_tag`), используйте его в условии: `{{ ne .entity.properties.last_tag "v1.0.0" }}`.

> ℹ Если планируется создание нескольких автоматизаций, каждая из которых должна реагировать на изменения параметров сущностей одного и того же ресурса, рекомендуется указывать условие в формате `{{ and (ne .entity.properties.repository_last_tag "v1.0.0") (eq .entity.slug "example") }}` для того, чтобы дополнительно фильтровать, какая из автоматизаций будет запускаться в каждом конкретном случае. Без добавления условия `(eq .entity.slug "example")` каждая из автоматизаций будет реагировать на изменения каждой сущности.

Выполнение:

Настройте выполнение действия:

- «Тип выполнения»: выберите «Действия».
- «Объект выполнения»: выберите действие «Обновление сервиса из шаблона» (или соответствующее действие, если оно названо иначе).

Запускать выполнение для связанных ресурсов:

Автоматизация просматривает изменения в сущности ресурса «Шаблоны», но действие будет обновлять сущности ресурса «Сервисы», которые связаны с шаблоном, в котором произошло обновление параметра `repository_last_tag`.

Необходимо включить параметр «Запускать выполнение для связанных ресурсов» и настроить связи ресурса:

- «Ресурс»: выберите «Шаблоны» (или соответствующий ресурс).
- «Связь»: выберите связь «Шаблоны-Сервисы» (или название вашей связи, если оно отличается).

При обновлении тега в ресурсе «Шаблоны» (или соответствующий ресурс) система найдёт все связанные с этим шаблоном сущности ресурса «Сервисы» (или соответствующий ресурс) и

запустит для каждой из них действие «Обновление сервиса из шаблона» (или аналогичное), создавая Merge Request с изменениями из новой версии шаблона.

Создание сервиса

Настройка платформы для создания сервисов на основе шаблона с последующим автоматическим обновлением при выпуске новой версии шаблона завершена.

Для создания сервиса необходимо следующее:

1. Перейдите во вкладку «Каталог» в ресурс «Сервисы».
2. В правом верхнем углу нажмите кнопку «Создать».
3. В открывшемся окне во вкладке «Основная информация» заполните требуемые поля. Если настройки были произведены в соответствии с данной инструкцией, заполнение данных в других вкладках не требуется. Данные будут заполнены автоматически в дальнейшем.
4. Справа от появившейся записи о новом сервисе нажмите на синюю кнопку с тремя вертикальными точками и выберите пункт «Запустить процесс».
5. В выпадающем списке выберите процесс `Создание сервиса из шаблона`.
6. При необходимости заполните переменные шаблона и запустите процесс.
7. При корректных настройках сервис будет создан. При наличии обновлений шаблона для сервиса будут формироваться MR с изменениями. Проверить корректность создания можно, перейдя в него и открыв вкладку «Запуски действий». Все действия должны быть в статусе `Success`.

Действия

Обзор

Действия — это механизм платформы для запуска операций во внешних инфраструктурных системах и сервисах. С их помощью можно, например:

- создавать проекты, переменные, ветки или теги в GitLab;
- создавать ресурсы в Kubernetes;
- получать ресурсы из Kubernetes;
- создавать секреты в Deckhouse Stronghold или HashiCorp Vault;
- создавать проекты в SonarQube, DefectDojo и других системах;
- создавать топики и ACL в Kafka.

Действие можно привязать к одному или нескольким ресурсам. После этого его можно запускать для любой сущности этих ресурсов.

Конфигурация действия

Основная информация

При создании или редактировании действия укажите основную информацию:

Название	Обязательность	Описание
Название	Да	Произвольное название действия
Идентификатор	Да	Идентификатор действия. Генерируется автоматически из названия
Ресурс	Нет	Один или несколько ресурсов, для которых будет доступен запуск действия
Иконка	Нет	Иконка, которая отображается в карточке действия
Владелец	Нет	Учётная запись пользователя, отвечающего за конфигурацию и работоспособность действия
Команда владелец	Нет	Команда, отвечающая за конфигурацию и работоспособность действия
Теги	Нет	Теги для классификации и поиска действий
Описание	Нет	Описание в Markdown. Отображается при запуске действия или процесса, содержащего действие

Конфигурация запроса

Тип действия

Действие может быть следующих типов:

- «Встроенное (BuiltIn)» — логика выполнения действия описана внутри платформы. Для встроенных действий необходимо выбрать один из преднастроенных бэкендов.
- «Вебхук (Webhook)» — логика выполнения действия полностью настраивается пользователем.

Параметры перезапуска

Если выполнение действия завершается с ошибкой, платформа может автоматически выполнить повторные попытки.

Количество перезапусков

Сколько раз повторять выполнение после ошибки. 0 — одна попытка без перезапусков.

Базовая задержка (сек.)

Задержка в секундах перед первым перезапуском; перед каждой следующей попыткой пауза удваивается (экспоненциальная задержка).

Формат тела запроса

Для вебхук-действий доступен выбор формата отправки тела запроса:

- JSON (по умолчанию) — тело запроса отправляется в формате JSON с заголовком `Content-Type: application/json`.
- «Form URL Encoded» — тело запроса отправляется в формате `application/x-www-form-urlencoded`.

> При выборе формата Form URL Encoded тело запроса должно содержать только плоские пары «ключ — значение». Например:
>
> ```
> token: {{ .credentials.token }}
> ```
>
> Вложенные структуры и массивы в этом формате не поддерживаются.

Расширенное логирование

Для вебхук-действий доступна опция расширенного логирования, которая включает детальное логирование всех деталей HTTP-запроса и ответа:

- URL запроса;
- HTTP-метод;
- HTTP-заголовки (включая токены);
- тело запроса;
- код статуса ответа;
- HTTP-заголовки ответа;
- тело ответа.

При включённой опции все эти данные записываются в лог выполнения действия. При выключённой опции логирование выполняется в стандартном режиме.

> Включение расширенного логирования может привести к записи конфиденциальной информации (токены, пароли и т. д.) в логи. Используйте эту опцию с осторожностью и только при необходимости отладки.

Тело запроса

Каждое действие отправляет HTTP-запрос на встроенный бэкенд или на URL вебхука бэкенда. Обычно запрос включает тело, в котором описывается, что именно будет отправлено. Тело запроса задаётся в YAML.

В тело запроса можно подставлять значения параметров из пользовательской формы с помощью шаблонов Go template.

Пример:

```
project_id: {{ .property.project_id }}
```

При таком теле запроса выражение `{{ .property.project_id }}` будет заменено на значение параметра `project_id`, введённое пользователем в форме запуска действия.

Для встроенных типов действий параметр «Тело запроса» является конфигурацией действия, при этом доступен пример данной конфигурации.

Пользовательская форма

Параметры

В разделе «Пользовательская форма» указываются параметры, доступные пользователю для заполнения при запуске действия. Типы параметров описаны в разделе «Параметры».

Для каждого параметра доступны опции:

- «Редактируемый» — позволяет изменить значение параметра при запуске действия. Если опция выключена, значение изменить нельзя.
- «Обязательный» — требует указания значения при запуске действия.
- «Скрытый» — параметр не отображается в пользовательской форме при запуске действия.

Для каждого параметра можно задать значение по умолчанию, которое будет подставляться в форму при запуске.

> Для нередактируемых или скрытых параметров рекомендуется задавать значение по умолчанию, так как при запуске действия пользователь не сможет их изменить.

Описание параметра отображается при запуске действия по кнопке со значком «info». Рекомендуется заполнять его, чтобы было проще понять назначение параметров и снизить риск ошибок при запуске.

В значениях параметров можно использовать шаблонные функции Go template. Например, выражение `{{ .entity.name }}` в значении параметра означает, что при запуске действия будет подставлено название сущности, для которой оно запускается.

Условия параметров

Параметры могут автоматически скрываться или отображаться в пользовательской форме в зависимости от значения параметра типа `Boolean`. Настройка выполняется в разделе «Условия параметров», где задаются правила показа или скрытия для выбранных параметров.

Обновление

Контексты шаблонизации

Все обновления выполняются только после успешного завершения действия. В Go-шаблонах доступны следующие контексты шаблонизации:

Контекст	Описание	Пример
`.property`	Значения из пользовательской формы	`{{ .property.repo }}`
`.response`	Разобранный ответ действия (JSON)	`{{ .response.id }}`
`.entity`	Параметры сущности каталога, для которой выполняется действие	`{{ .entity.slug }}`
`.workflow`	Параметры сценария	`{{ .workflow.branch }}`
`.process`	Параметры процесса	`{{ .process.releaseId }}`
`.global`	Глобальные переменные (идентификатор набора, затем ключ)	`{{ .global.myGroup.apiUrl }}`
`.team`	Переменные команды, выбранной при запуске	`{{ .team.token }}`
`.store`	Переменные из хранилища процесса	`{{ .store.myKey }}`

Обновление параметров сущности

Если включена опция «Обновление параметров сущности», платформа применяет правила обновления и записывает значения в параметры сущности.

Поле	Описание
Источник	Go-шаблон значения
Параметр сущности	Идентификатор параметра сущности в каталоге

Например, при выполнении действия «Создание проекта в GitLab» в поле `response` будет сохранена спецификация созданного проекта:

```
{
  "id": "1",
  "...": "..."
}
```

Если после создания проекта в GitLab нужно сразу заполнить параметр сущности `repository_id`, укажите в правилах обновления:

1. Источник: `{{ .response.id }}`.
2. Параметр сущности: `repository_id`.

Создание сущностей

Если включена опция «Создание сущностей», платформа автоматически создаст новые сущности в выбранных ресурсах по заданным правилам. Правила задаются отдельно для каждого ресурса.

Поле	Описание
Владелец создаваемых сущностей	Назначается владельцем всем сущностям, которые действие создаёт в каталоге
Команда-владелец создаваемых сущностей	Назначается командой-владельцем всем сущностям, которые действие создаёт в каталоге
Ресурс	Ресурс каталога, в котором будет создана сущность
Источник (идентификатор)	Go-шаблон для идентификатора создаваемой сущности
Источник (название)	Go-шаблон для названия создаваемой сущности
Дополнительные правила	Сопоставление Go-шаблона значения с параметром сущности

Идентификатор и название обязательны для каждой группы правил. Если в ресурсе уже есть сущность с таким идентификатором, создание пропускается.

Например, при выполнении действия «Создание проекта в GitLab» в ответе может быть:

```
{
  "id": "42",
  "name": "my-project",
  "...": "..."
}
```

Чтобы после успешного выполнения действия создать сущность в ресурсе «Проекты GitLab», укажите в правилах:

1. Ресурс: «Проекты GitLab».
2. Источник (идентификатор): `{{ .response.id }}`.
3. Источник (название): `{{ .response.name }}`.

При необходимости добавьте дополнительные правила для заполнения параметров сущности, например сопоставьте `{{ .response.path }}` с параметром `path`.

Создание связей сущности

Если включена опция «Создание связей сущности», платформа автоматически создаст новые связи для выбранной сущности по заданным правилам. Набор правил задаётся отдельно для каждого ресурса.

Поле	Описание
Ресурс	Любой из двух ресурсов, участвующих в нужной связи ресурсов: по выбранному ресурсу загружается список связей, в которых он указан как родительский или как дочерний. Можно выбрать ресурс сущности, для которой запускается действие, или ресурс другой стороны связи — в списке будет одна и та же связь, если оба ресурса входят в неё
Связь	Связь ресурсов в каталоге: в ней заданы родительский и дочерний ресурсы и роли сущностей при создании связи. По выбранной связи определяется, на какой стороне находится сущность запуска и какое из полей ниже задаёт идентификатор второй сущности из ответа действия
Идентификатор родительской сущности	Go-шаблон для идентификатора родительской сущности в каталоге
Идентификатор дочерней сущности	Go-шаблон для идентификатора дочерней сущности в каталоге

Обновление учётных данных пользователя

Если включена опция обновления учётных данных пользователя, то действие автоматически обновляет учётные данные пользователя в соответствии с правилами обновления.

Поле	Описание
Источник	Go-шаблон значения
Тип учётных данных	Тип обновляемых учётных данных

Например, при выполнении действия, которое возвращает новый API-ключ в ответе:

```
{
  "apiKey": "new-api-key-12345",
  "...": "..."
}
```

для обновления учётных данных укажите в правилах обновления:

1. Источник: `{{ .response.apiKey }}`.
2. Тип учётных данных: выберите тип учётных данных, которые будут обновлены.

Выбор пользователя для обновления учётных данных

По умолчанию учётные данные обновляются для пользователя, который запустил действие (инициатора).

Чтобы обновить учётные данные другого пользователя, включите опцию «Обновлять учётные данные конкретного пользователя» и выберите нужного пользователя.

Обновление хранилища процесса

Правила из раздела «Обновление хранилища процесса» применяются только при выполнении действия в рамках процесса: после успешного завершения действия значения записываются в хранилище запуска по заданным правилам, **в порядке списка**.

Поле	Описание
Условие	Необязательный Go-шаблон; пустое — правило всегда выполняется. Результат рендеринга: `true`, `false`, `1` или `0`
Операция	Записать строку, Записать JSON, Добавить строку, Добавить JSON, Слить JSON (поверхностно), Удалить
Цель	Dot-path в хранилище без ведущей точки, например `deploy.job_id` или `notification.items`
Источник	Go-шаблон значения

Другие действия и элементы процесса читают данные через `{{ .store.<путь> }}`. Внутри цикла в шаблонах доступен контекст `{{ .store._loop.* }}`.

Если правил нет (список пустой), при выполнении действия в процессе в хранилище ничего не записывается.

Подробное описание операций, примеры правил, цикл по коллекции и просмотр хранилища — в разделе «Хранилище процесса».

Безопасность

Условия выполнения

Действие будет выполнено, только если статус сущности соответствует одному из выбранных значений в поле статусов. Если ограничения не заданы, это условие не применяется.

Маскирование полей действия

Для каждого действия можно включить маскирование полей, которые могут содержать конфиденциальную информацию.

Если включить данную опцию, то в записях выполнения действия будут скрыты:

- значения всех заполненных параметров;
- тело запроса (`body`);
- ответ (`response`), сформированный по результатам выполнения.

Временный ответ

Для каждого действия можно включить опцию «Временный ответ». Она позволяет ограничить доступ к результату выполнения действия и скрыть его от других пользователей.

Если опция включена, то после успешного выполнения действия ответ:

- сохраняется как временный и отображается только пользователю, который запустил действие;
- удаляется из обычного поля ответа, чтобы он не был доступен другим пользователям.

Другие пользователи вместо содержимого ответа будут видеть только зашифрованный временный ответ.

Инициатор действия может удалить временный ответ вручную.

> Временный ответ доступен только инициатору действия. Остальные пользователи не могут просмотреть его содержимое и не могут удалить его.

Подтверждение

Для действия можно включить обязательное подтверждение и задать количество подтверждающих. В этом случае действие не будет запущено, пока не будет получено указанное число подтверждений от заданных пользователей.

Текущее число подтверждений и список пользователей, от которых ожидается подтверждение, отображаются в таблице запусков действий для соответствующей сущности.

Если подтверждение требуется от конкретного пользователя, ему будет отправлено уведомление в интерфейсе платформы.

Уведомления о подтверждении

При включённом подтверждении можно включить уведомления и указать способ отправки. Если выбрана отправка на URL, указывается адрес вебхука, при необходимости отключается проверка TLS-сертификата сервера и добавляются дополнительные HTTP-заголовки запроса.

Авторизация

Использовать внешний сервис

Выбор внешних сервисов, для которых может выполняться действие. В случае добавления нескольких внешних сервисов конкретный можно будет выбрать непосредственно при запуске.

URL и HTTP-заголовки

Адрес, куда будет отправлен запрос, и HTTP-заголовки.

В заголовках поддерживаются Go-шаблоны вида `{{ .credentials.<идентификатор типа учётных данных> }}` для подстановки учётных данных пользователя.

Выбрать учётную запись для запуска

По умолчанию доступ к внешним инфраструктурным сервисам выполняется с использованием учётных данных пользователя, который запустил действие. При необходимости можно явно указать, что действие должно выполняться от имени конкретной учётной записи.

Для действий, запускаемых как события автоматизации, указание учётной записи для запуска является обязательным.

Учётные данные

Для встроенных действий платформа заранее определяет набор требуемых учётных данных. Их идентификаторы подгружаются при выборе встроенного бэкенда. Для каждого идентификатора необходимо выбрать тип учётных данных, который будет использоваться.

Для вебхук-действий обращение к учётным данным возможно в теле запроса с использованием конструкции `{{ .credentials.<идентификатор типа учётных данных> }}`.

HTTP-метод

Для вебхук-действий задаётся HTTP-метод исходящего запроса.

Запуски действий

Записи запусков действий

При каждом запуске действия создаётся запись, содержащая инициатора, статус выполнения и подробный лог. Записи запусков для каждой сущности доступны в карточке сущности на вкладке «Запуски действий». Если для действия требуется подтверждение, оно выполняется на этой же вкладке.

Запись о запуске можно удалить или перезапустить действие. При перезапуске создаётся новая запись.

Логирование запусков

Для каждого запуска действия в базе данных создаётся запись, содержащая полный лог выполнения.

Параметр `actions.logging.enabled` в конфигурационном файле Deckhouse Development Platform (DDP) управляет выводом логов запуска в stdout: при значении `true` логи выводятся, при `false` не выводятся.

> Записи с полным логом запуска в базе данных создаются независимо от значения `actions.logging.enabled`.

Статусы действий

Для каждого запуска действия создаётся запись со статусом. Возможные статусы:

- «Created» — запись создана, но действие ещё не запускалось.
- «Unapproved» — действие ожидает подтверждения.
- «Running» — действие выполняется.
- «Failed» — действие завершилось с ошибкой.
- «Update failed» — действие выполнено, но обновить параметры сущности не удалось.
- «Success» — действие выполнено успешно.
- «Retrying» — действие завершилось с ошибкой, выполняется повторная попытка запуска.

Типы действий

CreateDefectdojoEngagement

CreateDefectdojoEngagement — создаёт новый engagement в системе DefectDojo. Действие использует DefectDojo API v2.

Пример запроса

```
name: example engagement
product: '1'
target_start: '2024-06-01'
target_end: '2024-06-30'
lead: '1'
```

Спецификация запроса

Список полей соответствует официальному API DefectDojo, `/api/v2/engagements`, подробнее — в документации Defectdojo.

Учётные данные

- `token` — API v2 Key пользователя, от имени которого будет запускаться выполнение действия.

CreateDefectdojoProduct

CreateDefectdojoProduct — создаёт новый продукт в системе DefectDojo. Действие использует DefectDojo API v2.

Пример запроса

```
name: example
description: example description
prod_type: 1
```

Спецификация запроса

Список полей соответствует официальному API DefectDojo, `/api/v2/products`, подробнее — в документации Defectdojo.

Учётные данные

- `token` — API v2 Key пользователя, от имени которого будет запускаться выполнение действия.

CreateGitlabBranches

CreateGitlabBranches — создаёт новые ветки в целевом репозитории.

Пример запроса

```
project_id: '0'
branches:
  - branch: new-branch
    ref: main
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо создать ветки
branches	Да	Список создаваемых веток
branches.branch	Да	Название новой ветки
branches.ref	Да	Название существующей ветки или SHA-хеш коммита

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие корректных реквизитов токена GitLab. Этот токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/repository/branches`. В случае успешного создания проекта GitLab возвращает информацию о созданных ветках.

CreateGitlabGroupVariables

CreateGitlabGroupVariables — создаёт переменные (variables) на уровне группы в GitLab.

Пример запроса

```
group_id: '0'
variables:
  - key: EXAMPLE_VARIABLE
    value: value
```

Спецификация запроса

Название	Обязательность	Описание
group_id	Да	Идентификатор группы, в котором необходимо создать переменные
variables	Да	Список создаваемых переменных

Список полей для переменных соответствует официальному GitLab Group-level Variables API, `/groups/:id/variables`, подробнее — в документации Gitlab.

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/groups/:id/variables`.

CreateGitlabMergeRequest

CreateGitlabMergeRequest — создаёт новый Merge Request (MR) в целевом репозитории. В Merge Request добавляются файлы, хранящиеся в репозитории-источнике. Файлы могут содержать переменные, значение которых будет подставлено в момент создания MR.

Пример запроса

```
source_project_id: '0'
source_project_branch: example
source_project_tag: v1.0.0
target_project_id: '0'
merge_request_spec:
  source_branch: example
  target_branch: '1'
  title: example
additionalIgnoreFiles:
  - .ignore
  - .example
values:
  key1: value1
  nested:
    enabled: true
    subkey: 123
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
source_project_id	Да	Идентификатор проекта, который служит источником для Merge Request	-
target_project_id	Да	Идентификатор целевого проекта, в котором будет сформирован Merge Request	-
merge_request_spec	Да	Спецификация, соответствующая GitLab Merge Requests API	-
source_project_tag	Нет	Тег в проекте-источнике, из которого будет сформирован Merge Request. Если не указан, используется ветка в проекте-источнике	-
source_project_branch	Нет	Ветка в проекте-источнике, из которой будет сформирован Merge Request	main
additionalIgnoreFiles	Нет	Список файлов, содержащих пути для исключения из MR. Заполняется по аналогии с .templateignore	-
values	Нет	Переменные, используемые при шаблонизации, в формате `ключ: значение`	-

Учётные данные

- `password` — пароль (токен) пользователя, от имени которого будет запускаться выполнение действия.
- `username` — имя пользователя, от которого будет запускаться выполнение действия.

Алгоритм работы

Платформа:

1. Клонирует репозиторий-шаблон для генерации MR по его идентификатору (`source_project_id`). Подробнее о шаблонизации.

2. Считывает файл `values.yaml`, хранящийся в корне репозитория, и определяет переменные по умолчанию для шаблонизации.
3. Считывает переменные, передаваемые при запуске действия, и объединяет (merge) их с переменными из `values.yaml`. Приоритет отдаётся переменным, передаваемым при запуске действия.
4. Считывает файл `.templateignore` и определяет директории и файлы, исключаемые из шаблонизации.
5. Рендерит файлы из шаблонов, учитывая `values.yaml` и переданных в действие переменных.
6. Изменяет удалённый (remote) репозиторий на целевой, согласно его ID (`target_project_id`), и выполняет git push в целевую ветку (`source_project_branch`), либо в основную ветку `main`.
7. Создаёт MR согласно заданным настройкам путём отправки POST-запроса в GitLab API.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/merge_requests`.

CreateGitlabProject

CreateGitlabProject — создаёт новый проект в GitLab. Действие осуществляет вызов GitLab API для создания проекта с указанными параметрами, такими как название, путь проекта, описание и другие настройки. Для аутентификации используется GitLab token, который должен быть предоставлен в учётных данных.

Пример запроса

```
name: example
path: example
description: example
default_branch: main
initialize_with_readme: false
namespace_id: '0'
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
name	Да	Название проекта, который будет создан в GitLab	-
path	Да	URL-совместимый путь проекта. Обычно совпадает с названием, но может отличаться	-
default_branch	Да	Название ветки, которая будет использоваться по умолчанию, например, «main»	-
namespace_id	Да	Идентификатор неймспейса (namespace) в GitLab, в котором будет создан проект	-
initialize_with_readme	Нет	Флаг, определяющий, нужно ли инициализировать проект с файлом README	false
description	Нет	Описание проекта, которое будет видно пользователям	-

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects`, передавая параметры запроса в формате JSON. В случае успешного создания проекта GitLab возвращает данные о вновь созданном проекте.

CreateGitlabProjectVariables

CreateGitlabProjectVariables — создаёт переменные на уровне проекта в GitLab.

Пример запроса

```
project_id: '0'
variables:
  - key: EXAMPLE_VARIABLE
    value: value
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо создать переменные
variables	Да	Список создаваемых переменных

Список полей для переменных соответствует официальному GitLab Project-level CI/CD variables API, `/projects/:id/variables`, подробнее — в документации Gitlab.

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/variables`.

CreateGitlabProjectWebhook

CreateGitlabProjectWebhook — создаёт вебхук в проекте GitLab.

Пример запроса

```
project_id: '0'
url: https://example.com
push_events: true
issues_events: true
merge_requests_events: true
pipeline_events: true
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо создать вебхук
url	Да	URL-адрес вебхука
push_events	Да	Запускать вебхук при push в репозиторий
issues_events	Да	Запускать вебхук при создании Issue
merge_request_events	Да	Запускать вебхук при создании Merge Request
pipeline_events	Да	Запускать вебхук при запуске Pipeline

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/hooks`.

CreateGitlabTag

CreateGitlabTag — создаёт новый тег в проекте GitLab.

Пример запроса

```
project_id: '0'
tag_name: v1.0.0
ref: main
message: Tag description
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо создать тег
tag_name	Да	Название тега
ref	Да	Название ветки, тег или SHA-хеш коммита, на который будет ссылаться новый тег
message	Да	Описание тега

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/repository/tags`.

CreateGitlabRelease

CreateGitlabRelease — создаёт релиз GitLab на основе уже существующего тега.

Пример запроса

```
project_id: '0'
tag_name: v1.0.0
name: Release v1.0.0
description: |
  ## Изменения:
  - Новая функция.
  - Исправления ошибок.
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо создать релиз
tag_name	Да	Название существующего тега, на основе которого формируется релиз
name	Да	Название релиза, отображаемое в GitLab
description	Нет	Описание релиза в формате Markdown

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/releases`.

CreateKafkaACLs

CreateKafkaACLs — создаёт новый набор ACL в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
acls:
  - topics:
      - example_1
    allow:
      - User:principal_2
      - Group:principal_3
    deny:
      - User:principal_4
      - Group:principal_5
    ops:
      - CREATE
      - READ
      - WRITE
      - DELETE
      - DESCRIBE
      - DESCRIBE_CONFIGS
      - ALTER
    pattern: LITERAL
  - topics:
      - example_6
    allow:
      - User:principal_7
    allow_hosts:
      - 127.0.0.1
    deny:
      - User:principal_8
    deny_hosts:
      - 127.0.0.1
    ops:
      - CREATE
    pattern: LITERAL
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
securityProtocol	Да	Протокол для подключения к Kafka. Подробнее — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL	-
saslMechanism	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512	-
acls	Да	Набор ACL, которые необходимо создать	-	-
acls.ops	Да	Список операций, для которых будет создано правило	Список возможных операций	-
acls.pattern	Да	Тип шаблона	Список возможных шаблонов	-
acls.topics	Нет	Список из названий топиков, для которых применяется правило	-	-

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
acls.groups	Нет	Список из названий групп, для которых применяется правило	-	-
acls.transactional_ids	Нет	Список из ID транзакций, для которых применяется правило	-	-
acls.tokens	Нет	Список токенов, для которых применяется правило	-	-
acls.allow	Нет	Список хостов, для которых разрешается операция	-	-
acls.deny	Нет	Список принципалов (user, group), для которых запрещается правило	-	-
acls.deny_hosts	Нет	Список хостов, для которых запрещается операция	-	-

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

Список возможных шаблонов

- ANY.
- MATCH.
- LITERAL.
- PREFIXED.

Список возможных операций

Подробнее — в документации Kafka.

Topic:

- ALL.
- ALTER.
- ALTER_CONFIGS.
- CREATE.
- DELETE.
- DESCRIBE.
- DESCRIBE_CONFIGS.
- READ.
- WRITE.

Group:

- ALL.
- DELETE.
- DESCRIBE.
- READ.

TransactionalID:

- ALL.
- DESCRIBE.
- WRITE.

Tokens:

- DESCRIBE.

CreateKafkaTopics

CreateKafkaTopics — создаёт новые топики в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
partitions: 1
replication_factor: 1
configs: {}
topics:
  - example_1
  - example_2
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
securityProtocol	Да	Протокол для подключения к Kafka. Подробнее — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
partitions	Да	Количество разделов (партиций), на которые будет разделён топик	-
replication_factor	Да	Количество копий (реплик) каждой партиции топика, которые необходимо разместить на разных брокерах	-
configs	Да	Конфигурация в формате ключ-значение для создаваемых топиков	Значения приведены в документации Kafka
topics	Да	Список названий топиков, которые необходимо создать	-

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

CreateKafkaUsers

CreateKafkaUsers — создаёт новых пользователей SASL/SCRAM в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
users:
  - user: example_user_1
    password: example_password_user_1
    mechanism: SCRAM-SHA-256
    iterations: 4096
  - user: example_user_2
    password: example_password_user_2
    mechanism: SCRAM-SHA-256
    iterations: 4096
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
securityProtocol	Да	Протокол для подключения к Kafka. Подробнее — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
users	Да	Набор пользователей, которых необходимо создать	-
users.user	Да	Имя создаваемого пользователя	-
users.password	Да	Пароль создаваемого пользователя	-
users.mechanism	Да	Механизм аутентификации создаваемого пользователя	SCRAM-SHA-256, SCRAM-SHA-512
users.iterations	Да	Количество итераций, которые будут применяться для хеширования пароля	От 4096 до 16384

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

CreateCodeScoringProject

CreateCodeScoringProject — создаёт новый проект в системе CodeScoring. Действие использует CodeScoring API для регистрации проекта с указанными параметрами: название проекта, URL репозитория, ID VCS системы и опция автоматического запуска SCA-анализа после клонирования репозитория.

Пример запроса

```
name: example-project
repository: https://gitlab.example.com/group/project.git
vcs_id: 2
run_sca_after_clone: true
```

Спецификация запроса

Название	Обязательность	Описание
name	Да	Название проекта в CodeScoring
repository	Да	URL репозитория (например, https://gitlab.example.com/group/project.git)
vcs_id	Да	ID VCS системы в CodeScoring (должен быть больше 0)
run_sca_after_clone	Нет	Автоматический запуск SCA-анализа после клонирования репозитория

Ответ

В ответе возвращается объект созданного проекта со следующей информацией: идентификатор проекта (pk), название, тип проекта, описание, информация о репозитории, статус проекта, права доступа, лицензия, количество зависимостей и уязвимостей, языки проекта, статус расписания сканирования и даты первого и последнего SCA-сканирования.

DeleteCodeScoringProject

DeleteCodeScoringProject — удаляет проект в системе CodeScoring по его ID.

Пример запроса

```
id: 1
```

Спецификация запроса

Название	Обязательность	Описание
id	Да	ID проекта в CodeScoring

CreateKeycloakClient

CreateKeycloakClient — создаёт нового клиента в Keycloak.

Пример запроса

```
realm: master
config:
  clientId: example
  name: example
  enabled: true
  clientAuthenticatorType: client-secret
  secret: secret
  defaultClientScopes:
  - roles
  - profile
  - email
  optionalClientScopes:
  - address
  - phone
  - offline_access
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
realm	Да	Realm в Keycloak, где требуется создать клиента	-
config	Да	Параметры создаваемого клиента в соответствии со спецификацией ClientRepresentation Keycloak	-

Учётные данные

- `username` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

CreateKubernetesResource

CreateKubernetesResource — создаёт новый ресурс или ресурсы в кластере Kubernetes или обновляет существующие.

Пример запроса

```
manifests:
  - apiVersion: v1
    kind: Namespace
    metadata:
      name: example1
  - apiVersion: v1
    kind: Namespace
    metadata:
      name: example2
```

Спецификация запроса

Название	Обязательность	Описание
manifests	Да	Манифесты Kubernetes, которые будут применены

Учётные данные

- `token` — токен сервисного аккаунта в Kubernetes.

GetKubernetesResource

GetKubernetesResource — получает ресурс из Kubernetes кластера.

Пример запроса

```
group: managed-services.deckhouse.io
version: v1alpha1
resource_type: postgres
resource_name: example-postgres
namespace: default
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
group	Да	API-группа ресурса. Указывает, к какой группе API относится запрашиваемый объект	Определение требуемых Group и Version
version	Да	Версия API ресурса	Определение требуемых Group и Version
resource_type	Да	Название ресурса во множественном числе, как в поле NAME вывода команды `kubectl api-resources`	-
resource_name	Да	Название конкретного ресурса, который необходимо получить	-
namespace	Да	Неймспейс, в котором находится ресурс	-

Ответ

При успешном выполнении действие возвращает объект ресурса в поле `resource`. Если ресурс не найден, действие завершается с ошибкой.

Название	Описание
`resource`	Объект ресурса в формате Kubernetes

Учётные данные

- `token` — токен сервисного аккаунта в Kubernetes.

CreateRepositoryFromTemplate

CreateRepositoryFromTemplate — создаёт новый репозиторий из шаблона в Gitlab. Механизм рендеринга основан на Go template и поддерживает все встроенные методы, а также расширения, добавленные в платформу.

Пример запроса

```
sourceBranch: main
sourceTag: v1.0.0
templateRepositoryUrl: https://gitlab.example.com/example-1.git
targetRepositoryUrl: https://gitlab.example.com/example-2.git
targetBranch: master
additionalIgnoreFiles:
  - .ignore
  - .example
values:
  key1: value1
  nested:
    enabled: true
    subkey: 123
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
templateRepositoryUrl	Да	URL шаблонного репозитория	-
targetRepositoryUrl	Да	URL репозитория, который будет создан в результате выполнения действия	-
values	Да	Переменные, используемые при шаблонизации, в формате `ключ: значение`	-
additionalIgnoreFiles	Нет	Список файлов, содержащих пути для исключения из целевого репозитория. Заполняется по аналогии с .templateignore	-
sourceTag	Нет	Тег шаблонного репозитория, который будет использоваться при шаблонизации. Если не указан, используется ветка шаблонного репозитория	-
sourceBranch	Нет	Ветка шаблонного репозитория, которая будет использоваться при шаблонизации	main
targetBranch	Нет	Ветка целевого репозитория, которая будет создана в результате выполнения действия	main

Учётные данные

- `password` — пароль (токен) пользователя, от имени которого будет запускаться выполнение действия.
- `username` — имя пользователя, от которого будет запускаться выполнение действия.

Алгоритм работы

Платформа:

1. Клонирует шаблонный репозиторий по указанному URL (`templateRepositoryUrl`), используя в качестве ref либо `sourceTag` , либо `sourceBranch` , либо ветку `main` .

2. Считывает файл `values.yaml`, хранящийся в корне репозитория, и определяет переменные по умолчанию для шаблонизации.
3. Считывает переменные, передаваемые при запуске действия, и делает их merge с переменными из `values.yaml`. Приоритет при merge отдаётся переменным, передаваемым при запуске действия.
4. Считывает файл `.templateignore` и определяет директории и файлы, исключаемые из шаблонизации.
5. Рендерит из шаблонов файлы, учитывая `values.yaml` и переданные в действие переменные.
6. Изменяет удалённый (remote) репозиторий на целевой (`targetRepositoryUrl`) и делает git push в целевую ветку (`targetBranch`), либо в основную ветку `main`.

Детали работы

Действие поддерживает шаблонизацию имён директорий и файлов. Для этого необходимо в их название добавить выражение в формате Go template. Например, директория `src/{{ .module }}/utils` при наличии value `module` со значением `example` будет отрендерена в директорию `src/example/utils` в целевом репозитории.

Если после рендеринга из шаблона содержимое файла будет отсутствовать, файл не создаётся. Например, файл с содержимым:

```
{{- if .createContent }}
- Это контент, который будет отображаться, если переменная createContent == true
{{- end }}
```

не будет создан, если переменная `createContent` имеет значение `false`. Аналогичным образом, не будут созданы файлы, изначально являющиеся пустыми.

При отсутствии переменных для шаблонизации одновременно в файле `values.yaml` и в переменных, передаваемых при запуске действия, рендеринг завершится с ошибкой и целевой репозиторий создан не будет.

Переменные шаблонного репозитория

Для добавления переменных по умолчанию, используемых при шаблонизации, необходимо создать в корне репозитория файл values.yaml с соответствующим содержимым.

Пример файла `values.yaml`:

```
module: example
createContent: false
```

Файл `values.yaml` является опциональным.

Исключение файлов

Некоторые файлы могут содержать переменные в формате Go template, которые необходимо сохранять при рендеринге репозитория из шаблона, например, Helm-чарты в директории `helm`. Директория `.git` игнорируется всегда.

Для исключения подобных файлов из механизма рендеринга следует добавить в корень репозитория файл `.templateignore` с соответствующим содержимым.

В каждой строке `.templateignore` задаётся одно правило — путь или маска. Если в строке есть `{{`, вся строка выполняется как один шаблон Go с теми же переменными и функциями, что при подстановке в имена файлов и каталогов (подробнее — «Как обрабатывается строка правила»). Если `{{` в строке нет, подстановка переменных из `values.yaml` и из запроса действия не делается: строка читается как есть и сравнивается с относительным путём на диске, допускаются литералы и символы маски `*` и `**`.

Пример файла `.templateignore` только с масками пути, без шаблонов подстановки, для игнорирования содержимого директорий `helm`, `docs`:

```
helm/**
docs/**
```

Добавление путей в игнор

1. В корне шаблонного репозитория создайте или отредактируйте файл `.templateignore` (по одному правилу на строку).
2. Каждая строка — это одно правило: путь от корня репозитория. В нём допускаются обычные символы пути и маски: звёздочка `*` в имени сегмента, последовательность `**` — для произвольной глубины вложенных каталогов. Платформа сопоставляет относительный путь с маской по встроенным правилам (аналогично распространённым соглашениям для масок в файлах игнорирования в системах контроля версий).
3. Чтобы подставить фрагмент пути из `values.yaml` или из поля `values` запроса действия, используйте в строке конструкции Go template (`{{ ... }}`). Если в строке есть `{{`, платформа обрабатывает всю строку от начала до конца как один шаблон Go: нельзя оставить часть строки «простым текстом» и шаблонизировать только середину пути. Примеры — в разделе «Примеры Go template в `.templateignore`».
4. Пустые строки и строки, начинающиеся с `#`, при разборе файла пропускаются — их можно использовать для комментариев.

Примеры без подстановки (только маски пути)

Отдельные файлы в корне:

```
package-lock.json
yarn.lock
LICENSE
.env.local
```

Каталоги целиком и типичные артефакты сборки:

```
vendor/**
node_modules/**
dist/**
build/tmp/**
```

Вложенность по маске:

```
docs/**/*.pdf
charts/*/values.schema.json
.github/workflows/**
```

Секреты по расширению во всём дереве:

```
**/*.pem
**/*.key
```

Как обрабатывается строка правила

Переменные для раскрытия правил те же, что объединены из `values.yaml` в корне шаблонного репозитория и из поля `values` в запросе действия (приоритет у `values` в запросе).

Для каждой непустой строки из `.templateignore` или из файла из `additionalIgnoreFiles` платформа делает следующее.

1. В список правил всегда попадает строка как в файле, без изменений. Именно её потом сравнивают с путём к файлу в первую очередь — это нужно, когда в именах на диске ещё есть фрагменты вроде `{{ .module }}` до переименования.
2. Если в строке есть `{{`, платформа один раз прогоняет всю строку через шаблонизатор Go template с теми же возможностями, что при подстановке в имена файлов и каталогов (встроенные функции платформы и набор Sprig). Если `{{` в строке нет, этот шаг пропускают.
3. Если шаг подстановки был и полученный текст отличается от строки в файле (включая случай «на выходе пусто»), в список правил добавляют вторую запись — уже с этим полученным текстом. Итого из одной строки файла может получиться две записи в списке. При проверке пути к файлу смотрят обе: достаточно совпадения с любой из них — файл попадает под правило.

Дальше при обходе дерева для каждого пути к файлу вычисляется путь относительно корня клонированной копии (с прямыми слешами). Путь сравнивается с каждым правилом: сначала по правилам сопоставления с маской (в том числе с использованием `*` и `**`), при необходимости — по точному совпадению строки правила и относительного пути.

Так одна строка в файле может задать два правила: например, исходное `charts/{{ .name }}/**` (чтобы не трогать путь до переименования каталогов), и раскрытое `charts/billing/**` (чтобы совпадало с путём после подстановки переменных в имена на диске). Поэтому правила работают и «до», и «после» этапа переименования каталогов по шаблону.

Если шаблон в строке синтаксически неверен или обращается к отсутствующему полю, раскрытие правил завершается ошибкой, и рендер репозитория не продолжается.

Поле `additionalIgnoreFiles` в действии задаёт имена дополнительных файлов в корне репозитория; в каждом из них — такие же строки-правила, как в `.templateignore`, с тем же раскрытием шаблонов. Но смысл другой: совпавшие пути убирают из рабочей копии (каталог или файл удаляют), и делают это дважды — в начале и в конце цепочки обработки, чтобы эти объекты не участвовали в дальнейших шагах и не попали в итоговый репозиторий.

Файл `.templateignore`, наоборот, означает «не шаблонизировать»: для совпавших путей платформа не подставляет переменные в содержимое файлов и не применяет к соответствующим каталогам переименование по шаблону в пути. Сами файлы и каталоги при этом не удаляются только из-за записи в `.templateignore` — они остаются в копии, но обрабатываются как обычный текст и обычные имена, без шага шаблонизации.

Примеры Go template в `.templateignore`

Ниже в каждом примере показаны фрагменты `values.yaml` (или эквивалентные поля в `values` запроса) и строки `.templateignore`. Несколько независимых правил задают несколькими строками файла: результат одной строки-шаблона — одна строка с маской; перевод строк внутри результата шаблона не делит правило на несколько.

Имя каталога в правиле берётся из `values.yaml` или из `values` действия:

`values.yaml`:

```
project: payment-gateway
```

`.templateignore`:

```
{{ .project }}/legacy/**
```

В набор правил попадут строки `{{ .project }}/legacy/**` и `payment-gateway/legacy/**`.

