Документация Deckhouse Code

Дата генерации: 22.09.2026

Документация

О продукте Deckhouse Code

Назначение и возможности

Deckhouse Code организует совместную работу команд разработки и эксплуатации: поддерживает хранение и версионирование кода, ревью кода, CI/CD-пайплайны и поставку в целевые среды.

Deckhouse Code закрывает следующие сценарии:

- хранение и версионирование Git-репозиториев, включая форки, защищённые ветки и pull-зеркалирование;
- рецензирование и слияние изменений через запросы на слияние с правилами апрувов и цепочками слияния;
- выполнение CI/CD-пайплайнов, описанных в `.gitlab-ci.yml`;
- управление проектами, группами и правами доступа;
- сканирование кода на уязвимости и отслеживание зависимостей — в Enterprise Edition.

Типы установки

Deckhouse Code поставляется в виде пакета для ОС Linux, образа Omnibus Docker, Helm-чарта или модуля Deckhouse Kubernetes Platform.

Тип установки	Что это	Когда применяется	Документация по установке
Пакет для ОС Linux	Deckhouse Code, установленный из пакета `.deb` или `.rpm` на одном хосте	Для хоста с поддерживаемым дистрибутивом Linux, без контейнера и кластера Kubernetes	Быстрый старт
Omnibus Docker	Deckhouse Code, запущенный как один контейнер, собранный из того же omnibus-пакета	Для хоста с Docker вместо прямой установки пакета	Быстрый старт
Helm-чарт	Deckhouse Code, развёрнутый в кластере Kubernetes с помощью Helm-чарта, без оператора Deckhouse Kubernetes Platform	Для кластера Kubernetes, управляемого независимо от Deckhouse Kubernetes Platform	Быстрый старт
Модуль Deckhouse Kubernetes Platform	Deckhouse Code, устанавливаемый и обслуживаемый оператором через ресурс CodeInstance	Для кластера Deckhouse Kubernetes Platform	Модуль Deckhouse Kubernetes Platform

Интеграции

Deckhouse Code поддерживает следующие интеграции:

- Jira — документация GitLab по интеграции с Jira.
- Jenkins — документация GitLab по интеграции с Jenkins.
- EvaProject — руководство по интеграции с EvaProject.

Дальнейшие шаги

Deckhouse Code доступен в редакциях Standard Edition и Enterprise Edition, различия описаны в разделе «Редакции».

Задачи администрирования инстанса описаны в разделе «Администрирование», начиная с его страницы «Обзор». Повседневная работа с проектами, репозиториями и запросами на слияние описана в разделе «Использование», начиная с его страницы «Обзор».

Редакции

Deckhouse Code поставляется в редакциях Standard Edition (SE) и Enterprise Edition (EE).

Доступность возможностей

Категория	Возможности	SE	EE
Типы установки	Пакет для ОС Linux, Omnibus Docker, Helm-чарт, модуль Deckhouse Kubernetes Platform	Да	Да
Проекты и группы	Группы и участники, уровни видимости проектов, импорт проектов, задачи, метки, вехи, доски, wiki проекта и группы	Да	Да
Исходный код	Git-репозитории с форками и веб-редактором, защищённые ветки, правила отправки изменений, pull-зеркалирование, CODEOWNERS, доступ к Git по SSH и HTTPS	Да	Да
Ревью кода и запросы на слияние	Запросы на слияние, правила апрувов, внешние проверки статуса, цепочки слияния, аналитика ревью кода, аналитика запросов на слияние	Да	Да
CI/CD	Пайплайны из `.gitlab-ci.yml`, секреты из внешнего Vault или Deckhouse Stronghold, сервисные аккаунты для пайплайнов	Да	Да
		Нет	Да

Категория	Возможности	SE	EE
Сканирование и отчётность по безопасности	SAST-сканирование с помощью Semgrep, список зависимостей из CycloneDX SBOM, соответствие лицензий		
Аналитические отчёты	Аналитика задач, аналитика CI/CD, групповая аналитика репозитория	Нет	Да
Аналитика репозитория проекта	Языки программирования, покрытие тестами и статистика коммитов проекта	Да	Да
Настройки безопасности и аудит	Настройки безопасности группы и проекта, настройки безопасности инстанса, ключи доступа, события аудита	Да	Да
Идентификация и доступ	Вход с персональной двухфакторной аутентификацией, провайдеры OmniAuth, синхронизация LDAP, ограничения входа, ограничения создания групп и проектов, ограничения протокола, обязательная двухфакторная аутентификация для инстанса	Да	Да