Сегмент пути из переменной и статический хвост:

`values.yaml`:

```
lang: ru
```

`.templateignore`:

```
apps/{{ .lang }}/messages.yaml
```

Условное правило в одной строке работает следующим образом: при `skipGenerated: false` подстановка даёт пустую строку, которая всё равно добавляется вторым правилом. Исходная строка с `{{` используется как маска пути и обычно не совпадает с реальными путями. При `skipGenerated: true` вторым правилом становится `generated/**`):

`values.yaml`:

```
skipGenerated: false
```

`.templateignore`:

```
{{- if .skipGenerated }}generated/**{{- end }}
```

Вариант «игнорировать только не production»:

`values.yaml`:

```
tier: staging
```

`.templateignore`:

```
{{- if ne .tier "prod" }}mock/**{{- end }}
```

Использование `printf` для сборки строки маски (удобно, если имя чарта в переменной):

`values.yaml`:

```
chartName: wordpress
```

`.templateignore`:

```
{{ printf "charts/%s/**" .chartName }}
```

Значение по умолчанию для «пустого» значения — функция `default` из набора Sprig. Поле в данных должно существовать (иначе при раскрытии сработает `missingkey=error`); для «не задано в YAML» заведите ключ с пустой строкой или используйте условие `if` / `index`:

`values.yaml`:

```
envName: ""
```

`.templateignore`:

```
{{ default "dev" .envName }}/secrets/**
```

При пустом `envName` в правило попадёт и `{{ default "dev" .envName }}/secrets/**`, и `dev/secrets/**`.

Опциональный сегмент пути (`with`):

`values.yaml`:

```
analyticsModule: tracking
```

`.templateignore`:

```
{{ with .analyticsModule }}{{ . }}/vendor/**{{ end }}
```

Если `analyticsModule` пусто, шаблон даёт пустую строку (см. правила раскрытия выше).

Доступ к полю вложенной структуры по строковому ключу `index`:

`values.yaml`:

```
regions:
  primary: eu-west
```

`.templateignore`:

```
configs/{{ index .regions "primary" }}/bootstrap.yaml
```

Удаление пробелов в сегменте имени — функция `trim` из набора Sprig:

`values.yaml`:

```
serviceName: " billing-api "
```

`.templateignore`:

```
{{ trim .serviceName " " }}/logs/**
```

Несколько фрагментов в одной маске:

`values.yaml`:

```
base: services
variant: canary
```

`.templateignore`:

```
{{ .base }}/{{ .variant }}/**/*.tmp
```

Примеры для `additionalIgnoreFiles`

В спецификации действия перечисляются имена файлов в корне репозитория (например, `.ship-ignore`, `.ci-remove`). Формат строк внутри таких файлов тот же: комментарии `#`, пустые строки, маски пути, при необходимости — шаблоны Go в строках.

Файл `.ship-ignore` в шаблоне:

```
# не попадает в целевой репозиторий
local/fixtures/**
scratchpad.md
```

Фрагмент запроса действия:

```
additionalIgnoreFiles:
  - .ship-ignore
```

Шаблон в файле для `additionalIgnoreFiles` (удаление каталога, имя из переменных):

Файл `.env-drop` в корне шаблона:

```
{{ .obsoleteDir }}/**
```

При `obsoleteDir: legacy-ui` из `values` после раскрытия в списке удаления окажутся и `{{ .obsoleteDir }}/**`, и `legacy-ui/**` — совпавшие пути будут удалены из копии перед финальными шагами.

> Не путайте назначение: `.templateignore` оставляет файлы на диске, но отключает для них переименование пути и рендер содержимого; списки из `additionalIgnoreFiles` удаляют совпавшие пути из рабочей копии.

Пример структуры директорий шаблонного репозитория

```
├── example-folder-01
│   ├── example-file-01
│   └── {{ .example }}-file-02
├── {{ .example }}-folder-02
│   └── ...
├── values.yaml
└── .templateignore
```

Если переменная `example` при рендере репозитория примет значение `new`, то итоговая структура после рендера будет выглядеть следующим образом:

```
├── example-folder-01
│   ├── example-file-01
│   └── new-file-02
├── new-folder-02
│   └── ...
├── values.yaml
└── .templateignore
```

Локальная отладка

Для локальной отладки шаблонов доступна утилита `ddp-render-dir`.

Утилита:

1. Создаёт копию исходной директории.
2. Выполняет рендеринг файлов в этой директории по тем же правилам, что и действие создания репозиториев из шаблонов.

Ключи командной строки для запуска:

- `--source-dir` — исходная директория, которую необходимо отрендерить.
- `--target-dir` — директория, в которую будет помещен результат рендеринга.
- `--values` (опционально) — путь к файлу `values.yaml` с переменными, которые будут использоваться при рендеринге.
- `--ignore-files` (опционально) — список файлов, содержащих пути для исключения из целевого репозитория.

CreateSonarqubeProject

CreateSonarqubeProject — создаёт новый проект в SonarQube. Действие использует SonarQube Web API для создания проекта с указанными параметрами, такими как ключ (project key), название проекта, главная ветка, параметры определения нового кода (new code definition) и видимость

проекта. Аутентификация осуществляется с использованием SonarQube token, который должен быть передан в учётных данных.

Пример запроса

```
project: example-project
name: example-project
mainBranch: develop
newCodeDefinitionType: NUMBER_OF_DAYS
newCodeDefinitionValue: '30'
visibility: public
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения	Значени по умолчан
project	Да	Уникальный идентификатор проекта (Project Key) в SonarQube	-	-
name	Да	Название проекта, отображаемое в интерфейсе SonarQube	-	-
mainBranch	Нет	Название главной ветки проекта	-	master
newCodeDefinitionType	Нет	Метод определения «нового кода»	PREVIOUS_VERSION, NUMBER_OF_DAYS, REFERENCE_BRANCH	-
newCodeDefinitionValue	Нет	Значение для определения «нового кода» (например, количество дней, если тип - NUMBER_OF_DAYS)	-	-
visibility	Нет	Видимость проекта	private, public	private

Учётные данные

- `token` — токен Sonarqube с типом User Token, сгенерированный пользователем, от имени которого будет запускаться выполнение действия.

CreateVaultSecret

CreateVaultSecret — создаёт секрет с одним или несколькими значениями в HashiCorp Vault.

Пример запроса

```
path: example/data/path
secrets:
  - key: key1
    value: value1
  - key: key2
    value: value2
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения	По умолчанию
path	Да	Путь, по которому будет сохранён секрет в Vault		-
allow_update	Нет	Определяет поведение действия при наличии существующего секрета	true, false, merge, merge_or_create	false
secrets	Да	Набор секретов, которые необходимо создать в Vault		-
secrets.key	Да	Название (идентификатор) секрета		-
secrets.value	Да	Значение секрета		-

Учётные данные

- `token` — Vault-токен, который имеет необходимые права для создания секретов.

Примечание

`allow_update` — определяет поведение действия при создании или обновлении секрета:

- `false` (по умолчанию) — действие завершается с ошибкой, если секрет уже существует;
- `true` — создаётся новая версия секрета со значениями, переданными в действии;
- `merge` — обновляются или создаются только те ключи секрета, которые указаны в действии. Существующие ключи, не упомянутые в действии, сохраняются без изменений. Если секрета по пути ещё нет, действие завершается с ошибкой;
- `merge_or_create` — как `merge`, если секрет уже существует; если секрета по пути нет, он создаётся (полная запись значений из действия).

DeleteVaultSecret

DeleteVaultSecret — удаляет секрет из HashiCorp Vault.

Пример запроса

```
path: example/data/path
```

Спецификация запроса

Название	Обязательность	Описание
path	Да	Путь, по которому находится секрет в Vault, который необходимо удалить

Учётные данные

- `token` — Vault-токен, который имеет необходимые права для удаления секретов.

DeleteDefectdojoProduct

DeleteDefectdojoProduct — удаляет продукт из DefectDojo. Действие использует DefectDojo API v2.

Пример запроса

```
id: 1
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
id	Да	Идентификатор продукта, который необходимо удалить	-

Учётные данные

- `token` — API v2 Key пользователя, от имени которого будет запускаться выполнение действия.

DeleteGitlabProject

DeleteGitlabProject — удаляет существующий проект в GitLab. Действие осуществляет вызов GitLab API для удаления проекта. Для аутентификации используется GitLab token, который должен быть предоставлен в учётных данных.

Пример запроса

```
project_id: 0
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, который необходимо удалить

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет DELETE-запрос по URL: `/api/v4/projects/:id`.

DeleteKafkaACLs

DeleteKafkaACLs — удаляет набор ACL в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
acls:
  - topics:
      - example_1
    allow:
      - User:principal_2
      - Group:principal_3
    allow_hosts:
      - 127.0.0.1
    deny:
      - User:principal_4
      - Group:principal_5
    deny_host:
      - 127.0.0.1
    ops:
      - CREATE
      - READ
      - WRITE
      - DELETE
      - DESCRIBE
      - DESCRIBE_CONFIGS
      - ALTER
    pattern: LITERAL
  - any_topic: true
    any_group: true
    any_transactional_id: true
    any_allow: true
    any_allow_hosts: true
    any_deny: true
    any_deny_hosts: true
    ops:
      - ANY
    pattern: ANY
  - any_resource: true
    any_allow: true
    any_allow_hosts: true
    any_deny: true
    any_deny_hosts: true
    ops:
      - ANY
    pattern: ANY
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
securityProtocol	Да	Протокол для подключения к Kafka. Подробнее — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL	-
saslMechanism	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512	-
acls	Да	Набор ACL, которые необходимо создать	-	-
acls.ops	Да	Список операций, для которых будет создано правило	Список возможных операций	-
acls.pattern	Да	Тип шаблона	Список возможных шаблонов	-
acls.any_resource	Нет	Все ресурсы	-	false
acls.topics	Нет	Список из названий топиков, для которых	-	-

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
		применять правило		
acls.any_topic	Нет	Все топики	-	false
acls.groups	Нет	Список из названий групп, для которых применять правило	-	-
acls.any_group	Нет	Все топики	-	false
acls.transactional_ids	Нет	Список из ID транзакций, для которых применять правило	-	-
acls.any_transactional_id	Нет	Любая транзакция	-	false
acls.tokens	Нет	Список токенов, для которых применять правило	-	-
acls.any_token	Нет	Все токены	-	false
acls.allow	Нет	Список принципалов (user, group), для которых разрешить правило	-	-
acls.any_allow	Нет	Любой принципал (user, group)	-	false
acls.allow_hosts	Нет	Список хостов, для которых	-	-

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
		разрешить операцию		
acls.any_allow_hosts	Нет	Любой хост, с которого разрешено проводить операцию	-	false
acls.deny	Нет	Список принципалов (user, group), для которых запретить правило	-	-
acls.any_deny	Нет	Любой принципал (user, group)	-	false
acls.deny_hosts	Нет	Список хостов, для которых запретить операцию	-	-
acls.any_deny_hosts	Нет	Любой хост, с которого запрещено проводить операцию	-	false

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

Список возможных pattern

- ANY.
- MATCH.
- LITERAL.
- PREFIXED.

Список возможных операций

Подробное описание — в документации Kafka.

Topic:

- ALL.
- ALTER.
- ALTER_CONFIGS.
- CREATE.
- DELETE.
- DESCRIBE.
- DESCRIBE_CONFIGS.
- READ.
- WRITE.

Group:

- ALL.
- DELETE.
- DESCRIBE.
- READ.

TransactionalID:

- ALL.
- DESCRIBE.
- WRITE.

Token:

- DESCRIBE.

DeleteKafkaTopics

DeleteKafkaTopics — удаляет существующие топики в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
topics:
  - example_1
  - example_2
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
auth_type	Да	Тип авторизации в Kafka	PLAINTEXT, SCRAM-SHA-256, SCRAM-SHA-512
topics	Да	Список названий топиков, которые необходимо удалить	-

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

DeleteKafkaUsers

DeleteKafkaUsers — удаляет существующих пользователей SASL/SCRAM в Kafka.

Пример запроса

```
securityProtocol: SASL_PLAINTEXT
saslMechanism: PLAIN
users:
  - user: example_user_1
    mechanism: SCRAM-SHA-256
  - user: example_user_2
    mechanism: SCRAM-SHA-256
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
securityProtocol	Да	Протокол для подключения к Kafka. Подробнее — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
saslMechanism	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
users	Да	Набор пользователей, которых необходимо удалить	-
users.user	Да	Имя удаляемого пользователя	-
users.mechanism	Да	Механизм аутентификации удаляемого пользователя	SCRAM-SHA-256, SCRAM-SHA-512

Учётные данные

- `user` — имя пользователя, от которого будет запускаться выполнение действия.
- `password` — пароль пользователя, от имени которого будет запускаться выполнение действия.

DeleteKubernetesResource

DeleteKubernetesResource — удаляет существующий ресурс в кластере Kubernetes.

Пример запроса

```
group: apps
version: v1
resource_type: deployments
resource_name: nginx-deployment
namespace: example
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения
group	Да	API-группа ресурса. Указывает, к какой группе API относится удаляемый объект	Определение требуемых Group и Version
version	Да	Версия API ресурса	Определение требуемых Group и Version
resource_type	Да	Тип удаляемого ресурса	pods, services, deployments, statefulsets, daemonsets, replicasets, jobs, cronjobs, nodes, namespaces, configmaps, secrets, persistentvolumes, persistentvolumeclaims, limitranges, resourcequotas, horizontalpodautoscalers, ingresses, networkpolicies, serviceaccounts, roles, clusterroles, rolebindings, clusterrolebindings, podsecuritypolicies, storageclasses, volumeattachments, events, endpoints, customresourcedefinitions
resource_name	Да	Название конкретного ресурса, который необходимо удалить	-
namespace	Да	Неймспейс, в котором находится ресурс	-

Учётные данные

- `token` — токен сервисного аккаунта в Kubernetes.

Определение требуемых Group и Version

Каждому типу ресурса соответствует своя группа API (Group) и версия (Version). Полный список API-ресурсов с их группами и версиями приведён в документации Kubernetes.

Если неизвестно, какие требуются Group и Version, можно использовать актуальные значения. Существует несколько способов их определить:

С помощью утилиты `d8 k`

Команда `d8 k explain` показывает `apiVersion` для ресурса.

Пример:

```
d8 k explain deployment
```

Вывод:

```
GROUP:      apps
KIND:       Deployment
VERSION:    v1

DESCRIPTION:
    Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
...
```

С помощью документации

Как искать в документации:

1. Найдите нужный ресурс (например, Deployment).
2. В заголовке будет указана API Group и версия. Пример для Deployment:

   ```
   apiVersion: apps/v1
   ```

   Здесь:

   - «API Group» — apps;
   - «Version» — v1.

DeleteSonarqubeProject

DeleteSonarqubeProject — удаляет проект из SonarQube.

Пример запроса

```
project: example-project
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
project	Да	Идентификатор (Project Key) проекта, который необходимо удалить	-

Учётные данные

- `token` — токен Sonarqube с типом User Token, сгенерированный пользователем, от имени которого будет запускаться выполнение действия.

DeleteSonarqubeProjects

DeleteSonarqubeProjects — удаляет один или несколько проектов из SonarQube.

Пример запроса

```
analyzedBefore: 2017-10-19T13:00:00+0200
onProvisionedOnly: 'false'
projects: example_project,another_project
q: example
qualifiers: TRK
```

Спецификация запроса

Название	Обязательность	Описание	Возможные значения	Примеры значе
analyzedBefore	Нет	Удалить все проекты, в которых последний анализ старше определённой даты	-	2017-10-19, 2017-10-19T13:0
onProvisionedOnly	Нет	Фильтровать проекты по значению `onProvisionedOnly`	true, false, yes, no	-
projects	Нет	Список ключей проектов, который необходимо удалить	-	my_project, anotl
q	Нет	Удалить все проекты, в названии или ключе которых содержится заданная подстрока	-	example
qualifiers	Нет	Фильтровать проекты по указанным квалификаторам	TRK, VW, APP	

Учётные данные

- `token` — токен Sonarqube с типом User Token, сгенерированный пользователем, от имени которого будет запускаться выполнение действия.

StartGitlabPipeline

StartGitlabPipeline — запускает выполнение пайплайна в GitLab.

Пример запроса

```
project_id: 0
ref: main
variables:
  - key: example-key
    value: example-value
```

Спецификация запроса

Название	Обязательность	Описание
project_id	Да	Идентификатор проекта, в котором необходимо запустить пайплайн
ref	Да	Название ветки, тег или SHA-хеш коммита, на котором будет запущен пайплайн
variables	Нет	Список переменных, которые необходимо передать в запускаемый пайплайн
variables.key	Да	Название переменной
variables.value	Да	Значение переменной

Учётные данные

- `token` — токен пользователя, от имени которого будет запускаться выполнение действия.

Примечание

Для выполнения действия необходимо наличие токена GitLab. Токен передаётся через механизм учётных данных и используется для аутентификации при вызове GitLab API (HTTP-заголовок `Private-Token`).

Действие осуществляет POST-запрос по URL: `/api/v4/projects/:id/pipeline`.

CreateVaultKubernetesAuthRole

CreateVaultKubernetesAuthRole — создаёт или обновляет роль аутентификации Kubernetes в HashiCorp Vault.

Пример запроса

```
mountPath: kubernetes
role: example
bound_service_account_names:
  - default
bound_service_account_namespaces:
  - default
optional:
  token_ttl: 1h
  token_max_ttl: 12h
  audience: vault
  token_policies:
    - default
```

Спецификация запроса

Название	Обязательность	Описание
mountPath	Обязательно	Путь монтирования Kubernetes auth backend в Vault (например, kubernetes)
role	Обязательно	Название роли, которая создаётся в Vault
bound_service_account_names	Обязательно	Список имён service account’ов, которым разрешён доступ через данную роль
bound_service_account_namespaces	Обязательно	Список неймспейсов (namespaces), в которых разрешён доступ через данную роль
optional	Необязательно	Дополнительные параметры роли (приведены в следующей таблице)

Поддерживаемые значения в optional:

Поле	Тип	Описание
token_ttl	string	Время жизни (TTL) токена, выданного при логине
token_max_ttl	string	Максимальное TTL токена
token_policies	[]string	Дополнительные политики, назначаемые при логине
audience	string	Значение JWT audience (aud), которое Vault ожидает от токена
token_period	string	Периодичность выдачи токена
token_explicit_max_ttl	string	Явный верхний предел TTL токена
token_num_uses	int	Ограничение на количество использований токена
token_type	string	Тип выдаваемого токена (например, service, batch)
alias_name_source	string	Источник alias name для identity
token_no_default_policy	bool	Исключение default policy из состава токена
token_bound_cidrs	[]string	Ограничение CIDR-диапазонов, откуда можно использовать выданный токен

Полный список поддерживаемых параметров приведён в официальной документации HashiCorp Vault.

Учётные данные

- `token` — Vault-токен, обладающий правами на создание/обновление ролей Kubernetes auth backend.

CreateNexusRepository

`CreateNexusRepository` — создаёт новый репозиторий любого поддерживаемого типа (maven, docker, npm и др.) в Nexus Repository Manager 3 с помощью REST API.
Параметры формата, типа и другие ключевые настройки полностью настраиваются и соответствуют Nexus API.

Пример запроса (Maven hosted)

```
description: |
  Maven hosted repo for internal Java build artifacts.
name: my-maven-repo
format: maven
type: hosted
online: true
storage:
  blobStoreName: default
  strictContentTypeValidation: true
  writePolicy: ALLOW
cleanup:
  policyNames:
    - maven-cleanup
maven:
  versionPolicy: RELEASE
  layoutPolicy: PERMISSIVE
```

Пример запроса (Docker group)

```
description: |
  Docker group repo aggregating hosted+proxy.
name: my-docker-group
format: docker
type: group
online: true
storage:
  blobStoreName: default
  strictContentTypeValidation: true
group:
  memberNames:
    - docker-hosted
    - docker-proxy
docker:
  v1Enabled: false
  forceBasicAuth: true
  httpPort: 5001
```

Спецификация запроса

Поле	Обязательность	Описание
`description`	Нет	Документация по назначению этого действия/репозитория. Не используется самим Nexus, только для UI
`name`	Да	Название создаваемого репозитория. Должно быть уникальным в рамках Nexus
`format`	Да	Формат (`maven` , `docker` , `npm` , `raw` и т. д.)
`type`	Да	Тип: `hosted` , `proxy` или `group`
`online`	Да	Доступен ли репозиторий (`true` / `false`)
`storage`	Да	Объект storage: `blobStoreName` , `strictContentTypeValidation` , `writePolicy`
`cleanup`	Нет	Привязанные политики очистки (`policyNames`)
`maven`	Для maven	Только для maven: `versionPolicy` , `layoutPolicy`
`proxy`	Для proxy	Прокси-репозиторий: `remoteUrl` , `contentMaxAge` , `metadataMaxAge`
`group`	Для group	Список включённых memberNames
`docker`	Для docker	docker-specific: `httpPort` , `v1Enabled` , `forceBasicAuth`
`component`	Очень редко	Только для некоторых нестандартных сценариев
`attributes`	Нет	Любые кастомные поля

Требования

- Используйте только те блоки (`maven` , `group` , `proxy` , `docker` и пр.), которые поддерживаются для вашего типа/формата.
- Для maven hosted обязательно `maven: {versionPolicy, layoutPolicy}` .
- Для group — обязательно `group.memberNames` .
- Для proxy — обязательно `proxy.remoteUrl` .
- Для docker — специфичные поля в `docker` .

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

DeleteNexusRepository

`DeleteNexusRepository` — удаляет существующий репозиторий из Nexus Repository Manager 3.

Пример запроса

```
name: my-repo-to-delete
```

Спецификация запроса

Поле	Обязательность	Описание
`name`	Да	Название репозитория, который требуется удалить

Алгоритм

- Выполняется `DELETE` по адресу `/service/rest/v1/repositories/{name}`, где `{name}` — это значение поля `name`.
- Если репозиторий найден и удалён — возвращается 204.
- Если не найден — возвращается ошибка 404.

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

CreateNexusPrivilege

CreateNexusPrivilege — создаёт новую привилегию в Nexus Repository Manager 3. Привилегии определяют права доступа к репозиториям и другим ресурсам Nexus.

Пример запроса (repository-view)

```
name: example-privilege
description: Example privilege description
type: repository-view
actions:
  - READ
  - BROWSE
format: maven2
repository: maven-releases
```

Пример запроса (repository-content-selector)

```
name: content-selector-privilege
description: Privilege with content selector
type: repository-content-selector
actions:
  - READ
format: maven2
repository: maven-releases
contentSelector: my-content-selector
```

Пример запроса (wildcard)

```
name: wildcard-privilege
description: Wildcard privilege
type: wildcard
pattern: nx-*
actions:
  - READ
```

Спецификация запроса

Поле	Обязательность	Описание
name	Да	Название создаваемой привилегии. Должно быть уникальным в рамках Nexus
description	Нет	Описание привилегии
type	Да	Тип привилегии: `repository-view`, `repository-content-selector`, `repository-admin`, `application`, `wildcard`
actions	Нет	Список действий, разрешённых привилегией (например, `READ`, `BROWSE`, `CREATE`, `UPDATE`, `DELETE`)
format	Нет	Формат репозитория (например, `maven2`, `docker`, `npm`). Используется для типов `repository-view`, `repository-content-selector`, `repository-admin`
repository	Нет	Название репозитория. Используется для типов `repository-view`, `repository-content-selector`, `repository-admin`
contentSelector	Нет	Название селектора контента. Обязателен для типа `repository-content-selector`. Если не указан или невалиден, тип автоматически преобразуется в `repository-view`
pattern	Нет	Шаблон для типа `wildcard`
domain	Нет	Домен для типа `application`
attributes	Нет	Дополнительные атрибуты в формате ключ-значение

Примечание

Для типа `repository-content-selector` селектор контента должен существовать в Nexus до создания привилегии. Если селектор контента не указан или невалиден, действие автоматически преобразует тип привилегии в `repository-view`.

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

AssignNexusPrivilege

AssignNexusPrivilege — назначает привилегии существующей роли в Nexus Repository Manager 3. Действие получает текущую конфигурацию роли и объединяет существующие привилегии с новыми.

Пример запроса

```
roleId: example-role
privileges:
  - example-privilege
  - another-privilege
```

Спецификация запроса

Поле	Обязательность	Описание
roleId	Да	Идентификатор роли, которой назначаются привилегии
privileges	Да	Список названий привилегий, которые необходимо назначить роли

Алгоритм работы

1. Получает текущую конфигурацию роли из Nexus.
2. Объединяет существующие привилегии роли с новыми привилегиями из запроса.
3. Обновляет роль с объединённым списком привилегий.

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

Примечание

Роль должна существовать в Nexus до назначения привилегий. Если роль не найдена, действие завершится с ошибкой. Все указанные привилегии также должны существовать в Nexus.

DeleteNexusPrivilege

DeleteNexusPrivilege — удаляет привилегию из Nexus Repository Manager 3.

Пример запроса

```
name: example-privilege
```

Спецификация запроса

Поле	Обязательность	Описание
name	Да	Название привилегии, которую требуется удалить

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

CreateNexusRole

CreateNexusRole — создаёт новую роль в Nexus Repository Manager 3. Роли объединяют привилегии и могут включать другие роли.

Пример запроса

```
id: example-role
name: Example Role
description: Example role description
privileges:
  - nx-repository-view-*-*-read
  - nx-repository-view-maven2-*-browse
roles: []
```

Спецификация запроса

Поле	Обязательность	Описание
id	Да	Уникальный идентификатор роли
name	Да	Название роли
description	Нет	Описание роли
privileges	Нет	Список названий привилегий, которые назначаются роли
roles	Нет	Список идентификаторов других ролей, которые включаются в данную роль

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

AssignNexusRole

AssignNexusRole — назначает роли существующему пользователю в Nexus Repository Manager 3. Действие получает текущую конфигурацию пользователя и объединяет существующие роли с новыми.

Пример запроса

```
userId: example-user
roles:
  - example-role
  - another-role
```

Спецификация запроса

Поле	Обязательность	Описание
userId	Да	Идентификатор пользователя, которому назначаются роли
roles	Да	Список идентификаторов ролей, которые необходимо назначить пользователю

Алгоритм работы

1. Получает текущую конфигурацию пользователя из Nexus.
2. Объединяет существующие роли пользователя с новыми ролями из запроса.
3. Обновляет пользователя с объединённым списком ролей.

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

Примечание

Пользователь должен существовать в Nexus до назначения ролей. Если пользователь не найден, действие завершится с ошибкой. Все указанные роли также должны существовать в Nexus.

DeleteNexusRole

DeleteNexusRole — удаляет роль из Nexus Repository Manager 3.

Пример запроса

```
id: example-role
```

Спецификация запроса

Поле	Обязательность	Описание
id	Да	Идентификатор роли, которую требуется удалить

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

CreateNexusUser

CreateNexusUser — создаёт нового пользователя в Nexus Repository Manager 3.

Пример запроса

```
userId: example-user
firstName: First
lastName: Last
emailAddress: user@example.com
password: password
status: active
roles:
  - nx-admin
```

Спецификация запроса

Поле	Обязательность	Описание
userId	Да	Уникальный идентификатор пользователя
firstName	Да	Имя пользователя
lastName	Да	Фамилия пользователя
emailAddress	Да	Email-адрес пользователя
password	Да	Пароль пользователя
status	Да	Статус пользователя: `active` или `disabled`
roles	Нет	Список идентификаторов ролей, которые назначаются пользователю при создании

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

DeleteNexusUser

DeleteNexusUser — удаляет пользователя из Nexus Repository Manager 3.

Пример запроса

```
userId: example-user
```

Спецификация запроса

Поле	Обязательность	Описание
userId	Да	Идентификатор пользователя, которого требуется удалить

Учётные данные

- `token` — строка base64(`admin:password`), используется как Basic Auth при запросах к Nexus.

Wait

Wait ожидает заданное число секунд; дополнительно может применяться случайная добавка к длительности (джиттер). Предназначено для использования в процессах в качестве элемента задержки, в том числе для ожидания применения результатов предыдущего действия.

Пример запроса

```
duration_seconds: 10
max_jitter_seconds: 0
description: "Waiting for release"
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
duration_seconds	Да	Базовая длительность ожидания в секундах (0–86400, т. е. до 24 часов)	-
max_jitter_seconds	Нет	Максимальная случайная добавка к ожиданию в секундах (0–N). 0 — джиттер отключён	0
description	Нет	Описание для отображения в логах и ответе действия	-

Fail

Fail эмулирует ошибку исполнения действия. Предназначен для использования в процессах в качестве отладочного элемента.

Пример запроса

```
fail: true
```

Спецификация запроса

Название	Обязательность	Описание	Значение по умолчанию
fail	Нет	Если `true`, действие завершается с ошибкой; если `false`, действие завершается успешно	`true`

Процессы

Обзор

Процессы — это механизм автоматизации сложных бизнес-сценариев, который позволяет создавать визуальные схемы выполнения действий с поддержкой условной логики, параллельного выполнения и обработки ошибок.

Основные концепции

Элементы процесса

Процесс состоит из различных типов элементов:

- «Начало» — точка входа в процесс.
- «Задача» — выполнение конкретного действия.
- «Эксклюзивный шлюз» — условное ветвление.
- «Параллельный шлюз» — объединение нескольких веток процесса в одну.
- «Цикл» — повторение части процесса фиксированное число раз или обход JSON-массива из хранилища.
- «Шаблон» — вычисление Go-шаблона и запись результата в хранилище процесса.
- «Примечание» — текстовый блок.
- «Ошибка» — немедленная остановка процесса со статусом `Failed`.
- «Конец» — завершение процесса.

Создание процесса

Основная информация

Для создания процесса перейдите в раздел «Самообслуживание» → «Процессы» и нажмите кнопку «Создать».

Заполните следующие поля:

- «Название» — название процесса.
- «Описание» — подробное описание назначения процесса.
- «Ресурс» — один или несколько ресурсов, для которых может запускаться процесс.
- «Владелец» — пользователь, ответственный за процесс.
- «Команда владелец» — команда, ответственная за процесс.
- «Теги» — теги для категоризации процесса.
- «Иконка» — иконка для отображения в интерфейсе.

Настройка процесса

Настройка процесса происходит в визуальном редакторе на вкладке «Конфигурация».

> Для настройки процесса необходимо сначала заполнить основные поля и создать его. После этого станут доступны вкладки «Конфигурация» и «Параметры».

Добавление элементов

1. Добавьте элемент, выбрав его тип на панели редактора.
2. Настройте параметры элемента в боковой панели (действие, условия и др.).
3. Свяжите элементы друг с другом.

Типы элементов

Задача

Каждая задача — это запуск конкретного действия, созданного ранее в разделе «Самообслуживание» → «Действия».

Эксклюзивный шлюз

Эксклюзивный шлюз позволяет настраивать ветвление процесса по условиям. По умолчанию проверяется статус предыдущей задачи, и в зависимости от него выполнение идёт по «успешной» или «неуспешной» ветке.

В качестве условий для эксклюзивного шлюза можно задать либо проверку статуса предыдущей задачи, либо сравнение значений из шаблонов:

- Для проверки статуса предыдущей задачи используйте конструкцию `{{ .prev_task.status }}`.

- Для проверки значения из хранилища — `{{ .store.<путь> }}`, в том числе вложенные ключи: `{{ .store.notification.status }}`.
- Внутри цикла — поля текущего элемента: `{{ .store._loop.item.scan_type }}`, флаги `{{ .store._loop.first }}` и `{{ .store._loop.last }}`.

Доступна проверка нескольких условий с комбинацией их результатов через оператор `AND` или `OR`.

Если проверка условий выполняется, процесс идёт по ветке «Истина» (зелёная); если нет — по ветке «Ложь» (красная). Порты «Истина» и «Ложь» можно размещать на любой стороне шлюза; связь можно перенести на другой порт.

Параллельный шлюз

Применяется для объединения нескольких веток в одну по заданным условиям.

В конфигурации параллельного шлюза можно настроить параметры ожидания:

- Ожидать выполнения всех входящих элементов для продолжения выполнения.
- Ожидать выполнения как минимум одного входящего элемента для продолжения выполнения.

При этом можно настроить, что именно считать «выполнением»:

- Только успешное завершение входящих в шлюз задач.
- Переход входящих задач в любой финальный статус (`Failed`, `Skipped` и т. д.).

Параллельный шлюз также может использоваться для разделения веток, если подключается после элементов «Эксклюзивный шлюз» или «Цикл», либо в качестве вспомогательного элемента для повышения читаемости схемы процесса.

Цикл

Элемент «Цикл» повторяет ветку процесса либо фиксированное число раз, либо по каждому элементу JSON-массива из хранилища. После завершения всех итераций активируется элемент, подключённый к выходу из цикла.

Режим цикла

- **Фиксированное число** — поле «Количество итераций» от 1 до 10000. На каждой итерации в `{{ .store._loop.item }}` доступен номер итерации (1, 2, 3…).
- **По коллекции** — поле «Шаблон коллекции»: Go-шаблон, который при входе в цикл должен дать JSON-массив. Обычно указывают путь к массиву, записанному предыдущей задачей: `{{ .store.notification.engagements }}`. Каждый элемент массива — одна итерация; текущий элемент доступен как `{{ .store._loop.item }}`.

Если массив пуст или ключ отсутствует, тело цикла не выполняется, процесс сразу переходит по связи «Выход из цикла».

Подробнее — в разделе «Хранилище процесса».

Исходящие связи

Из элемента «Цикл» должны выходить ровно две связи к разным элементам:

1. «Тело цикла» — от этого порта строится ветка, которая выполняется на каждой итерации.
2. «Выход из цикла» — по этой ветке выполнение продолжается после последней итерации.

Порты «Тело цикла» и «Выход из цикла» можно размещать на любой стороне элемента; связь можно перенести на другой порт.

Использование

Типовые сценарии:

- повторная проверка статуса внешней системы с фиксированным числом попыток;
- обход списка объектов из ответа API: задача записывает массив в хранилище, цикл в режиме «По коллекции» обрабатывает каждый элемент;
- вложенные циклы — контекст родителя доступен через `{{ .store._loop.parent }}`.

Шаблон

Элемент «Шаблон» вычисляет Go-шаблон без дополнительного вызова действия и записывает результат в хранилище как строку.

Настройте:

- «Тело шаблона» — текст с плейсхолдерами `{{ .store.* }}`, `{{ .process.* }}`, внутри цикла — `{{ .store._loop.* }}`;
- «Ключ в хранилище» — dot-path назначения, например `rendered_message`;
- «Подсказка формата» — как отображать значение при просмотре запуска (`text`, `markdown`, `html`, `json`); не влияет на содержимое хранилища.

Примеры и ограничения — в разделе «Хранилище процесса».

Таймер

Элемент **Таймер** приостанавливает выполнение процесса до наступления заданного момента времени, после чего активируется следующий элемент.

Пока процесс ожидает только срабатывания таймера (других активных задач нет), запуск переводится в статус «Ожидание» (`Wait`). На панели визуализации запуска отображается баннер с ориентировочным временем возобновления.

Исходящие связи

От элемента «Таймер» должна исходить ровно одна связь к следующему элементу. Подключение возможно только через левый (вход) и правый (выход) порты.

При повторном прохождении таймера в том же запуске (например, через цикл) момент срабатывания рассчитывается заново.

Режимы расписания

В конфигурации таймера выберите «Расписание»:

- «Задержка после входа в элемент» — фиксированная пауза с момента, когда процесс дошёл до таймера. Задайте «Задержка (секунды)» от 1 до 1 209 600 (14 суток).
- «Произвольное расписание» — срабатывание в заданное время по календарю в выбранном часовом поясе (IANA, например `Europe/Moscow`).

Для режима «Произвольное расписание» укажите «Шаблон»:

- «Определённый день недели» — «День недели», «Время суток» (час и минута), «Часовой пояс».
- «Определённое число каждого месяца» — «Число месяца» (1–31), «Время суток», «Часовой пояс». Если в месяце нет такого числа, используется последний день месяца.
- «Каждые N дней» — «Каждые N дней» (1–365), «Время суток», «Часовой пояс». Первое срабатывание — не ранее чем через N календарных дней от дня, когда процесс дошёл до таймера; при повторном входе в элемент расчёт выполняется снова.

«Время суток» задаётся в выбранном часовом поясе (поле «Часовой пояс»), если он отличается от пояса браузера.

Использование

Типовые сценарии: пауза перед повторной проверкой статуса внешней системы, отложенный запуск следующего шага, ожидание окна обслуживания или регулярного слота по календарю.

Примечание

Элемент Примечание предназначен для текстовых пояснений на схеме процесса: он не выполняется при запуске и не соединяется с другими элементами.

В конфигурации примечания можно задать:

- «Текст» — содержимое с поддержкой markdown разметки.
- «Цвет фона» — цвет заливки фона.
- «Цвет текста» — цвет текста.

Примечание всегда позиционируется за остальными элементами и может использоваться для визуального выделения частей процесса. Также это единственный элемент, размер которого можно изменить.

Ошибка

Элемент «Ошибка» принудительно переводит запущенный процесс в статус `Failed`: при достижении этого элемента выполнение всех действий процесса прерывается.

От элемента не должно исходить связей к следующим элементам.

Параметры процесса

На вкладке `Параметры` настраиваются параметры процесса, которые могут использоваться во всех действиях процесса.

Конфигурация и использование параметров процесса описаны в разделе «Шаблонизация».

Хранилище процесса

Хранилище процесса — JSON-объект для передачи данных между шагами одного запуска: идентификаторы, массивы, промежуточные результаты и тексты из элемента «Шаблон».

Полное описание правил записи, операций, контекста цикла и примеров сценариев — в разделе «Хранилище процесса».

Кратко:

- **Одно хранилище на запуск** — у каждого экземпляра процесса своё хранилище; его можно просмотреть на вкладке «Хранилище» при визуализации запуска.
- **Вложенные пути** — ключи задаются через точку: `notification.module_name`, `ctx.job.id`; поддерживаются индексы массивов: `items[0].status`.
- **Запись по правилам** — данные попадают в store после успешного выполнения действия (раздел «Обновление хранилища процесса») или при проходе элемента «Шаблон». Доступны операции записи строки и JSON, добавления, слияния и удаления, а также условие на каждое правило.
- **Чтение** — Go-шаблоны `{{ .store.<путь> }}` в конфигурации действий и условиях шлюзов (см. «Хранилище процесса»).

Если ключа нет в хранилище, шаблон `{{ .store.<путь> }}` завершит шаг с ошибкой.

Запуск процесса

Ручной запуск

1. Перейдите к сущности, для которой нужно запустить процесс.
2. В меню сущности выберите «Запустить процесс».
3. Выберите нужный процесс из списка.
4. Заполните параметры процесса.
5. Нажмите «Запустить».

Параметры запуска

При запуске процесса доступны:

- «Общие параметры процесса» — параметры, определённые в конфигурации процесса.
- «Параметры действий» — параметры для каждого действия в процессе.
- «Переменные окружения» — дополнительные переменные для выполнения.

Управление выполнением

Статусы процесса

Процесс может находиться в следующих статусах:

- «Создан» — процесс создан, но не запущен.
- «Выполняется» — процесс находится в процессе выполнения.
- «Приостановлен» — выполнение процесса приостановлено.
- «Завершен» — процесс успешно завершен.
- «Неудачно» — процесс завершился с ошибкой.
- «Отменен» — выполнение процесса было отменено.

Управление выполнением

Для активных процессов доступны следующие операции:

- «Приостановить» — временно остановить выполнение.
- «Возобновить» — продолжить выполнение после приостановки.
- «Остановить» — полностью остановить выполнение.
- «Принудительный перезапуск» — перезапустить процесс с начала.

Отслеживание состояния

В разделе «Запуски процессов» можно просмотреть:

- Список всех запусков процессов для сущности.
- Детальную информацию о каждом запуске.
- Логи выполнения действий.
- Статус каждого элемента процесса.
- Хронологию выполнения процесса.
- Содержимое хранилища процесса на вкладке «Хранилище».

Таймлайн

На панели визуализации запуска процесса доступна вкладка «Таймлайн», на которой отображается хронология выполнения процесса: для каждого элемента, который участвовал в текущем запуске, показаны время начала и окончания, длительность и статус.

Лог процесса

Для каждого запуска процесса доступен подробный лог выполнения: панель на схеме запуска, отдельное окно или док внизу экрана. Лог и JSON хранилища можно скачать с панели диалога запуска процесса.

Примеры использования

Создание проекта с настройкой

1. «Начало» — запуск процесса.

2. «Задача» — создание проекта в GitLab.
3. «Эксклюзивный шлюз» — проверка успешности создания.
4. «Задача» (при успехе) — настройка переменных проекта.
5. «Задача» (при ошибке) — отправка уведомления об ошибке.
6. «Конец» — завершение процесса.

Развертывание приложения

1. «Начало» — запуск процесса развертывания.
2. «Параллельный шлюз» — разделение на ветки.
3. «Задача» (ветка 1) — создание namespace в Kubernetes.
4. «Задача» (ветка 2) — создание секретов в Vault.
5. «Параллельный шлюз» — ожидание завершения обеих веток.
6. «Задача» — развертывание приложения.
7. «Конец» — завершение процесса.

Ограничения

- Процессы не могут содержать более 100 элементов.
- Максимальное время выполнения процесса — 24 часа.
- Количество одновременных запусков процесса ограничено настройками системы.
- Некоторые действия могут быть недоступны для использования в процессах.

Хранилище процесса

Хранилище процесса — это JSON-объект, общий для одного запуска процесса. Через него задачи передают друг другу данные: идентификаторы, массивы для обхода в цикле, промежуточные результаты и сформированные тексты.

Запись выполняется **только по правилам** в конфигурации действий и элементов «Шаблон». Чтение — через Go-шаблоны `{{ .store.<путь> }}` в теле запроса, условиях шлюзов и других полях с поддержкой шаблонизации.

Модель данных

- **Одно хранилище на запуск** — у каждого экземпляра процесса своё хранилище; оно сохраняется между шагами и доступно при просмотре запуска.
- **Вложенные пути** — ключи задаются dot-path без ведущей точки: `notification.module_name`, `ctx.job.id`. Промежуточные объекты создаются автоматически.
- **Индексы массивов** — в пути можно указывать элемент массива: `items[0].status`, `branches[2].name`.
- **Типы значений** — в зависимости от операции в хранилище попадают строки, числа, объекты и массивы JSON. Операции «Записать строку» и «Добавить строку» всегда работают со строками; операции с суффиксом JSON сохраняют разобранную структуру.

> Служебный ключ `_loop` заполняется движком процесса во время выполнения цикла. Не записывайте в `_loop` вручную правилами действий — используйте его только для чтения в шаблонах.

Правила обновления хранилища

Правила настраиваются в действии: «Конфигурация» → «Обновление» → «Обновление хранилища процесса». Они выполняются **после успешного завершения** действия, **в порядке списка**. Каждое следующее правило видит изменения, внесённые предыдущими правилами того же действия.

В конфигурации задачи процесса правила отображаются в режиме только для чтения; для изменения откройте действие из боковой панели задачи.

Поля правила

Поле	Обязательность	Описание
Условие	Нет	Go-шаблон. Пустое значение — правило выполняется всегда. После рендеринга результат должен быть `true`, `false`, `1` или `0`; иначе правило пропускается
Операция	Да	Способ изменения значения по пути «Цель»
Цель	Да	Dot-path в хранилище без ведущей точки
Источник	Да*	Go-шаблон значения. Для операции «Удалить» не используется

Операции

Операция	Результат
Записать строку	Заменить значение по пути текстовой строкой из источника; JSON не разбирается
Записать JSON	Заменить значение разобранным JSON (объект, массив, число, булево) из шаблона источника
Добавить строку	Дописать текст к существующей строке по пути; если ключа нет — создаётся строка. Не применяется к числам и массивам
Добавить JSON	Добавить один JSON-элемент в массив по пути; если массива нет — создаётся пустой массив
Слить JSON (поверхностно)	Объединить JSON-объект из источника с объектом по пути: ключи верхнего уровня перезаписываются, вложенные объекты заменяются целиком
Удалить	Удалить ключ по пути

Для операций **Записать JSON**, **Добавить JSON** и **Слить JSON** удобно использовать функцию `toJSON` и прямую ссылку на поле ответа, например `{{ .response.items }}` — платформа подставит значение без лишней сериализации.

Примеры правил

Сохранить идентификатор из ответа действия:

Условие	Операция	Цель	Источник
	Записать строку	`deploy.job_id`	`{{ .response.id }}`

Записать массив для последующего цикла:

Условие	Операция	Цель	Источник
	Записать JSON	`notification.engagements`	`{{ toJSON .response.engagements }}`

Дописать строку в лог:

Условие	Операция	Цель	Источник
	Добавить строку	`run.log`	`step {{ .store._loop.index }}: ok\n`

Обновить только часть объекта:

Условие	Операция	Цель	Источник
	Слить JSON (поверхностно)	`meta`	`{{ toJSON (dict "status" "ready" "updated_by" .entity.slug) }}`

Выполнить правило только на первой итерации цикла:

Условие	Операция	Цель	Источник
`{{ .store._loop.first }}`	Записать строку	`run.started_at`	`{{ now }}`

Удалить временный ключ после использования:

Условие	Операция	Цель	Источник
	Удалить	`temp.payload`	

Панель «Пути хранилища»

В редакторе правил действия доступна панель **Пути хранилища**: она показывает ключи контекста цикла (`_loop.item` , `_loop.index` и др.).

Чтение из хранилища

В шаблонах процесса используйте:

```
{{ .store.<путь> }}
```

Для вложенных полей путь совпадает с «Целью» в правилах записи:

```
{{ .store.notification.module_name }}
{{ .store.items[0].id }}
```

Параметры запуска процесса доступны отдельным контекстом:

```
{{ .process.<идентификатор_параметра> }}
```

Подробнее о контекстах шаблонизации — в разделе «Шаблонизация».

> ⚠ Если ключа нет в хранилище (действие ещё не выполнялось, правило не сработало или условие было ложным), шаблон `{{ .store.<путь> }}` завершит шаг с ошибкой. Убедитесь, что предшествующие задачи записали нужные данные.

Контекст цикла `_loop`

Пока выполняется тело цикла, в хранилище доступен объект `_loop`:

Ключ	Описание
`_loop.item`	Текущий элемент коллекции (режим «По коллекции») или номер итерации, начиная с 1 (режим «Фиксированное число»)
`_loop.index`	Индекс итерации, начиная с 0
`_loop.total`	Общее число итераций
`_loop.first`	`true` на первой итерации
`_loop.last`	`true` на последней итерации
`_loop.parent`	Контекст родительского цикла при вложенных циклах
`_loop.loop_element_uuid`	UUID элемента «Цикл»

Примеры в шаблонах:

```
{{ .store._loop.item.name }}
{{ .store._loop.item.scan_type }}
{{ .store._loop.index }}
```

В условиях эксклюзивного шлюза внутри цикла:

```
{{ eq .store._loop.item.branch .store.current_branch_tag }}
```

Цикл по коллекции из хранилища

Элемент «Цикл» поддерживает два режима (поле «Режим цикла»):

1. **Фиксированное число** — тело выполняется заданное количество раз (1–10000). В `_loop.item` — номер итерации (1, 2, 3…).
2. **По коллекции** — при входе в цикл вычисляется Go-шаблон «Шаблон коллекции»; каждый элемент полученного **JSON-массива** — одна итерация.

Шаблон коллекции

Рекомендуемый вариант — путь к массиву, уже записанному в хранилище предыдущей задачей:

```
{{ .store.notification.engagements }}
```

Альтернатива — собрать массив в шаблоне:

```
{{ toJSON .store.modules }}
```

Граничные случаи:

- Пустой массив или отсутствующий ключ — 0 итераций, переход по связи «Выход из цикла»; в логе процесса появится предупреждение.
- Значение по пути не является JSON-массивом — ошибка валидации или выполнения цикла.

Сценарий: обход списка из API

1. **Задача** «Получить engagements» — правило **Записать JSON**, цель `notification.engagements`, источник `{{ toJSON .response.engagements }}`.
2. **Цикл**, режим **По коллекции**, шаблон коллекции `{{ .store.notification.engagements }}`.
3. **Задача** в теле цикла — в теле запроса использовать `{{ .store._loop.item.id }}`, `{{ .store._loop.item.name }}`.
4. **Эксклюзивный шлюз** в теле — условие по полю элемента, например левое значение `{{ .store._loop.item.status }}`, правое `active`.
5. Связь **Выход из цикла** — продолжение после обхода всех элементов.

Элемент «Шаблон»

Элемент «Шаблон» вычисляет Go-шаблон при проходе процесса и записывает **строку** в хранилище. JSON из результата не разбирается — в store сохраняется текст как есть.

Поле	Описание
Тело шаблона	Go-шаблон; доступны `{{ .store.* }}`, `{{ .process.* }}`, внутри цикла — `{{ .store._loop.* }}`
Ключ в хранилище	Dot-path назначения, например `rendered_message` или `notification.summary`
Подсказка формата	Подсказка при просмотре запуска (`text`, `markdown`, `html`, `json`); на запись не влияет

Пример тела:

```
## Отчёт по {{ .store._loop.item.name }}

Статус: {{ .store._loop.item.status }}
Владелец: {{ .store._loop.item.owner }}
```

Последующие задачи читают результат через `{{ .store.rendered_message }}` или соответствующий путь.

Просмотр хранилища при выполнении

На экране визуализации запуска процесса откройте вкладку **Хранилище**:

- **Дерево** — навигация по вложенным ключам; справа — тип и значение выбранного узла.
- **JSON** — полное содержимое хранилища в формате JSON.

Доступны копирование значения, обновление данных и скачивание JSON запуска (кнопка на панели диалога запуска).

В боковой панели задачи на схеме запуска отображаются **Правила обновления хранилища** выбранного действия — удобно сверять фактическое содержимое store с конфигурацией.

Лог процесса и отладка

На панели визуализации доступны:

- вкладка **Лог выполнения** (в том числе в отдельном окне или снизу схемы);
- **Скачать лог** — текстовый лог запуска;
- сообщения о пустой коллекции цикла и пропущенных правилах хранилища.

При ошибке шаблона или невалидном JSON в правиле с операцией JSON соответствующее правило может быть пропущено (с предупреждением в логе), остальные правила того же действия продолжают выполняться.

Связанные разделы

- Обзор — элементы схемы, запуск и управление.
- Обновление хранилища в действиях — где включаются правила.
- Шаблонизация — синтаксис Go-шаблонов и контексты.

Источники данных

Обзор

Источники данных — механизм Deckhouse Development Platform (DDP), который позволяет синхронизировать информацию из внешних инфраструктурных систем (например, GitLab или Kubernetes) и преобразовывать её в параметры той или иной сущности.

Также источники данных позволяют:

- Создавать новые сущности.
- Удалять сущности, которые были удалены из внешних инфраструктурных систем.
- Создавать связи между сущностями.

Каждый источник данных привязан к определенному ресурсу, и, соответственно, может оперировать сущностями только данного ресурса. При этом, связи могут создаваться между сущностями привязанного ресурса и любыми другими сущностями.

Типы источников данных

- Встроенные (BuiltIn) — источники данных, для которых логика опроса инфраструктурных систем поставляется в рамках DDP.
- Вебхуки (Webhooks) — механизм приёма данных во входящих POST-запросах от внешних систем, подробнее в разделе «Вебхуки».

Параметры источников данных

Некоторые типы источников данных поддерживают настраиваемые параметры, позволяющие дополнительно конфигурировать их поведение. Подробная информация о параметрах для каждого типа источника данных приведена в разделе «Типы источников данных».

Правила источников данных

Для каждого источника данных администратором системы настраиваются правила, по которым будут:

- Создаваться новые сущности.
- Создаваться новые связи.
- Производиться поиск соответствующих сущностей для обновления их параметров.
- Производиться поиск сущностей для удаления.

Правила описываются в формате `"source": "target"`. Для описания правил используется синтаксис Go template.

В DDP реализованы 5 типов правил:

- Правила фильтрации (filter rules).
- Правила создания (create rules).
- Правила сопоставления (match rules).
- Правила обновления (update rules).
- Правила создания связей (create relation rules).

Правила фильтрации

После получения данных из внешней системы применяются правила фильтрации: каждое следующее правило обрабатывает уже отфильтрованный набор элементов.

В диалоге редактирования источника данных откройте вкладку «Правила». В блоке «Правила фильтрации» для каждого правила задаются три поля:

1. Действие — оставить только элементы, у которых значение выбранного поля совпало с регулярным выражением (вариант «Оставить только совпадения»), или убрать из набора такие элементы (вариант «Убрать совпадения»).
2. Источник — путь к полю в структуре данных в формате Go template.
3. Условие — регулярное выражение для проверки значения этого поля.

Если правил нет, дальше по цепочке проходят все полученные элементы. Если действие не выбрано, подставляется вариант «Оставить только совпадения».

Чтобы оставить только записи с нужным значением поля (например, чтобы в поле `name` значение начиналось с `first`), выберите «Оставить только совпадения». В источнике укажите выражение для этого поля, например `{{ .name }}`, в условии — регулярное выражение для подходящих значений. Чтобы не обрабатывать записи, которые совпадают с шаблоном, выберите «Убрать совпадения» и задайте шаблон в условии.

> Если в условии указано невалидное регулярное выражение, правило не выполняется; ошибка попадает в лог синхронизации и обработка источника данных продолжается.

Правила создания

Правила создания определяют, каким образом информация, полученная от бэкенда источника данных, будет использоваться при создании новых сущностей.

Например, при следующей спецификации одного из ресурсов внешней инфраструктурной системы, полученного через источник данных:

```
...
metadata:
  name: example
  namespace: default
...
```

и следующих правилах создания:

```
"match": [
  {
    "source": "{{ .metadata.name }}",
    "target": "Name"
  },
  {
    "source": "{{ .metadata.namespace }}-{{ .metadata.name }}",
    "target": "Slug"
  }
]
```

после синхронизации источника данных будет создана сущность, название (Name) которой будет иметь значение `example`, а идентификатор (Slug) — значение `default-example`.

Правила сопоставления

Правила сопоставления определяют, каким образом производить поиск существующих, либо только что созданных источником данных сущностей для обновления их параметров.

Правила обновления

Правила обновления определяют, какие параметры сущности автоматически заполняются на основе спецификации соответствующего ресурса из внешней системы. Сущность, параметры которой необходимо заполнить, определяется на основе правил сопоставления.

Импорт из спецификации

> Экспериментальный функционал

С использованием механизма импорта, правила обновления и параметры обновляемого ресурса можно сформировать автоматически по JSON, YAML или OpenAPI-спецификации данных.

Настройка в интерфейсе

1. На вкладке «Основная информация» выберите ресурс, к которому будет привязан источник данных.
2. В диалоге редактирования источника данных откройте вкладку «Правила».
3. В блоке «Правила обновления» нажмите кнопку «Импортировать из спецификации».

Шаги по настройке

1. «Загрузка данных» — выберите формат (автоопределение, JSON, YAML, OpenAPI 3.x, Swagger 2.0) и вставьте содержимое спецификации в поле «Данные». Если в спецификации несколько схем, на следующем шаге в поле «Выберите схему для импорта» можно выбрать схему для маппинга.
2. «Настройка маппинга» — в таблице отображаются столбцы «Путь в данных», «Тип в источнике», «Идентификатор параметра», «Название параметра», «Тип параметра». Можно редактировать идентификатор, название и тип параметра, отметить галочками строки для применения. Строки, для которых в ресурсе уже есть параметр с таким идентификатором или названием, помечаются меткой «Существует».
3. «Применение» — нажмите «Применить»: в блок «Правила обновления» источника данных добавятся сформированные правила. Новые параметры ресурса будут созданы при сохранении источника данных.

Поведение системы при импорте

1. «Автосоздание параметров ресурса» — для полей спецификации, по которым в ресурсе ещё нет параметра, предлагаются новые параметры; они добавляются в ресурс при сохранении источника данных.
2. «Автозаполнение правил обновления» — по каждому выбранному полю создаётся правило: источник задаётся выражением Go template по пути в данных (например, `{{ .metadata.name }}`), параметр — идентификатор параметра ресурса. После нажатия «Применить» эти правила появляются в блоке «Правила обновления» на вкладке «Правила» источника данных.

Поддерживаемые форматы

Автоопределение, JSON, YAML, OpenAPI 3.x, Swagger 2.0. Типы из спецификации (string, integer, array, object и т.д.) сопоставляются с типами параметров DDP; при необходимости тип можно изменить на шаге «Настройка маппинга».

Правила создания связей

Правила создания связей определяют, какие связи создаются для сущности, соответствующей правилам сопоставления. При задании правил создания связей заполняется:

- Связь ресурса.
- Поле из спецификации ресурса внешней инфраструктурной системы, на основании которого выполняется поиск сущности для установления связи.

Например, при следующей спецификации одного из ресурсов внешней инфраструктурной системы, полученного через источник данных:

```
...
metadata:
  name: example
  namespace: default
...
```

и следующем правиле создания связей:

```
"createRelations": [
  {
    "resourceRelationUuid": "<12345678-1234-5678-1234-567812345678>",
    "parentEntitySlug": "{{ .metadata.namespace }}",
    "childEntitySlug": ""
  }
]
```

будет:

- Если указан идентификатор родительской сущности (`parentEntitySlug`), поиск выполняется только среди сущностей родительского ресурса.
- Если указан идентификатор дочерней сущности (`childEntitySlug`), поиск выполняется только среди сущностей дочернего ресурса.
- В рамках одного правила необходимо указывать либо родительский, либо дочерний идентификатор.
- Если заданы оба идентификатора, поиск выполняется только по родительскому ресурсу.

Если в родительском ресурсе существует сущность с идентификатором `default`, создаётся связь между этой сущностью и сущностью ресурса, к которому привязан источник данных и которая соответствует правилам сопоставления.

Синхронизация

Каждый источник данных имеет несколько параметров настройки синхронизации:

- Периодическая синхронизация — включение или выключение периодической синхронизации.
- Периодичность запуска — cron-выражение, определяющее интервал запуска периодической синхронизации.

При включенном параметре «Периодическая синхронизация» источник данных будет автоматически обновлять информацию из внешних инфраструктурных систем с интервалом, определенным в параметре «Периодичность запуска».

Также каждый источник данных может быть синхронизирован вручную в любой момент времени через веб-интерфейс (пункт «Синхронизировать» в меню источника данных), либо через API.

Удаление сущностей

Каждый источник данных имеет несколько параметров удаления сущностей:

- Удаление несуществующих сущностей.
- Удаление только сущностей, созданных этим источником данных.

При включении параметра «Удаление несуществующих сущностей» источник данных будет сравнивать список ресурсов, полученных из внешней инфраструктурной системы и список существующих сущностей ресурса. При обнаружении в DDP сущностей, которые отсутствуют в списке полученных, они будут удалены из DDP.

При включении параметра «Удаление только сущностей, созданных этим источником данных» источник данных будет дополнительно сравнивать, каким образом была создана сущность, которую необходимо удалить. Если сущность была создана этим источником данных, то она будет удалена, если нет, то источник данных перейдёт к обработке следующей сущности в списке без удаления текущей.

Поиск создателя (origin) сущности производится по двум полям в спецификации сущности:

```
{
    "origin": {
        "type": "<datasource | manual | webhook | action>",
        "uuid": "12345678-1234-5678-1234-567812345678>"
    }
}
```

Логирование запусков

Для каждого запуска источника данных создается запись в БД, содержащая полный лог выполнения. Максимальное количество подобных записей регулируется параметром «максимальное количество записей» в конфигурации источника данных. При достижении максимального количества старые записи будут удаляться.

Параметр `datasources.logging.enabled` в конфигурационном файле DDP позволяет выводить в stdout, либо скрывать логи синхронизации источника данных. При значении `true` логи будут выводиться в stdout, при значении `false` логи выводиться не будут.

> Записи в БД с полным логом запуска источника данных создаются независимо от значения параметра `datasources.logging.enabled`.

Системная учётная запись

При настройке источника данных необходимо выбрать системную учётную запись для запуска синхронизации.

Реквизиты данной учётной записи будут использоваться источником данных для доступа к опрашиваемой инфраструктурной системе.

Помимо системной учётной записи необходимо указание того, какие ее реквизиты будут использованы при запуске источника данных. Для этого в секции «Безопасность / Учётные данные» добавляются новые учётные данные с определенным идентификатором.

Значение данного идентификатора может быть подставлено в конфигурацию источника данных с использованием синтаксиса Go template.

Например, если в заголовке для источника данных необходимо передать секретное значение с идентификатором `token`, то оно может быть указано в следующем виде:

```
{{ .credentials.token }}
```

Типы источников данных

DefectdojoProducts

Источник данных типа `DefectdojoProducts` возвращает список продуктов в Defectdojo.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис DefectDojo».

Спецификация ответа

Платформа выполняет GET-запрос по URL: `/api/v2/products`. Возвращаются все доступные значения. Спецификация ответа.

Конфигурация

- `URL` — URL DefectDojo в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

GitlabGroups

Источник данных типа `GitlabGroups` возвращает список групп в GitLab.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис GitLab».

Спецификация ответа

Платформа выполняет GET-запрос к API GitLab по URL: `/api/v4/groups`. Платформа возвращает все доступные значения. Спецификация ответа.

Конфигурация

- `URL` — URL GitLab в формате `https://gitlab.com`, без части `/api/v4`.

Параметры

Настраиваемые параметры отсутствуют.

GitlabProjects

Источник данных типа `GitlabProjects` возвращает список проектов в GitLab.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис GitLab».

Спецификация ответа

В зависимости от конфигурации параметров, платформа выполняет GET-запрос к API GitLab и возвращает соответствующую спецификацию.

Если значение параметра `all` равно `true`, выполняется GET-запрос по URL: `/api/v4/projects`. Платформа возвращает все доступные значения. Спецификация ответа.

Если значение параметра `all` равно `false`, выполняется GET-запрос по URL: `/api/v4/groups/:id/projects`. Платформа возвращает все доступные значения. Спецификация ответа.

Если значение параметра `tags` равно `true`, платформа дополнительно получает git-теги. Для получения git-тегов выполняется GET-запрос по URL: `/api/v4/projects/:id/repository/tags`. Платформа получает список всех git-тегов и расширяет спецификацию ответа полем `ddp_repository_tags`, которое соответствует спецификации ответа list-project-repository-tags.

Конфигурация

- `URL` — URL GitLab в формате `https://gitlab.com`, без части `/api/v4`.

Параметры

Название	Обязательность	Описание	Возможные значения	По умо
all	Опционально	В явном виде указывает, что необходимо собирать репозитории всех групп, к которым есть доступ	true, false	false
group_ids	Обязательно, если `all` в значении `false`	Источник данных будет собирать проекты групп с указанным ID. ID групп указываются через запятую	Пример: 1001,1002	-
tags	Опционально	Платформа дополнительно получает список всех git тегов и расширяет спецификацию ответа полем `ddp_repository_tags`, которое соответствует спецификации ответа list-project-repository-tags	true, false	false
include_subgroups	Опционально	Если указан параметр `group_ids`, то параметр `include_subgroups` определяет, собирать ли проекты подгрупп указанных групп	true, false	false
tags_order_by	Опционально	Поле для сортировки тегов. Описание параметра — в GitLab Tags API	`updated`, `name`, `version`	upd
tags_sort	Опционально	Направление сортировки тегов. Описание параметра — в GitLab Tags API	`asc`, `desc`	des
tags_search	Опционально	Строка для поиска тегов по имени. Описание параметра — в GitLab Tags API	Строка	-

HarborArtifacts

Источник данных типа `HarborArtifacts` собирает информацию о всех артефактах в Harbor.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Harbor».

Спецификация ответа

Платформа получает список всех проектов и репозиториев, которые содержатся в этих проектах, затем по каждому из доступных проектов выполняется GET-запрос к API Harbor: `/api/v2.0/projects/{project_name}/repositories/{repository_name}/artifacts`. Спецификация ответа доступна в интерфейсе Harbor.

Конфигурация

- `URL` — URL Harbor в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

HarborProjects

Источник данных типа `HarborProjects` собирает информацию о всех проектах в Harbor.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Harbor».

Спецификация ответа

Платформа получает список всех доступных проектов, выполняя GET-запросы к API Harbor: `/api/v2.0/projects`. Спецификация ответа доступна в интерфейсе Harbor.

Конфигурация

- `URL` — URL Harbor в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

HarborRepositories

Источник данных типа `HarborRepositories` собирает информацию о всех репозиториях в Harbor.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Harbor».

Спецификация ответа

Платформа получает список всех проектов, затем получает список всех репозиториев в каждом из проектов, выполняя GET-запросы к API Harbor: `/api/v2.0/projects/{project_name}/repositories`. Спецификация ответа доступна в интерфейсе Harbor.

Конфигурация

- `URL` — URL Harbor в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

HarborTags

Источник данных типа `HarborTags` собирает информацию о всех тегах у артефактов в Harbor.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Harbor».

Спецификация ответа

Платформа получает список всех проектов и репозиториев, которые содержатся в этих проектах, затем по каждому из доступных проектов выполняется GET-запрос к API Harbor: `/api/v2.0/projects/{project_name}/repositories/{repository_name}/artifacts`. Затем происходит сбор всех тегов по всем артефактам (поле `tags`) и результат возвращается в виде массива. Спецификация ответа доступна в интерфейсе Harbor.

Конфигурация

- `URL` — URL Harbor в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

HelmReleases

Источник данных типа `HelmReleases` возвращает список всех HelmReleases в кластере Kubernetes.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Kubernetes».

Аутентификация в Kubernetes описана в официальной документации.

Спецификация ответа

Платформа возвращает все HelmReleases в кластере Kubernetes. Спецификация:

```
[
  {
    "name": "string",              // Название.
    "info": {
      "first_deployed": "string",  // Дата и время первого деплоя.
      "last_deployed": "string",   // Дата и время последнего деплоя.
      "deleted": "string",         // Дата и время удаления (может быть пустой
строкой).
      "description": "string",     // Описание.
      "status": "string",          // Статус.
      "notes": "string"            // Заметки.
    },
    "chart": {
      "metadata": {
        "name": "string",          // Название чарта.
        "home": "string",          // Домашняя страница.
        "sources": "array",        // Массив строк с URL-адресами источников.
        "version": "string",       // Версия чарта.
        "description": "string",   // Описание чарта.
        "keywords": "array",       // Массив строк с ключевыми словами.
        "maintainers": "array",    // Массив объектов с информацией о мейнтейнерах.
        "icon": "string",          // URL иконки.
        "apiVersion": "string",    // Версия API.
        "appVersion": "string"     // Версия приложения.
      },
      "templates": [               // Массив объектов с шаблонами.
        {
          "name": "templates/NOTES.txt",
          "data": "..."
        },
        {
          "name": "templates/_helpers.tpl",
          "data": "..."
        }
        // ... другие шаблоны.
      ],
      "values": "object",          // Объект с доступными настройками Helm-чарта.
      "schema": "null",            // Схема (может быть null).
      "files": [                   // Массив объектов с файлами.
        {
          "name": ".helmignore",
          "data": "..."
        },
        {
          "name": "LICENSE",
          "data": "..."
        }
        // ... другие файлы.
```

```
      ]
    },
    "config": {        // Текущая конфигурация.
      "caSecretName": "string",
      "cache": {
        "enabled": "boolean",
        "expireHours": "integer"
      }
      // ... другие настройки.
    },
    "manifest": "string", // Отрендеренные манифесты.
    "version": "integer", // Версия HelmRelease.
    "namespace": "string" // Неймспейс, где развёрнут релиз.
  },
  // ... другие ресурсы типа HelmRelease.
]
```

Конфигурация

- `URL` — URL Kubernetes API в формате `https://api.example.com`.

Параметры

Настраиваемые параметры отсутствуют.

KafkaAcls

Источник данных типа `KafkaAcls` собирает информацию о доступных ACL.

Авторизация

Платформа поддерживает аутентификацию в Kafka с помощью SASL/PLAIN, SASL/SCRAM.

Спецификация ответа

Платформа запрашивает информацию о настроенных ACL в Kafka. Полученные данные предоставляются в следующем формате:

```
[
  {
    "Cluster": "string",              // Название кластера Kafka.
    "ResourceType": "string",         // Тип ресурса (TOPIC, GROUP и т. д.).
    "ResourceName": "string",         // Название ресурса.
    "PatternType": "string",          // Тип паттерна (LITERAL, PREFIXED и т. д.).
    "Principal": "string",            // Principal пользователя (например:
"User:Alice").
    "Host": "string",                 // Хост (обычно "*" для любого хоста).
    "Operation": "string",            // Операция (READ, WRITE, DESCRIBE и т. д.).
    "PermissionType": "string"        // Тип разрешения (ALLOW, DENY).
  },
  // ... другие записи ACL.
]
```

Конфигурация

- `URL` — URL Kafka в формате `example.com`.

Параметры

Название	Обязательность	Описание	Возможные значения
SecurityProtocol	Обязательно	Протокол для подключения к Kafka — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
SaslMechanism	Опционально	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
User	Обязательно	Имя пользователя для подключения к Kafka	-
Pass	Обязательно	Пароль пользователя для подключения к Kafka	-

KafkaBrokers

Источник данных типа `KafkaBrokers` собирает информацию о доступных брокерах.

Авторизация

Платформа поддерживает аутентификацию в Kafka с помощью SASL/PLAIN, SASL/SCRAM.

Спецификация ответа

Платформа осуществляет несколько запросов к Kafka с целью получения сведений о доступных брокерах. Полученные данные предоставляются в следующем формате:

```
{
  "Cluster": "string",          // Название кластера Kafka (из
kafkaBrokerMetadata.Cluster).
  "Leader": "boolean",          // Является ли брокер лидером-контроллером (true/
false).
  "NodeID": "number",           // Уникальный ID брокера (из broker.NodeID).
  "Port": "number",             // Порт брокера (из broker.Port).
  "Host": "string",             // Хост брокера (из broker.Host).
  "Rack": "string|null",        // Рек (зона доступности) брокера (из broker.Rack).
  "Configs": {                  // Конфигурации брокера.
    {
      "Name": "number",         // Уникальный ID брокера.
      "Configs": [              // Массив отдельных конфигураций.
        {
          "Key": "listeners",       // Ключ конфигурации.
          "Value": "CLIENT://:9092,INTERNAL://:9094", // Значение конфигурации.
          "Source": "STATIC_BROKER_CONFIG" // Источник конфигурации.
        },
        {
          "Key": "log.retention.hours", // Ключ конфигурации.
          "Value": 168,                 // Значение конфигурации.
          "Source": "DEFAULT_CONFIG"    // Источник конфигурации.
        },
        {
          "...": "..."                  // Другие параметры.
          // Полный список параметров — в официальной документации:
          // https://kafka.apache.org/documentation/#brokerconfigs.
        }
      ]
    }
  }
}
```

Конфигурация

- `URL` — URL Kafka в формате `example.com`.

Параметры

Название	Обязательность	Описание	Возможные значения
SecurityProtocol	Обязательно	Протокол для подключения к Kafka — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
SaslMechanism	Опционально	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
User	Обязательно	Имя пользователя для подключения к Kafka	-
Pass	Обязательно	Пароль пользователя для подключения к Kafka	-

KafkaTopics

Источник данных типа `KafkaTopics` собирает информацию о доступных топиках в Kafka.

Авторизация

Платформа поддерживает аутентификацию в Kafka с помощью SASL/PLAIN, SASL/SCRAM.

Спецификация ответа

Платформа осуществляет несколько запросов к Kafka с целью получения сведений о доступных топиках. Полученные данные предоставляются в следующем формате:

```
{
  "Cluster": "string",           // Название кластера Kafka.
  "Topic": "string",             // Название топика.
  "ID": "string",                // Уникальный идентификатор топика.
  "IsInternal": "boolean",       // Является ли топик внутренним (Пример внутреннего
топика: __consumer_offsets).
  "Partitions": "number",        // Количество партиций в топике.
  "Configs": {                   // Конфигурации топика (динамические параметры).
    "retention.ms": 604800000,   // Пример параметра: время хранения сообщений.
    "cleanup.policy": "delete",  // Пример параметра: политика очистки.
    "...": "..."                 // Другие параметры.
    // Полный список параметров — в официальной документации:
    // https://kafka.apache.org/documentation/#topicconfigs.
  }
}
```

Конфигурация

- `URL` — URL Kafka в формате `example.com`.

Параметры

Название	Обязательность	Описание	Возможные значения
SecurityProtocol	Обязательно	Протокол для подключения к Kafka — в документации Kafka	PLAINTEXT, SASL_PLAINTEXT, SASL_SSL
SaslMechanism	Опционально	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL — в документации Kafka	PLAIN, SCRAM-SHA-256, SCRAM-SHA-512
User	Обязательно	Имя пользователя для подключения к Kafka	-
Pass	Обязательно	Пароль пользователя для подключения к Kafka	-

KubernetesResources

Источник данных типа `KubernetesResources` возвращает список ресурсов Kubernetes. Тип возвращаемых ресурсов задаётся через параметры источника данных. Поддерживаются как встроенные в Kubernetes типы ресурсов, так и любые кастомные ресурсы.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Kubernetes».

Спецификация ответа

Спецификация ответа зависит от типа ресурса, который планируется получить. Для получения точной информации необходимо обратиться к документации. Для встроенных в Kubernetes ресурсов доступна официальная документация.

При отсутствии возможности обратиться к документации можно получить описание спецификации с помощью команды `d8 k explain`.

Пример для deployment:

```
d8 k explain deployment --recursive
```

Примерный вывод:

```
GROUP:      apps
KIND:       Deployment
VERSION:    v1

DESCRIPTION:
    Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
  apiVersion <string>
  kind <string>
  metadata <ObjectMeta>
    annotations <map[string]string>
    creationTimestamp <string>
...
```

Описание спецификации находится внутри `FIELDS:`.

Конфигурация

- `URL` — URL Kubernetes API в формате `https://api.example.com`.

Параметры

Название	Обязательность	Описание	Возможные значения
apiGroup	Опционально	API-группа ресурса. Для ресурсов в core API-группе поле не задаётся	См. определение требуемых Group и Version
version	Обязательно	Версия API ресурса	См. определение требуемых Group и Version
isNamespaced	Обязательно	Принадлежность ресурса неймспейсам. Проверить принадлежность можно с помощью команды `d8 k api-resources`	true, false
namespace	Опционально	Неймспейс, из которого будут собираться ресурсы. Если не указан, ресурсы будут собираться из всех неймспейсов. Значение параметра учитывается, только если значение `isNamespaced` равно `true`	
resource	Обязательно	Название ресурса. Указывается маленькими буквами во множественном числе, как в поле NAME вывода команды `d8 k api-resources`	

Примеры

Собрать все ingress-ресурсы Kubernetes-кластера:

```
apiGroup: networking.k8s.io
version: v1
isNamespaced: true
resource: ingresses
```

Собрать все поды Kubernetes-кластера:

```
version: v1
isNamespaced: true
resource: pods
```

Собрать все поды Kubernetes-кластера в неймспейсе `d8-development-platform`:

```
version: v1
isNamespaced: true
resource: pods
namespace: d8-development-platform
```

Собрать все неймспейсы Kubernetes-кластера:

```
version: v1
isNamespaced: false
resource: namespaces
```

Собрать все кастомные ресурсы ModuleRelease Kubernetes-кластера:

```
apiGroup: deckhouse.io
version: v1alpha1
isNamespaced: false
resource: modulereleases
```

Определение требуемых Group и Version

Каждому типу ресурса соответствует своя версия и группа. Полный список API-ресурсов с их группами и версиями — в документации Kubernetes.

Если неизвестно, какие требуются Group и Version, можно попробовать подставить актуальные значения. Есть несколько вариантов, как их посмотреть, например с помощью утилиты `d8 k`: команда `d8 k explain` показывает `version` и `apiGroup` для ресурса.

Пример:

```
d8 k explain deployment
```

Вывод:

```
GROUP:      apps
KIND:       Deployment
VERSION:    v1

DESCRIPTION:
    Deployment enables declarative updates for Pods and ReplicaSets.

FIELDS:
...
```

NexusArtifacts

Источник данных типа `NexusArtifacts` возвращает список артефактов в Nexus.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Nexus».

Спецификация ответа

Платформа формирует список доступных репозиториев, затем по каждому репозиторию выполняется GET-запрос к API Nexus: `/service/rest/v1/components`. Спецификация ответа.

Конфигурация

- `URL` — URL Nexus в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

NexusRepositories

Источник данных типа `NexusRepositories` возвращает список репозиториев в Nexus.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Nexus».

Спецификация ответа

Платформа выполняет GET-запрос к API Nexus: `/service/rest/v1/repositories`. Спецификация ответа.

Конфигурация

- `URL` — URL Nexus в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

PrometheusMetrics

Источник данных типа `PrometheusMetrics` возвращает результат запроса PromQL в Prometheus.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис Prometheus».

Спецификация ответа

Платформа выполняет GET-запрос к API Prometheus: `/api/v1/query`. Спецификация ответа.

Конфигурация

- `URL` — URL Prometheus API в формате `https://example.com/api/v1/query`.
- «Query» — запрос в формате PromQL, на основе которого будет сформирован ответ.

Параметры

Настраиваемые параметры отсутствуют.

SonarqubeProjects

Источник данных типа `SonarqubeProjects` возвращает список проектов в SonarQube.

Авторизация

Конфигурация авторизации описана в разделе «Внешний сервис SonarQube».

Спецификация ответа

Платформа выполняет GET-запрос к API SonarQube: `/api/projects/search`. Система собирает все доступные `components` и возвращает их в виде массива. Спецификация `components`.

Конфигурация

- `URL` — URL SonarQube в формате `https://example.com`.

Параметры

Настраиваемые параметры отсутствуют.

GenericAPI

Источник данных типа `GenericAPI` позволяет подключаться к любому REST API и получать данные в формате JSON. Поддерживаются различные типы пагинации и настраиваемые параметры запроса.

Авторизация

GenericAPI поддерживает любые типы аутентификации через настройку заголовков HTTP. Наиболее распространённые способы:

Bearer Token:

```
Authorization: Bearer <токен>
```

Basic Authentication:

```
Authorization: Basic <base64-encoded-credentials>
```

API Key:

```
X-API-Key: <ключ>
```

Конфигурация

- `URL` — базовый URL API в формате `https://api.example.com`.
- `Method` — HTTP-метод (GET, POST, PUT, PATCH, DELETE).
- `Query` — дополнительные query-параметры (например, для фильтрации или поиска).

Параметры

Название	Обязательность	Описание	Возможные значения
`paginationType`	Опционально	Тип пагинации для обработки больших объёмов данных	`none`, `offset`, `cursor`, `page`, `link_header`
`path`	Опционально	Путь к эндпоинту API, который будет добавлен к базовому URL	Любая строка, начинающаяся с `/`
`dataPath`	Опционально	Путь к данным в ответе (цепочка ключей через точку)	Любая строка
`responseFormat`	Опционально	Формат ответа	`array`, `map`
`mapKeyField`	Опционально	При `responseFormat` равном `map`: название поля, в которое записать ключ из JSON-объекта	Любая строка
`pageParam`	Опционально	Название параметра для номера страницы (для page-пагинации)	Любая строка
`offsetParam`	Опционально	Название параметра смещения (для offset-пагинации)	Любая строка
`limitParam`	Опционально	Название параметра для количества элементов на странице (для offset-пагинации)	Любая строка
`sizeParam`	Опционально		Любая строка

Название	Обязательность	Описание	Возможные значения
		Название параметра для размера страницы (для page-пагинации)	
`cursorParam`	Опционально	Название параметра для курсора (для cursor-пагинации)	Любая строка
`pageSize`	Опционально	Количество элементов на странице для пагинации	Положительное целое число
`requestBody`	Опционально	Тело запроса для POST/PUT/PATCH-методов	Любая строка

Спецификация ответа

Формат ожидаемого ответа задаётся параметром `responseFormat` (по умолчанию `array`).

Пример ответа для `responseFormat: array`:

```
[
  {
    "id": 1,
    "name": "Example Item",
    "description": "Description of the item",
    "created_at": "2023-01-01T00:00:00Z"
  },
  {
    "id": 2,
    "name": "Another Item",
    "description": "Another description",
    "created_at": "2023-01-02T00:00:00Z"
  }
]
```

Пример ответа для `responseFormat: map`:

```
{
  "billing": { "title": "Billing", "owner": "team-a" },
  "shipping": { "title": "Shipping", "owner": "team-b" }
}
```

Перед обработкой ответ с `responseFormat: map` преобразуется в массив с учётом значения параметра `mapKeyField`.

При `mapKeyField: example` в обработку передаётся массив с добавленным полем `example` со значением, взятым из ключа объекта:

```
[
  { "example": "billing", "title": "Billing", "owner": "team-a" },
  { "example": "shipping", "title": "Shipping", "owner": "team-b" }
]
```

Если `mapKeyField` не задан (пустая строка), в обработку передаётся массив без добавления нового поля:

```
[
  { "title": "Billing", "owner": "team-a" },
  { "title": "Shipping", "owner": "team-b" }
]
```

Типы пагинации

- `none` — без пагинации, получает все данные одним запросом.
- `offset` — пагинация по смещению и лимиту:
  - Использует параметры `offsetParam` и `limitParam`.
  - Смещение начинается с `0` и увеличивается на число полученных записей.
  - Пример: `?offset=0&limit=100`, затем `?offset=100&limit=100`.
- `page` — пагинация по номеру страницы и размеру:
  - Использует параметры `pageParam` и `sizeParam`.
  - Пример: `?page=1&size=20`.
- `cursor` — пагинация по курсору:
  - Использует параметр `cursorParam`.
  - Пример: `?cursor=eyJpZCI6MTIzfQ==`.
- `link_header` — пагинация через заголовок Link: - Использует заголовок `Link` в ответе для определения следующей страницы. - Стандарт RFC 5988.