Категория	Возможности	SE	EE
Поиск	Расширенный поиск, индексирование поиска через OpenSearch	Да	Да
Интеграции	Вебхуки для событий группы, проекта и подгруппы	Да	Да
Эксплуатация	Лимиты ресурсов и настройки производительности, мониторинг, режим обслуживания, резервное копирование и восстановление, обновление, диагностика	Да	Да

Страницы, доступные только в Enterprise Edition, отмечены в боковом меню документации значком «EE».

Установка

Пакет для ОС Linux

Требования

Пакет для ОС Linux разворачивает Deckhouse Code на одном сервере: веб-интерфейс, Git, база данных и фоновые задачи работают на этой же машине.

Операционная система

Поддерживаемые дистрибутивы, архитектура — `x86_64`:

- РЕД ОС 8.x;
- Ubuntu 24.04 LTS.

Системные зависимости устанавливаются из репозиториев ОС, поэтому доступ к ним нужен на время установки. Доступ в интернет не требуется.

Ресурсы

Значения рассчитаны на типовой инстанс:

Ресурс	Значение
Оперативная память	8 ГБ; минимум — 6 ГБ и раздел подкачки (swap)
Диск	от 20 ГБ под каталог данных `/var/opt/gitlab`

Для инстансов, обслуживающих сотни активных пользователей и больше, запросите у вендора рекомендации по подбору ресурсов.

Установочный пакет

Установочный пакет получите у вендора, имя файла зависит от ОС:

ОС	Файл пакета
РЕД ОС 8.x	`deckhouse-code-<VERSION>.el8.x86_64.rpm`
Ubuntu 24.04 LTS	`deckhouse-code_<VERSION>_amd64.deb`

`<VERSION>` — версия полученного пакета, например `1.0.0-0`.

Сетевые порты

Инстанс принимает соединения на портах, которые открывают на файрволе сервера:

Порт	Назначение
`22`	Git по SSH, обслуживается системным sshd
`80`	Перенаправление на HTTPS
`443`	Веб-интерфейс и Git по HTTPS

Быстрый старт

Сначала инстанс запускается по HTTP, затем на нём включается TLS. Перед началом убедитесь, что сервер отвечает требованиям.

В командах и файлах конфигурации замените `<HOSTNAME>` на FQDN инстанса, а `<VERSION>` — на версию полученного пакета, например `1.0.0-0`.

> CLI-утилиты (`gitlab-ctl`, `gitlab-backup`), файл конфигурации (`/etc/gitlab/gitlab.rb`) и служебные каталоги (`/opt/gitlab`, `/var/opt/gitlab`) сохраняют префикс `gitlab-`: Deckhouse Code работает на движке GitLab, и системные имена оставлены без изменений.

Подготовка

Имя инстанса `<HOSTNAME>` используется в адресе веб-интерфейса, в адресах репозиториев для Git, в параметре `external_url` и в полях CN и SAN сертификата — везде одно и то же имя.

1. Проверьте, что имя разрешается в IP-адрес машины:

   ```
   getent hosts <HOSTNAME>
   ```

2. Если DNS-записи A или AAAA нет, добавьте имя в файл `/etc/hosts` на сервере:

   ```
   echo "<IP_ADDRESS> <HOSTNAME>" | sudo tee -a /etc/hosts
   ```

   `<IP_ADDRESS>` — IP-адрес машины инстанса. Ту же строку добавьте на каждой машине, с которой будут открывать веб-интерфейс и работать с Git, включая вашу рабочую станцию.

Установка пакета

Установка сама запускает первичную настройку (`gitlab-ctl reconfigure`), она занимает несколько минут. Адрес инстанса берётся из переменной `EXTERNAL_URL` при первой установке; TLS включается на следующем шаге.

РЕД ОС 8

Установите системные зависимости:

```
sudo dnf install -y libxcrypt-compat policycoreutils-python-utils
```

Пакет `libxcrypt-compat` даёт библиотеку `libcrypt.so.1`, без которой приложение не запускается: в минимальной поставке РЕД ОС этой библиотеки нет. Пакет `policycoreutils-python-utils` даёт утилиту `semanage`, которая выставляет контексты SELinux при первичной настройке.