Примеры конфигурации

JSONPlaceholder API (без пагинации):

```
url: https://jsonplaceholder.typicode.com
path: /users
paginationType: none
dataPath: .
```

GitLab API (page-пагинация):

```
url: https://gitlab.example.com
path: /api/v4/projects
paginationType: page
pageParam: page
sizeParam: per_page
pageSize: 20
headers:
  Authorization: Bearer <token>
```

DefectDojo API (offset-пагинация):

```
url: https://defectdojo.example.com
path: /api/v2/engagements
paginationType: offset
limitParam: limit
pageSize: 100
dataPath: results
query: "product=42"
headers:
  Authorization: Token <token>
```

GitHub API (link header пагинация):

```
url: https://api.github.com
path: /repos/owner/repo/issues
paginationType: link_header
headers:
  Authorization: Bearer <token>
  Accept: application/vnd.github.v3+json
```

API с вложенными данными:

```
url: https://api.example.com
path: /data
dataPath: response.data.items
paginationType: page
pageParam: page
sizeParam: limit
query: "filter=active&sort=name"
```

POST-запрос с телом:

```
url: https://api.example.com
path: /search
method: POST
requestBody: '{"query": "example", "filters": {"status": "active"}}'
query: "include=metadata&format=json"
headers:
  Content-Type: application/json
  Authorization: Bearer <token>
```

Виджеты

Обзор

Виджеты — карточки для визуализации данных, хранящихся в платформе, а также информации из инфраструктурных сервисов. В отличие от источников данных, виджеты получают информацию из инфраструктурных сервисов непосредственно в момент их отображения в интерфейсе.

Виджеты могут быть добавлены на дашборды; дашборды, в свою очередь, могут быть привязаны:

- к статическим страницам (каталог, самообслуживание, главная страница, администрирование);
- к карточкам сущностей.

Конфигурация

Конфигурация виджета включает общие параметры и набор полей, специфичных для конкретного типа виджета.

В конфигурации поддерживается использование синтаксиса Go template для шаблонизации при обработке виджета, например:

- `{{ .entity.name }}` — подстановка значения параметра сущности «name».
- `{{ .credentials.token }}` — подстановка учётных данных с названием «token».

Для каждого виджета доступно задание области видимости:

- «Global» — виджет не поддерживает получение параметров сущности через механизм Go template;
- «Resource» — виджет поддерживает получение параметров сущности через механизм Go template. Виджеты с областью видимости Resource можно прикрепить только к страницам сущности.

В конфигурации виджетов возможно задание учётной записи, с данными которой виджет будет взаимодействовать с инфраструктурными системами, а также выбрать тип учётных данных, который будет использоваться.

Если учётная запись не задана, виджет будет использовать учётные данные пользователя, взаимодействующего с платформой в текущий момент.

Типы виджетов

AI-чат

Виджет «AI-чат» позволяет отправлять запросы к языковой модели через выбранный AI-провайдер: задаются общие инструкции («Глобальный промпт») и набор кнопок быстрых вопросов («Быстрые вопросы»).

Конфигурация

Название	Обязательность	Описание
Глобальный промпт	Нет	Общие инструкции к каждому запросу; при отправке объединяются с промптом выбранной кнопки быстрого вопроса.
Быстрые вопросы	Да	Набор кнопок с подписью и текстом промпта (до 20 штук); для корректной работы виджета необходимо добавить хотя бы один вопрос.

Для каждого быстрого вопроса задаются поля:

Название	Обязательность	Описание
Название вопроса	Да	Короткая подпись, отображается на нижней панели виджета и в чате.
Промпт	Да	Инструкция для модели при нажатии на эту кнопку; допускается использование Go-шаблонизации.

При заполнении промпта рекомендуется в явном виде указывать названия MCP-инструментов, которые должна вызвать модель при подготовке ответа.

Пример промпта для быстрого вопроса:

```
1. Вызови MCP tool get_external_data для внешнего сервиса «Deckhouse Code» и получи
пайплайны для проекта с ID {{ .entity.properties.deckhouse_code_id }}.
2. Выведи таблицу с последними 10 пайплайнами.
```

Использование виджета

Для использования виджета у пользователя должен быть добавлен как минимум один AI-провайдер.

У чата не предусмотрена история:

- Выводится только один ответ на последний заданный вопрос.
- Ответ не сохраняется при переходе на другую страницу или при обновлении страницы.

Перед отправкой вопроса пользователь может кастомизировать промпт, кликнув на пункт «Отправить с изменением промпта» в выпадающем меню кнопки вопроса.

API

Виджет позволяет вывести спецификацию API из файла в репозитории GitLab или по ссылке в формате OpenAPI (Swagger) или Protobuf. При выводе спецификации OpenAPI из файла в формате YAML или JSON виджет отображает интерфейс Swagger. Во всех остальных случаях виджет отображает спецификацию в виде текста.

Общая конфигурация

Название	Обязательность	Описание	Возможные значения	Значение по умолчанию
Тип спецификации	Да	Тип спецификации	OpenAPI (Swagger), Protocol Buffers	-
Тип источника	Да	Тип источника, из которого будет загружаться файл со спецификацией	URL, GitLab	-

Конфигурация типа источника: URL

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	Ссылка на файл со спецификацией	-
Заголовки	Нет	Заголовки для доступа к файлу со спецификацией	-

Конфигурация типа источника: GitLab

Название	Обязательность	Описание	Значение по умолчанию
GitLab URL	Да	URL GitLab	-
ID проекта	Да	Идентификатор проекта, из которого будет браться файл со спецификацией	-
Ветка	Да	Ветка, из которой будет браться файл со спецификацией	-
Путь к файлу	Да	Путь к файлу со спецификацией относительно корня репозитория	-

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Bitbucket. Pull Requests

Виджет позволяет отображать данные о Pull Requests (PR) в Bitbucket и выполнять действия с ними.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Пример
Ключ проекта	Да	Часть URL репозитория, которая идёт сразу после `/projects/`	Для репозитория `.../projects/MYTEAM/repos/backend` укажите `MYTEAM`
Идентификатор репозитория	Да	Часть URL репозитория, которая идёт сразу после `/repos/`	Для репозитория `.../projects/MYTEAM/repos/backend` укажите `backend`

Фильтрация по статусу

Виджет позволяет фильтровать отображаемые Pull Requests по статусу. В настройках запроса виджета можно выбрать один из следующих статусов:

- «Открыт» — показывает только открытые PR.
- «Слит» — показывает только слитые PR.
- «Отклонён» — показывает только отклонённые PR.
- «Все» — показывает PR в любом статусе.

По умолчанию отображаются только открытые PR.

Дополнительные возможности виджета

При активированной функции действий в настройках виджет позволяет выполнять следующие действия с Pull Requests:

- «Слить» — слияние открытого запроса на слияние (доступно только для открытых PR).
- «Закрыть» — отклонение (decline) запроса на слияние.
- «Просмотр изменений» — просмотр диффа (изменений) в запросе на слияние.
- «Комментарии» — просмотр и добавление комментариев к PR.
- «Создать PR» — создание нового Pull Request с указанием исходной и целевой ветки, ревьюеров, названия и описания.

> Для выполнения действий с PR требуются соответствующие права доступа в репозитории Bitbucket.

GitHub. Pull Requests

Виджет отображает Pull Requests (PR) репозитория на GitHub и позволяет просматривать изменения, создавать, сливать и закрывать PR.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы». В настройках внешнего сервиса или в конфигурации виджета в поле «URL» необходимо указать `https://api.github.com`.

Учётная запись и автор действий

Запросы к GitHub выполняются с токеном из учётных данных того пользователя платформы, от имени которого вызывается действие. Если в настройках виджета включено «Выбрать учётную запись для виджета», используются учётные данные выбранного пользователя платформы, а не текущего.

При создании, слиянии и закрытии Pull Request в GitHub автором PR и исполнителем действий считается учётная запись GitHub, которой принадлежит этот токен. Логин и имя в интерфейсе GitHub могут не совпадать с именем в профиле Deckhouse Development Platform (DDP).

Конфигурация

Название	Обязательность	Описание	Пример
Владелец репозитория	Да	Владелец репозитория (организация или пользователь).	Для `https://github.com/example/my-repo` укажите `example`
Репозиторий	Да	Название репозитория без `.git`.	Для `https://github.com/example/my-repo` укажите `my-repo`

Статус

В настройках запроса виджета можно фильтровать PR по статусу:

- «Открыт» — только открытые PR (не черновики).
- «Черновик» — только черновики.
- «Закрыт» — только закрытые PR.
- «Все» — любые PR.

По умолчанию отображаются открытые PR. В таблице отображаются: номер, название, описание, статус, метки, автор, дата создания, дата обновления; для каждого PR доступны действия через меню.

Действия

- «Изменения» — просмотр списка изменённых файлов и диффа по каждому файлу.
- «Слить» — слияние открытого PR (доступно только для открытых PR, не черновиков).
- «Закрыть» — закрытие PR без слияния.

- «Создать PR» — создание нового Pull Request. В диалоге указываются название, исходная ветка, целевая ветка и описание.

> Для выполнения действий с PR требуются соответствующие права доступа в репозитории GitHub.

GitHub. Actions

Виджет показывает запуски GitHub Actions в репозитории и позволяет просматривать jobs, артефакты, а также производить действия над ними.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы». В настройках внешнего сервиса в поле «URL» необходимо указать `https://api.github.com`.

Учётная запись и автор действий

Запросы к GitHub выполняются с токеном из учётных данных того пользователя платформы, от имени которого вызывается действие. Если в настройках виджета включено «Выбрать учётную запись для виджета», используются учётные данные выбранного пользователя платформы, а не текущего.

При запуске, отмене запуска и перезапуске workflow, работе с артефактами и логами в GitHub инициатором считается учётная запись GitHub, которой принадлежит токен. Логин в интерфейсе GitHub может не совпадать с именем в профиле DDP.

Конфигурация

Название	Обязательность	Описание	Пример
Владелец репозитория	Да	Владелец репозитория (организация или пользователь).	Для `https://github.com/example/my-repo` укажите `example`
Репозиторий	Да	Название репозитория без `.git`.	Для `https://github.com/example/my-repo` укажите `my-repo`

Параметры запроса

В настройках запроса виджета можно задать фильтры:

- «Ветка» — только запуски с указанной head-веткой.
- «Событие» — только запуски с выбранным типом события.
- «Статус» — только запуски в выбранном статусе или с выбранным итогом (conclusion).
- «Workflow» — только запуски для выбранного файла workflow.
- «Кто запустил» — только запуски, начатые указанным пользователем GitHub.

- «Фильтр по дате создания» — интервал дат создания запуска (дата начала и дата окончания).

Действия

В виджете доступны следующие действия:

- «Запустить workflow» — ручной запуск workflow с триггером `workflow_dispatch`: выбираются «Workflow» и «Ветка или тег»; при объявленных во входном YAML параметрах отображаются «Входные параметры».
- «Перезапустить workflow», «Перезапустить упавшие jobs», «Отменить workflow» — для выбранного запуска.
- «Перезапустить job» — для job в статусе «завершён» с итогом failure или cancelled.
- Просмотр логов job, скачивание артефактов, открытие запуска на GitHub, дерево jobs и шагов.

> Для действий с workflow и артефактами нужны соответствующие права в репозитории GitHub.

CodeScoring. Зависимости

Виджет позволяет вывести таблицу с зависимостями продукта на основе данных из CodeScoring с указанием названия зависимости, версии, лицензии, количества уязвимостей и другой информации для каждой зависимости.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL CodeScoring	-
ID проекта	Да	Идентификатор проекта в CodeScoring	-

CodeScoring. Уязвимости

Виджет позволяет вывести таблицу с уязвимостями продукта на основе информации из CodeScoring с указанием кода уязвимости, уровня критичности, наличия эксплойта, исправленной версии для каждой уязвимости.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL CodeScoring	-
ID проекта	Да	Идентификатор проекта в CodeScoring	-

CodeScoring. Секреты

Виджет позволяет вывести таблицу найденных секретов проекта из CodeScoring. Поддерживаются запуск либо отмена сканирования секретов по выбранной ветке или тегу.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL CodeScoring	-
ID проекта	Да	Идентификатор проекта в CodeScoring	-

ClickHouse. Метрики (диапазон)

Виджет позволяет построить линейный график на основе read-only SQL-запроса к ClickHouse. В запросе можно использовать плейсхолдеры `{{from}}` и `{{to}}` для подстановки границ временного интервала.

Пример корректного запроса для виджета:

```
SELECT
  toStartOfMinute(timestamp) AS time,
  avg(value) AS value,
  service AS series
FROM metrics
WHERE timestamp >= {{from}} AND timestamp < {{to}}
GROUP BY time, series
ORDER BY time
```

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Запрос	Да	Read-only SQL-запрос. Используйте `{{from}}` и `{{to}}` для подстановки интервала	-
База данных	Нет	Название базы ClickHouse, передаётся в заголовке `X-ClickHouse-Database`	-
Интервал по умолчанию	Нет	Интервал, используемый при открытии виджета и при обновлении, если в параметрах запроса интервал не задан	Последний час
Колонка времени	Да	Колонка с меткой времени для оси X графика	-
Колонка значения	Да	Колонка с числовыми значениями для оси Y графика	-
Колонка серии	Нет	Колонка, используемая как название серии в легенде графика	-
Пороговое значение	Нет	Порог, отображаемый в виде горизонтальной линии на графике	-
Минимальное значение	Нет	Начальная точка отсчёта для вертикальной оси графика	-
Максимальное значение	Нет	Предельная точка отсчёта для вертикальной оси графика	-

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

ClickHouse. Метрики (значение)

Виджет позволяет вывести одно число на основе read-only SQL-запроса к ClickHouse, задать для него единицу измерения и настроить пороговое значение. Запрос должен возвращать одну строку; отображается значение первой колонки.

Пример корректного запроса для виджета:

```
SELECT count() AS value
FROM events
WHERE timestamp >= {{from}} AND timestamp < {{to}}
```

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Запрос	Да	Read-only SQL-запрос. Используйте `{{from}}` и `{{to}}` для подстановки интервала	-
База данных	Нет	Название базы ClickHouse, передаётся в заголовке `X-ClickHouse-Database`	-
Интервал по умолчанию	Нет	Интервал, используемый при открытии виджета и при обновлении, если в параметрах запроса интервал не задан	Последний час
Количество цифр после запятой	Нет	Точность, с которой будет выводиться полученное значение	-
Единица измерения	Нет	Постфикс, с которым будет выводиться полученное значение	-
Отображать пороговое значение	Нет	Отображать пороговое значение в формате <значение метрики> / <пороговое значение>	false
Пороговое значение	Нет	Пороговое значение	-
Меньшее значение считается лучше	Нет	Метрика считается «хорошей», когда её значение ниже заданного порогового значения	false
Порог предупреждения (%)	Нет	Граница между красным и оранжевым цветами. Если значение метрики превышает этот процент от порога, оно получит оранжевый цвет	60
Порог успеха (%)	Нет	Граница между оранжевым и зелёным цветами. Если значение метрики превышает этот процент от порога, оно получит зелёный цвет	90

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

ClickHouse. Таблица

Виджет позволяет вывести результат read-only SQL-запроса к ClickHouse в виде таблицы с сортировкой и постраничной навигацией. В запросе можно использовать плейсхолдеры `{{from}}` и `{{to}}`.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Запрос	Да	Read-only SQL-запрос. Используйте `{{from}}` и `{{to}}` для подстановки интервала	-
База данных	Нет	Название базы ClickHouse, передаётся в заголовке `X-ClickHouse-Database`	-
Интервал по умолчанию	Нет	Интервал, используемый при открытии виджета и при обновлении, если в параметрах запроса интервал не задан	Последний час
Размер страницы	Да	Количество строк, загружаемых за один запрос	50

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

ClickHouse. Топ N

Виджет позволяет построить горизонтальную столбчатую диаграмму на основе read-only SQL-запроса к ClickHouse. Запрос должен возвращать колонки с метками и числовыми значениями; количество отображаемых строк ограничивается параметром «Лимит».

Пример корректного запроса для виджета:

```
SELECT service AS label, count() AS value
FROM events
WHERE timestamp >= {{from}} AND timestamp < {{to}}
GROUP BY label
ORDER BY value DESC
```

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Запрос	Да	Read-only SQL-запрос. Используйте `{{from}}` и `{{to}}` для подстановки интервала	-
База данных	Нет	Название базы ClickHouse, передаётся в заголовке `X-ClickHouse-Database`	-
Интервал по умолчанию	Нет	Интервал, используемый при открытии виджета и при обновлении, если в параметрах запроса интервал не задан	Последний час
Колонка метки	Да	Колонка для подписей столбцов диаграммы	-
Колонка значения	Да	Колонка с числовыми значениями для длины столбцов	-
Лимит	Да	Максимальное количество строк для диаграммы	10

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

DefectDojo. Продукт

Виджет позволяет просматривать уязвимости продукта в DefectDojo с разбивкой по engagements и уровням критичности.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL DefectDojo. Указывается без пути к API (`/api/v2`)	-
Название продукта	Да	Название продукта в DefectDojo	-
Уровни уязвимостей	Да	Уровни критичности уязвимостей, которые подгружаются при открытии виджета и при смене engagement	Critical, High, Medium, Low, Info

Параметры запроса

В настройках запроса виджета можно изменить уровни уязвимостей — загружаются только уязвимости выбранных уровней критичности. По умолчанию используются уровни из конфигурации виджета. Изменение уровней приводит к повторной загрузке данных из DefectDojo.

Дополнительные возможности виджета

Фильтры

- «Engagement» — выбор engagement продукта. По умолчанию выбирается последний созданный engagement (с наибольшим идентификатором). При смене engagement данные перезагружаются.
- «Фильтры» — панель фильтров по уровням критичности, тегам, тестам и компонентам. Фильтры по тегам, тестам и компонентам применяются только к уже загруженным уязвимостям и не вызывают новых запросов к DefectDojo.

Вкладки

- «Обзор» — уязвимости по уровню критичности, по тегам (топ-10), по тестам (топ-10), по компонентам (топ-10). Диаграммы строятся по отфильтрованному набору уязвимостей.
- «Детали» — таблица уязвимостей с деталями по каждой из них.

> Если уязвимостей для выбранного engagement и уровней критичности больше 1000, в виджете отображаются первые 1000 записей и предупреждение о частичной загрузке. Фильтры по тегам, тестам и компонентам действуют только в пределах загруженного набора.

DefectDojo. Уязвимости в продукте (детали)

Виджет позволяет вывести таблицу с уязвимостями продукта на основе информации из DefectDojo с указанием уровня критичности, описания и даты обнаружения для каждой уязвимости.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL DefectDojo. Указывается без пути к API (`/api/v2`)	-
Название продукта	Да	Название продукта в DefectDojo	-

Дополнительные возможности виджета

При просмотре виджета доступна настройка следующих параметров:

- «Активные уязвимости» — если включено, то загружаются уязвимости продукта с флагом ‘Active’ = true. Если отключено, то загружаются уязвимости продукта с флагом ‘Active’ = false. Включено по умолчанию.
- «Дублирующиеся уязвимости» — если включено, то загружаются уязвимости продукта с флагом ‘Duplicate’ = true. Если отключено, то загружаются уязвимости продукта с флагом ‘Duplicate’ = false. Отключено по умолчанию.

DefectDojo. Уязвимости в продукте (общая статистика)

Виджет позволяет вывести график с общим количеством уязвимостей продукта на основе информации из DefectDojo с разбивкой по уровням критичности.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL DefectDojo. Указывается без пути к API (`/api/v2`)	-
Название продукта	Да	Название продукта в DefectDojo	-

Дополнительные возможности виджета

При просмотре виджета доступна настройка следующих параметров:

- «Активные уязвимости» — если включено, то загружаются уязвимости продукта с флагом ‘Active’ = true. Если отключено, то загружаются уязвимости продукта с флагом ‘Active’ = false. Включено по умолчанию.
- «Дублирующиеся уязвимости» — если включено, то загружаются уязвимости продукта с флагом ‘Duplicate’ = true. Если отключено, то загружаются уязвимости продукта с флагом ‘Duplicate’ = false. Отключено по умолчанию.

Docker образы

Виджет позволяет отображать данные о доступных образах в docker registry. На виджет выводятся все доступные теги и команда docker pull. Поддерживается поиск.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Docker Registry. Используется для получения данных о доступных образах	-
Название	Нет	Название репозитория, из которого будут загружаться данные в виджет. Пример: `repo`. Без указания названия, будут получены все доступные образы	-

GitLab. Запросы слияния

Виджет позволяет отображать данные о Merge Requests (MR) в платформе GitLab и выполнять действия с ними.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL GitLab API. Используется для получения данных из GitLab	-
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Фильтрация по статусу

Виджет позволяет фильтровать отображаемые Merge Requests по статусу. В настройках запроса виджета можно выбрать один из следующих статусов:

- «Открытые» — показывает только открытые MR.
- «Закрытые» — показывает только закрытые MR.
- «Слитые» — показывает только слитые MR.
- «Заблокированные» — показывает только заблокированные MR.

По умолчанию отображаются только открытые MR.

Дополнительные возможности виджета

При активированной функции действий в настройках виджет позволяет выполнять следующие действия с Merge Requests:

- «Слить» — слияние открытого запроса на слияние (доступно только для открытых MR).
- «Закрыть» — закрытие запроса на слияние.
- «Отметить как черновик/готово» — изменение статуса черновика запроса на слияние.
- «Просмотр изменений» — просмотр диффа (изменений) в запросе на слияние.

> Для выполнения действий с MR требуются соответствующие права доступа в репозитории GitLab.

GitLab. Пайплайны

Виджет позволяет отображать данные о пайплайнах в платформе GitLab.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL GitLab API. Используется для получения данных из GitLab	-
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Дополнительные возможности виджета

Запуск пайплайнов

Виджет позволяет запускать пайплайны в GitLab напрямую из DDP.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ref	Да	Целевая ветка или тег для запуска пайплайна	-
Переменные	Нет	Переменные в формате ключ-значение, которые будут переданы в запускаемый пайплайн	-

GitLab. Редактор пайплайна

Виджет позволяет редактировать конфигурацию пайплайна GitLab CI/CD (файл `.gitlab-ci.yml`) и создавать запросы на слияние с изменениями.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL GitLab API. Используется для получения данных из GitLab	-
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Отображаемые данные

Виджет отображает:

- «Редактор кода» — Monaco Editor для редактирования файла `.gitlab-ci.yml`.
- «Дифф-просмотр» — отображение изменений между оригинальной и редактируемой версией конфигурации.

Дополнительные возможности виджета

Создание запроса на слияние

Виджет позволяет создавать запросы на слияние с изменениями конфигурации пайплайна.

Параметры запроса на слияние

Название	Обязательность	Описание	Значение по умолчанию
Заголовок MR	Да	Краткий заголовок, описывающий цель запроса на слияние	-
Описание MR	Нет	Подробное описание запроса на слияние и изменений	-
Название новой ветки	Да	Название новой ветки, которая будет содержать ваши изменения	-
Целевая ветка	Да	Ветка, в которую будет выполнен запрос на слияние	main
Сообщение коммита	Да	Описание изменений, внесенных в конфигурацию пайплайна	-

Ограничения

- Виджет работает только с файлом `.gitlab-ci.yml` в корне проекта.

- Для создания запроса на слияние требуются права на запись в репозиторий.
- Максимальный размер файла конфигурации ограничен возможностями GitLab API.

GitLab. Статистика пайплайнов

Виджет позволяет отображать статистику пайплайнов в платформе GitLab, включая общую статистику, распределение по статусам, источникам, участникам и веткам.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL GitLab API. Используется для получения данных из GitLab	-
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Отображаемые данные

Виджет отображает следующую статистику:

Основные метрики

- «Общее количество пайплайнов» — общее число пайплайнов за выбранный период.
- «Процент успеха» — процент успешно выполненных пайплайнов.
- «Процент неудач» — процент неудачно выполненных пайплайнов.
- «Средняя длительность» — среднее время выполнения пайплайнов.

Распределение по статусам

- Успешные пайплайны.
- Неудачные пайплайны.
- Отмененные пайплайны.
- Пропущенные пайплайны.
- Ручные пайплайны.

Распределение по источникам

- Push (коммиты).
- Merge requests (запросы слияния).
- Schedule (по расписанию).
- Web (через веб-интерфейс).

Топ участников

- Список участников с наибольшим количеством запущенных пайплайнов.
- Аватары участников (при наличии).
- Количество пайплайнов для каждого участника.

Активность веток

- Список веток с наибольшим количеством пайплайнов.
- Количество пайплайнов для каждой ветки.

Параметры запроса

Название	Обязательность	Описание	Значение по умолчанию
Начальная дата	Да	Начальная дата для анализа пайплайнов в формате ISO 8601. Пример: `2024-01-01T00:00:00Z`	-
Конечная дата	Да	Конечная дата для анализа пайплайнов в формате ISO 8601. Пример: `2024-01-31T23:59:59Z`	-
Ветка	Нет	Фильтр по конкретной ветке. Если не указана, анализируются все ветки	-

Ограничения

- Виджет анализирует максимум 100 пайплайнов за один запрос для оптимизации производительности.
- Статистика рассчитывается только для пайплайнов с валидными данными (имеющими статус и время выполнения).
- Данные обновляются при каждом обновлении виджета.

GitLab. Теги

Виджет позволяет отображать данные о тегах проекта в платформе GitLab.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL GitLab API. Используется для получения данных из GitLab	-
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Дополнительные возможности виджета

Создание тегов

Виджет позволяет создать теги в GitLab напрямую из DDP.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Название	Да	Название создаваемого тега	-
Создать из	Да	Ветка или существующий тег, от которого создаётся новый тег	-
Описание	Нет	Описание создаваемого тега	-

Bitbucket. Теги

Виджет позволяет отображать данные о тегах репозитория в Bitbucket.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Пример
Ключ проекта	Да	Часть URL репозитория, которая идёт сразу после `/projects/`	Для репозитория `.../projects/MYTEAM/repos/backend` укажите `MYTEAM`
Репозиторий	Да	Часть URL репозитория, которая идёт сразу после `/repos/`	Для репозитория `.../projects/MYTEAM/repos/backend` укажите `backend`

Отображаемые данные

Виджет отображает список тегов репозитория с информацией о каждом теге:

- «Название тега» — название тега.
- «Коммит» — хеш коммита, сообщение коммита, автор, дата создания, ссылка на коммит в Bitbucket.

Дополнительные возможности виджета

Создание тегов

Виджет позволяет создавать теги в Bitbucket напрямую из DDP.

Конфигурация

Название	Обязательность	Описание
Название	Да	Название создаваемого тега
Создать из	Да	Ветка или существующий тег, от которого создаётся новый тег (выбирается из списка)
Описание	Нет	Описание создаваемого тега

GitHub. Теги

Виджет отображает теги репозитория GitHub с информацией о коммите (автор, дата, описание) и позволяет создавать новые теги.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы». В настройках внешнего сервиса в поле «URL» необходимо указать `https://api.github.com`.

Учётная запись и автор действий

Запросы к GitHub выполняются с токеном из учётных данных того пользователя платформы, от имени которого вызывается действие. Если в настройках виджета включено «Выбрать учётную запись для виджета», используются учётные данные выбранного пользователя платформы, а не текущего.

«Аннотированный тег» (поле «Описание» заполнено): в метаданных git-тега поля автора аннотации (`tagger`) заполняются из имени и email пользователя платформы, выполнившего действие (как в профиле в DDP). Если имя не задано, может подставляться email.

«Лёгкий тег» (без описания): отдельный автор тега в git не задаётся; создаётся ссылка на коммит.

Создание тега через API выполняется от учётной записи GitHub по токену; данные `tagger` при этом берутся из профиля DDP и могут не совпадать с логином GitHub.

Конфигурация

Название	Обязательность	Описание	Пример
Владелец репозитория	Да	Владелец репозитория (организация или пользователь).	Для `https://github.com/example/my-repo` укажите `example`
Репозиторий	Да	Название репозитория без `.git`.	Для `https://github.com/example/my-repo` укажите `my-repo`

Отображаемые данные

В таблице отображаются колонки: тег, описание, автор коммита, ссылка на коммит, дата создания коммита; для каждого тега доступно действие «Просмотр» (просмотр описания коммита).

Дополнительные возможности виджета

Создание тега

Виджет позволяет создавать теги в GitHub. В диалоге «Создать тег» указываются:

Название	Обязательность	Описание	Значение по умолчанию
Название тега	Да	Уникальное название тега, например `v1.0.0` или `release-2024-01`	—
Создать из	Да	Ветка или существующий тег, от которого создаётся новый тег	—
Описание	Нет	Аннотация к тегу (например, описание релиза). Если указано, создаётся аннотированный тег	—

> Для создания тегов требуются права на запись в репозиторий GitHub.

GitLab. Релизы

Виджет отображает список релизов GitLab-проекта, подсвечивает последний релиз и показывает связанную информацию: тег, ссылку на коммит, автора, дату публикации и описание (поддерживает Markdown).

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Дополнительные возможности виджета

Создание релиза

Виджет позволяет создать релиз в GitLab напрямую из DDP:

Название	Обязательность	Описание	Значение по умолчанию
Название релиза	Да	Название релиза, отображаемое в списке	-
Тег	Да	Существующий тег, на основе которого будет сформирован релиз (выбирается из списка тегов проекта)	-
Описание	Нет	Описание релиза в формате Markdown	-

Созданный релиз автоматически появляется в списке, а последний релиз подсвечивается.

GitLab. Участники

Виджет позволяет отображать данные об участниках проекта в GitLab. Подробнее об участниках.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
ID проекта	Да	ID проекта, из которого будут загружаться данные в виджет. Пример: `12345`	-

Просмотр репозитория

Виджет позволяет просматривать структуру и содержимое файлов в репозитории. Поддерживаются репозитории GitLab, Bitbucket и GitHub.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

- GitLab.
- Bitbucket.
- GitHub.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Провайдер	Да	Сервис, в котором размещён репозиторий (GitLab, GitHub, Bitbucket)	-
Ветка / Тег	Нет	Название ветки, тег или SHA коммита (по умолчанию: main)	main
Путь	Нет	Путь к директории в репозитории (оставьте пустым для корня)	-
Рекурсивно	Нет	Получать файлы рекурсивно из поддиректорий	false

Конфигурация для GitLab

Название	Обязательность	Описание	Значение по умолчанию
ID проекта	Да	ID проекта в GitLab (например, 12345).	-

Конфигурация для Bitbucket

Название	Обязательность	Описание	Пример	Значение по умолчанию
Ключ проекта	Да	Ключ проекта в Bitbucket (например, MYPROJ)	MYPROJ	-
Идентификатор репозитория	Да	Идентификатор репозитория в Bitbucket (например, my-repo)	my-repo	-

Конфигурация для GitHub

Название	Обязательность	Описание	Значение по умолчанию
Владелец репозитория	Да	Владелец репозитория (организация или пользователь). Пример: для `https://github.com/example/my-repo` укажите «example»	-
Репозиторий	Да	Название репозитория без `.git`. Пример: для `https://github.com/example/my-repo` укажите «my-repo»	-

Jenkins. Пайплайны

Виджет отображает данные о пайплайнах в Jenkins и позволяет управлять сборками.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Jenkins. Используется для получения данных из Jenkins	-
Название	Да	Название пайплайна в Jenkins. Поддерживается вложенность: `folder1/folder2/jobName`	-

Отображаемые данные

Виджет автоматически определяет тип пайплайна и отображает соответствующее представление.

Обычные пайплайны

Для обычных пайплайнов виджет отображает:

- «Список сборок» — таблица со всеми сборками пайплайна с информацией о номере, статусе, длительности, времени выполнения и пользователе.
- «Последняя сборка» — информация о последней выполненной сборке.
- «Последняя успешная сборка» — информация о последней успешной сборке.
- «Последняя неудачная сборка» — информация о последней неудачной сборке.

Multibranch пайплайны

Для multibranch пайплайнов виджет отображает:

- «Список веток» — таблица со всеми ветками с информацией о статусе, количестве сборок и последней сборке для каждой ветки.
- Всю информацию, описанную в разделе «обычные пайплайны», в разрезе каждой ветки.

Дополнительные возможности виджета

Виджет позволяет выполнять следующие действия:

Для обычных пайплайнов

- «Запустить сборку» — запуск новой сборки. Если у сборки есть параметры, отображается диалог для их ввода:
  - Строковые параметры;
  - Пароли;
  - Выбор из списка;
  - Булевые значения.
- «Отменить сборку» — отмена выполняющейся сборки.
- «Повторить сборку» — повторный запуск последней сборки.
- «Просмотр логов» — просмотр логов выполнения сборки.

Для multibranch пайплайнов

- «Запустить сборку ветки» — запуск новой сборки для конкретной ветки. Если у сборки есть параметры, отображается диалог для их ввода.
- «Получить сборки ветки» — загрузка списка сборок для конкретной ветки.
- «Сканировать multibranch» — запуск сканирования multibranch пайплайна для обнаружения новых веток.
- «Просмотр логов» — просмотр логов выполнения сборки.

Jira. Задачи

Виджет позволяет отображать задачи из Jira на основе JQL-запроса.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Jira. Используется для получения данных из Jira	-
JQL	Да	JQL-запрос для фильтрации задач. Пример: `project = PROJ AND status = Open`	-

Параметры запроса

Название	Обязательность	Описание	Значение по умолчанию
JQL	Нет	JQL-запрос для фильтрации задач. Если не указан, используется JQL из конфигурации	Из конфигурации
Максимум результатов	Нет	Максимальное количество задач для отображения (от 1 до 1000)	50

Дополнительные возможности виджета

- «Просмотр описания» — при клике на кнопку «Просмотр описания» открывается диалоговое окно с полным описанием задачи.
- «Переход в Jira» — клик по ключу задачи открывает задачу в Jira в новой вкладке.
- «Динамическая фильтрация» — возможность изменить JQL-запрос и максимальное количество результатов прямо в виджете без изменения конфигурации.

Helm. Релизы

Виджет позволяет отображать данные о Helm-релизах в Kubernetes и производить rollback на предыдущие версии.

Данные, отображаемые на виджете:

- «Список релизов Helm» — информация о текущих релизах, созданных с помощью Helm в указанном неймспейсе Kubernetes.
- «Манифесты релизов» — манифесты, связанные с Helm-релизами в указанном неймспейсе Kubernetes. Это включает в себя файлы YAML, которые определяют конфигурацию и состояние ресурсов.

- «Values» — переменные, которые использовались для развёртывания Helm-релизов.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Namespace	Нет	Неймспейс, из которого будут загружаться данные в виджет. Пример: `default`	-
Релиз	Нет	Название релиза, из которого будут загружаться данные в виджет. Пример: `my-release`	-

Iframe

> ⚠ Виджет Iframe работает только при включённой опции `allowIframe: true` в конфигурации заголовков безопасности (`security.headers.csp.allowIframe`). По умолчанию эта опция отключена, поэтому виджет не будет отображать контент до изменения конфигурации.

Виджет позволяет отображать данные из внешнего источника.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL внешнего источника. Используется для отображения данных в виджете	-

Kafka. ACLs

Виджет позволяет отображать список ACLs кластера Kafka.

Для каждого ACL отображается следующая информация:

- Субъект.
- Тип ресурса.
- Шаблон.
- Тип шаблона.
- Хост.

- Операция.
- Тип разрешения.

Аутентификация

Для работы с виджетом требуется учётная запись пользователя. Система поддерживает следующие методы аутентификации:

- PLAINTEXT.
- SCRAM-SHA-256.
- SCRAM-SHA-512.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Kafka кластера	-
Протокол аутентификации	Да	Протокол для подключения к Kafka. Подробнее	-
Механизм SASL	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее	-
Пользователь Kafka	Да	Username учётной записи для взаимодействия с Kafka	-
Пароль	Да	Пароль учётной записи для взаимодействия с Kafka	-
Типы ресурсов	Нет	Фильтр по типам ресурсов	-
Типы шаблонов	Нет	Фильтр по типам шаблонов	-
Операции	Нет	Фильтр по операциям	-
Типы разрешений	Нет	Фильтр по типам разрешений	-
Субъекты	Нет	Фильтр по субъектам. Поддерживается шаблонизация и регулярные выражения	-
Хосты	Нет	Фильтр по хостам. Поддерживается шаблонизация и регулярные выражения	-

Дополнительные возможности виджета

При активированной функции действий в настройках виджет позволяет:

- Создавать новые правила ACL.
- Удалять существующие правила ACL.

Kafka. Топики

Виджет позволяет отображать различные данные о Kafka топиках.

Для каждого топика доступно:

- Общая информация о топике: основные параметры, конфигурация и статус.
- Информация о партициях: лидер и оффсеты, количество реплик и т. д.
- Информация о консьюмерах: список активных потребителей, их группы, текущие оффсеты и лаги.
- Сообщения: просмотр содержимого сообщений топиков.
- Поиск сообщений: фильтрация сообщений по timestamp и offset.
- Настройки топика: просмотр конфигурации топика в виде таблицы ключ-значение.

Аутентификация

Для работы с виджетом требуется учётная запись пользователя. Система поддерживает следующие методы аутентификации:

- PLAINTEXT.
- SCRAM-SHA-256.
- SCRAM-SHA-512.

> Доступность информации в виджете определяется уровнем прав подключённой учётной записи.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Kafka кластера	-
Протокол аутентификации	Да	Протокол для подключения к Kafka. Подробнее	-
Механизм SASL	Нет	Механизм аутентификации, который будет использовать SASL. Обязателен при использовании протокола SASL_PLAINTEXT или SASL_SSL. Подробнее	-
Пользователь Kafka	Да	username учётной записи для взаимодействия с Kafka	-
Пароль	Да	пароль учётной записи для взаимодействия с Kafka	-
Топики Kafka	Нет	Название топика или регулярное выражение для фильтрации отображаемых топиков в виджете; при пустом значении отображаются все доступные пользователю топики.	-

Дополнительные возможности виджета

При активированной функции действий в настройках виджет позволяет:

- Создавать новые топики;
- Удалять существующие топики;
- Отправлять сообщение в топик;
- Очищать топик от сообщений.

Kubernetes deployments

Виджет Kubernetes deployments позволяет выводить основную информацию обо всех deployments в кластере Kubernetes. Доступна фильтрация по неймспейсу и/или по label selector.

Для каждого ресурса Deployment доступны:

- «Просмотр спецификации и статуса Deployment».
- «Масштабирование количества реплик Deployment». Для применения изменений после выбора требуемого количества реплик необходимо нажать кнопку «Сохранить» с иконкой дискеты.

- «Просмотр информации о подах», управляемых Deployment, и контейнерах этих подов, включая просмотр логов каждого контейнера.
- «Просмотр и редактирование ресурсов контейнеров». Виджет отображает все настроенные ресурсы контейнеров, включая CPU, Memory, ephemeral-storage и другие типы ресурсов. Редактирование доступно только для CPU и Memory в секциях `requests` и `limits`. Изменения применяются на уровне Deployment и распространяются на все поды, управляемые данным Deployment. При очистке значений CPU или Memory соответствующие ресурсы удаляются из конфигурации контейнера. Остальные ресурсы (например, `ephemeral-storage`) отображаются, но не могут быть отредактированы через виджет.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Kubernetes API	Да	URL API сервера Kubernetes. Используется для получения данных из Kubernetes	-
Namespace	Нет	Kubernetes namespace из которого будут загружаться deployment. В случае, если namespace не указан, виджет будет пытаться загрузить все deployment кластера. Пример: `default`	-
Label selector	Нет	Селекторы для фильтрации получаемых deployment. Перечисляются через запятую. Пример: `app.kubernetes.io/name=example`	-

Kubernetes ingresses

Виджет позволяет отображать данные об Ingress в кластере Kubernetes.

Для каждого Ingress доступны:

- Просмотр спецификации Ingress в виде YAML-конфигурации.
- Правила Ingress.
- Настройки TLS.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL API сервера Kubernetes. Используется для получения данных из Kubernetes	-
Namespace	Нет	Неймспейс Kubernetes из которого будут загружаться ingresses. В случае, если неймспейс не указан, виджет будет пытаться загрузить все ingress кластера. Пример: `default`	-
Label selector	Нет	Селекторы для фильтрации получаемых ingress. Перечисляются через запятую. Пример: `app.kubernetes.io/name=example`	-

Kubernetes pods

Виджет позволяет отображать данные о подах в кластере Kubernetes.

Для каждого pod доступны:

- Просмотр спецификации пода в виде YAML-конфигурации.
- Логи контейнеров.
- Различная информация о состоянии пода: статус, количество перезапусков и др.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL API сервера Kubernetes. Используется для получения данных из Kubernetes	-
Namespace	Нет	Неймспейс, из которого будут загружаться данные в виджет. Пример: `default`	-
Label selector	Нет	Селекторы для фильтрации получаемых подов. Перечисляются через запятую. Пример: `app.kubernetes.io/name=example`	-

Markdown

Виджет обеспечивает отображение текста, написанного в формате Markdown.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Markdown	Да	Текст в формате Markdown. Отображается в виджете в отформатированном виде	-

Nexus artifacts

Виджет позволяет выводить список артефактов в репозитории Nexus.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Nexus API. Используется для получения данных из Nexus	-
Repository	Да	Название репозитория, данные из которого будут отображаться в виджете. Пример: `my-repo`	-
Name	Нет	Название артефакта, данные о котором будут отображаться в виджете	-

Opensearch index

Виджет Opensearch index позволяет отобразить данные из определённого index или index pattern в платформе. Данные по умолчанию сортируются от более новых к более старым. Доступен полнотекстовый поиск для фильтрации отображаемых данных. Для каждой записи (строки таблицы) доступно отображение в формате «ключ-значение», либо в JSON. При указании index pattern в виджете будет выводиться ссылка на страницу Discover в OpenSearch Dashboards.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
API URL	Да	URL Opensearch API. Используется для получения данных из Opensearch	-
Dashboards URL	Да	URL Opensearch Dashboards. Используется при генерации ссылки для перехода в Opensearch и просмотра данных непосредственно в системе	-
Index pattern	Да	Название index pattern из которого будут загружаться данные в виджет. Может содержать символ «*». Примеры: `security-auditlog`, `security-auditlog-*`	-
Timestamp field	Нет	Название поля с timestamp. Значение поля выводится в таблице с данными в отдельной колонке	@timestamp

Prometheus. Метрики (диапазон)

Виджет строит график по результату запроса `query_range` к Prometheus. Запрос в формате PromQL должен возвращать тип **Matrix** (одну или несколько временных серий).

Пример запроса с плейсхолдерами:

```
sum by (status) (rate(http_requests_total[{{rateInterval}}]))
```

Плейсхолдеры в запросе

Виджет подставляет в запрос значения, рассчитанные из активного интервала:

Плейсхолдер	Описание
`{{range}}`	Длительность выбранного интервала
`{{rateInterval}}`	Окно диапазона для функций `rate()` и `increase()`
`{{interval}}`	Шаг между точками на графике (рассчитывается автоматически)

Шаг запроса выбирается по длительности интервала (не более 1100 точек на графике, минимум 30 секунд). Поле «Шаг разрешения» в конфигурации больше не используется.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Prometheus	-
Запрос	Да	PromQL-запрос для графика диапазона	-
Метка	Да	Имя метки Prometheus, используемое как название серии в легенде графика	-
Интервал по умолчанию	Нет	Интервал при загрузке виджета и при обновлении, если пользователь не выбрал другой диапазон в панели виджета	Последний час
Пороговое значение	Нет	Горизонтальная пунктирная линия на графике	-
Минимальное значение	Нет	Нижняя граница оси Y; оставьте пустым для автоматического масштабирования	-
Максимальное значение	Нет	Верхняя граница оси Y; оставьте пустым для автоматического масштабирования	-
InsecureSkipVerify	Нет	Отключение проверки подлинности TLS/SSL-сертификата Prometheus	false

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Prometheus. Метрики (значение)

Виджет показывает одно число по PromQL-запросу к Prometheus. Запрос должен возвращать тип **Scalar** или тип **Vector** с одним значением.

Пример запроса без плейсхолдеров:

```
sum(machine_cpu_cores)
```

Пример с плейсхолдером (длительность интервала из панели виджета):

```
sum(increase(http_requests_total[{{range}}]))
```

Плейсхолдеры в запросе

Используйте плейсхолдеры, если в выражении нужна длительность интервала, выбранного в панели виджета. Запрос выполняется на момент окончания этого интервала.

Плейсхолдер	Описание
`{{range}}`	Длительность выбранного интервала
`{{rateInterval}}`	Окно диапазона для функций `rate()` и `increase()`
`{{interval}}`	Шаг запроса, рассчитанный из интервала

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL Prometheus	-
Запрос	Да	PromQL-запрос, возвращающий одно значение	-
Интервал по умолчанию	Нет	Интервал при загрузке виджета и при обновлении, если пользователь не выбрал другой диапазон в панели виджета	Последний час
Количество цифр после запятой	Нет	Точность, с которой будет выводиться полученное значение	-
Единица измерения	Нет	Постфикс, с которым будет выводиться полученное значение	-
Отображать пороговое значение	Нет	Отображать пороговое значение в формате <значение метрики> / <пороговое значение>	false
Пороговое значение	Нет	Пороговое значение	-
Меньшее значение считается лучше	Нет	Метрика считается «хорошей», когда её значение ниже заданного порогового значения	false
Порог предупреждения (%)	Нет	Граница между красным и оранжевым цветами. Если значение метрики превышает этот процент от порога, оно получит оранжевый цвет	60
Порог успеха (%)	Нет	Граница между оранжевым и зелёным цветами. Если значение метрики превышает этот процент от порога, оно получит зелёный цвет	90
InsecureSkipVerify	Нет	Отключение проверки подлинности TLS/SSL-сертификата Prometheus	false

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

SonarQube

Виджет позволяет отображать данные о метриках в платформе SonarQube.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	Адрес SonarQube, например, `https://sonarqube.example.com`	-
Ключ проекта	Да	Идентификатор проекта в SonarQube	-
Ветка	Нет	Ветка проекта для которой будут браться метрики	Согласно настройкам проекта в Sonarqube
Метрики	Да	Метрики проекта, которые будут выводиться в виджете. В конфигурации указывается Metric key	

Список возможных метрик для текущей версии SonarQube.

Дополнительные возможности виджета

Виджет позволяет просматривать данные не только для ветки по умолчанию, но и для любой другой ветки.

Svacer. Ветка

Виджет позволяет просматривать результаты статического анализа кода по ветке проекта в Svacer: прогресс ревью маркеров, распределение находок по критичности, динамику срабатываний на ветке, сравнение снимков и таблицу маркеров с пагинацией.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Название проекта	Да	Название проекта в Svacer	-
Название ветки	Да	Название ветки в Svacer; используется как ветка по умолчанию при открытии виджета	-
Кэшировать полный список маркеров	Нет	Кэширует на бэкенде DDP полный список маркеров выбранного снимка из Svacer; ускоряет пагинацию на вкладке «Находки»	Включено
Время жизни кэша (сек.)	Нет	Сколько секунд хранить ответ в памяти DDP; действует при включённом кэше. Допустимый диапазон при указании значения: 30–86400	180
Порог маркеров в снимке	Нет	Число маркеров, после которого снимок считается большим	10 000
Стратегия для больших снимков	Нет	Поведение при превышении порога маркеров (см. ниже)	Гибрид

Стратегии для больших снимков:

- «Гибрид» — вкладка «Находки» без фильтров недоступна; обзор собирается облегчёнными запросами к Svacer.
- «Без ограничений» — всегда загружается полный список маркеров; при большом снимке отображается только предупреждение.

Параметры запроса

При просмотре виджета доступны следующие параметры:

- «Ветка» — ветка Svacer для загрузки данных. Список формируется по проекту из настроек виджета. По умолчанию используется ветка из конфигурации виджета.
- «Снимок» — снимок ветки для загрузки данных. Значение «Последний снимок» соответствует актуальному снимку на ветке в момент обновления виджета.

- «Фильтры»:
  - «Статус ревью» — фильтр по статусу разметки маркера (подтверждено, ложное срабатывание, неясно, без решения, не исправлять).
  - «Критичность» — фильтр по уровню критичности находки.
  - «Чекер» — фильтр по имени чекера (например, `gosec.G402`).

Вкладки

- «Обзор» — прогресс ревью, число маркеров без ревью, срабатывания на ветке, распределение по критичности, динамика срабатываний, разбивка ревью по статусам и сравнение с предыдущим снимком (новые, исправленные, совпавшие и без изменений).
- «Находки» — таблица маркеров с колонками «Критичность», «Надёжность», «Место», «Чекер», «Ревью», «Сообщение» и «Теги». Поддерживается пагинация и переход к маркеру в Svacer по ссылке.

В нижней панели виджета отображаются сводные показатели по выбранному снимку: число маркеров в снимке, маркеры без ревью и отфильтрованные находки по уровням критичности.

> При стратегии «Гибрид» и числе маркеров в снимке выше порога вкладка «Находки» недоступна без фильтра по статусу ревью, критичности или чекеру.

S3 bucket

Виджет позволяет просматривать содержимое S3-совместимых хранилищ объектов, таких как Amazon S3, Yandex Object Storage, MinIO и другие.

Для каждого объекта доступно:

- Просмотр списка объектов в контейнере (bucket) с информацией о размере, дате изменения и классе хранения.
- Поиск объектов по префиксу (пути).
- Загрузка файлов из bucket.
- Просмотр детальной метаинформации объектов (размер, тип контента, метаданные, настройки кеширования и т. д.).
- Навигация по папкам bucket.

Аутентификация

Для работы с виджетом требуется учётная запись с правами доступа к S3 Bucket. Система поддерживает следующие методы аутентификации:

- Access Key ID и Secret Access Key.
- Поддержка различных S3-совместимых провайдеров через настройку эндпоинта.

> В отличие от других виджетов, S3 Bucket виджет не поддерживает использование внешних сервисов для передачи учётных данных. Все параметры аутентификации указываются непосредственно в конфигурации виджета.

Использование шаблонов для учётных данных

Для повышения безопасности можно использовать механизм шаблонизации с учётными данными:

- `{{ .credentials.accessKeyId }}` — подставить Access Key ID из учётных данных.
- `{{ .credentials.secretAccessKey }}` — подставить Secret Access Key из учётных данных.

> Доступность информации в виджете определяется уровнем прав подключённой учётной записи.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Название bucket	Да	Название S3 Bucket для просмотра	-
Endpoint	Да	Эндпоинт URL S3-совместимого хранилища (например, `https://storage.yandexcloud.net`)	-
Регион	Да	Регион, в котором находится Bucket	-
Access Key ID	Да	Идентификатор ключа доступа для аутентификации	-
Secret Access Key	Да	Секретный ключ доступа для аутентификации	-
Префикс	Нет	Префикс (путь) для фильтрации объектов при первоначальной загрузке	-
Максимум объектов	Нет	Максимальное количество объектов для отображения за один запрос (по умолчанию 100)	100

Дополнительные возможности виджета

Поиск объектов

Виджет позволяет искать объекты по префиксу (пути). При поиске список объектов обновляется в соответствии с заданным префиксом.

> Поиск работает только при вводе символов с начала названия объекта. Поиск по символам из середины названия не поддерживается.

Если в конфигурации виджета задан начальный префикс, поиск в виджете ограничивается этим префиксом. Загрузка и отображение файлов с другим префиксом недоступны.

Загрузка файлов

Виджет позволяет загружать файлы из bucket напрямую в браузер. Для каждого файла доступна кнопка загрузки.

Детальная информация об объектах

При клике на иконку документа для каждого объекта отображается детальная информация:

- Основные параметры: ключ, размер, дата изменения, класс хранения, тип контента, ETag.
- Информация о контенте: кодировка, язык, диспозиция.
- Настройки кеширования: Cache-Control, срок действия.
- Безопасность: серверное шифрование.
- Пользовательские метаданные.

Подгрузка дополнительных объектов

При наличии большого количества объектов в bucket доступна функция «Загрузить ещё» для пошаговой загрузки объектов без потери производительности.

Vault. Секреты

Виджет позволяет просматривать секреты в HashiCorp Vault или Deckhouse Stronghold. Поддерживается работа с KV v2 секретами.

> Виджет не передаёт значения секретов пользователю. На клиентскую сторону передаются только метаданные секретов (версия, время создания и т. д.) и структура ключей без их значений.

Для каждого секрета доступно:

- Просмотр иерархической структуры секретов и директорий.
- Просмотр метаданных секрета: версия, время создания, время удаления, статус уничтожения.

- Просмотр ключей секрета в формате таблицы «ключ/значение» (вместо значений отображается плейсхолдер).
- Навигация по вложенным секретам и директориям.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Путь	Да	Путь к секрету или директории в Vault. Необходимо явно указывать путь с `/data/`. Примеры: `services/data/`, `services/data/example`	-
Префикс UI	Нет	Префикс для URL интерфейса. Используйте `vault` для HashiCorp Vault или `stronghold` для Deckhouse Stronghold	-

Особенности работы с путями

Для работы с KV v2 секретами путь должен явно содержать `/data/`. Примеры корректных путей:

- `services/data/` — для просмотра всех секретов в директории `services`.
- `services/data/example` — для просмотра конкретного секрета `example`.
- `services/data/nested/secret` — для вложенных секретов.

Отображаемые данные

Виджет отображает следующую информацию:

Структура секретов

- «Директории» — отображаются с завершающим слешем (например, `nested/`) и всегда помечаются как директории, даже если содержат ключи.
- «Секреты» — отображаются без завершающего слеша и содержат ключи.

Метаданные секрета

Для каждого секрета отображаются следующие метаданные (если доступны):

- «Версия» — версия секрета в KV v2.
- «Время создания» — дата и время создания секрета.
- «Время удаления» — дата и время удаления секрета (для удалённых версий).
- «Статус уничтожения» — индикатор того, что секрет был уничтожен.

Ключи секрета

Ключи секрета отображаются в формате таблицы «ключ/значение»:

- Ключ — полный путь к ключу в структуре секрета (например, `database.host`).
- Значение — всегда маскируется символами `********` и не может быть раскрыто.

График

Виджет позволяет выводить информацию об объектах DDP в виде одного из следующих типов графиков:

- Столбчатая диаграмма;
- Кольцевая диаграмма;
- Круговая диаграмма;
- Полярная диаграмма;
- Радарная диаграмма.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Тип графика	Да	Тип визуализации графика	-
Название таблицы	Да	Название таблицы в базе данных, из которой будут браться записи для визуализации	-
Название поля	Да	Название поля, по которому будет происходить агрегация записей	-
Фильтры	Нет	Поля, по которым будут фильтроваться полученные записи, и их значения	-
Тип агрегации	Да	Принцип, по которому будут группироваться полученные записи	-
Параметры агрегации	Нет	Выбор временного периода и шага группировки при агрегации записей по дате	-

При настройке виджета следует учитывать, что названия полей в базе данных могут отличаться от названий полей в спецификации объектов. Общий принцип таков: формат camelCase в

спецификации объектов при сохранении структур в базу данных преобразуется в snake_case. Например:

- Поле `createdAt` в спецификации следует указывать в конфигурации виджета как `created_at`.
- Поле `resourceUuid` в спецификации следует указывать в конфигурации виджета как `resource_uuid`.

Доступно обращение к вложенным значениям. В таком случае разделителем для вложенности служит символ точки. Например, чтобы выполнить агрегацию по статусу сущностей, виджет следует настроить следующим образом:

Название таблицы	Название поля
`entities`	`health.status`

Типы агрегации

Дата

Данные на графике будут отсортированы и сгруппированы по выбранным временным интервалам.

В параметрах агрегации можно задать параметры:

- «Единица измерения шага» — например: секунды, минуты, часы, дни и т. д.
- «Количество единиц в одном шаге» — например: 5 минут, 2 часа, 1 день и т. п.

Это позволяет управлять детализацией отображения данных во времени и адаптировать график под нужный масштаб анализа.

Значение

Данные на графике отображаются в отсортированном порядке — по значениям.
Для каждого уникального значения в исходном наборе данных:

- Выполняется подсчёт количества вхождений.
- На графике отображается пара: значение — количество.

Это позволяет быстро увидеть распределение и частоту повторения различных значений.

Разбивка по интервалам

Тип агрегации «Разбивка по интервалам» позволяет гибко настроить отображение данных на графике, разделяя значения по заданным числовым диапазонам (интервалам). Это удобно для построения гистограмм и анализа распределения данных.

Доступны два режима настройки интервалов:

1. «Автоматическая разбивка по количеству интервалов».

Указывается только количество интервалов, на которые нужно разделить доступные данные.
Интервалы будут рассчитаны автоматически — равномерно от минимального до максимального значения.

2. «Ручное задание границ интервалов».

   Указывается массив числовых границ интервалов.
   Например: `0, 10, 20, 50`

   В этом случае:

   - Числа будут автоматически отсортированы по возрастанию.
   - Интервалы сформируются на основе отсортированных значений: `[0, 10)`, `[10, 20)`, `[20, 50]`

В параметрах агрегации должно быть указано хотя бы одно из двух:

- `Количество` — количество интервалов;
- `Границы` — границы интервалов.

Примеры:

- `Количество = 5` — построится 5 равных интервалов на основании данных.
- `Границы = 100, 0, 50` — после сортировки: `[0, 50, 100]`, график будет построен по интервалам `[0, 50)`, `[50, 100]`.

Квоты ресурсов Kubernetes

Виджет позволяет отображать данные о квотах ресурсов в кластере Kubernetes.

Для каждой квоты происходит визуализация занятых ресурсов.

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
URL	Да	URL API сервера Kubernetes. Используется для получения данных из Kubernetes	-
Namespace	Да	Неймспейс, из которого будут загружаться данные в виджет. Пример: `default`	-

Процентное значение

Виджет позволяет отображать заданное процентное значение.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ресурс	Нет	Ресурс, из которого извлекаются необходимые значения при обработке шаблона	-
Процентное значение	Нет	Значение, которое будет выводиться на виджет. Шаблонизация поддерживается. Пример без шаблонизации: `100`. Пример с шаблонизацией: `{{ .entity.properties.id }}`	-

Таблица сущностей

Виджет позволяет отображать сущности, созданные в DDP, в виде таблицы.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ресурс	Да	Ресурс, сущности которого отображаются в таблице	-
Показывать действия	Нет	Необходимость отображения действий с сущностями (возможность запуска действий и сценариев, возможность удаления и др.)	false

Kanban доска сущностей

Виджет отображает сущности выбранного ресурса на Kanban-доске.

Отображаемые данные

- «Колонки» — настроенные колонки доски и колонка «Без статуса» для сущностей без подходящего значения параметра состояния.
- «Карточки сущностей» — для каждой сущности отображаются название, описание (если указано), статус проверок, владелец и дата обновления.
- «Перемещение карточек» — перетаскивание карточки между колонками обновляет значение параметра состояния сущности (доступно при наличии прав на изменение сущностей).

Конфигурация

Название	Обязательность	Описание	Зн по ум
Ресурс	Да	Ресурс, сущности которого отображаются на доске	-
Параметр состояния	Да	Параметр сущности, определяющий колонку карточки. Поддерживаются типы `String`, `Number`, `Boolean`, `Enum`, `List`, `Date`, `Percentage`, `URL`	-
Колонки	Да	Список колонок доски. Для каждой колонки задаются: название (заголовок на доске), значение (значение параметра состояния; для параметров типа `Enum` и `List` выбирается из доступных опций) и цвет (цвет тега в заголовке). Порядок колонок настраивается перетаскиванием	-

Особенности

- Сущности без подходящего значения параметра состояния отображаются в колонке «Без статуса».
- При отсутствии прав на изменение сущностей перемещение карточек недоступно.

Временная шкала сущностей

Виджет отображает сущности выбранного ресурса на временной шкале.

Отображаемые данные

- «График временной шкалы» — горизонтальная диаграмма, где каждая сущность отображается в виде полосы, показывающей период времени (от даты начала до даты окончания).
- «Информация о сущностях» — при наведении на полосу отображается название сущности, дата начала и дата окончания периода.
- «Сортировка» — сущности отсортированы от самых старых (сверху) к самым новым (снизу).

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ресурс	Да	Ресурс, для которого отображается временная шкала	-
Поле даты начала	Да	Поле, из которого берется дата начала периода. Может быть системным полем (`createdAt`, `updatedAt`) или параметром типа `Date`	-
Поле даты окончания	Да	Поле, из которого берется дата окончания периода. Может быть системным полем (`createdAt`, `updatedAt`) или параметром типа `Date`	-

Особенности

- Виджет автоматически масштабирует временную шкалу для отображения всех сущностей;
- Сущности с некорректными датами (дата начала позже даты окончания) автоматически исключаются из отображения.

Календарь сущностей

Виджет отображает сущности выбранного ресурса в календаре.

Отображаемые данные

- «Недельный календарь» — сетка из 7 дней текущей недели.
- «Сущности по датам» — для каждого дня отображаются все сущности, у которых дата в выбранном поле соответствует этому дню.
- «Информация о сущностях» — для каждой сущности отображаются название и описание (если указано).
- «Навигация по неделям» — кнопки для перехода к предыдущей и следующей неделе.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ресурс	Да	Ресурс, для которого отображается календарь	-
Поле даты	Да	Поле, из которого берется дата для отображения сущности в календаре. Может быть системным полем (`createdAt`, `updatedAt`) или параметром типа `Date`	-

Особенности

- Виджет отображает текущую неделю по умолчанию (с понедельника по воскресенье).
- Доступна навигация между неделями с помощью кнопок «Предыдущая неделя» и «Следующая неделя».
- Для каждого дня отображается дата в формате `ДД.ММ`.
- Сущности отображаются в виде карточек с возможностью перехода на страницу сущности.
- Сущности с пустой или нулевой датой автоматически исключаются из отображения.

Статус сущности

Виджет отображает информацию о статусе сущности и результатах проверок статуса.

Отображаемые данные

Виджет показывает следующую информацию.

Общий статус

- «Прогресс-бар» — визуальное отображение общего статуса сущности с указанием процента успешно пройденных проверок.
- «Счётчик успешных проверок» — количество пройденных проверок из общего числа настроенных проверок статуса.

Список проверок

Для каждой проверки статуса отображается:

- «Название проверки» — название правила проверки.
- «Статус» — результат выполнения проверки:
  - «Пройдено» — проверка успешно пройдена.
  - «Не пройдено» — проверка не пройдена (ошибок выполнения нет).
  - «Ошибка» — при выполнении проверки произошла ошибка.
- «Время последней проверки» — дата и время последнего выполнения проверки.

- «Сообщение об ошибке» — текст ошибки (отображается, если проверка завершилась с ошибкой).

Статистика

В нижней части виджета отображается сводная статистика по проверкам:

- «Пройдено» — количество успешно пройденных проверок.
- «Не пройдено» — количество проверок, которые не были пройдены (без ошибок выполнения).
- «Ошибка» — количество проверок, завершившихся с ошибкой.

Заблокированные действия

Виджет автоматически определяет и отображает действия, которые недоступны при текущем статусе сущности.

- Условия отображения:
  - действие должно быть доступно для ресурса, связанного с сущностью;
  - у действия должны быть настроены разрешённые статусы;
  - текущий статус сущности не входит в список разрешённых статусов для этого действия.
- Отображаемая информация:
  - название действия;
  - описание действия (если указано).

Конфигурация

Виджет не требует дополнительной конфигурации.

Для работы виджета необходимо настроить проверки статуса для ресурса, связанного с сущностью, подробнее — в документации.

Особенности

Виджет имеет следующие особенности:

- если для сущности не настроено ни одной проверки статуса, виджет отображает сообщение о том, что проверки отсутствуют;
- если данные о проверках статуса недоступны, виджет отображает сообщение об отсутствии данных.

Статистика событий

Виджет отображает статистику событий, происходящих с сущностями в DDP. Виджет содержит три таба:

1. «Статистика событий» — график, показывающий количество событий по типам за выбранный временной период с настраиваемой группировкой по времени.
2. «Топ сущностей» — таблица с сущностями, для которых было сгенерировано максимальное количество событий.
3. «События в Redis» — таблица со стримами событий из Redis, показывающая для каждого стрима:
    - название стрима (кликабельное для просмотра всех событий);
    - ресурс, к которому относится стрим;
    - количество событий в стриме;
    - информацию о последнем событии (сущность, ресурс, тип события, время).

Параметры запроса

Название	Обязательность	Описание	Значение по умолчанию
Дата от	Да	Начальная дата для выборки событий	3 дня назад
Дата до	Да	Конечная дата для выборки событий	текущая дата
Интервал	Нет	Интервал группировки событий на графике (секунды, минуты, часы, дни, недели, месяцы, годы)	час
Шаг интервала	Нет	Количество единиц интервала для группировки	1
Топ сущностей	Нет	Количество сущностей с максимальным количеством событий для отображения в таблице	10

Типы событий

Виджет поддерживает следующие типы событий:

- `ENTITY_CREATED` — создание сущности.
- `ENTITY_UPDATED` — обновление сущности.
- `ENTITY_DELETED` — удаление сущности.

Особенности

- График показывает события за выбранный временной период с настраиваемой группировкой по времени (по умолчанию — по часам).

- Таблица отображает все события для каждой сущности (без фильтрации по дате).
- Для удалённых сущностей отображается их название, извлечённое из спецификации события.
- Вкладка «События в Redis» позволяет отслеживать события, хранящиеся в Redis Streams:
  - Для каждого стрима отображается количество событий и информация о последнем событии.
  - При клике на название стрима открывается диалог со всеми событиями из этого стрима.
  - Стримы автоматически привязываются к ресурсам по UUID, указанному в названии стрима.
  - При просмотре событий из стрима отображаются последние 1000 событий (новые первыми). Если в стриме больше 1000 событий, более старые события не отображаются.
- Каждая строка в таблице содержит информацию о последнем событии для сущности.
- Доступен просмотр детальной истории изменений для каждой сущности.
- События для удалённых ресурсов не отображаются (удаляются из БД при удалении ресурса).

Числовое значение

Виджет позволяет отображать заданное числовое значение.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
Ресурс	Нет	Ресурс, из которого извлекаются необходимые значения при обработке шаблона	-
Числовое значение	Нет	Значение, которое будет выводиться на виджет. Шаблонизация поддерживается. Пример без шаблонизации: `100`. Пример с шаблонизацией: `{{ .entity.properties.id }}`	-

Kaiten. Карточки пространства

Виджет позволяет отображать структуру задач в пространстве Kaiten в виде многоуровневой таблицы «Доска → Карточки», просматривать задачи на всех уровнях организации работы и получать информацию о критичных параметрах карточек (статус, срочность, блокировки, исполнители и др.).

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
ID пространства	Да	Идентификатор пространства в Kaiten	-

Параметры запроса

Название	Обязательность	Описание	Значение по умолчанию
Мои задачи	Нет	Фильтр по текущему пользователю	false
Создано после	Да	Начальная дата для выборки	1 месяц назад
Создано до	Да	Конечная дата для выборки	сейчас

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Отображаемые данные

Каждая карточка содержит:

- Название карточки.
- Колонка (статус в доске).
- Статус (очередь, в работе, готово).
- Линия.
- Владелец (аватар, имя, email).
- Участники.
- Срок (дата дедлайна, срочность).
- Блокированная/незаблокированная.

Kaiten. Статистика пространства

Виджет предоставляет агрегированные метрики и статистику карточек пространства Kaiten за выбранный период. Позволяет анализировать эффективность работы команды и выявлять узкие места в бизнес-процессах.

Конфигурация

Название	Обязательность	Описание	Значение по умолчанию
ID пространства	Да	Идентификатор пространства в Kaiten	-

Параметры запроса

Название	Обязательность	Описание	Значение по умолчанию
Создано после	Да	Начальная дата для анализа	1 месяц назад
Создано до	Да	Конечная дата для анализа	сейчас

Авторизация

Конфигурация авторизации описана в разделе «Внешние сервисы».

Отображаемые данные

Виджет содержит четыре вкладки:

Общие показатели

Основные метрики:

- В очереди: задачи в очереди на выполнение.
- Выполнено: завершённые задачи.
- В работе: активные задачи.

Дополнительные метрики:

- Заблокировано: количество заблокированных задач.
- Блокирующих: количество задач, блокирующих другие.
- Архивировано: количество задач в архиве.
- Срочных: количество срочных задач.
- В среднем на выполнение: среднее время выполнения (в минутах).

Статистика по чеклистам:

- Всего с чек-листом: общее количество задач с чек-листами.
- Чеклист полностью выполнен: задачи с полностью выполненными чек-листами.
- Чеклист не выполнен: задачи с невыполненными чек-листами.

По пользователю

- Список пользователей с количеством назначенных задач.
- Визуализация в виде прогресс-баров.
- Количество задач на каждого пользователя.

Забытые задачи

Карточки, которые не обновлялись с момента создания.

Последние обновлённые

Десять последних обновлённых карточек.

Очередь задач

Виджет позволяет отслеживать состояние очереди задач и работу воркеров, обрабатывающих задачи в фоновом режиме. Виджет отображает статистику очереди, информацию о воркерах (консьюмерах) и детали всех задач в очереди.

Отображаемые данные

Виджет состоит из трех основных разделов:

Статистика очереди

В верхней части виджета отображаются четыре ключевых показателя:

- «Размер очереди» — общее количество задач в очереди.
- «Ожидающие задачи» — количество задач, ожидающих обработки.
- «Активные воркеры» — количество активных воркеров (консьюмеров), обрабатывающих задачи.
- «Задачи в очереди» — общее количество задач, включая новые и обрабатываемые.

Таблица воркеров

Таблица содержит информацию о каждом активном воркере:

- «Название консьюмера» — идентификатор воркера (консьюмера).
- «Ожидающие задачи» — количество задач, назначенных данному воркеру и ожидающих обработки.
- «Время простоя» — время с момента последней активности воркера.

> В таблице отображаются только активные воркеры. Воркеры, которые не обрабатывают задачи и неактивны более 5 минут, автоматически скрываются из списка.

Таблица задач

Таблица содержит детальную информацию о всех задачах в очереди:

- «UUID задачи» — уникальный идентификатор задачи.

- «Тип» — тип задачи (например, `health_check`).
- «UUID ресурса» — идентификатор ресурса или сущности, к которой относится задача.
- «Консьюмер» — название консьюмера, обрабатывающего задачу.
- «Время простоя» — время с момента доставки задачи воркеру.
- «Время доставки» — время, когда задача была доставлена воркеру.
- «Статус» — текущий статус задачи:
  - «Новая» — задача добавлена в очередь, но ещё не назначена воркеру.
  - «В обработке» — задача назначена воркеру и обрабатывается.

Конфигурация

Виджет не требует дополнительной конфигурации и работает сразу после добавления на дашборд.

Технологический радар

Виджет позволяет визуализировать технологии, инструменты и практики, используемые в компании, с их разбивкой по уровням зрелости (Adopt, Trial, Assess, Hold).

На виджете отображается круговая диаграмма: четыре квадранта, четыре кольца и набор элементов с номером, названием и привязкой к квадранту и кольцу. Наполнение радара настраивается в конфигурации виджета.

Конфигурация

Название	Обязательность	Описание
Квадранты	Да	Названия квадрантов
Элементы	Нет	Список элементов на радаре (не более 200) и их конфигурация

Конфигурация элемента

Название	Обязательность	Описание
Название	Да	Название элемента
Номер	Да	Целое число от 0 до 9999
Описание	Нет	Описание элемента, в диалоге просмотра отображается в формате Markdown
Квадрант	Да	Квадрант, к которому относится элемент
Кольцо	Да	Кольцо, к которому относится элемент: Adopt, Trial, Assess или Hold

Проверки статуса

Обзор

Для каждого ресурса можно настроить набор правил, по которым будет определяться статус его сущностей. Возможно добавление произвольного количества правил. Поле `Условие` определяет, должны ли выполняться:

- все правила (`AllOf`);
- хотя бы одно из них (`AnyOf`).

> Если хотя бы одна из проверок завершилась с ошибкой, то статус сущности будет выставлен в `error` независимо от результата остальных проверок и условия.

Расписание

Планировщик проверок запускается каждую минуту. Для каждого правила в поле «Расписание» можно задать cron-выражение из пяти полей — тогда проверка по этому правилу выполняется только в указанные моменты времени. Если поле пустое, правило выполняется при каждом запуске планировщика (каждую минуту).

Пример: `0 * * * *` — проверка в начале каждого часа.

Если правило в текущий момент по расписанию не выполняется, при расчёте статуса сущности используется результат последней проверки по этому правилу.

Логи последних нескольких проверок доступны в меню ресурса в разделе «Проверки статуса».

Статусы сущности

Возможны четыре варианта статуса:

- `healthy` — правила заданы, и параметры сущности соответствуют этим правилам;
- `unhealthy` — правила заданы, но параметры сущности им не соответствуют;
- `unknown` — правила не заданы либо проверка не может быть выполнена по каким-либо причинам;
- `error` — во время выполнения хотя бы одного из правил произошла ошибка.

Для каждой сущности при клике на плашку со статусом открывается таблица с результатами выполнения правил и дополнительной информацией. В таблице отображаются только правила, для которых уже выполнялась хотя бы одна проверка.

> Событие `ENTITY_UPDATED` генерируется только при изменении статуса сущности.

Типы проверок статуса

Property

Правило типа `Property` проверяет, соответствует ли конкретный параметр сущности заданному шаблонному выражению.

Конфигурация правила состоит из одного параметра — выражения. Для описания выражения используется синтаксис Go template.

Примеры выражений:

- `{{ eq .entity.properties.lifecycle "deployed" }}` — значение свойства `lifecycle` должно быть `"deployed"`;
- `{{ lt .entity.properties.vulnerabilities 10 }}` — значение свойства `vulnerabilities` должно быть меньше `10`.

Prometheus

Правило типа Prometheus проверяет, соответствует ли указанная метрика заданному пороговому значению. В конфигурации указывается запрос PromQL, который должен возвращать результат типа Scalar или Vector с одним значением.

Поддерживается шаблонизация, например:

```
avg(ingress_nginx_detail_request_seconds_sum{location="/{{ .entity.slug }}"})
```

В данном примере вместо выражения `{{ .entity.slug }}` подставится идентификатор сущности.

Параметры конфигурации

Название	Описание	Возможные значения
Запрос	Запрос в формате PromQL к метрике в Prometheus	
Оператор	Оператор сравнения между результатом запроса и пороговым значением	Equal, NotEqual, LessThan, GreaterThan
Порог	Пороговое значение для сравнения с результатом запроса	

> Для проверки каждой сущности выполняется отдельный запрос к Prometheus. Рекомендуется учитывать это при планировании нагрузки на систему.

Авторизация

Конфигурация авторизации описана в разделе внешний сервис Prometheus.

Gitlab Pipeline

Правило типа `GitlabPipeline` проверяет, соответствует ли статус последнего pipeline в Gitlab для выбранного Ref (ветка или тег) заданному статусу.

Во всех текстовых полях поддерживается шаблонизация, например, при подобном выражении в поле `Ref`:

```
{{ .entity.properties.mainBranch }}
```

во время проверки подставится значение параметра `mainBranch` проверяемой сущности.

Параметры конфигурации

Название	Описание	Возможные значения
ID проекта	ID проекта в Gitlab	
Ref	Ветка или тег для которых будет искаться последний pipeline	
Статус	Статус pipeline, который будет считаться успешным	created, waiting_for_resource, preparing, pending, running, success, failed, canceled, skipped, manual, scheduled

> Для проверки каждой сущности выполняется отдельный запрос к Gitlab. Рекомендуется учитывать это при планировании нагрузки на систему.

Авторизация

Конфигурация авторизации описана в разделе внешний сервис GitLab.

DefectDojo Findings

Правило типа `DefectDojoFindings` выполняет проверку по количеству активных уязвимостей различной критичности (severity) для заданного продукта в DefectDojo.

Проверка проводится по условиям в блоке `conditions`, где можно указать ожидаемое количество уязвимостей для каждой severity и оператор сравнения (например, Critical < 5).

Во всех текстовых полях поддерживается шаблонизация. Например, можно подставить название продукта из параметров сущности:

```
{{ .entity.properties.defectdojo_product_key }}
```

Параметры конфигурации

Название	Описание	Возможные значения
Название продукта	Идентификатор продукта в DefectDojo	
Условия	Условия для сравнения количества уязвимостей по severity	

Условия

Каждое условие представляет объект следующей формы:

```
conditions:
  - severity: Critical
    operator: "<"
    value: "5"
  - severity: High
    operator: "<="
    value: "10"
```

Поле	Описание	Возможные значения
Уровень критичности	Уровень критичности	Total, Critical, High, Medium, Low, Info
Оператор	Оператор сравнения	==, !=, <, <=, >, >=
Значение	Целевое значение для сравнения	

Авторизация

Конфигурация авторизации описана в разделе внешний сервис DefectDojo.

CodeScoring Vulnerabilities

Правило типа `CodeScoringVulnerabilities` выполняет проверку по количеству уязвимостей различной критичности (severity) для заданного проекта в CodeScoring.

Проверка проводится по условиям в блоке `conditions`, где можно указать ожидаемое количество уязвимостей для каждой severity и оператор сравнения. Поддерживается проверка как по CVSS2, так и по CVSS3 метрикам.

Во всех текстовых полях поддерживается шаблонизация. Например, можно подставить ID проекта из параметров сущности:

```
{{ .entity.properties.codescoring_project_id }}
```

Параметры конфигурации

Название	Описание	Возможные значения
ID проекта	Идентификатор проекта в CodeScoring	
Условия	Условия для сравнения количества уязвимостей по severity	

Условия

Каждое условие представляет объект следующего формата:

```
conditions:
  - severity: CRITICAL
    operator: "<"
    value: "5"
    cvss: "cvss3"
  - severity: HIGH
    operator: "<="
    value: "10"
    cvss: "cvss2"
```

Поле	Описание	Возможные значения
Уровень критичности	Уровень критичности	Для CVSS3: CRITICAL, HIGH, MEDIUM, LOW, NONE, UNKNOWN. Для CVSS2: HIGH, MEDIUM, LOW, NONE
Оператор	Оператор сравнения	==, !=, <, <=, >, >=
Значение	Целевое значение для сравнения	
CVSS	Версия CVSS метрики	cvss2, cvss3

Авторизация

Конфигурация авторизации описана в разделе внешний сервис CodeScoring.

SonarQube Metrics

Правило типа `SonarqubeMetrics` выполняет проверку по метрикам проекта в SonarQube на основании заданных условий.

Проверка выполняется с использованием REST API вызова SonarQube `/api/measures/component`, где сравниваются текущие значения указанных метрик с ожидаемыми значениями, заданными в разделе «Условия».

Во всех текстовых полях поддерживается шаблонизация, например, можно подставить ключ компонента из параметров сущности:

```
{{ .entity.properties.sonarqube_project_key }}
```

Параметры конфигурации

Название	Обязательность	Описание	Возможные значения
Ключ проекта	Да	Идентификатор проекта в SonarQube	
Ветка	Нет	Ветка проекта для которой будут браться метрики	
Условия	Да	Условия для сравнения метрик в SonarQube	

Условия

Каждое условие представляет объект следующей формы:

```
conditions:
  - metric: coverage
    operator: "<"
    value: "5"
  - metric: bugs
    operator: "<="
    value: "10"
```

Поле	Описание	Возможные значения
Метрика	Метрика SonarQube. В конфигурации указывается Metric key	См. список метрик в официальной документации SonarQube
Оператор	Оператор сравнения	==, !=, <, <=, >, >=
Значение	Целевое значение для сравнения	

Список возможных метрик для текущей версии SonarQube.

Авторизация

Конфигурация авторизации описана в разделе внешний сервис SonarQube.

SonarQube Quality Gate

Правило типа `SonarqubeQualityGate` проверяет статус Quality Gate проекта в SonarQube. Проверка выполняется с использованием REST API вызова SonarQube `/api/qualitygates/project_status`.

Правило возвращает `true` (выполняется), если Quality Gate проекта имеет статус `OK`, и `false` (не выполняется) во всех остальных случаях (статусы `WARN`, `ERROR`, `NONE`).

Во всех текстовых полях поддерживается шаблонизация, например, можно подставить ключ проекта из параметров сущности:

```
{{ .entity.properties.sonarqube_project_key }}
```

Параметры конфигурации

Название	Обязательность	Описание	Возможные значения
Ключ проекта	Да	Идентификатор проекта в SonarQube	
Ветка	Нет	Ветка проекта для которой будет проверяться Quality Gate. Если не указана, проверяется основная ветка	

> Для проверки каждой сущности выполняется отдельный запрос к SonarQube. Рекомендуется учитывать это при планировании нагрузки на систему.

Авторизация

Конфигурация авторизации описана в разделе внешний сервис SonarQube.

URL

Правило типа `URL` проверяет доступность HTTP/HTTPS-эндпоинта: отправляется запрос с заданными параметрами, результат оценивается по одному или нескольким условиям — выражениям Go template. В условиях можно проверять код ответа, заголовки, тело ответа и параметры сущности.

Во всех строковых полях (URL, тело запроса, заголовки) доступна шаблонизация с подстановкой данных сущности (`.entity`), например: `{{ .entity.slug }}` .

Параметры конфигурации

Название	Обязательность	Описание	Примеры
URL	Да	Полный адрес, по которому выполняется запрос	`https://example.com`
Метод	Нет	HTTP-метод запроса	GET, POST
Query	Нет	Значение query, подставляемое в исходящий запрос	
Сохранять детали	Нет	Если включено — в таблице проверок сохраняются и отображаются тело ответа и статус (код, заголовки). Если выключено — только условие, результат и ошибка	
Условия	Нет	Выражения Go template, по которым оценивается ответ	См. ниже
При нескольких условиях	Нет	`AllOf` — должны выполниться все условия; `AnyOf` — достаточно одного	AllOf, AnyOf
Тело запроса	Нет	Тело запроса в формате YAML. После подстановки шаблонов преобразуется в JSON и отправляется	

Если условия не заданы, по умолчанию используется проверка кода ответа 200: `{{ eq .status.code 200 }}` .

Условия

В условиях доступны:

- `.status` — данные ответа: `.status.code` (код HTTP), `.status.status` (строка статуса), `.status.headers` (заголовки), `.status.contentLength` . `.status.headers` — карта

«название заголовка → массив значений» (названия в нижнем регистре). Пример: первый элемент заголовка — `{{ index (index .status.headers "content-type") 0 }}`; перебор — `{{ range index .status.headers "set-cookie" }}...{{ end }}`.

- `.response` — тело ответа с приведёнными типами (для JSON-ответа).
- `.entity` — параметры проверяемой сущности.

Примеры условий:

```
{{ eq .status.code 200 }}
{{ eq (index (index .status.headers "content-type") 0) "application/json" }}
{{ gt .status.contentLength 0 }}
```

Пример тела запроса

Поле «Тело запроса» задаётся в формате YAML. Пример:

```
id: "{{ .entity.id }}"
name: "{{ .entity.name }}"
```

Проверка результата

Проверка считается успешной, если при режиме AllOf выполняются все условия, при AnyOf — хотя бы одно. Если при вычислении условия возникает ошибка (например, ответ не в JSON, а в условии используется `.response`), проверка завершается с ошибкой.