Установите пакет Deckhouse Code:

```
sudo EXTERNAL_URL="http://<HOSTNAME>" \
  rpm -i ./deckhouse-code-<VERSION>.el8.x86_64.rpm
```

Режим SELinux `enforcing` поддерживается: модуль политики устанавливается при первичной настройке, ручные действия не нужны.

Ubuntu 24.04

Отдельно устанавливать системные зависимости не нужно: они объявлены в пакете, и `apt` возьмёт их из репозиториев ОС.

Установите пакет Deckhouse Code:

```
sudo EXTERNAL_URL="http://<HOSTNAME>" \
  apt install -y ./deckhouse-code_<VERSION>_amd64.deb
```

Префикс `./` в пути к файлу обязателен: без него `apt` ищет пакет с таким именем в репозиториях.

Что получилось

На `http://<HOSTNAME>` работает инстанс: веб-интерфейс, Git поверх HTTP, база данных и фоновые задачи. Проверьте состояние — команды выполняются на самой машине инстанса:

```
# Все сервисы в состоянии run.
sudo gitlab-ctl status
# Ожидается код 200.
curl -s -o /dev/null -w "%{http_code}\n" http://localhost/-/health
# Первый пароль администратора.
sudo cat /etc/gitlab/initial_root_password
```

Откройте `http://<HOSTNAME>` в браузере и войдите под пользователем `root` с паролем из файла `/etc/gitlab/initial_root_password`.

> ⚠ Сохраните пароль: файл `initial_root_password` хранится 24 часа, и любой следующий запуск `gitlab-ctl reconfigure` удалит его, если с установки прошло больше суток. Само значение пароля при этом не меняется.

TLS

По HTTP пароли и содержимое репозиториев передаются в открытом виде, поэтому сразу после первого входа включите TLS.

Подготовьте каталог для сертификатов:

```
sudo mkdir -p /etc/gitlab/ssl && sudo chmod 755 /etc/gitlab/ssl
```

Разместите в нём сертификат и ключ под именами `<HOSTNAME>.crt` и `<HOSTNAME>.key`.

Свой сертификат

Выпустите в вашем центре сертификации сертификат на имя `<HOSTNAME>` и скопируйте файлы на сервер:

```
sudo install -m 600 <KEY_FILE> /etc/gitlab/ssl/<HOSTNAME>.key
# Сертификат вместе с цепочкой.
sudo install -m 644 <CERT_FILE> /etc/gitlab/ssl/<HOSTNAME>.crt
```

`<KEY_FILE>` и `<CERT_FILE>` — пути к полученным файлам ключа и сертификата.

Самоподписанный сертификат

На тестовом стенде выпустите сертификат на месте. Имя инстанса в поле SAN обязательно, иначе клиенты отклонят сертификат:

```
sudo openssl req -x509 -nodes -newkey rsa:2048 -days 825 \
  -keyout /etc/gitlab/ssl/<HOSTNAME>.key \
  -out /etc/gitlab/ssl/<HOSTNAME>.crt \
  -subj "/CN=<HOSTNAME>/O=Deckhouse Code/C=RU" \
  -addext "subjectAltName=DNS:<HOSTNAME>"
sudo chmod 600 /etc/gitlab/ssl/<HOSTNAME>.key
```

Включите HTTPS в файле `/etc/gitlab/gitlab.rb`:

```
external_url 'https://<HOSTNAME>'
letsencrypt['enable'] = false
nginx['redirect_http_to_https'] = true
nginx['ssl_certificate'] = "/etc/gitlab/ssl/<HOSTNAME>.crt"
nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/<HOSTNAME>.key"
```

Без строки `letsencrypt['enable'] = false` настройка с адресом `https://` запускает получение сертификата через Let's Encrypt по HTTP-01 и в закрытом контуре завершается ошибкой.