Автоматизации

Автоматизации — механизм реагирования платформы на изменения спецификации сущностей.

Каждая автоматизация может подписываться на обновления сущностей какого-либо ресурса по определённым правилам. При возникновении события обновления автоматизация запускает выполнение действия, процесса или синхронизации источника данных.

Автоматизация может запускать выполнение как для сущности, которая вызвала триггер, так и для связанных с ней сущностей в соответствии с настройками.

> Если выполнение настроено для связанных сущностей, оно не будет запущено для сущности, вызвавшей событие. При настройке запуска для связанных сущностей поддерживается только выполнение действий или процессов.

Интерфейс настройки автоматизации

Форма создания и редактирования автоматизации разделена на логические секции.

Триггеры

Секция для настройки условий, при которых автоматизация будет запускаться. Каждая автоматизация может содержать произвольное количество триггеров.

При настройке каждого триггера указываются:

- «Ресурс» — ресурс, события с сущностями которого будут являться триггером для запуска автоматизации;
- «Условие» — условие в формате Go template, описывающее, изменения какого параметра будет считаться триггером для запуска автоматизации и какое именно значение должен принять изменённый параметр.

Примеры условий триггера:

- `{{ ne .entity.properties.id 0 }}` — автоматизация будет запускаться, если значение параметра `id` изменилось и после изменения стало не равным нулю;
- `{{ eq .entity.properties.status "Failed" }}` — автоматизация будет запускаться, если значение параметра `status` изменилось на `Failed` с любого другого.

Выполнение

Секция для настройки того, что будет выполняться при срабатывании триггера автоматизации.

Настраиваются:

- «Тип выполнения» — тип объекта, который будет выполняться: действие, процесс или синхронизация источника данных;
- «Объект выполнения» — конкретный объект (действие, процесс или источник данных), который будет выполняться.

Для действий и процессов доступна дополнительная настройка:

- «Запускать выполнение для связанных ресурсов» — при включении этой опции выполнение будет автоматически запускаться для всех сущностей, имеющих соответствующую связь с изменённой сущностью. При этом выполнение не будет запущено на сущность, вызвавшую событие.

Синхронизация

Секция для настройки периодической синхронизации — автоматического запуска автоматизации по расписанию.

Настраиваются:

- «Периодическая синхронизация» — включение/выключение периодической синхронизации;
- «Периодичность запуска» — cron-выражение для настройки частоты запуска автоматизации (например, `0 0 * * *` для ежедневного запуска в полночь).

Триггеры

Механизм работы триггеров автоматизаций основан на событиях (events), которые генерируются при каждом создании, изменении либо удалении любой сущности любого ресурса.

Автоматизации:

1. Прослушивают события указанных в конфигурации триггеров ресурсов.
2. Проверяют, изменилось ли значение параметра, указанного в условиях триггера.
3. При изменении сравнивают текущее значение параметра с условием.
4. При соответствии условию запускают выполнение.

> ⚠ Новые автоматизации обрабатывают только события, которые произошли после их создания. События, произошедшие до создания автоматизации, не будут обработаны.

Вебхуки

По механизму действия вебхуки похожи на источники данных. При этом, если источник данных работает по pull-модели, то есть сам опрашивает инфраструктурные системы и пытается загрузить из них данные, то вебхуки ожидают, что данные будут отправлены извне, то есть работают по push-модели.

При этом настройка правил обработки (создание, сопоставление, обновление, создание связей) аналогична настройке подобных правил в источниках данных.

У каждого вебхука есть уникальный URL, который формируется по шаблону: `<url backend>/webhooks/<UUID вебхука>`. Вебхук принимает POST-запросы с массивом данных любого формата, который обрабатывается на основе правил вебхука.

Внешние сервисы

Внешние сервисы — это механизм, позволяющий настраивать параметры авторизации для внешних инфраструктурных систем (например, GitLab, Kubernetes, DefectDojo и др.).

Настройка внешних сервисов осуществляется в разделе «Администрирование» → «Внешние сервисы».

Конфигурация

Конфигурация внешнего сервиса включает следующие параметры:

Название параметра	Описание
«Название»	Название внешнего сервиса — отображается в интерфейсе
«Идентификатор»	Уникальный человекочитаемый идентификатор (slug)
«Владелец»	Пользователь, ответственный за внешний сервис
«Команда-владелец»	Команда, которой принадлежит внешний сервис
«URL»	Базовый URL для обращения к внешнему сервису
«Учётные данные»	Список доступных учётных данных (например, токенов), которые могут быть подставлены в запросы
«Заголовки»	HTTP-заголовки, автоматически добавляемые к запросам
«Отключить проверку SSL»	Отключение проверки SSL-сертификата внешнего сервиса (например, при использовании самоподписанных сертификатов). Рекомендуется вместо этого добавить нужные сертификаты в «доверенные»
«Системная учётная запись»	Учётная запись, которая будет использоваться для периодически запускаемых заданий: источников данных, проверок статуса и т. д

Использование внешних сервисов

Внешние сервисы могут быть подключены к различным объектам платформы:

- действиям,
- виджетам,
- источникам данных,
- проверкам статуса сущностей.

Подключение сервиса выполняется на вкладке «Авторизация» в настройках соответствующего объекта.

Для каждого объекта можно в явном виде переопределить параметры, заданные во внешнем сервисе.
Например, если внешний сервис содержит токен авторизации, но в конкретном действии используются другие учётные данные, можно указать альтернативные значения. В этом случае будет переопределён только указанный параметр, остальные значения будут взяты из конфигурации внешнего сервиса.

Особенности подключения к объектам

Действия

- К одному действию можно подключить несколько внешних сервисов.
- Можно выбрать сервис по умолчанию.
- Параметр «Системная учётная запись» не используется в действиях.
- При запуске действия можно выбрать, для какого внешнего сервиса оно будет запускаться. Если ни один сервис не выбран, будет использоваться сервис по умолчанию. Если сервис по умолчанию не задан, то будет использоваться первый внешний сервис в списке.

Виджеты

- Для одного виджета также поддерживается подключение нескольких внешних сервисов.
- Один из них может быть выбран как используемый по умолчанию.
- В последующих релизах будет добавлена возможность изменения сервиса во время просмотра виджета.
- Параметр «Системная учётная запись» не используется в виджетах.

Источники данных

- Поддерживается подключение только одного внешнего сервиса.
- Если в сервисе задана системная учётная запись, она используется по умолчанию при синхронизации.

Проверки статуса

- Проверке может быть назначен один внешний сервис.
- Если у него указана системная учётная запись — она используется при запуске автоматических проверок.

Заголовки авторизации для внешних сервисов

При настройке внешних сервисов необходимо указать соответствующие HTTP-заголовки для авторизации. Ниже приведён список внешних сервисов, поддерживаемых платформой, с указанием типа авторизации и необходимых заголовков.

> Приведённые ниже примеры демонстрируют возможные способы авторизации для каждого сервиса. В некоторых сервисах могут использоваться другие механизмы. Платформа поддерживает все варианты, которые можно передать через HTTP-заголовки.

CodeScoring

Тип авторизации: API Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`<токен>`

Пример:

```
Authorization: <ваш-api-token>
```

ClickHouse

Тип авторизации: Basic Authentication или заголовки `X-ClickHouse-User` / `X-ClickHouse-Key`.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`
`X-ClickHouse-User`	`<имя-пользователя>`
`X-ClickHouse-Key`	`<пароль>`

Пример Basic Authentication:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

Пример с заголовками ClickHouse:

```
X-ClickHouse-User: <имя-пользователя>
X-ClickHouse-Key: <пароль>
```

DefectDojo

Тип авторизации: API v2 Key Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Token <токен>`

Пример:

```
Authorization: Token <ваш-defectdojo-api-v2-key>
```

Bitbucket

Тип авторизации: Bearer Token (Personal Access Token).

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-bitbucket-personal-access-token>
```

Docker Registry

Тип авторизации: Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

GitLab

Тип авторизации: Personal Access Token или Project Access Token.

Заголовки:

Заголовок	Формат значения
`Private-Token`	`<токен>`

Пример:

```
Private-Token: <ваш-gitlab-token>
```

Подробнее о том, как получить GitLab token можно узнать в официальной документации GitLab.

GitHub

Тип авторизации: Personal Access Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-github-token>
```

Токен создаётся в настройках GitHub: «Settings» → «Developer settings» → «Personal access tokens».

Harbor

Тип авторизации: Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

Jenkins

Тип авторизации: Basic Authentication (Username/Password).

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`, где:
   - `username` — имя пользователя в Jenkins.
   - `password` — пароль пользователя.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

Jira

Тип авторизации: Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

Kaiten

Тип авторизации: API Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-kaiten-api-token>
```

Kubernetes

Тип авторизации: Bearer Token (Service Account Token или User Token).

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-kubernetes-token>
```

Nexus

Тип авторизации: Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

OpenSearch

Тип авторизации: Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Basic <base64-encoded-credentials>`

Пример:

1. Сформируйте строку `username:password`.
2. Закодируйте её в Base64: `echo -n "username:password" | base64`.
3. Добавьте заголовок:

```
Authorization: Basic <base64-encoded-credentials>
```

Prometheus

Тип авторизации: Bearer Token или Basic Authentication.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>` или `Basic <base64-encoded-credentials>`

Пример Bearer Token:

```
Authorization: Bearer <ваш-токен>
```

Пример Basic Authentication:

```
Authorization: Basic <base64-encoded-credentials>
```

SonarQube

Тип авторизации: Bearer Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-токен>
```

Svacer

Тип авторизации: Bearer Token.

Заголовки:

Заголовок	Формат значения
`Authorization`	`Bearer <токен>`

Пример:

```
Authorization: Bearer <ваш-токен>
```

Vault

Тип авторизации: Token Authentication.

Заголовки:

Заголовок	Формат значения
`X-Vault-Token`	`<токен>`

Пример:

```
X-Vault-Token: <ваш-vault-token>
```

Доверенные сертификаты

Механизм доверенных сертификатов позволяет загрузить в Deckhouse Development Platform (DDP) корневые и промежуточные сертификаты (CA) или сертификаты серверов. Платформа использует их при проверке TLS/SSL при подключении к внешним сервисам по HTTPS — например, при обращении к API внешних сервисов в источниках данных, виджетах и т. д.

Это даёт возможность безопасно работать с сервисами, использующими самоподписанные или корпоративные сертификаты, без отключения проверки SSL.

> Для Dex, PostgreSQL и Redis, используемых платформой в качестве служебных сервисов, сертификаты необходимо добавлять через конфигурацию модуля: соединение с этими компонентами устанавливается до включения механизма доверенных сертификатов.

Настройка доверенных сертификатов осуществляется в разделе «Администрирование» → «Доверенные сертификаты».

Конфигурация

При добавлении доверенного сертификата указываются следующие параметры:

Параметр	Описание
«Название»	Отображаемое название сертификата в интерфейсе (например, «Корпоративный CA»)
«Сертификат (PEM)»	Содержимое сертификата в формате PEM. Обязателен при создании

Сертификат задаётся в формате PEM, например:

```
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAKL...
...
-----END CERTIFICATE-----
```

После сохранения в карточке сертификата отображаются данные, извлечённые из PEM: субъект, издатель, срок действия, альтернативные имена и т. д.

При редактировании сертификата поле «Сертификат (PEM)» скрыто. Сохранённое значение не возвращается API из соображений безопасности. Чтобы обновить сертификат, создайте новую запись и, при необходимости, удалите старую.

Использование

Загруженные доверенные сертификаты автоматически добавляются в пул корневых сертификатов, используемый DDP Backend при исходящих HTTPS-запросах. Этот пул применяется при:

- обращении к внешним сервисам (действия, виджеты, источники данных, проверки статуса);
- запросах к API инфраструктурных систем (GitLab, Kubernetes, Vault, Prometheus и др.).

Таким образом, если внешний сервис использует сертификат, выпущенный вашим или корпоративным CA, достаточно добавить соответствующий корневой или промежуточный сертификат в раздел «Доверенные сертификаты» — после этого платформа будет доверять таким соединениям без отключения проверки SSL.

> Вместо отключения проверки SSL во внешнем сервисе (опция «Отключить проверку SSL») рекомендуется добавлять нужные сертификаты в доверенные — это сохраняет проверку подлинности и защиту соединения.

Изменения списка доверенных сертификатов (добавление, изменение, удаление) применяются к новым исходящим соединениям без перезапуска DDP.

Права доступа

Управление доверенными сертификатами регулируется глобальными разрешениями:

- `read:trusted-certificates` — просмотр списка и сведений о сертификатах;
- `create:trusted-certificates` — добавление сертификатов;
- `update:trusted-certificates` — редактирование названия существующего сертификата;
- `delete:trusted-certificates` — удаление сертификатов.

Подробнее о ролевой модели в документации по RBAC.

Наборы данных

Наборы данных — это предварительно настроенные конфигурации, предназначенные для быстрого развёртывания готовых интеграций с внешними системами и настройки необходимых компонентов платформы.

Каждый набор данных содержит:

- Дашборды для визуализации данных.
- Виджеты для отображения информации.
- Источники данных для синхронизации.
- Внешние сервисы для подключения к API.
- Типы учётных данных для аутентификации.
- Роли и права доступа.
- Другие необходимые компоненты платформы.

Управление наборами данных

Просмотр доступных наборов

В разделе «Администрирование» → «Наборы данных» отображается список всех доступных наборов данных. Каждая карточка содержит:

- «Название» — название набора данных.
- «Краткое описание» — общее описание функциональности.
- Кнопка «Описание» — открывает подробную информацию о наборе данных, включая:
  - подробное описание с полной информацией о компонентах;
  - список объектов, которые будут созданы при применении.
- Кнопка «Применить» — запускает процесс установки набора данных.
- Кнопка «Удалить» — удаляет ранее применённый набор данных.

Применение набора данных

1. Нажмите кнопку «Применить» на карточке нужного набора данных.
2. Если набор данных требует настройки параметров, откроется форма конфигурации.
3. Заполните необходимые параметры и нажмите «Применить».
4. Если параметры не требуются, появится окно подтверждения.
5. После успешного применения все компоненты набора данных будут созданы в системе.

Удаление набора данных

1. Нажмите кнопку «Удалить» на карточке примененного набора данных.
2. Подтвердите удаление в диалоговом окне.
3. Все компоненты набора данных будут удалены из системы.

Настройка после применения

После применения некоторых наборов данных может потребоваться дополнительная настройка:

1. Перейдите в «Профиль» → «Учётные данные».
2. Заполните необходимые учётные данные (токены, пароли и т.д.).
3. Сохраните изменения.

Доступные наборы данных

GitLab

Набор данных предназначен для работы с GitLab-репозиториями и анализа процессов CI/CD.

Дашборды:

- «Статистика репозитория» — для анализа статистики пайплайнов.
- «Управление репозиторием» — для управления репозиториями.

Виджеты:

- «Статистика пайплайнов GitLab» — для детальной аналитики пайплайнов.
- «GitLab пайплайны» — для управления пайплайнами.
- «Редактор пайплайнов GitLab» — для редактирования конфигурации пайплайнов.

Настройка:

Перед применением создайте внешний сервис для GitLab и выберите его при настройке набора данных.

Параметры настройки:

- «Внешний сервис» — внешний сервис GitLab, который будет использоваться набором данных.

Требования:

- Настроенный внешний сервис GitLab с доступом к API.
- Учётные данные для внешнего сервиса с правами на чтение репозиториев и пайплайнов.

Применение:

1. Создайте внешний сервис для GitLab в разделе «Администрирование» → «Внешние сервисы».
2. Нажмите «Применить» на карточке набора данных GitLab.
3. В форме конфигурации выберите созданный внешний сервис и нажмите «Применить».
4. Перейдите в «Профиль» → «Учётные данные».
5. Заполните учётные данные, указанные во внешнем сервисе.
6. Сохраните изменения.

После этого в разделе «Каталог» появятся ваши репозитории GitLab, а на дашбордах будет отображаться статистика и управление пайплайнами.

Yandex Cloud

Набор данных предназначен для подключения к API Yandex Cloud.

Создаваемые объекты:

- внешний сервис «Yandex Cloud»;
- тип учётных данных «Yandex Cloud сервисный аккаунт» — JSON-ключ сервисного аккаунта;
- тип учётных данных «Yandex Cloud IAM-токен» — короткоживущий токен для запросов к API;
- действие «Yandex Cloud: получить IAM токен» — получение IAM-токена по JSON-ключу сервисного аккаунта и сохранение его в учётных данных.

Настройка:

Набор данных настраивает подключение к API Yandex Cloud через внешний сервис. IAM-токен имеет ограниченный срок действия, поэтому его нужно периодически обновлять с помощью действия «Yandex Cloud: получить IAM токен».

> После применения укажите в профиле супер-администратора JSON-ключ сервисного аккаунта, затем выполните действие «Yandex Cloud: получить IAM токен» для заполнения IAM-токена.

Параметры настройки:

Название	Описание	Значение по умолчанию
«URL API»	Базовый URL API Yandex Cloud	`https://api.cloud.yandex.net`

Применение:

1. Примените набор данных Yandex Cloud.
2. Перейдите в профиль супер-администратора.
3. Заполните учётные данные «Yandex Cloud сервисный аккаунт» JSON-ключом.
4. Выполните действие «Yandex Cloud: получить IAM токен».
5. Подключите внешний сервис «Yandex Cloud» к действиям, виджетам или источникам данных, которые обращаются к API Yandex Cloud.

После этого запросы через внешний сервис будут использовать актуальный IAM-токен из учётных данных пользователя.

Vault

Набор данных предназначен для подключения к API HashiCorp Vault.

Создаваемые объекты:

- внешний сервис «Vault»;
- тип учётных данных «Vault токен» — токен для запросов к API.

Настройка:

Набор данных настраивает подключение к API Vault через внешний сервис с авторизацией по заголовку `X-Vault-Token`. Подробнее о формате авторизации — в разделе «Внешние сервисы».

> После применения укажите в профиле супер-администратора токен Vault в учётных данных «Vault токен».

Параметры настройки:

Название	Описание	Значение по умолчанию
«URL Vault»	Базовый URL API HashiCorp Vault	`https://vault.example.com`

Применение:

1. Примените набор данных Vault.
2. Укажите URL вашего сервера Vault.
3. Перейдите в профиль супер-администратора.
4. Заполните учётные данные «Vault токен».
5. Подключите внешний сервис «Vault» к действиям, виджетам или источникам данных, которые обращаются к API Vault.

После этого запросы через внешний сервис будут использовать токен из учётных данных пользователя. Например, внешний сервис можно подключить к виджету «Vault. Секреты» для просмотра секретов в KV v2.

Deckhouse managed PostgreSQL

Набор данных для развёртывания managed PostgreSQL в Deckhouse Kubernetes Platform.

Набор данных подготавливает ресурс, действия и процессы для управления жизненным циклом managed PostgreSQL: создание инстанса в кластере через внешний сервис Kubernetes и сохранение реквизитов подключения в Vault или Stronghold.

Настройка:

Перед применением настройте внешние сервисы Kubernetes и Vault (или Stronghold).

> Подробное описание компонентов набора данных доступно по кнопке «Описание» на карточке в разделе «Наборы данных».

Параметры настройки:

Название	Описание	Значение по умолчанию
«Внешний сервис Kubernetes»	Kubernetes кластер, в котором будет разворачиваться managed PostgreSQL	—
«Внешний сервис Vault или Stronghold»	Внешний сервис для записи реквизитов подключения в KV v2	—
«Префикс пути Vault»	Префикс пути KV v2 для сохранения реквизитов (с `/data/`)	`services/data/`

Deckhouse managed Valkey

Набор данных для развёртывания managed Valkey в Deckhouse Kubernetes Platform.

Набор данных подготавливает ресурс, действия и процессы для управления жизненным циклом managed Valkey: создание инстанса в кластере через внешний сервис Kubernetes и сохранение реквизитов подключения в Vault или Stronghold.

Настройка:

Перед применением настройте внешние сервисы Kubernetes и Vault (или Stronghold).

> Подробное описание компонентов набора данных доступно по кнопке «Описание» на карточке в разделе «Наборы данных».

Параметры настройки:

Название	Описание	Значение по умолчанию
«Внешний сервис Kubernetes»	Kubernetes кластер, в котором будет разворачиваться managed Valkey	—
«Внешний сервис Vault или Stronghold»	Внешний сервис для записи реквизитов подключения в KV v2	—
«Префикс пути Vault»	Префикс пути KV v2 для сохранения реквизитов (с `/data/`)	`services/data/`

Deckhouse managed Kafka

Набор данных для развёртывания managed Kafka в Deckhouse Kubernetes Platform.

Набор данных подготавливает ресурс, действия и процессы для управления жизненным циклом managed Kafka: создание инстанса в кластере через внешний сервис Kubernetes и сохранение реквизитов подключения в Vault или Stronghold.

Настройка:

Перед применением настройте внешние сервисы Kubernetes и Vault (или Stronghold).

> Подробное описание компонентов набора данных доступно по кнопке «Описание» на карточке в разделе «Наборы данных».

Параметры настройки:

Название	Описание	Значение по умолчанию
«Внешний сервис Kubernetes»	Kubernetes кластер, в котором будет разворачиваться managed Kafka	—
«Внешний сервис Vault или Stronghold»	Внешний сервис для записи реквизитов подключения в KV v2	—
«Префикс пути Vault»	Префикс пути KV v2 для сохранения реквизитов (с `/data/`)	`services/data/`

Сценарии

Сценарии — механизм, позволяющий объединять действия в цепочку, управляя ходом выполнения.

Сценарии, как и действия, привязываются к конкретным ресурсам и могут быть запущены только для сущностей того ресурса, к которому они привязаны.

Тип запуска действий

Действия в одном сценарии могут запускаться параллельно либо последовательно.

При выборе параллельного запуска все действия запустятся одновременно. При выборе последовательного запуска действия будут запускаться в порядке их расположения в списке действий сценария. При этом, если одно из действий выполнилось неуспешно, все последующие будут созданы, но не будут запущены.

> При выборе последовательного запуска и в случае, если какое-либо из вышестоящих действий требует подтверждения, следующие за ним запустятся сразу после того, как данное действие перейдёт в статус «Unapproved».

Каждое действие сценария может быть обязательным либо опциональным:

- если действие является обязательным для сценария, то пользователь, запускающий сценарий, не может выключить выполнение данного действия;
- если действие является опциональным, то при запуске сценария пользователь может отключить его, и действие не запустится.

Запуск сценария

При запуске сценария пользователю будет предложена форма, в которой он сможет заполнить параметры запуска каждого действия, входящего в сценарий. При этом, если действие является опциональным, оно может быть выключено.

Параметры сценария

Конфигурация и использование параметров сценария описаны в разделе «Шаблонизация».

Руководство пользователя

Обзор

Данный документ представляет собой руководство пользователя Deckhouse Development Platform (DDP) и является частью эксплуатационной документации продукта.

Интерфейс

Интерфейс Deckhouse Development Platform (DDP) состоит из следующих разделов:

- «Главная» — стартовая страница платформы. Здесь можно размещать один или несколько дашбордов с виджетами.
- «Каталог» — сервисный каталог для просмотра ресурсов, сущностей и связей, а также для запуска действий и сценариев.
- «Самообслуживание» — раздел для настройки источников данных, действий, вебхуков, автоматизаций, сценариев, дашбордов и виджетов. Доступ к разделу может быть ограничен ролевой моделью.
- «AI» — раздел для настройки MCP-серверов, пользовательских инструментов и коллекций инструментов. Подробнее — в разделе «Управление MCP». Доступ к разделу может быть ограничен ролевой моделью.
- «Администрирование» — раздел для управления командами, пользователями, политиками доступа и учётными данными. Доступ к разделу может быть ограничен ролевой моделью.

Глобальный поиск

В верхней части боковой панели раздела «Каталог» доступно поле «Поиск». Оно позволяет быстро находить сущности по всей платформе.

Ограничения

- Максимальная длина запроса: 255 символов.
- Поиск работает только по сущностям. Ресурсы, команды и другие объекты платформы в результаты не попадают.

Каталог

Каталог — это раздел платформы, который содержит список всех ресурсов, их параметров и связей, а также предоставляет пользователям возможность запускать сценарии для сущностей ресурсов.

Наполнение каталога возможно в ручном режиме, либо автоматически с использованием источников данных или вебхуков.

Ресурсы

Ресурс — это шаблон или тип сущности в каталоге платформы. Ресурс определяет структуру данных, параметры и свойства, которые будут наследовать все сущности данного типа. Например, ресурс «Репозитории GitLab» описывает структуру данных для репозиториев GitLab, а каждая конкретная сущность этого ресурса представляет отдельный репозиторий.

Основной принцип, заложенный в платформу, заключается в том, что модель данных настраивается администратором платформы. Администратор может самостоятельно создавать те ресурсы, которые ему необходимы.

Именование ресурсов

При именовании ресурсов следуйте правилам:

- «Название ресурса» может быть любым и не требует специальных префиксов или разделителей.
- «Идентификатор ресурса» должен быть уникальным в пределах всей платформы.

Группировка ресурсов

Ресурсы в каталоге могут быть организованы в иерархическую структуру. Это позволяет логически группировать связанные ресурсы и улучшает навигацию в каталоге.

Для создания группы ресурсов необходимо:

1. «Создать ресурсы» — создать все необходимые ресурсы, которые будут входить в группу. Например, ресурс «GitLab» может быть родительским для ресурсов «Репозитории», «Группы» и т. д.
2. «Связать дочерние ресурсы с родительскими» — в нижней части сайдбара каталога включите переключатель «Изменить порядок ресурсов», затем перетащите дочерний ресурс на родительский. Дочерние ресурсы будут автоматически привязаны к родительскому и отобразятся в интерфейсе как вложенные элементы.
3. «Настроить отображение» — в сайдбаре каталога родительские ресурсы отображаются с иконкой раскрытия/сворачивания, позволяющей показывать или скрывать дочерние элементы.

> Для изменения порядка и группировки ресурсов (перетаскивания в сайдбаре каталога) требуется глобальное разрешение `update:resources-order`. Включите переключатель «Изменить порядок ресурсов» в нижней части сайдбара, чтобы активировать режим перетаскивания. Подробнее о правах доступа — в разделе «Ролевая модель».

> Для просмотра дочернего ресурса у пользователя должны быть права на просмотр (`read:resources`) всей иерархии родительских ресурсов. Подробнее о правах доступа — в разделе «Ролевая модель».

Отображение полного пути

В интерфейсе платформы (селекторы ресурсов, теги, графы связей) отображается полный путь ресурса с цепочкой родительских ресурсов, разделённых символом `/`. Например, если ресурс «Репозитории» является дочерним для ресурса «GitLab», то в интерфейсе будет отображаться «GitLab / Репозитории».

Страница ресурса

По умолчанию после создания ресурса на его странице будет отображаться таблица с сущностями ресурса. При клике на сущность в таблице можно перейти на её страницу.

Дашборды

В форме редактирования ресурса задаются три независимых варианта использования дашбордов:

1. «Использовать дашборд на странице ресурса» — выбранный дашборд выводится на странице ресурса вместо таблицы или карточек сущностей.
2. «Использовать дашборд на странице сущности» — выбранный дашборд выводится на странице сущности вместо встроенного содержимого вкладки «Обзор».
3. «Дополнительные вкладки с дашбордами» — дополнительные дашборды в виде вкладок на странице каждой сущности ресурса. Порядок отображения дашбордов зависит от порядка в конфигурации ресурса, доступно изменение порядка перетаскиванием. Один и тот же дашборд нельзя добавить несколько раз для одного и того же ресурса.

Режим просмотра сущностей

Когда на странице ресурса отображается список сущностей, над списком доступен переключатель «Таблица» и «Карточки».

1. «Таблица» — сущности в виде таблицы с сортировкой, пагинацией и поиском; при необходимости доступно массовое удаление.
2. «Карточки» — сущности в виде карточек с основными параметрами; список подгружается при прокрутке страницы.
3. Выбранный режим сохраняется в браузере отдельно для каждого ресурса.

Связи между ресурсами

Каждый ресурс может быть связан с другим ресурсом или сам с собой двусторонней связью. Связи настраиваются администратором платформы. После настройки связей для ресурса каждая сущность данного ресурса может быть связана с одной либо несколькими сущностями другого ресурса.

Допускается создание как вертикальных связей (сущность из одного ресурса связывается с сущностью из другого ресурса), так и горизонтальных (связываются сущности в рамках одного ресурса).

У каждой связи можно указать параметры родительского и дочернего ресурса, через которые будут связаны сущности. Данные параметры будут отображаться на графе связей при нажатии на соединительную линию между карточками ресурсов или сущностей.

Сущности

Сущность — это конкретная единица каждого ресурса. Например, если ресурс называется «Группы» и является дочерним для ресурса «GitLab», то каждая группа в GitLab, зарегистрированная в платформе, является сущностью этого ресурса.

Сущности наследуют параметры ресурсов, при этом возможно задание параметров отдельно для каждой сущности.

Именование сущностей

При именовании сущностей следуйте правилам:

- «Название сущности» может быть любым и не требует специальных префиксов или разделителей.
- «Идентификатор сущности» должен быть уникальным в пределах ресурса.

Страница сущности

Страница сущности состоит из встроенных панелей и вкладок с дашбордами. Встроенные панели:

- «Обзор» — содержит блоки с указанием владельца сущности, её описанием, параметрами и связями.
- «Запуски действий» — содержит список действий, которые запускались для данной сущности.
- «Запуски процессов» — содержит список процессов, которые запускались для данной сущности.
- «События» — содержит список событий, сгенерированных для данной сущности.

Вкладки с дашбордами из поля «Дополнительные вкладки с дашбордами» в настройках ресурса отображаются на странице сущности рядом со встроенными панелями; порядок вкладок соответствует порядку в настройках ресурса.

Связи между сущностями

Связи для каждой сущности отображаются на странице сущности в виде таблицы и в виде графа. Каждая сущность может быть связана с одной или несколькими сущностями того же или другого ресурса, в зависимости от настроенных связей между ресурсами.

Связи между сущностями могут создаваться в ручном режиме либо автоматически при помощи источников данных или вебхуков.

События

При создании, удалении либо изменении спецификации любой сущности генерируется событие соответствующего типа: `ENTITY_CREATED`, `ENTITY_DELETED`, `ENTITY_UPDATED`. Эти события нужны для следующих целей:

- «Аудит» — фиксация изменений, происходящих с сущностью в течение её жизненного цикла.
- «Настройка автоматизированных реакций» — настройка реакций на изменения спецификации сущности.

Время хранения событий ограничено и может быть настроено через файл конфигурации платформы.

Избранное

Пользователь может добавить любую сущность в избранное. Для этого на странице сущности нажмите на круглую кнопку со всплывающей подсказкой «Добавить в избранное» рядом с названием сущности. Посмотреть добавленные в избранное сущности можно в разделе каталога «Избранное».

> Каждый пользователь видит в разделе «Избранное» только его добавленные сущности.

Выгрузка сущностей в CSV

Платформа предоставляет возможность выгрузки сущностей в CSV-файл для последующего анализа и обработки данных.

Выгрузка сущностей одного ресурса

Для выгрузки сущностей конкретного ресурса:

1. Откройте страницу ресурса в каталоге.
2. Рядом с названием ресурса нажмите на круглую кнопку со всплывающей подсказкой «Выгрузить сущности ресурса в CSV».
3. В выгруженный файл будут включены все сущности ресурса с учётом применённых фильтров, сортировки и пагинации.

Выгрузка сущностей нескольких ресурсов

Для выгрузки сущностей из нескольких ресурсов одновременно:

1. В нижней части сайдбара каталога нажмите кнопку «Выгрузить сущности».
2. В открывшемся диалоге выберите один или несколько ресурсов.
3. Нажмите кнопку «Скачать .csv».
4. Все сущности выбранных ресурсов будут объединены в один CSV-файл.

> Для выгрузки сущностей требуется глобальное разрешение `read:entities`. Подробнее о правах доступа — в разделе «Ролевая модель».

AI-ассистент

> Экспериментальный функционал

AI-ассистент — это интеллектуальный помощник, встроенный в Deckhouse Development Platform (DDP). Он отвечает на вопросы о платформе, анализирует данные каталога и выполняет различные задачи с использованием инструментов MCP (Model Context Protocol).

AI-ассистент использует настраиваемые AI-провайдеры для обработки запросов и может работать с различными языковыми моделями, включая OpenAI GPT, Ollama и любые модели, доступные через совместимый REST API.

Подключение AI-провайдера

> Пользователь самостоятельно настраивает AI-провайдеры в своём профиле.

Для подключения нового AI-провайдера:

1. Откройте «Профиль» → вкладка «AI-провайдеры».
2. Нажмите «Добавить».
3. Заполните «Название», «Модель», «URL», «Метод» и «Заголовки»; при необходимости укажите «Поле ответа» и «Шаблон тела запроса».
4. При использовании токенов в заголовках предпочтительно сохранить учётные данные и подставлять их через шаблонизацию.
5. Нажмите «Сохранить».

Примеры для распространённых API приведены в разделе «Примеры конфигурации».

Учётные данные для провайдеров

Для безопасного хранения токенов и ключей используется система учётных данных: они шифруются в базе и подставляются в заголовки запросов через шаблонизацию.

Добавление учётных данных

Для добавления новых учётных данных:

1. В форме создания или редактирования провайдера нажмите «Управление учётными данными».
2. В диалоге нажмите «Добавить учётные данные».
3. Введите пару «Ключ» / «Значение» (например, `api_key` и секрет).
4. Нажмите «Сохранить».

Изменение учётных данных

Обратите внимание, что:

- Ключ существующих учётных данных нельзя изменить: удалите запись и создайте новую.
- Значение можно обновить, введя новое.
- После сохранения значения не показываются в интерфейсе.

Шаблонизация в заголовках

Вместо токена в открытом виде используйте подстановку:

```
Authorization: Bearer {{ .credentials.api_key }}
```

где `Authorization` — заголовок, `credentials` — место хранения учётных данных в зашифрованном виде, а `api_key` — ключ, заданный в учётных данных.

Поле ответа

В «Поле ответа» задается путь к тексту ответа модели (например, `choices.0.message.content`) в теле ответа API (тело ответа возвращается в JSON-формате) . Если значение поля не задано, платформа попытается найти текст ответа автоматически.

> Чтобы подобрать путь, выполните тестовый запрос к API, откройте ответ и найдите в теле ответа (возвращается в формате JSON) название поля с текстом ответа модели.

Шаблон тела запроса

В поле «Шаблон тела запроса» вводится структура в формате JSON, которая отправляется в API.

Используются следующие переменные:

- `{{.prompt}}` — текст запроса пользователя.
- `{{.model}}` — модель из настроек провайдера.

Примеры структуры

OpenAI-совместимый чат:

```
{
  "model": "{{.model}}",
  "messages": [
    {
      "role": "user",
      "content": "{{.prompt}}"
    }
  ],
  "temperature": 0.7
}
```

Другой формат сообщений:

```
{
  "messages": [
    {
      "content": "{{.prompt}}",
      "role": "user"
    }
  ],
  "model": "{{.model}}",
  "stream": false
}
```

Минимальный вариант:

```
{
  "query": "{{.prompt}}",
  "model_name": "{{.model}}"
}
```

> Шаблон должен быть валидным JSON. В него можно добавлять поля вроде `temperature`, `max_tokens` и т. д.

Примеры конфигурации

API OpenAI (Chat Completions)

1. «Название» — по желанию (например, `ChatGPT`).
2. «Модель» — например, `gpt-4` или `gpt-3.5-turbo`.
3. «URL» — `https://api.openai.com/v1/chat/completions`.
4. «Метод» — `POST`.
5. «Заголовки» — например, `Authorization: Bearer {{ .credentials.openai_api_key }}` (ключ `openai_api_key` заведите в «Управлении учётными данными»).
6. «Поле ответа» — `choices.0.message.content`.
7. «Шаблон тела» — как в первом примере в разделе «Шаблон тела запроса».

Ollama (`/api/generate`)

1. «URL» — `http://localhost:11434/api/generate` (или адрес вашей Ollama).
2. «Метод» — `POST`.
3. «Поле ответа» — `response`.
4. «Шаблон тела»:

```
{
  "model": "{{.model}}",
  "prompt": "{{.prompt}}",
  "stream": false
}
```

Произвольный REST API

- «URL» — эндпоинт вашего сервиса (например, `https://api.example.com/v1/chat`).
- «Метод» — обычно `POST`.
- «Заголовки» — например, `Authorization: Bearer {{ .credentials.api_key }}`, `Content-Type: application/json`.
- «Поле ответа» — путь к полю с текстом в вашем JSON.
- «Шаблон тела» — на базе первого примера в разделе «Шаблон тела запроса», при необходимости добавьте поля (`max_tokens` и т. д.).

Работа с AI-ассистентом

Панель ассистента открывается справа; кнопка запуска — в правом нижнем углу экрана.

Выбор провайдера

Вверху панели чата — список провайдеров. Если доступен один провайдер, он выбирается автоматически.

Чаты

Диалог ведётся в чатах. Слева — список чатов и «Новый чат».

Чаты имеют следующие настройки:

1. До 20 чатов на пользователя. При достижении лимита удалите старые чаты через меню «⋯».
2. До первого сообщения у чата служебное название. После первого вопроса название подставляется по тексту (можно «Переименовать»).
3. Удаление чата вместе с историей сообщений в нем необратимо.

Передача контекста

Переключатель «Передавать контекст» задаётся для каждого чата отдельно:

1. «Включено» — в запрос уходит контекст переписки в этой теме.
2. «Выключено» — только текущее сообщение. Расход токенов ниже, без «памяти» в чате.

При включённом контексте и длинной истории старые реплики не отправляются целиком каждый раз: недавние сообщения идут полностью, более ранняя часть сжимается в резюме отдельным запросом к тому же провайдеру.

> ⚠ Включение передачи контекста увеличивает расход токенов при взаимодействии с моделью.

Инструменты MCP

AI-ассистент использует встроенные инструменты платформы и инструменты из MCP-коллекций, к которым у пользователя есть доступ.

В панели чата раскройте **Доступные инструменты**, чтобы посмотреть название, тип (**internal**, **external**, **custom**), аргументы и пример. Клик по примеру вставляет текст в поле ввода. В одном запросе модель может вызвать несколько инструментов.

- Встроенные инструменты (**internal**) и их параметры описаны в документации MCP-сервера.
- Подключение MCP-серверов, пользовательских MCP-инструментов и настройка коллекций — в разделе «Управление MCP».

MCP-сервер

> ⚠ Экспериментальный функционал

MCP-сервер — компонент Deckhouse Development Platform (DDP), который реализует протокол MCP (Model Context Protocol) и обеспечивает взаимодействие внешних AI-клиентов (таких как LM Studio, Claude Desktop и других) с платформой. Сервер работает по JSON-RPC 2.0 и

предоставляет набор инструментов для работы с ресурсами платформы и проксирования запросов во внешние инфраструктурные сервисы.

MCP — это открытый протокол для взаимодействия AI-моделей с внешними системами. Подробнее о протоколе можно узнать на официальном сайте MCP.

> Помимо встроенных инструментов платформы, MCP-сервер может вызывать инструменты из MCP-коллекций, к которым у пользователя есть доступ. Настройка MCP-серверов, пользовательских MCP-инструментов и коллекций выполняется в разделе «Управление MCP».

Доступные инструменты

Ниже перечислены **встроенные** инструменты платформы. Инструменты типов **external** и **custom** доступны после синхронизации каталога и добавления в MCP-коллекцию.

get_resources

Получить список ресурсов.

Параметры: нет.

Возвращает: список ресурсов.

Пример:

```
Получи список ресурсов
```

---

get_external_services

Получить список внешних сервисов (GitLab, SonarQube и т. д.).

Параметры: нет.

Возвращает: список внешних сервисов.

Пример:

```
Получи список внешних сервисов
```

---

get_resource_entities

Получить все сущности выбранного ресурса.

Параметры:

Название	Тип	Обязательность	Описание
`resource_uuid`	строка	Да	UUID ресурса

Возвращает: список сущностей ресурса.

Пример:

```
Получи все сервисы и покажи название и дату создания
```

---

get_entity

Получить одну сущность по UUID.

Параметры:

Название	Тип	Обязательность	Описание
`entity_uuid`	строка	Да	UUID сущности

Возвращает: данные одной сущности.

Пример:

```
Получи сущность с UUID 3fa85f64-5717-4562-b3fc-2c963f66afa6
```

---

get_entity_relations

Получить связи сущности.

Параметры:

Название	Тип	Обязательность	Описание
`resource_uuid`	строка	Да	UUID ресурса
`entity_slug`	строка	Да	Идентификатор сущности

Возвращает: список связей сущности.

Пример:

```
Получи связи сущности «api-gateway» ресурса «Сервисы»
```

---

get_external_data

Выполнить HTTP-запрос к внешнему сервису с учётными данными пользователя.

Параметры:

Название	Тип	Обязательность	Описание
`external_service_uuid`	строка	Да	UUID внешнего сервиса
`query`	строка	Да	Описание запроса (например: «получить пайплайны проекта 123»)
`api_path`	строка	Да	Путь к API с параметрами запроса (например, пагинация)
`method`	строка	Нет	HTTP-метод. По умолчанию: `GET`
`body`	строка	Нет	Тело запроса для POST/PUT/PATCH (JSON-строка)

Учётные данные и заголовки берутся из настроек внешнего сервиса в платформе.

Возвращает: результат HTTP-запроса к внешнему сервису.

Пример:

```
Получи список проектов из внешнего сервиса GitLab
```

---

get_actions

Получить список действий.

Параметры: нет.

Возвращает: список действий.

Пример:

```
Получи список действий
```

---

get_datasources

Получить список источников данных.

Параметры: нет.

Возвращает: список источников данных.

Пример:

```
Получи список источников данных
```

---

get_processes

Получить список процессов.

Параметры: нет.

Возвращает: список процессов.

Пример:

```
Получи список процессов
```

Подключение MCP-сервера

LM Studio

1. Получение параметров:
   - Войдите в Deckhouse Development Platform.
   - Получите ваш API-токен в разделе «Профиль».
   - Запишите URL вашей платформы (например, `https://ddp.example.com`).
2. Настройка в LM Studio:
   - Откройте LM Studio.
   - Перейдите в настройки (Settings).
   - Найдите раздел «MCP Servers» или «Model Context Protocol».
   - Нажмите «Add Server» или «Добавить сервер».
3. Конфигурация сервера (заполните следующие поля):
   - «Server Name»: `DDP MCP Server` (или любое удобное название).
   - «Server URL»: `https://<DOMAIN>/api/v2/mcp`.
   - «Transport»: `HTTP` или `JSON-RPC`.
   - «Authentication»:
     - «Type»: `Bearer Token` или аналог (заголовок `Authorization`).
     - «Header»: `Authorization: Bearer <ваш_api_токен>`.
     - «Token»: вставьте ваш API-токен платформы (из раздела «Профиль»).
4. Проверка подключения:
   - Сохраните конфигурацию.
   - LM Studio попытается подключиться к серверу.

    - Если подключение успешно, вы увидите список доступных инструментов:
        - `get_resources` — получить список ресурсов.
        - `get_external_services` — получить список внешних сервисов (GitLab, SonarQube и т. д.).
        - `get_resource_entities` — получить все сущности выбранного ресурса.
        - `get_entity` — получить одну сущность по UUID.
        - `get_entity_relations` — получить связи сущности по ресурсу и идентификатору.
        - `get_external_data` — выполнить HTTP-запрос к внешнему сервису с учётными данными пользователя.
        - `get_actions` — получить список действий.
        - `get_datasources` — получить список источников данных.
        - `get_processes` — получить список процессов.

После успешного подключения вы можете использовать инструменты в диалогах с моделями.

Все вызовы будут выполняться с вашими правами доступа.

Подключение в других MCP-клиентах

MCP-сервер Deckhouse Development Platform совместим с любыми клиентами, поддерживающими протокол MCP через JSON-RPC 2.0.

Общие шаги подключения:

1. «URL эндпоинта»: `https://your-platform.com/api/v2/mcp`.
2. «Протокол»: JSON-RPC 2.0.
3. «Аутентификация»:
    - Заголовок: `Authorization: Bearer YOUR_API_TOKEN`.
    - Где `YOUR_API_TOKEN` — ваш API-токен платформы (раздел «Профиль»).
4. «Метод»: POST.

Пример запроса к MCP-серверу

HTTP-заголовки:

```
Authorization: Bearer your-api-token-here
Content-Type: application/json
```

Тело запроса:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/call",
  "params": {
    "name": "get_resource_entities",
    "arguments": {
      "resource_uuid": "uuid-нужного-ресурса"
    }
  }
}
```

Пример ответа от MCP-сервера

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "[{\"uuid\":\"...\",\"name\":\"Service 1\",\"properties\":{...}},...]"
      }
    ]
  }
}
```

Безопасность

Аутентификация:

- Все запросы к MCP-серверу должны быть аутентифицированы с помощью API-токена из раздела «Профиль». Токен передаётся в заголовке `Authorization: Bearer <api_token>`.
- Права доступа соответствуют правам вашего пользователя в платформе.

Права доступа:

- Инструменты работают с теми же правами, что и пользователь.
- Если у вас нет доступа к ресурсу, инструмент вернёт ошибку доступа.
- Данные фильтруются в соответствии с вашими правами RBAC.

Устранение неполадок

Не удаётся подключиться к серверу

Если не получается подключиться к серверу:

- Проверьте правильность URL (должен заканчиваться на `/api/v2/mcp`).
- Убедитесь, что API-токен действителен.
- Проверьте, что платформа доступна с вашего компьютера.
- Проверьте настройки файрвола и прокси.

Ошибка аутентификации

Если не проходит аутентификация:

- Проверьте правильность формата токена.
- Убедитесь, что токен не истёк.
- Убедитесь, что используется заголовок `Authorization: Bearer <api_token>`.

Инструмент возвращает ошибку доступа

Если при обращении к инструменту возвращается ошибка доступа:

- Убедитесь, что у вашего пользователя есть права на доступ к запрашиваемому ресурсу.
- Проверьте правильность названия или идентификатора ресурса.
- Обратитесь к администратору платформы для проверки прав доступа.

Данные не возвращаются

В случае отсутствия ответа:

- Проверьте правильность параметров запроса.
- Убедитесь, что ресурс существует и содержит сущности.
- Проверьте логи платформы для получения детальной информации об ошибке.

Команды и пользователи

В платформе существуют две встроенные сущности — команды и пользователи. Основным источником данных для авторизации является Dex, установленный в Deckhouse Kubernetes Platform.

Пользователи обычно появляются при первой авторизации через Dex; администратор также может создать учётную запись заранее. Создание команд вручную в платформе не поддерживается.

Принадлежность пользователя к командам определяется на основе информации из Dex и обновляется при каждой аутентификации пользователя.

Каждый пользователь может состоять в произвольном количестве команд. Принадлежность пользователя к командам определяется на основе групп в Dex.

Синхронизация

При первой авторизации пользователя в платформу для него создаётся внутренняя учётная запись. Команды пользователя синхронизируются при первой и каждой последующей авторизации пользователя.

Фильтрация групп при синхронизации

По умолчанию при синхронизации в DDP попадают все группы пользователя. Чтобы ограничить набор синхронизируемых команд, можно настроить правила фильтрации.

Настройка доступна в разделе «Администрирование» → «Команды» по кнопке «Правила синхронизации». Для настройки правил требуется разрешение `edit:team-filter-rules`.

Правила применяются по порядку цепочкой: каждое правило фильтрует результат предыдущего. Пустой список правил означает отсутствие фильтрации — синхронизируются все группы.

Режимы правил:

- «Включить» — оставить только команды, совпадающие с паттерном. Остальные исключаются.
- «Исключить» — оставить только команды, не совпадающие с паттерном. Совпадающие исключаются.

«Паттерн» задаётся в виде регулярного выражения (regex). Например:

- `^dev-.*` — группы, начинающиеся с `dev-`;
- `^team-(frontend|backend)$` — группы `team-frontend` и `team-backend`;
- `.*-prod$` — группы, заканчивающиеся на `-prod`.

Порядок правил можно менять перетаскиванием. При пустом паттерне правило не применяется (пропускается).

> Если пользователь входил в состав команды, а затем команда была отфильтрована правилами, то при следующей авторизации пользователь автоматически перестанет быть участником этой команды. Сама команда при этом останется в системе, её необходимо будет удалить вручную в разделе «Администрирование» → «Команды».

Параметры пользователей

Последняя активность

Время последней активности пользователя в платформе. Отображается в таблице пользователей в разделе «Администрирование» → «Пользователи». Обновляется при авторизации, либо при

автоматической ротации JWT-токена пользователя (интервал ротации зависит от конфигурации Dex и по умолчанию равен 10 минутам).

> Время последней активности не обновляется при использовании API-токена, выписанного пользователем, либо при каких-либо действиях пользователя в платформе в интервале между автоматическими ротациями JWT токена.

Сервисные пользователи

Сервисный пользователь — учётная запись платформы для автоматизации и интеграций (скрипты, технические сценарии).

От обычного пользователя сервисный отличается тем, что не может авторизоваться в веб-интерфейсе через Dex. В таблице «Администрирование» → «Пользователи» у таких записей в колонке «Последняя активность» обычно отображается «никогда».

При этом для сервисного пользователя доступны:

- настройка учётных данных для внешних сервисов;
- создание API-токенов для вызова API DDP;
- роли и команды — по тем же правилам RBAC, что и для остальных пользователей.

Учётную запись можно пометить как сервисную при создании в разделе «Пользователи» или переключателем «Сервисный пользователь» в таблице. Для этого администратору нужно разрешение `edit:users`. Супер-администратор не может быть сервисным пользователем.

Поскольку сервисный пользователь не может войти через браузер, учётные данные и API-токены для него настраивают через имперсонацию: администратор с разрешением `impersonate:users` входит в сессию «как этот пользователь» и заполняет профиль от его имени.

На странице «Администрирование» → «Лицензия» сервисные пользователи учитываются отдельно и не входят в показатель «Лицензируемые».

Параметры

Для каждого ресурса, действия или сценария администратор платформы может добавить неограниченное количество параметров одного из следующих типов:

- «Array» — список значений;
- «Boolean» — булево значение;
- «Date» — дата;
- JSON — текст в JSON-формате;
- «Entities» — сущность, один из параметров которой можно выбрать в качестве значения;
- «Enum» — перечисление значений с ключом и отображаемым значением;
- «List» — список значений с возможностью выбора одного из них;

- Markdown — текст в Markdown-формате;
- «Number» — число;
- «Object» — произвольный объект в JSON-формате;
- «Percentage» — процент;
- «String» — строка;
- YAML — текст в YAML-формате;
- «Teams» — команды;
- `URL` — строка в формате URL;
- «Users» — пользователи.

Ресурсы

После добавления параметра для ресурса он будет отображаться у всех сущностей данного ресурса в их карточках и в таблице каталога. Для каждого параметра можно задать значение по умолчанию. Значение каждого параметра можно изменить для каждой сущности по отдельности.

Заполненность параметров проверяется для каждой сущности, результат проверки выводится в карточке сущности в заголовке.

Синхронизация параметров ресурса

По различным причинам у сущностей могут появиться параметры, которых нет у ресурса. Для того чтобы удалить подобные параметры, доступна кнопка «Синхронизировать параметры» в меню ресурса.

При синхронизации параметров:

- для каждого параметра ресурса будет получен его идентификатор;
- для каждой сущности ресурса будет получен список её параметров;
- те параметры сущности, название которых не соответствует ни одному из идентификаторов параметров ресурса, будут удалены из спецификации сущности.

Действия и сценарии

Для каждого действия и сценария задаётся пользовательская форма, состоящая из параметров, которые пользователь должен заполнить при запуске.

Ограничения

Идентификатор каждого параметра должен:

- содержать символы `a-z`, `A-Z`, цифры либо подчёркивания;
- не начинаться с цифры.

Конфигурация

- «Редактируемый параметр» — для каждого параметра можно настроить разрешение либо запрет редактирования пользователем. В случае запрета редактирования пользователь при запуске действий или сценариев не сможет изменить значение параметра, то есть всегда будет использоваться значение по умолчанию.
- «Обязательный параметр» — каждый параметр может быть обязательным либо опциональным. Значение обязательного параметра не может быть пустым при запуске действий или сценариев. При этом значение опционального параметра может оставаться пустым без влияния на работоспособность действия.
- «Скрытый параметр» — скрытый параметр не отображается в таблицах и карточках сущностей, а также при запуске действий и сценариев.

Типы параметров

Date

Параметр типа «Date» может принимать значение даты с определённым пользователем форматом. При этом в спецификации значение всегда хранится в формате ISO 8601 (`YYYY-MM-DDTHH:mm:ss.sssZ`).

Конфигурация параметра

- «Формат» — настройка отображения даты. Если формат не задан явно, то используется формат по умолчанию: `YYYY-MM-DDTHH:mm:ss.sssZ` . Описание конфигурации формата доступно в документации Day.js. Настройка формата влияет на:
  - отображение даты в таблицах и карточках сущностей;
  - значение параметра при запуске действий или сценариев.
- «Текущая дата по умолчанию» — подстановка текущей даты в качестве значения по умолчанию при редактировании сущностей, а также при запуске действий или сценариев.
- «Значение по умолчанию» — заранее заданное значение по умолчанию. Не применяется, если активирован переключатель «Текущая дата по умолчанию».

> При запуске действий или сценариев и использовании текущей даты по умолчанию, параметр не должен быть скрытым в пользовательской форме. В противном случае текущая дата подставлена не будет.

При запуске действий или сценариев в параметр типа «дата» нельзя подставить значения с использованием Go template.

List

Параметр типа «List» позволяет выбрать одно значение из предопределённого списка.

Enum

Параметр типа «Enum» позволяет выбрать один элемент из предопределённого списка. В отличие от типа «List», параметр «Enum» использует пару «ключ — значение», где ключ хранится в спецификации, а значение отображается пользователю. Это позволяет изменять отображаемый текст без изменения ключей.

Управление MCP

Раздел **AI** → **MCP** предназначен для настройки инструментов, которые AI-ассистент и встроенный MCP-сервер могут вызывать от имени пользователя.

Доступ к разделу **AI** контролируется глобальным разрешением `view:ai-page`. Настройка MCP-серверов, инструментов и коллекций — отдельными глобальными разрешениями `read:` / `edit:` (см. ролевую модель).

Обзор

Подраздел	Назначение
MCP-серверы	Подключение upstream MCP-серверов и синхронизация их каталога инструментов
MCP-коллекции	Группы инструментов и разграничение доступа пользователей
MCP-инструменты	Пользовательские MCP-инструменты без upstream MCP-сервера
Каталог	Просмотр всех доступных вам инструментов

В AI-ассистенте инструменты отображаются с типом:

- **internal** — встроенные инструменты платформы (описаны в документации MCP-сервера);
- **external** — инструменты, полученные из подключённого upstream MCP-сервера;
- **custom** — пользовательские MCP-инструменты.

Вызов **external** и **custom** инструментов доступен только если они входят во **включённую** MCP-коллекцию, к которой у пользователя есть доступ.

MCP-серверы

Подключите upstream MCP-сервер, чтобы импортировать его инструменты в каталог платформы.

1. Перейдите в **AI** → **MCP** → **MCP-серверы**.
2. Нажмите **Подключить**.
3. На вкладке **Основная информация** укажите **Название**, **Идентификатор**, **Описание**, **Владелец** и **Команда**.

4. На вкладке **Конфигурация**:
    1. Включите переключатель **Включить**.
    2. Выберите **Транспорт** (`HTTP` или `SSE`).
    3. Укажите **URL** upstream MCP-сервера.
    4. При необходимости добавьте **HTTP-заголовки** и **учётные данные** для авторизации.
5. Нажмите **Сохранить**.

После сохранения откройте карточку сервера и нажмите **Синхронизировать**, чтобы загрузить каталог инструментов. На вкладке **Инструменты** включите нужные инструменты и при необходимости задайте **Теги**.

MCP-инструменты

Создайте пользовательский MCP-инструмент, который AI-ассистент вызывает напрямую, без upstream MCP-сервера.

1. Перейдите в **AI** → **MCP** → **MCP-инструменты**.
2. Нажмите **Добавить**.
3. На вкладке **Основная информация** укажите **Название**, **Описание**, **Владелец**, **Команду** и **Теги**.
4. На вкладке **Конфигурация**:
    1. Включите переключатель **Включить**.
    2. Задайте **Схему аргументов** (JSON Schema с корневым типом `object`).
    3. При необходимости укажите **Сопоставление path-параметров** — JSON-объект, который сопоставляет имена аргументов с сегментами `{placeholder}` в URL executor.
    4. При необходимости укажите **Сопоставление query-параметров** — JSON-объект, который сопоставляет имена аргументов с именами query-параметров URL.
5. На вкладке **Авторизация** укажите **URL** executor endpoint, **HTTP-метод**, **HTTP-заголовки** и **учётные данные**. В URL можно использовать сегменты `{placeholder}` для path-параметров и плейсхолдеры учётных данных, например `{{ .credentials.tenant_id }}`.
6. Нажмите **Сохранить**.

Пример. В схеме аргументов задан аргумент `space_id`, URL executor — `https://api.example.com/v1/spaces/{spaceId}/boards`, сопоставление path-параметров — `{"space_id": "spaceId"}`. При вызове инструмента с `space_id=42` запрос уйдёт на `https://api.example.com/v1/spaces/42/boards`.

Для **GET** и **DELETE** аргументы, не сопоставленные с path-параметрами, передаются только как query-параметры. Для **POST**, **PUT** и **PATCH** аргументы, не сопоставленные с path- или query-параметрами, отправляются в JSON-теле запроса.

MCP-коллекции

MCP-коллекция объединяет инструменты из каталога и определяет, какие из них доступны пользователям AI-ассистента и внешним MCP-клиентам.

1. Перейдите в **AI** → **MCP** → **MCP-коллекции**.
2. Нажмите **Создать**.
3. На вкладке **Основная информация** укажите **Название**, **Идентификатор**, **Описание**, **Владелец** и **Команду**.
4. На вкладке **Конфигурация**:
    1. Включите переключатель **Включить**.
    2. В поле **Инструменты** выберите инструменты из доступного каталога. Каждый пункт содержит идентификатор MCP-сервера в скобках.
5. Нажмите **Сохранить**.

Чтобы пользователь мог вызывать инструменты коллекции, откройте меню карточки коллекции и выберите **Настроить доступ**. Назначьте пользователям или командам роль с разрешением `use:mcp-collections`. Список разрешений — в ролевой модели.

Владелец коллекции и супер-администратор получают доступ к коллекции автоматически.

Каталог

Раздел **Каталог** показывает все инструменты, доступные текущему пользователю с учётом MCP-коллекций и прав доступа. Раздел предназначен для просмотра; редактирование выполняется на страницах **MCP-серверы**, **MCP-инструменты** и **MCP-коллекции**.

Связь с AI-ассистентом и MCP-сервером

- В AI-ассистенте в блоке **Доступные инструменты** отображаются встроенные инструменты и инструменты из доступных коллекций.
- MCP-сервер платформы возвращает и вызывает те же инструменты из коллекций, что доступны пользователю, с учётом его прав RBAC.

Учётные данные

Учётные данные — это механизм безопасного хранения и использования реквизитов доступа к внешним сервисам (токены, пароли, ключи API и т. д.) в Deckhouse Development Platform (DDP).

Как это работает

Всё взаимодействие с инфраструктурными сервисами в DDP происходит с использованием учётных данных конкретного пользователя. Учётные данные шифруются перед сохранением в базе данных и расшифровываются только при необходимости их использования в действиях, источниках данных и виджетах.

Подробнее о механизме работы, шифровании и настройке — в документации.

Заполнение учётных данных

Администратор платформы создаёт типы учётных данных в разделе «Администрирование» → «Учётные данные». Для каждого типа учётных данных вы можете указать свои персональные реквизиты в профиле в разделе «Учётные данные».

Например, если администратор создал тип учётных данных «Kubernetes token», то вы в своём профиле сможете добавить персональный токен для доступа в Kubernetes.

Особенности работы с учётными данными

- После сохранения учётных данных вы не сможете посмотреть их значение, только обновить его.
- В профиле отображается информация о том, заполнены ли те или иные учётные данные.
- Учётные данные используются автоматически в действиях, источниках данных и виджетах, где они указаны в конфигурации.

Подход к использованию учётных данных в действиях, источниках данных и виджетах описан в соответствующих разделах документации.

Шаблонизация

Deckhouse Development Platform поддерживает шаблонизацию на базе Go template: выражения в фигурных скобках подставляются в полях конфигурации, где это предусмотрено (действия, виджеты, источники данных и т.д.). Ниже — встроенные функции и контекстные переменные (`{{ .property.* }}`, `{{ .entity.* }}` и другие).

Помимо стандартных функций доступны:

- Встроенные функции платформы.
- Функции библиотеки sprig.

> Функции библиотеки sprig не поддерживаются в источниках данных.

Встроенные функции

extractPart

`extractPart` разделяет входную строку `s` по указанному разделителю `delimiter` и возвращает часть по указанному индексу `index`. Если индекс выходит за пределы массива, то возвращается ошибка.

Параметры:

- `s` — входная строка, которую нужно разделить.
- `delimiter` — разделитель, используемый для разделения входной строки.
- `index` — индекс части, которую нужно вернуть после разделения.

Возвращает:

- Строку, представляющую часть с указанным индексом после разделения входной строки.
- Если индекс выходит за пределы, возвращается ошибка.

Пример в шаблоне:

```
{{ extractPart "ddp/demo-service" "/" 1 }} // Вывод: "demo-service"
```

extractLastPart

`extractLastPart` разделяет входную строку `s` по указанному разделителю `delimiter` и возвращает последнюю часть. Если строка пустая или разделитель не найден, возвращается оригинальная строка.

Параметры:

- `s` — входная строка, которую нужно разделить.
- `delimiter` — разделитель, используемый для разделения входной строки.

Возвращает:

- Строку, представляющую последнюю часть после разделения входной строки.

Пример в шаблоне:

```
{{ extractLastPart "ddp/demo-service" "/" }} // Вывод: "demo-service"
```

toJSON

`toJSON` сериализует заданное значение в строку JSON. Принимает любой тип данных. При невозможности сериализации возвращается ошибка.

Параметры:

- `v` — значение, которое нужно сериализовать в JSON.

Возвращает:

- JSON-строку, представляющую входное значение.
- Ошибку, возникшую в ходе сериализации.

Пример в шаблоне:

```
{{ toJSON . }} // Вывод: JSON-представление объекта в контексте
```

toYAML

`toYAML` сериализует заданное значение в строку YAML. Принимает любой тип данных. При невозможности сериализации возвращается ошибка.

Параметры:

- `v` — значение, которое нужно сериализовать в YAML.

Возвращает:

- Строку с YAML-представлением входного значения.
- Ошибку, возникшую в ходе сериализации.

Пример в шаблоне:

```
{{ toYAML . }} // Вывод: YAML-представление объекта в контексте
```

fromYAML

`fromYAML` разбирает строку с YAML в `map` для использования в шаблоне (например, для объединения с `dict` или доступа по ключам).

Параметры:

- `s` — строка с YAML.

Возвращает:

- `map` с данными из строки.
- Пустой `map`, если строка пустая.
- Ошибку, если YAML некорректен.

Пример в шаблоне:

```
{{ index (fromYAML "first: a\nsecond: b\n") "first" }} // Вывод: "a"
```

replaceChar

`replaceChar` заменяет все вхождения указанного символа в строке на другой указанный символ.

Параметры:

- `s` — исходная строка, в которой нужно выполнить замену.
- `oldChar` — символ, который нужно заменить.
- `newChar` — символ, на который нужно заменить.

Возвращает:

- Изменённую строку, где все вхождения `oldChar` заменены на `newChar`.

Пример в шаблоне:

```
{{ replaceChar "hello/world" "/" "-" }} // Вывод: "hello-world"
```

toSlug

`toSlug` приводит произвольную строку к виду, допустимому для идентификатора:

- В результате остаются строчные латинские буквы и цифры, между фрагментами — дефисы.
- Ведущий дефис в результате отсутствует.
- Длина результата не больше 64 символов.
- Символы вне допустимого набора заменяются одним дефисом.
- Крайние дефисы удаляются.

Параметры:

- `s` — исходная строка.

Возвращает:

- Строку в форме идентификатора или пустую строку.

Пример в шаблоне:

```
{{ toSlug "My Service/Name!" }} // Вывод: "my-service-name"
```

filteredItems

`filteredItems` фильтрует массив элементов (maps) по запрашиваемому значению ключа. Возвращает массив элементов, у которых значение по заданному ключу соответствует целевому значению.

Параметры:

- `data` — массив данных, в котором выполняется фильтрация.
- `key` — ключ, по которому будет выполнена проверка значения.
- `value` — целевое значение для сравнения.

Возвращает:

- Массив элементов, у которых значение по заданному ключу соответствует целевому значению.
- Ошибку, если возникает проблема во время фильтрации.

Пример в шаблоне:

```
{{ filteredItems .items "name" "Alice" }} // Вывод: [{"name": "Alice", "age": 30}]
```

anyOf

`anyOf` проверяет, соответствует ли хотя бы один элемент массива условию сравнения.

Параметры:

- `operator` — оператор сравнения (eq, ne, lt, le, gt, ge).
- `value` — целевое значение для сравнения.
- `key` — ключ, по которому будет извлечено значение для сравнения.
- `data` — массив данных (maps).

Возвращает:

- `true`, если хотя бы одно значение из массива данных соответствует условию.

Пример в шаблоне:

```
{{ anyOf "eq" "value" "key" .data }} // Вывод: true, если хотя бы одна запись
удовлетворяет условию
```

allOf

`allOf` проверяет, соответствуют ли все элементы массива условию сравнения.

Параметры:

- `operator` — оператор сравнения (eq, ne, lt, le, gt, ge).
- `value` — целевое значение для сравнения.
- `key` — ключ, по которому будет извлечено значение для сравнения.
- `data` — массив данных (maps).

Возвращает:

- `true`, только если все значения соответствуют условию.

Пример в шаблоне:

```
{{ allOf "gt" 10 "age" .users }} // Вывод: true, если все пользователи старше 10 лет
```

getFieldValue

`getFieldValue` получает значение поля по ключу из структуры, представленной в виде map.

Параметры:

- `items` — структура данных в виде `map`.
- `key` — ключ, по которому нужно получить значение.

Возвращает:

- Значение, полученное по ключу.
- Ошибку, если структура отсутствует или ключ не найден.

Пример в шаблоне:

```
{{ getFieldValue .item "name" }} // Вывод: Значение поля "name" из структуры .item
```

findValueInDictArray

`findValueInDictArray` ищет в массиве словарей элемент, где значение по заданному ключу соответствует указанному значению, и возвращает значение другого ключа.

Параметры:

- `data` — массив словарей (map[string]interface{}), в которых выполняется поиск.
- `filterKey` — ключ, по которому выполняется фильтрация.
- `filterValue` — значение, с которым должно совпадать значение по filterKey.
- `targetKey` — ключ, значение которого требуется получить из найденного элемента.

Возвращает:

- Значение, соответствующее targetKey в найденном словаре.
- Ошибку, если элемент не найден или targetKey отсутствует.

Пример в шаблоне:

```
{{ findValueInDictArray .items "environment" "test"
"url" }}  // Вывод: Значение ключа "url" из первого найденного словаря, где
"environment" равно "test".
```

generatePassword

`generatePassword` генерирует случайный пароль с заданными параметрами.

Параметры:

- `length` — длина пароля (по умолчанию: 16).
- `includeUppercase` — включать заглавные буквы A-Z (по умолчанию: true).
- `includeLowercase` — включать строчные буквы a-z (по умолчанию: true).
- `includeNumbers` — включать цифры 0-9 (по умолчанию: true).

Возвращает:

- Строку со сгенерированным паролем.
- Ошибку, если невозможно сгенерировать пароль с заданными параметрами.

Примеры в шаблоне:

```
{{ generatePassword }}                                    // Вывод: Случайный пароль
длиной 16 символов
{{ generatePassword 12 }}                                 // Вывод: Случайный пароль
длиной 12 символов
{{ generatePassword 8 true true true }}                   // Вывод: Пароль длиной 8
символов
{{ generatePassword 10 false true true }}                 // Вывод: Пароль длиной 10
символов только из строчных букв и цифр
```

encodeUnicode

`encodeUnicode` преобразует строку в последовательность Unicode escape-последовательностей. Каждый символ строки кодируется в формате `\uXXXX`, где `XXXX` - четырехзначное шестнадцатеричное представление кодовой точки символа.

Параметры:

- `s` — строка, которую нужно закодировать в Unicode escape-последовательности.

Возвращает:

- Строку, где каждый символ представлен в виде `\uXXXX` escape-последовательности.

Пример в шаблоне:

```
{{ encodeUnicode "Привет" }} // Вывод: "\u041f\u0440\u0438\u0432\u0435\u0442"
{{ encodeUnicode "Hello" }}  // Вывод: "\u0048\u0065\u006c\u006c\u006f"
```

decodeUnicode

`decodeUnicode` декодирует строку, содержащую Unicode escape-последовательности, обратно в обычную строку. Функция обрабатывает escape-последовательности вида `\uXXXX`, где `XXXX` - четырехзначное шестнадцатеричное число, представляющее кодовую точку Unicode.

Параметры:

- `s` — строка с Unicode escape-последовательностями, которую нужно декодировать.

Возвращает:

- Декодированную строку, где все Unicode escape-последовательности преобразованы в символы.
- Ошибку, если декодирование не удалось (например, при некорректном формате escape-последовательностей).

Пример в шаблоне:

```
{{ decodeUnicode "\u041f\u0440\u0438\u0432\u0435\u0442" }} // Вывод: "Привет"
{{ decodeUnicode "\u0048\u0065\u006c\u006c\u006f" }}      // Вывод: "Hello"
```

jwtSign

`jwtSign` формирует подписанный JWT по переданным claims, ключу подписи и алгоритму.

Параметры:

- `claims` — `map` или JSON-строка. Стандартные поля: «sub», «iss», «aud», «exp», «iat», «nbf».
- `signingKey` — ключ подписи: для HS256/HS384/HS512 строка-секрет, для RS256/ES256 и др. — PEM закрытого ключа.
- `algorithm` — алгоритм подписи: HS256, HS384, HS512, RS256, RS384, RS512, ES256, ES384, ES512, PS256 и др.
- `headers` — заголовки JWT (опционально): `map` или JSON-строка, например «kid», «cty». Пустая строка или пустой `map` — заголовки не задаются.

Возвращает:

- Строку с подписанным JWT.
- Ошибку при неверных параметрах или ключе.

Примеры в шаблоне:

```
{{ jwtSign (dict "sub" "user-123" "iss" "ddp") .credentials.secret "HS256" "" }}
```

Глобальные переменные

Глобальные переменные — общие переменные, которые могут быть переиспользованы в шаблонизации при запуске действий.

Чтобы подставить значение глобальной переменной, в поле с поддержкой шаблонов укажите конструкцию вида:

```
{{ .global.<slug>.<key> }}
```

где:

- `global` — указывает на то, что идёт обращение к глобальным переменным.
- `slug` — идентификатор набора глобальных переменных.
- `key` — ключ переменной, значение которой необходимо подставить.

⚠ Глобальные переменные хранятся в базе данных DDP в открытом виде, их значение может быть получено пользователями через веб-интерфейс. Не рекомендуется помещать конфиденциальные данные в глобальные переменные.

Настройка глобальных переменных

Настройка глобальных переменных производится в разделе «Самообслуживание» → «Глобальные переменные».

Применяются следующие правила именования:

- Название набора глобальных переменных не должно быть пустым.
- Идентификатор набора глобальных переменных не может быть пустым и должен соответствовать следующим условиям:
  - Содержать символы `a-z`, `A-Z`, цифры, либо подчеркивания.
  - Не начинаться с цифры.
- Ключ каждой переменной набора не должен быть пустым и должен соответствовать следующим условиям:
  - Содержать символы `a-z`, `A-Z`, цифры, либо подчеркивания.
  - Не начинаться с цифры.
- Значение каждой переменной набора не должно быть пустым.

Командные переменные

Во всех действиях, сценариях и процессах можно использовать командные переменные.

Командные переменные настраиваются в разделе «Администрирование» → «Команды» в меню редактирования команды.
Каждый пользователь может редактировать переменные тех команд, в которые он входит - это можно сделать в профиле пользователя.

Чтобы получить значение командной переменной, используйте конструкцию:

```
{{ .team.<variable_name> }}
```

При запуске действия пользователь должен выбрать команду, чьи переменные будут подставлены.
При запуске сценария или процесса команда выбирается один раз — её переменные используются во всех действиях внутри.

Переменные действий

Параметры действия

Параметры действия доступны через контекст `{{ .property.* }}` и содержат значения, переданные при запуске действия.

Чтобы получить значение параметра действия, используйте конструкцию:

```
{{ .property.<property_slug> }}
```

где:

- `property` — указывает на то, что идёт обращение к параметрам действия.
- `property_slug` — идентификатор параметра, значение которого необходимо подставить.

Идентификаторы параметров можно посмотреть на вкладке `Пользовательская форма` в окне конфигурации действия.

Примеры использования:

```
{{ .property.environment }}   // Значение параметра "environment"
{{ .property.count }}         // Значение параметра "count"
{{ .property.url }}           // Значение параметра "url"
```

Ответ действия

Ответ действия доступен через контекст `{{ .response.* }}` и содержит данные, возвращённые после выполнения действия.

Чтобы взять значение из ответа действия, используйте конструкцию:

```
{{ .response.<field_name> }}
```

где:

- `response` — указывает на то, что идёт обращение к ответу действия.
- `field_name` — название поля в ответе, значение которого необходимо подставить.

Формат ответа смотрите в документации по конкретному действию или в интерфейсе DDP:

1. Откройте меню действия (кнопка с тремя точками в карточке действия).
2. Выберите пункт «Запуски действия».
3. Откройте конфигурацию одного из запусков действия.
4. Найдите в таблице колонку «Response».

Примеры использования:

```
{{ .response.status }}        // Статус ответа
{{ .response.data.id }}       // ID из данных ответа
{{ .response.headers.auth }}  // Значение заголовка авторизации
```

⚠ Контекст `{{ .response.* }}` может использоваться только в полях, описывающих правила обновления сущности после запуска действия.

Учётные данные

Учётные данные доступны во всех действиях, сценариях, процессах, виджетах, источниках данных и внешних сервисах через контекст `{{ .credentials.* }}`.

Для получения значения учётных данных используйте следующую конструкцию:

```
{{ .credentials.<credentials_slug> }}
```

где:

- `credentials` — указывает на то, что идёт обращение к учётным данным.
- `credentials_slug` — идентификатор учётных данных, значение которого необходимо подставить.

Идентификатор `credentials_slug` совпадает с тем, что задаётся во вкладке «Авторизация» в разделе «Учётные данные» в диалогах конфигурации объектов DDP.

Примеры использования:

```
{{ .credentials.token }}             // Токен доступа
{{ .credentials.username }}          // Имя пользователя
{{ .credentials.password }}          // Пароль
{{ .credentials.accessKeyId }}       // Access Key ID для S3
{{ .credentials.secretAccessKey }}   // Secret Access Key для S3
{{ .credentials.apiKey }}            // API ключ
{{ .credentials.bearerToken }}       // Bearer-токен
```

Сущность

Сущность доступна в виджетах с областью видимости `Resource`, действиях, процессах и сценариях через контекст `{{ .entity.* }}`.

Чтобы получить значение поля сущности, используйте конструкцию:

```
{{ .entity.<field_name> }}
```

где:

- `entity` — указывает на то, что идёт обращение к сущности.
- `field_name` — название поля сущности, значение которого необходимо подставить.

Основные поля сущности

```
{{ .entity.uuid }}           // UUID сущности
{{ .entity.slug }}           // Идентификатор сущности
{{ .entity.name }}           // Название сущности
{{ .entity.description }}    // Описание сущности
```

Параметры сущности

Параметры сущности доступны через контекст `{{ .entity.properties.* }}` и содержат пользовательские параметры, настроенные для конкретной сущности.

Чтобы получить значение параметра сущности, используйте конструкцию:

```
{{ .entity.properties.<property_slug> }}
```

где:

- `entity` — указывает на то, что идёт обращение к сущности.
- `properties` — указывает на параметры сущности.
- `property_slug` — идентификатор параметра сущности, значение которого необходимо подставить.

Примеры использования:

```
{{ .entity.properties.projectId }}     // ID проекта из параметров сущности
{{ .entity.properties.branch }}        // Ветка Git из параметров сущности
{{ .entity.properties.environment }}   // Окружение из параметров сущности
{{ .entity.properties.apiUrl }}        // URL API из параметров сущности
{{ .entity.properties.version }}       // Версия из параметров сущности
```

Параметры процесса

Для каждого процесса доступно задание общих параметров, значения которых могут быть использованы во всех действиях, входящих в процесс.

Чтобы получить значение параметра процесса, используйте конструкцию:

```
{{ .process.<property_slug> }}
```

где:

- `process` — указывает на то, что идёт обращение к процессу.
- `property_slug` — идентификатор параметра процесса, значение которого необходимо подставить.

Тип и значение по умолчанию параметров задаются в интерфейсе редактирования процесса. При этом для каждого конкретного параметра пользователь может при запуске процесса переопределить значение по умолчанию, если редактирование параметра разрешено.

Примеры использования:

```
{{ .process.deploymentUrl }}    // URL для деплоя из параметров процесса
{{ .process.branch }}           // Ветка Git из параметров процесса
{{ .process.environment }}      // Окружение из параметров процесса
```

Параметры сценария

Для каждого сценария доступно задание общих параметров, значения которых могут быть использованы во всех действиях, входящих в сценарий.

Чтобы получить значение параметра сценария, используйте конструкцию:

```
{{ .workflow.<property_slug> }}
```

где:

- `workflow` — указывает на то, что идёт обращение к сценарию.
- `property_slug` — идентификатор параметра сценария, значение которого необходимо подставить.

Тип и значение по умолчанию параметров задаются в интерфейсе редактирования сценария. При этом для каждого конкретного параметра пользователь может при запуске сценария переопределить значение по умолчанию, если редактирование параметра разрешено.

Примеры использования:

```
{{ .workflow.apiEndpoint }}       // API-эндпоинт из параметров сценария
{{ .workflow.notificationEmail }} // Email для уведомлений из параметров сценария
{{ .workflow.retryAttempts }}     // Количество попыток повтора из параметров сценария
```

Хранилище процесса

Хранилище доступно только в процессах и используется для передачи данных между шагами. Запись — правилами в действиях и элементом «Шаблон» (см. «Хранилище процесса»); чтение — плейсхолдерами в шаблонах.

Чтобы прочитать значение, используйте:

```
{{ .store.<путь> }}
```

Путь совпадает с полем «Цель» в правилах записи. Поддерживаются вложенные ключи и индексы массивов:

```
{{ .store.projectId }}
{{ .store.notification.engagements }}
{{ .store.items[0].status }}
```

Контекст цикла процесса

Пока выполняется тело цикла внутри процесса, в шаблонах доступен объект `_loop`:

```
{{ .store._loop.item }}    // текущий элемент коллекции или номер итерации
{{ .store._loop.index }}   // индекс с 0
{{ .store._loop.total }}   // число итераций
{{ .store._loop.first }}   // true на первой итерации
{{ .store._loop.last }}    // true на последней итерации
```

При вложенных циклах контекст родителя: `{{ .store._loop.parent }}`.

История изменений

v1.6.2

> Релиз выпущен 14.07.2026

Новые возможности

MCP

В форме редактирования пользовательских MCP-инструментов добавлено поле «Сопоставление path-параметров».

Улучшения интерфейса

В редакторе процессов для эксклюзивного шлюза и цикла улучшена работа с исходящими портами: порты «Истина»/«Ложь» и «Тело цикла»/«Выход из цикла» можно размещать на разных сторонах элемента, связь можно переносить на другой порт. Подробнее — в разделе «Процессы».

Исправления

- Исправлена ошибка в диалоге запуска процесса: параметры типа Entities корректно подставляют значение по умолчанию и проходят проверку обязательности.
- Исправлена ошибка виджета «Svacer. Ветка»: шаблонизация названия ветки в конфигурации виджета применяется корректно.
- Исправлена ошибка при создании сценария для ресурса с проверками статуса.

v1.6.1

> Релиз выпущен 13.07.2026

Исправления

- Исправлена ошибка шаблонизации в теле действия: конструкции `if`, `else`, `range` и `with`, в том числе с обрезкой пробелов (`{{- ... -}}`), корректно обрабатываются при полной отрисовке шаблона.
- Исправлена ошибка шаблонизации: сокращённая подстановка параметров действия (`{{ .slug }}`) работает в теле действия при полной отрисовке шаблона.
- Исправлена ошибка при запуске сценария из веб-интерфейса, из-за которой запрос завершался с ошибкой после успешного старта.

v1.6.0

Релиз выпущен 13.07.2026

Несовместимые изменения

Виджеты Prometheus

Обновлены виджеты «Prometheus. Метрики (диапазон)» и «Prometheus. Метрики (значение)». Существующие виджеты продолжают открываться, но для корректного отображения данных может потребоваться перенастройка:

- у виджета **«Prometheus. Метрики (диапазон)»** удалено поле «Шаг разрешения» — шаг между точками рассчитывается автоматически; сохранённое значение в конфигурации игнорируется;
- запросы PromQL теперь поддерживают плейсхолдеры `{{range}}`, `{{rateInterval}}` и `{{interval}}` — проверьте запросы с фиксированными окнами (`[5m]`, `[1h]` и т. п.) и при необходимости замените их на плейсхолдеры;

Новые возможности

Пользователи

- Добавлена имперсонация пользователей. Подробнее — в разделе «Имперсонация».
- Добавлена возможность создания «сервисных пользователей».

Безопасность

- Добавлен механизм ротации ключа шифрования для безопасной смены `security.secretKey` без потери учётных данных. Подробнее — в разделе «Ротация ключа шифрования».

Аудит-логи

Добавлена выгрузка аудит-логов в CSV. Подробнее — в разделе «Аудит-логи».

Источники данных

В источник данных типа «GenericAPI» добавлена поддержка JSON-ответов в формате map.

Шаблонизация

Добавлена функция шаблонизации «toSlug».

Процессы

- Добавлен элемент «Таймер» — пауза выполнения процесса до заданного момента времени.

- Добавлен элемент «Шаблон» — вычисление Go-шаблона и запись результата в хранилище процесса без вызова действия.
- Расширена работа с хранилищем процесса: вложенные пути, операции записи JSON, условия на правила, цикл по коллекции из хранилища.
- На панель визуализации запуска процесса добавлена вкладка «Таймлайн», на которой можно посмотреть этапы выполнения процесса с разбивкой по времени.
- Добавлена возможность создания новых действий, либо редактирования существующих, из окна конфигурации элемента процесса.
- Добавлено поле «Дата окончания» для элементов процессов и действий.

Виджеты

- Добавлены виджеты для визуализации данных из ClickHouse: «ClickHouse. Метрики (диапазон)», «ClickHouse. Метрики (значение)», «ClickHouse. Таблица», «ClickHouse. Топ N».
- Добавлен виджет «DefectDojo. Продукт» для просмотра уязвимостей продукта в DefectDojo с разбивкой по engagement и уровням критичности.
- Добавлен виджет «Kanban доска сущностей» для отображения сущностей ресурса на Kanban-доске.
- Добавлен виджет «Svacer. Ветка» для просмотра статистики по ветке проекта в Svacer.
- Обновлены виджеты «Prometheus. Метрики (диапазон)» и «Prometheus. Метрики (значение)»: выбор интервала в панели виджета, плейсхолдеры в PromQL, автоматический расчёт шага графика.

Наборы данных

- Добавлен набор данных Yandex Cloud для подключения к API Yandex Cloud.
- Добавлен набор данных Vault для подключения к API HashiCorp Vault.
- Добавлен набор данных Deckhouse managed PostgreSQL для развёртывания managed PostgreSQL в Deckhouse Kubernetes Platform.
- Добавлен набор данных Deckhouse managed Valkey для развёртывания managed Valkey в Deckhouse Kubernetes Platform.
- Добавлен набор данных Deckhouse managed Kafka для развёртывания managed Kafka в Deckhouse Kubernetes Platform.

Проверки статуса

Для правил проверки статуса добавлено поле «Расписание».

Действия

- Добавлен механизм автоматического создания сущностей после успешного выполнения действия.
- Добавлено действие GetKubernetesResource для получения спецификации ресурса из Kubernetes кластера.

MCP

Добавлен раздел **AI → MCP** для подключения upstream MCP-серверов, пользовательских MCP-инструментов и MCP-коллекций. Подробнее — в разделе «Управление MCP».

Улучшения интерфейса

- Переработаны боковые панели для разделов «Каталог», «Самообслуживание» и «Администрирование»
- Добавлена возможность создания новых учётных данных из окна конфигурации внешних сервисов, действий, виджетов и источников данных.

Исправления

- Удалена возможность создания параметра типа «Entities» для ресурсов.
- Исправлена offset-пагинация в «GenericAPI» источниках данных.
- Исправлена ошибка при синхронизации большого количества групп пользователя во время авторизации.

v1.5.0

> Релиз выпущен 04.05.2026

Несовместимые изменения

Действия

В правилах обновления хранилища процесса для получения значений из ответа действия вместо синтаксиса `{{ .key }}` необходимо использовать `{{ .response.key }}`.

Новые возможности

Виджеты

Добавлены новые виджеты:

- «AI-чат» — для предоставления пользователям преднастроенных вопросов при взаимодействии с AI-провайдерами. Подробнее — в разделе «AI-чат».
- «GitHub. Actions» — для просмотра запусков и управления GitHub Actions.
- «Технологический радар» — для визуализации технологий, инструментов и практик, используемых в компании. Подробнее — в разделе «Технологический радар».

Связи

В связях ресурсов теперь можно указывать параметры родительского и дочернего ресурса, через которые будут связаны сущности. Подробнее — в разделе «Связи между ресурсами».

Источники данных

- Query для всех типов источников данных теперь настраивается единообразно. Из конфигурации удалено неиспользуемое поле «Тип query».
- Оптимизирован алгоритм работы источников данных: увеличена скорость синхронизации и снижено количество обращений к БД.

Проверки статуса

В проверках статуса для правил типа URL добавлена возможность настроить поле «Query» для задания параметров запроса.

Действия

Для всех правил обновления после запуска действия теперь доступны все контексты шаблонизации.

Избранное

Сущности теперь можно добавлять в избранное. Подробнее — в разделе «Избранное».

Шаблонизация

Добавлена встроенная функция `fromYAML` для разбора YAML в `map`.

Улучшения интерфейса

- Улучшена визуализация графа связей для ресурсов и сущностей.
- На странице ресурса теперь можно переключать отображение списка сущностей между таблицей и карточками. Подробнее — в разделе «Режим просмотра сущностей».
- Дашборды, которые отображаются на страницах сущностей ресурса, теперь настраиваются в конфигурации ресурса. Подробнее — в разделе «Дашборды».

Исправления

- Исправлена ошибка в автоматизации, из-за которой после изменения идентификатора автоматизации происходило дублирование запуска действий и процессов.
- Исправлена ошибка в конфигурации виджетов, из-за которой после изменения типа виджета, привязанного к дашборду, он переставал отображаться.
- Исправлена сортировка по дате последней активности на странице «Администрирование» → «Пользователи».

v1.4.0

Релиз выпущен 15.04.2026

Несовместимые изменения

Доступ к API и MCP

Доступ к DDP через API или MCP с использованием заголовка `X-API-TOKEN` больше не поддерживается. Используйте заголовок `Authorization: Bearer <API-TOKEN>`.

Расширения PostgreSQL

Для работы поиска по сущностям требуется расширение PostgreSQL `pg_trgm`.

Включите его, выполнив в базе данных команду:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
```

Проверки статуса

В проверках статуса для правил типа URL расчёт результата теперь выполняется по одному или нескольким условиям (выражениям Go template). Поле «Ожидаемый статус» в конфигурации правила больше не используется. Если условия не заданы, подставляется выражение по умолчанию: `{{ eq .status.code 200 }}`. Подробнее — в разделе «Типы проверок статуса».

AI-ассистент

- В связи с изменением подхода к организации чатов для AI-ассистента текущая история пользовательских диалогов будет очищена.
- Встроенные AI-провайдеры OpenAI и Ollama удалены. Подключение к любому провайдеру теперь настраивается единообразно. Настройка описана в разделе «AI-ассистент».

Источники данных Kubernetes

Удалены встроенные типы источников данных `KubernetesNamespaces`, `KubernetesDeployments`, `KubernetesIngresses` и `KubernetesPods`; для синхронизации ресурсов кластера используйте KubernetesResources.

Новые возможности

AI-ассистент и MCP-сервер

- Добавлены новые инструменты MCP:
  - `get_actions`;

    - `get_datasources`;
    - `get_processes`;
    - `get_entity`;
    - `get_entity_relations`;
    - `get_resources`;
    - `get_external_services`.

- В AI-ассистент добавлены чаты, опциональная передача контекста диалога с моделью и автоматическая суммаризация истории чата.
- AI-ассистент теперь поддерживает множественный вызов инструментов в одном запросе.

Доверенные сертификаты

Добавлена возможность управления доверенными сертификатами через интерфейс DDP.

Интерфейс

«Глобальный поиск» — добавлен поиск по сущностям платформы.

Виджеты

Добавлены новые виджеты:

- «GitHub. Pull Requests» — для просмотра и управления Pull Requests в GitHub.
- «GitHub. Теги» — для просмотра и создания тегов в репозитории GitHub.
- «Bitbucket. Теги» — для отображения и создания тегов репозитория в Bitbucket.
- «CodeScoring. Секреты» — для отображения секретов проекта в CodeScoring.

Изменения в существующих виджетах:

- Изменена визуализация для виджета «GitLab. Участники»: отображается вклад каждого из участников в репозиторий (активность по коммитам).
- В виджете «Просмотр репозитория» добавлена поддержка Bitbucket и GitHub.

Действия

Добавлены новые действия:

- «Wait» — для паузы на заданное время.
- «Fail» — для принудительного завершения с ошибкой.

Изменения в существующих действиях: в действии `CreateVaultSecret` для параметра `allow_update` добавлено значение `merge_or_create`.

Источники данных

Добавлена поддержка нескольких последовательных правил фильтрации и исключения элементов по условию.

Шаблонизация

- Добавлена встроенная функция `jwtSign` для формирования подписанного JWT в Go template.
- В правилах файла `.templateignore` и дополнительных файлов исключения для шаблонов репозитория добавлена поддержка шаблонов Go template с теми же переменными, что и для содержимого файлов. Более подробно об этом — в документе Типы действий.

Ролевая модель

Добавлен пресет роли «Platform engineer».

Процессы

- Обновлен визуальный редактор процессов.
- Добавлен новый элемент «Примечание» для текстовых пояснений на схеме процесса.
- Добавлен новый элемент «Цикл» — механизм для повторения части процесса заданное число раз.
- Добавлен новый элемент «Ошибка» — механизм для немедленной остановки процесса.
- Добавлены условия выполнения для эксклюзивного шлюза и параллельного шлюза.
- На странице «Самообслуживание» → «Сценарии» добавлена кнопка для автоматической миграции сценариев в процессы.

Автоматизации

Добавлена возможность запуска процессов по триггеру в автоматизациях. При этом механизм выполнения сценариев по триггеру больше не поддерживается.

Проверки статуса

Обновлена логика настройки и работы правил проверки статуса типа URL, теперь можно делать проверку по полям возвращаемого ответа.

Безопасность

Access- и refresh-токены сессии пользователя больше не хранятся в local storage браузера: они передаются только через HttpOnly cookie.

Улучшения интерфейса

- Добавлена возможность создания нового виджета непосредственно из окна редактирования дашборда.
- Ошибки при взаимодействии с БД теперь выводятся в интерфейсе в человекочитаемом формате.

Исправления

Исправлена синхронизация источников данных при отсутствующих правилах сопоставления: удаление несуществующих сущностей, обновление параметров или создание связей сущностей теперь запускается только при наличии хотя бы одного правила сопоставления. Создание новых сущностей остаётся возможным, даже если правила сопоставления не заданы.

Предупреждение об удалении

В ближайших релизах будет удалён механизм сценариев. Рекомендуется использовать процессы в качестве более функциональной замены. Мигрировать существующие сценарии в процессы можно при помощи кнопки автоматической миграции на странице «Самообслуживание» → «Сценарии».

> ⚠ В действиях, выполняемых в составе сценариев, подстановка параметров через шаблонные выражения формата `{{ .workflow.<идентификатор параметра> }}` не будет автоматически изменена на `{{ .process.<идентификатор параметра> }}`. При наличии подобных конструкций измените их вручную.

v1.3.1

> ℹ Релиз выпущен 06.03.2026

Исправления

Исправлена ошибка, из-за которой дочерние ресурсы в каталоге не отображались у пользователей без разрешения `update:resources-order`.

v1.3.0

> ℹ Релиз выпущен 03.03.2026

Новые возможности

Источники данных

- Добавлены подсказки при настройке параметров источников данных.
- Добавлен импорт параметров — по JSON, YAML или OpenAPI-спецификации данных можно автоматически создать параметры ресурса и заполнить правила обновления источника данных.

Виджеты

Добавлены новые виджеты:

- Просмотр репозитория — для просмотра структуры и содержимого файлов в репозиториях.
- Jenkins. Пайплайны — для управления сборками в Jenkins.
- Jira. Задачи — для просмотра задач в Jira.
- Bitbucket. Pull Requests — для просмотра и управления Pull Requests в Bitbucket.
- Vault. Секреты — для просмотра секретов в HashiCorp Vault или Deckhouse Stronghold.

Процессы и действия

- Добавлен механизм обмена данными между действиями процесса: результаты выполнения действия можно записать в хранилище и использовать в последующих действиях — хранилище процесса.
- Добавлена возможность отмены запущенных действий и процессов.

Команды

Добавлены правила фильтрации при синхронизации команд.

Каталог

Добавлена возможность выгрузки сущностей нескольких ресурсов в CSV-файл.

Ролевая модель

- Добавлено глобальное разрешение `edit:team-filter-rules` для настройки правил фильтрации при синхронизации команд.
- Добавлено глобальное разрешение `edit:user-access-credentials-types` для управления типами учётных данных.

Параметры

Добавлен новый тип параметра Enum.

Учётные данные

Добавлена возможность хранения пользовательских учётных данных в HashiCorp Vault или Deckhouse Stronghold.

Улучшения интерфейса

- Пункты верхнего и боковых меню, а также карточки в разделе «Самообслуживание» теперь можно открыть в новой вкладке: нажмите средней кнопкой мыши или, удерживая Ctrl (Windows/Linux) либо Cmd (macOS), нажмите левой кнопкой мыши.
- Улучшена форма редактирования источников данных.

Исправления

- Решена проблема с отображением команды-владельца в карточках раздела «Самообслуживание».
- Исправлено удаление линий в редакторе процессов.

Предупреждение об удалении

В релизе 1.4.0 будут удалены механизмы источников данных для конкретных ресурсов Kubernetes: `KubernetesNamespaces`, `KubernetesDeployments`, `KubernetesIngresses`, `KubernetesPods`. Для замены рекомендуется использовать общий источник данных KubernetesResources.

v1.2.1

> Релиз выпущен 04.02.2026

Новые возможности

Действия

- Добавлена возможность выбора формата тела запроса для вебхук-действий.
- Добавлена опция расширенного логирования для вебхук-действий.
- Добавлена возможность автоматического обновления учётных данных пользователя после выполнения действия.

v1.2.0

> Релиз выпущен 03.02.2026

Несовместимые изменения

Иконки

Иконки теперь выбираются из централизованного каталога (встроенные иконки из библиотек или загруженные пользователями). Если ранее использовались иконки, заданные через URL, они перестанут отображаться.

Новые возможности

AI-ассистент

Добавлен механизм безопасного хранения учётных данных для AI-провайдеров.

Действия

- Добавлен параметр конфигурации `actions.logging.enabled`, позволяющий управлять выводом логов действий в stdout.
- Добавлена возможность настройки временного ответа для действий.
- Добавлена панель управления действиями, которая позволяет просмотреть все действия, запущенные пользователем, подтвердить назначенные действия или удалить временный ответ.
- Добавлены действия для управления Nexus Repository Manager 3:
  - `CreateNexusPrivilege` — создание привилегий;
  - `AssignNexusPrivilege` — назначение привилегий ролям;
  - `DeleteNexusPrivilege` — удаление привилегий;
  - `CreateNexusRole` — создание ролей;
  - `AssignNexusRole` — назначение ролей пользователям;
  - `DeleteNexusRole` — удаление ролей;
  - `CreateNexusUser` — создание пользователей;
  - `DeleteNexusUser` — удаление пользователей.

Владение объектами

Добавлена автоматическая установка текущего пользователя в качестве владельца создаваемых объектов.

Ролевая модель

- Добавлены новые глобальные разрешения для управления иконками: `create:icons` и `delete:icons`.
- Добавлен пресет глобальной роли `Developer`.

Исправления

- Решена проблема с принудительной перегенерацией идентификатора объекта при открытии формы редактирования.
- Исправлена ролевая модель для редактирования переменных команд.
- Исправлено поведение параметра конфигурации `datasources.logging.enabled`.
- Исправлено зависание при создании секрета в Vault без параметра `allow_update`.
- Исправлена работа механизмов подтверждения и пропуска действий в процессах.
- Исправлен выбор внешних сервисов для действий в процессах.

v1.1.1

> Релиз выпущен 17.12.2025

Исправления

- Решена проблема с доступом к обновленным свойствам сущности в последовательных сценариях.
- Решена проблема с удалением командных переменных.

v1.1.0

Релиз выпущен 15.12.2025

Несовместимые изменения

HTTP-заголовки безопасности включены по умолчанию

По умолчанию включены HTTP-заголовки безопасности (Content Security Policy, X-Frame-Options и др.), что может привести к следующим проблемам:

- «Iframe виджеты перестанут работать» — виджеты, использующие iframe для отображения внешнего контента, будут заблокированы политикой CSP.
- «Иконки из внешних источников не будут отображаться» — если для объектов DDP используются иконки, загружаемые из внешних источников, они будут заблокированы.

Чтобы сохранить предыдущее поведение:

1. Отключите заголовки безопасности полностью:

```
security:
  headers:
    enabled: false
```

2. Или настройте политики для поддержки iframe виджетов и внешних изображений:

```
security:
  headers:
    enabled: true
    csp:
      allowIframe: true
      allowExternalImages: true
```

Подробнее о настройке — в разделе «HTTP-заголовки безопасности».

Группировка ресурсов в каталоге

Изменён механизм группировки ресурсов в каталоге. Ранее ресурсы группировались по префиксу в названии с разделителем `:` (например, «Gitlab: группы», «Gitlab: проекты»). Теперь группировка осуществляется через иерархическую структуру с использованием родительских ресурсов.

Что изменилось:

- Ресурсы больше не группируются по префиксу в названии с разделителем `:`.
- Для группировки ресурсов необходимо создать родительский ресурс и привязать к нему дочерние ресурсы через drag and drop в сайдбаре каталога.
- В интерфейсе каталога ресурсы отображаются в виде иерархического дерева с возможностью раскрытия/сворачивания групп.

Требуется ручная перенастройка:

- Если у вас были ресурсы с группировкой по двоеточию (например, «Gitlab: группы», «Gitlab: проекты»), необходимо:
  1. Создать родительский ресурс (например, «Gitlab»),
  2. Убрать префикс из названий дочерних ресурсов (например, «Группы», «Проекты»),
  3. Привязать дочерние ресурсы к родительскому через drag and drop в сайдбаре каталога (перетащить дочерний ресурс на родительский),
  4. Скорректировать настройки ролевой модели, разрешить пользователям просмотр не только дочернего ресурса, но и всей иерархии родительских.
  5. Учесть, что идентификатор ресурса должен оставаться уникальным в пределах всей платформы и может сохранить префикс (например, `gitlab-groups`, `gitlab-projects`), в то время как название ресурса может быть любым.

Важно:

- Название ресурса может быть любым, но идентификатор должен оставаться уникальным в пределах всей платформы.
- При миграции существующих ресурсов рекомендуется сохранить идентификатор с префиксом для обратной совместимости, но изменить отображаемое название.

Изменения в ролевой модели

В связи с исправлением ошибок в ролевой модели, права на запуск действий и просмотр внешних сервисов по умолчанию больше не предоставляются пользователям.

Чтобы сохранить прежнее поведение, рекомендуется добавить в роль по умолчанию следующие глобальные разрешения:

- Глобальное разрешение на запуск действий (`run:actions`).
- Глобальное разрешение на просмотр внешних сервисов (`read:external-services`).

Новые возможности

AI-ассистент и MCP-сервер

> ⚠ Экспериментальный функционал

Добавлена поддержка AI-ассистента и MCP-сервера для интеграции с внешними AI-моделями и клиентами.

AI-ассистент

AI-ассистент — интеллектуальный помощник, интегрированный в интерфейс платформы, который предоставляет пользователям доступ к AI-моделям через удобный чат-интерфейс:

- «Настраиваемые провайдеры» — поддержка различных AI-провайдеров (OpenAI, локальные модели и др.).
- «Интеграция с MCP-инструментами» — AI-ассистент автоматически использует доступные инструменты MCP-сервера для выполнения запросов.

MCP-сервер

MCP-сервер — сервер, реализующий протокол MCP (Model Context Protocol) для предоставления инструментов внешним AI-клиентам:

- «Протокол MCP» — стандартизированный интерфейс для взаимодействия с AI-моделями через JSON-RPC 2.0.
- «Доступные инструменты»:
  - `get_resource_entities` — получение сущностей ресурсов платформы для анализа,
  - `get_external_data` — выполнение HTTP-запросов к внешним сервисам (GitLab, SonarQube, Kubernetes и др.).
- «Интеграция с внешними клиентами» — поддержка подключения через LM Studio, Claude Desktop и другие MCP-совместимые клиенты.
- «Безопасность» — аутентификация через API-токены, права доступа соответствуют правам пользователя.

Виджеты

Kubernetes deployments

Обновлён виджет «Kubernetes Deployments»: добавлена возможность просмотра и редактирования ресурсов контейнеров (CPU и Memory для requests и limits).

Статистика событий

Обновлён виджет «Статистика событий»: добавлена вкладка «События в Redis» для отслеживания событий, хранящихся в Redis Streams.

CodeScoring виджеты

Добавлены виджеты для интеграции с платформой анализа безопасности кода CodeScoring:

- «CodeScoring. Зависимости» — просмотр зависимостей продукта с информацией о версиях, лицензиях и количестве уязвимостей.
- «CodeScoring. Уязвимости» — таблица уязвимостей с уровнем критичности (CVSS2/CVSS3), наличием эксплойта и исправленными версиями.

Оба виджета поддерживают запуск и отмену SCA-анализа, пагинацию и фильтрацию данных.

Очередь задач

Добавлен виджет «Очередь задач» для мониторинга очереди задач и работы воркеров:

- «Статистика очереди» — размер очереди, количество ожидающих задач, активных воркеров.
- «Таблица воркеров» — информация о каждом активном воркере (название, ожидающие задачи, время простоя).
- «Таблица задач» — детальная информация о всех задачах в очереди с их статусами.
- «Фильтрация неактивных воркеров» — автоматическое скрытие воркеров, неактивных более 5 минут.

Виджет не требует дополнительной конфигурации и может быть добавлен на любой дашборд.

GitLab. Релизы

Добавлен виджет «GitLab. Релизы»:

- отображает список релизов проекта с подсветкой актуального (последнего) релиза;
- показывает тег, ссылку на коммит, дату и автора релиза, описание поддерживает Markdown;
- позволяет создать релиз прямо из платформы: выбрать существующий тег, указать название и оформить описание.

Статус сущности

Добавлен виджет «Статус сущности», который:

- отображает общий статус сущности;
- показывает детальную информацию по каждой проверке (текущий статус, время последней проверки, сообщения об ошибках);
- автоматически определяет и отображает действия, недоступные при текущем статусе сущности.

Временная шкала сущностей

Добавлен виджет «Временная шкала сущностей», который отображает сущности выбранного ресурса на временной шкале.

Календарь сущностей

Добавлен виджет «Календарь сущностей», который отображает сущности выбранного ресурса в календаре.

Воркеры

Добавлена поддержка воркеров для асинхронной обработки задач:

- «Фоновая обработка задач» — задачи выполняются в фоновом режиме, не блокируя основной сервер.
- «Масштабируемость» — настраиваемое количество реплик воркеров и параллельных задач.

Параметры конфигурации:

- `workers.replicas` — количество реплик воркеров (по умолчанию: 2);
- `workers.maxTasks` — максимальное количество параллельных задач на воркер (по умолчанию: 10).

Проверки статуса

CodeScoring Vulnerabilities

Добавлена проверка статуса «CodeScoring Vulnerabilities» — проверка количества уязвимостей по метрикам CVSS2 и CVSS3 для проекта в CodeScoring с настраиваемыми пороговыми значениями для каждого уровня критичности.

SonarQube Quality Gate

Добавлена проверка статуса «SonarQube Quality Gate» — проверка статуса Quality Gate проекта в SonarQube.

Действия

CodeScoring

Добавлены действия для работы с CodeScoring:

- `CreateCodeScoringProject` — регистрирует новый проект в CodeScoring с указанием репозитория, VCS системы и опцией автоматического запуска SCA-анализа.
- `DeleteCodeScoringProject` — удаляет проект в CodeScoring по его ID.

GitLab

`CreateGitlabRelease` — позволяет создавать релизы на основе существующих тегов GitLab.

Vault

`CreateVaultSecret` — расширена опция `allow_update`: добавлено значение `merge`. При выставлении значения `merge` обновляются или создаются только те ключи секрета, которые

указаны в действии. Существующие ключи, не упомянутые в действии, сохраняются без изменений.

Шаблонизация

Добавлены функции для работы с Unicode строками:

- `encodeUnicode` — кодирует строку в Unicode escape-последовательности;
- `decodeUnicode` — декодирует Unicode escape-последовательности обратно в строку.

Исправления

- Решена проблема с назначением прав доступа к внешним сервисам в интерфейсе.
- Решена проблема с назначением прав доступа к запуску действий (`run:actions`).
- Решена проблема, из-за которой не выполнялась первичная инициализация базы данных при установке.

v1.0.1

Релиз выпущен 10.11.2025

Новые возможности

Действия

HashiCorp Vault

Обновлено действие `CreateVaultSecret` — добавлен параметр `allow_update` . При значении `true` действие создаст новую версию секрета, если секрет с таким именем уже существует. При значении `false` действие вернёт ошибку, если секрет уже существует.

Добавлено новое действие `DeleteVaultSecret` — удаляет секрет из HashiCorp Vault.

Исправления

Исправлена ошибка с датой создания действия при его перезапуске.

v1.0.0

Релиз выпущен 31.10.2025

Новые возможности

Виджеты

Kaiten виджеты

Добавлены два виджета для интеграции с платформой управления проектами Kaiten.

- «Kaiten. Карточки пространства» — просмотр задач в пространстве:
  - Многоуровневое отображение структуры «Доска → Карточки».
  - Фильтрация по текущему пользователю и диапазону дат создания.
  - Отображение статусов, владельцев, участников, сроков и блокировок.
- «Kaiten. Статистика пространства» — аналитика и метрики по задачам:
  - Общие показатели (в очереди, в работе, выполнено).
  - Статистика по пользователям и чеклистам.
  - Долго не обновлявшиеся и последние обновленные карточки.
  - Среднее время выполнения задач.

Kafka виджеты

Расширены возможности работы с Kafka ACL.

- Действия для виджета Kafka Topics:
  - Создание и удаление топиков.
  - Отправка сообщений в топик.
  - Очистка топиков от сообщений.
- «Виджет Kafka ACLs» — управление доступом в Kafka кластере:
  - Просмотр списка правил доступа (ACL).
  - Фильтрация по типам ресурсов, шаблонам, операциям и разрешениям.
  - Создание и удаление правил доступа.

GitLab виджеты

Добавлены новые возможности для работы с GitLab.

- «GitLab. Редактор пайплайна» — редактирование `.gitlab-ci.yml`:
  - Редактирование конфигурации пайплайнов.
  - Просмотр различий между версиями.
  - Создание Merge Requests с изменениями.
- «GitLab. Статистика пайплайнов» — аналитика по пайплайнам:
  - Общие метрики (количество, процент успеха/неудач, средняя длительность).
  - Распределение по статусам и источникам.

    - Статистика по участникам и веткам.

Расширены функции виджета «GitLab. Запросы слияния»:

- «Действия виджета» — добавлена возможность выполнения действий с Merge Requests (MR):
    - Слияние и закрытие MR.
    - Изменение статуса черновика.
    - Просмотр изменений в дифф-редакторе.

S3 bucket виджет

Добавлен виджет для работы с S3-совместимыми хранилищами:

- Просмотр содержимого бакета.
- Поиск объектов по префиксу.
- Загрузка файлов из бакета.
- Просмотр детальной информации об объектах (метаданные, кэширование, шифрование).
- Поддержка AWS S3, Yandex Object Storage, MinIO.

Виджет статистики событий

Виджет для анализа событий сущностей в системе:

- График событий по типам за выбранный период.
- Топ сущностей по количеству событий.
- Настраиваемая группировка по времени.
- История изменений для каждой сущности.

Источники данных

GenericAPI

Добавлен тип источника данных Generic API для работы с произвольными инфраструктурными сервисами:

- Подключение к любому сервису с REST API.
- Поддержка разных типов пагинации для получения больших объёмов данных.
- Настраиваемые параметры запросов (заголовки, авторизация, query-параметры).

Действия

Nexus

Добавлены действия для работы с репозиториями Nexus:

- `CreateNexusRepository` — создание репозиториев любого поддерживаемого типа (maven, docker, npm и др.) в Nexus Repository Manager 3.
- `DeleteNexusRepository` — удаление репозиториев из Nexus Repository Manager 3.

Наборы данных

GitLab

Добавлен набор данных GitLab для быстрого подключения к GitLab и начала сбора и анализа репозиториев:

- Преднастроенные дашборды для просмотра статистики и управления репозиториями.
- Готовые виджеты для аналитики пайплайнов.
- Автоматическая настройка внешнего сервиса и источника данных.
- Синхронизация репозиториев GitLab в каталог платформы.

Улучшения функциональности

Внешние сервисы

Добавлен механизм внешних сервисов:

- Централизованная настройка подключений к инфраструктурным сервисам.
- Переиспользование настроек подключений для виджетов, действий, источников данных и проверок статуса.

Ролевая модель

Добавлены новые разрешения:

- `control:processes` — запуск, остановка и удаление запущенных процессов.
- `run:widget-actions` — выполнение действий виджетов.

Исправления

Виджеты

- Исправлена кнопка перезагрузки виджета в карточке сущности.
- Исправлено отображение логов действий виджетов.

Сущности и таблицы

- Исправлена функция массового удаления в таблице сущностей.
- Исправлена работа кнопки удаления нескольких сущностей, теперь она отключается, когда не выбрано ни одной сущности.
- Исправлено удаление сущности с глобальными правами доступа.

Процессы и автоматизации

- Добавлен статус «Частично завершено» для задач в процессах.
- Исправлены ошибки истории процессов.
- Добавлена информация о запусках действий в истории процесса.
- Исправлена потоковая передача логов через WebSocket при запуске действий.

- Исправлено удаление связей сущностей после выполнения автоматизации.

Самообслуживание и каталог

- Исправлена сортировка по ресурсу в разделе «Самообслуживание».

Информация о продукте

Описание функциональных характеристик

Общая информация

Программное обеспечение «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» обеспечивает выполнение следующих функций:

- ведение сервисного каталога, включающего информационные системы, микросервисы, репозитории кода, окружения, инфраструктурные ресурсы, команды, ответственных лиц и другие связанные сущности;
- настройка иерархической модели данных, атрибутов сущностей и ролей владения;
- агрегация данных и автоматическая синхронизация сервисного каталога с внешними источниками и инфраструктурными и корпоративными системами организации;
- построение и отображение связей между информационными системами, инфраструктурными ресурсами, владельцами и связанными сущностями;
- формирование дашбордов, отчетов, графиков, списков и аналитических представлений;
- настройка пользовательского интерфейса, страниц, виджетов и отображаемых данных;
- предоставление пользователям (разработчикам и командам эксплуатации) функций самообслуживания через веб-интерфейс;
- создание новых микросервисов на основе шаблонов репозиториев и стандартных конфигураций;
- автоматизация заказа инфраструктурных ресурсов организации, таких как базы данных, файловые хранилища, виртуальные машины и т.п.;
- развёртывание информационных систем и композиций микросервисов в целевых окружениях;
- создание, обновление и автоматическое удаление временных тестовых окружений;
- запуск задач по анализу и поиску уязвимостей в исходном коде и артефактах сборки;
- настройка и выполнение комплексных процессов автоматизации с условиями, таймерами, параллельными шагами, триггерами событий и маршрутизацией согласований и подтверждений действий уполномоченными пользователями;
- разграничение прав доступа пользователей и аудит операций.

Архитектура

- Архитектура ПО описана в разделе «Архитектура».
- Механизмы безопасности, реализованные в ПО, описаны в разделе «Безопасность».

Поддержание жизненного цикла

Общая информация

Поддержание жизненного цикла программного обеспечения «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» включает:

- выпуск обновлений ПО, включающих в себя обновление функционала и пользовательского интерфейса;
- устранение обнаруженных уязвимостей;
- сопровождение и устранение неисправностей, выявленных при эксплуатации.

Устранение неисправностей, выявленных в ходе эксплуатации

При возникновении неисправностей, выявленных в ходе эксплуатации, предусматривается следующая последовательность действий:

- диагностика неисправности;
- при невозможности самостоятельной диагностики и устранения неисправности обращение в службу технической поддержки для диагностики неисправности.

Устранение неисправностей, выявленных в ходе эксплуатации, выполняется путем:

- изменения конфигурации текущей версии ПО;
- изменения конфигурации окружения, используемого для установки ПО. Например, добавление аппаратных ресурсов для обеспечения корректности работы ПО;
- установки обновлённой версии ПО, содержащей программный код для устранения неисправности.

Совершенствование ПО

Процесс совершенствования ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» включает в себя следующие этапы:

- определение функционала, добавляемого в ПО;
- выявление неисправностей в текущей версии дистрибутива ПО;
- проектирование технического решения, формирование технического задания на разработку;
- реализация функционала, описанного в техническом задании;
- тестирование функционала, добавленного в ПО;
- подготовка документации по обновленной версии дистрибутива ПО;
- подготовка обновлённой версии дистрибутива ПО.

Определение функционала, добавляемого в ПО

Определение функционала, добавляемого в ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» при обновлении, происходит с использованием продуктового подхода. В качестве используемых методов применяются:

- проведение опросов пользователей ПО с целью выявления уровня удовлетворённости;
- проведение опросов пользователей ПО с целью выявления потребностей, возникающих при разработке программных продуктов;
- анализ процессов разработки программных продуктов, реализуемых пользователями ПО, их визуализация и определение мест, подлежащих оптимизации;
- анализ обращений пользователей в службу технической поддержки.

Выявление неисправностей в текущей версии дистрибутива ПО

Выявление неисправностей в текущей версии дистрибутива ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» производится на основе анализа:

- метрик систем мониторинга;
- инцидентов, возникающих при эксплуатации ПО;
- обращений пользователей в службу технической поддержки.

Проектирование технического решения, формирование технического задания на разработку

Проектирование технического решения осуществляется на основе информации о неисправностях, выявленных в текущей версии ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» и функционала, определённого как необходимого в следующей версии.

При проектировании технического решения и формировании технического задания учитываются:

- квалификация и количество сотрудников, участвующих в разработке ПО;
- целевые сроки реализации выпуска обновлённой версии дистрибутива ПО;
- критичность обнаруженных в ходе эксплуатации неисправностей и уязвимостей безопасности;
- практики и подходы к решению задач по разработке ПО.

На этапе формирования технического задания определяются следующие детальные требования:

- к функционалу обновления ПО;
- к пользовательскому интерфейсу ПО;

- к сценариям тестирования ПО.

Реализация функционала, описанного в техническом задании

Реализация функционала, описанного в техническом задании, осуществляется сотрудниками, обладающими квалификацией, указанной в разделе «Сведения о персонале».

При разработке ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» применяется система контроля версий, а также прочие современные практики разработки программных продуктов.

Тестирование функционала, добавленного в ПО

Тестирование функционала, добавленного в ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)», производится с целью:

- поиска неисправностей в добавляемом функционале;
- недопущения некорректного поведения ПО при при эксплуатации.

При тестировании ПО применяются следующие виды тестов:

- сканирование программного кода и собираемого дистрибутива на отсутствие известных уязвимостей;
- модульные тесты, запускаемые автоматически при появлении нового программного кода ПО в системе контроля версий и проверяющие корректность работы отдельных модулей ПО;
- интеграционные тесты, запускаемые автоматически при появлении нового программного кода ПО в системе контроля версий и проверяющие корректность интеграции отдельных модулей программного обеспечения;
- сквозные тесты, запускаемые автоматически при появлении нового программного кода ПО в системе контроля версий и проверяющие корректность работы сценариев, реализуемых пользователями программного обеспечения.

Подготовка документации по обновлённой версии дистрибутива ПО

Документация по обновлённой версии дистрибутива ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» описывает:

- функционал, добавленного в ПО;
- неисправности, исправленные в обновлении дистрибутива ПО;
- инструкции по обновлению ПО при необходимости.

Подготовка обновлённой версии дистрибутива ПО

Подготовка обновлённой версии дистрибутива ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» происходит путем автоматизированной сборки дистрибутива при условии положительного результата, полученного на этапе тестирования функционала.

Обновлённая версия дистрибутива публикуется в хранилище артефактов, доступном пользователям ПО.

Информация об обновлении версии ПО доводится до пользователей путём публикации сообщения об обновлении на официальном сайте ПО в разделе «История изменений».

Сведения о персонале

Требования к персоналу, задействованному в процессе разработки ПО

Персонал, задействованный в процессе разработки ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)», обладает следующими компетенциями:

- навыки работы с ПО, применяемым для контейнерной оркестрации;
- навыки разработки на языках программирования Go и TypeScript;
- понимание практик «Инфраструктура как код», «Непрерывная интеграция» и «Непрерывное развёртывание»;
- навыки работы с системами мониторинга и логирования;
- навыки работы с системами хранения секретов и артефактов;
- навыки работы с инструментами, используемыми для развёртывания программных продуктов;
- навыки работы с инструментами для описания инфраструктуры в виде кода.

Сведения о персонале, задействованном в процессе разработки ПО

Должность	Количество человек
Руководитель продукта	1
Менеджер продукта	1
Технический менеджер продукта	1
Разработчик	3

Требования к персоналу, задействованному в процессе сопровождения ПО

Персонал, задействованный в процессе сопровождения ПО «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)», обладает следующими навыками:

- эксплуатация ПО, применяемого для контейнерной оркестрации;
- эксплуатация операционных систем семейства Unix;
- управление учётными записями пользователей;
- эксплуатация систем мониторинга и логирования;
- диагностика и выявление неисправностей ПО.

Сведения о персонале, задействованном в процессе сопровождения ПО

Должность	Количество человек
Руководитель продукта	1
Менеджер продукта	1
Технический менеджер продукта	1
Разработчик	3

Фактический адрес размещения инфраструктуры разработки

115088, г. Москва, улица Угрешская, дом 12, строение 4, офис 47А.

Фактический адрес размещения разработчиков

Разработка осуществляется дистанционно штатными специалистами АО «Флант» вне места нахождения юридического лица, его филиала, представительства, вне стационарного рабочего места, территории или объекта, находящихся под контролем работодателя (дистанционная работа, ст. 312.1 ТК РФ).

Фактический адрес размещения службы поддержки

Поддержка осуществляется дистанционно штатными специалистами АО «Флант» вне места нахождения юридического лица, его филиала, представительства, вне стационарного рабочего места, территории или объекта, находящихся под контролем работодателя (дистанционная работа, ст. 312.1 ТК РФ).

Лицензирование

Общая информация

Программное обеспечение «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» лицензируется на ограниченный срок или бессрочно. Разные типы лицензий предоставляют разные уровни доступа к поддержке и обновлениям.

Стоимость и условия покупки лицензии рассчитываются индивидуально. Свяжитесь с нами для проведения демонстрации и приобретения.

Типы лицензий

Срочная лицензия

Право на использование в течение заданного периода времени.

Срок использования

Лицензия предоставляется на 1, 2 или 3 года.

Срок предоставления обновлений

Обновления предоставляются в течение 1, 2 или 3 лет.

Сценарии использования

Срочная лицензия подходит для следующих сценариев:

- среды разработки и тестирования, а также проекты с жизненным циклом менее 3 лет;
- сценарии с бюджетированием по модели OPEX.

Состав лицензии

В рамках срочной лицензии предоставляются:

- гарантийная техническая поддержка (доступ к обновлениям, исправления уязвимостей и ошибок) в течение срока действия лицензии;
- возможность продления лицензии после окончания срока действия.

Бессрочная лицензия

Предоставляет продукт в использование навсегда (срок получения обновлений ограничен).

Срок использования

Лицензия предоставляется бессрочно.

Срок предоставления обновлений

Обновления предоставляются в течение 1, 2, 3 или 5 лет.

Сценарии использования

Бессрочная лицензия подходит для следующих сценариев:

- длительное использование;
- сценарии с бюджетированием по модели CAPEX;
- среды, где размер инсталляций не будет уменьшаться.

Состав лицензии

В рамках бессрочной лицензии предоставляются:

- гарантийная техническая поддержка (доступ к обновлениям, исправления уязвимостей и ошибок) в течение выбранного срока действия (1, 2, 3 или 5 лет);
- возможность продления доступа к обновлениям путём приобретения дополнительных лицензий на обновления.

Метрики лицензирования

Программное обеспечение «Express 42 Platform. Платформа для реализации DevOps-практик (Инфраструктурный оркестратор на основе Kubernetes, внутренняя CI/CD-платформа для команд разработки, SDLC & TM-мониторинг) (альтернативное наименование — Deckhouse Development Platform)» лицензируется по количеству активных учетных записей, заведённых в продукте.

Техническая поддержка

Виды технической поддержки

Гарантийная техническая поддержка

Доступ к обновлениям решения в рамках установленного срока поддержки с новой функциональностью, исправлениями ошибок и решениями инцидентов безопасности.

Входит в основную лицензию и не требует дополнительной оплаты.

Включает

- Приём и анализ сообщений об ошибках в течение недели.
- Лучшие практики и рекомендации по развёртыванию, настройке и эксплуатации.
- Возможность оставить предложения по доработке продуктов Deckhouse и документации.

Техническая поддержка «Стандарт»

Консультации по вопросам эксплуатации от инженеров Deckhouse.

Режим работы

С 10:00 до 19:00 в рабочие дни.

Время реакции и первичного ответа

Время реакции — до 5 минут в рабочее время. Ответ на обращения — от 25 минут в рабочее время (в зависимости от критичности).

Включает

- Помощь с локализацией и устранением ошибок, включая сбор диагностической информации.
- Моделирование проблемных ситуаций на стенде техподдержки при наличии технической возможности.
- Консультации по вопросам эксплуатации.
- Рекомендации по лучшим практикам настройки и использования.

Техническая поддержка «Стандарт+»

Расширенная техническая поддержка от инженеров Deckhouse.

Режим работы

Круглосуточная поддержка без выходных.

Время реакции и первичного ответа

Время реакции — до 5 минут. Ответ на обращения — до 25 минут (в зависимости от критичности).

Включает

- Всё, что входит в «Стандарт».