Примените конфигурацию:

```
sudo gitlab-ctl reconfigure
```

Проверка

Проверьте состояние сервисов, ответ по HTTPS и перенаправление с HTTP:

```
# Все сервисы в состоянии run.
sudo gitlab-ctl status
# Параметр --resolve направляет запрос на loopback: адрес /-/health отвечает
# только с самой машины, проверка имени в сертификате при этом сохраняется.
# Ожидается health=200 tls=0.
curl --cacert /etc/gitlab/ssl/<HOSTNAME>.crt \
     --resolve '<HOSTNAME>:443:127.0.0.1' \
     -o /dev/null -w "health=%{http_code} tls=%{ssl_verify_result}\n" \
     https://<HOSTNAME>/-/health
# Перенаправление с HTTP на HTTPS, ожидается https://<HOSTNAME>/.
curl -sI http://<HOSTNAME>/ | grep -i location
```

Самоподписанный сертификат браузер и Git не примут, пока центр сертификации не добавлен в доверенные. Для Git укажите путь к файлу сертификата на машине клиента:

```
git config --global http.https://<HOSTNAME>.sslCAInfo <CERT_FILE>
```

Смените пароль пользователя `root` в веб-интерфейсе и удалите файл первоначального пароля:

```
sudo rm -f /etc/gitlab/initial_root_password
```

Настройка после установки

Эти настройки выполняются до передачи инстанса пользователям. Изменения в файле `/etc/gitlab/gitlab.rb` применяются командой `sudo gitlab-ctl reconfigure`.

Язык интерфейса

Язык задаётся для всего инстанса администратором и отдельно каждым пользователем для себя.

Русский язык для всего инстанса действует на страницу входа и на всех пользователей, которые не выбирали язык вручную. Войдите администратором, в правом верхнем углу выберите «Admin», в левой панели перейдите в «Settings» → «Preferences», прокрутите страницу до раздела «Localization», в поле «Default language» выберите «Russian - русский» и нажмите «Save changes».

Пользователь меняет язык для себя: аватар в правом верхнем углу → «Preferences» → «Localization» → «Language» → «Russian - русский» → «Save changes». Интерфейс переключится после обновления страницы. Личный выбор имеет приоритет над языком инстанса.

Почта (SMTP)

Без почты не работают сброс пароля, приглашения и уведомления. Задайте параметры подключения к серверу почты в файле `/etc/gitlab/gitlab.rb`:

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "<SMTP_HOST>"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "<SMTP_USER>"
gitlab_rails['smtp_password'] = "<SMTP_PASSWORD>"
gitlab_rails['smtp_domain'] = "<DOMAIN>"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['gitlab_email_from'] = "deckhouse-code@<DOMAIN>"
```

Примените конфигурацию и отправьте тестовое письмо на свой адрес:

```
sudo gitlab-ctl reconfigure
sudo gitlab-rails runner \
  "Notify.test_email('<ADMIN_EMAIL>', 'Тест', 'Работает').deliver_now"
```

Сетевой доступ (файрвол)

Инстансу нужны порты `22` (Git по SSH через системный sshd), `80` (перенаправление на HTTPS) и `443` (веб-интерфейс и Git по HTTPS). Откройте их средствами вашей ОС.

firewalld (РЕД ОС)

```
sudo systemctl enable --now firewalld
sudo firewall-cmd --permanent --add-service={ssh,http,https}
sudo firewall-cmd --reload
```

ufw (Ubuntu)

```
sudo ufw allow OpenSSH
sudo ufw allow 80,443/tcp
sudo ufw enable
```

Ресурсы

При памяти 6–8 ГБ закрепите число рабочих процессов, чтобы автоматическая настройка не заняла всю память. Задайте значения в файле `/etc/gitlab/gitlab.rb`:

```
puma['worker_processes'] = 2
sidekiq['concurrency'] = 10
```

Примените конфигурацию:

```
sudo gitlab-ctl reconfigure
```

Проверьте, что на сервере настроен раздел подкачки (swap): он покрывает пики потребления памяти при миграциях во время обновления и при тяжёлых операциях с репозиториями.

Порт SSH

Если служба SSH инстанса слушает порт, отличный от 22, укажите его в `/etc/gitlab/gitlab.rb`, чтобы URL для клонирования в веб-интерфейсе содержали этот порт:

```
gitlab_rails['gitlab_shell_ssh_port'] = 2222
```

Примените конфигурацию:

```
sudo gitlab-ctl reconfigure
```

Расширенный поиск (OpenSearch)

Расширенный поиск работает на внешнем кластере OpenSearch. Задайте подключение в `/etc/gitlab/gitlab.rb` и направьте задачи индексации в отдельную очередь Sidekiq:

```
gitlab_rails['fe_search'] = {
  'advanced_search_enabled' => true,
  'opensearch' => {
    'url' => 'https://<OPENSEARCH_HOST>:9200',
    'username' => '<OPENSEARCH_USER>',
    'password' => '<OPENSEARCH_PASSWORD>'
  }
}
sidekiq['routing_rules'] = [
  ['feature_category=fe_global_search', 'global-search-indexing'],
  ['*', 'default']
]
```

`<OPENSEARCH_HOST>` — адрес кластера OpenSearch, `<OPENSEARCH_USER>` и `<OPENSEARCH_PASSWORD>` — учётные данные аккаунта, которому принадлежат индексы. Примените конфигурацию:

```
sudo gitlab-ctl reconfigure
```

Затем переиндексируйте инстанс и включите расширенный поиск в настройках, как описано на странице «Расширенный поиск».

Диагностика

Ошибки ниже встречаются при установке пакета для ОС Linux и первичной настройке инстанса. Если нужного случая здесь нет, соберите вывод команд из раздела «Общая диагностика» и обратитесь к вендору.

Приложение не запускается: нет библиотеки libcrypt.so.1

Где проявляется. РЕД ОС, вывод установки пакета и первичной настройки:

```
libcrypt.so.1: cannot open shared object file
```

Причина. В минимальной поставке РЕД ОС нет библиотеки `libcrypt.so.1`.

Решение. Установите пакет с библиотекой и повторите настройку:

```
sudo dnf install -y libxcrypt-compat
sudo gitlab-ctl reconfigure
```

Настройка завершается ошибкой SELinux

Где проявляется. РЕД ОС, выполнение `gitlab-ctl reconfigure`, ресурс `bash[Set proper security context on ssh files for selinux]`:

```
ValueError: Type gitlab_shell_t is invalid
semanage: command not found
```

Причина. На сервере нет утилит SELinux из пакета `policycoreutils-python-utils` либо модуль политики не загрузился при первичной настройке.

Решение. Установите утилиты, повторите настройку и проверьте, что модуль политики загружен:

```
sudo dnf install -y policycoreutils-python-utils
sudo gitlab-ctl reconfigure
sudo semodule -l | grep -i gitlab
```

Имя инстанса недоступно: curl возвращает 000

Причина. Имя `<HOSTNAME>` не разрешается в IP-адрес на сервере или на машине, с которой открывают веб-интерфейс, либо порт `80` или `443` недоступен: закрыт файрволом или не запущен nginx.

Решение.

1. Проверьте имя командой `getent hosts <HOSTNAME>`. Если имени нет, создайте DNS-запись A или AAAA на IP-адрес машины, а для стенда без DNS добавьте строку в `/etc/hosts`. Имя должно совпадать с `external_url` и с полями CN и SAN сертификата.
2. Если имя разрешается в ожидаемый IP-адрес, проверьте на сервере состояние nginx и занятые порты:

   ```
   sudo gitlab-ctl status nginx
   sudo ss -tlnp | grep -E ':(80|443)\b'
   ```

3. Проверьте вход по HTTP с рабочей станции:

   ```
   curl -sS -o /dev/null -w "HTTP %{http_code}\n" \
     http://<HOSTNAME>/
   ```

> Любой HTTP-код (после включения TLS обычно 301 или 302) означает, что nginx доступен. Код 000 означает, что соединение не установлено. Если nginx запущен, а порт закрыт снаружи, откройте его по разделу «Сетевой доступ (файрвол)».

Веб-интерфейс отвечает 502 после настройки или перезапуска

Причина. Приложение ещё запускается: первые одну-две минуты после `reconfigure` или `restart` код 502 ожидаем.

Решение. Подождите и проверьте состояние сервисов командой `sudo gitlab-ctl status`. Если 502 держится дольше нескольких минут, читайте лог командой `sudo gitlab-ctl tail puma`: частая причина — нехватка памяти, порядок настройки описан в разделе «Ресурсы».

Настройка завершается ошибкой Let’s Encrypt

Где проявляется. Выполнение `gitlab-ctl reconfigure` после того, как в `external_url` задан адрес `https://`, а файлов сертификата на диске нет.

Причина. Без сертификата на диске включается получение сертификата через Let’s Encrypt, что в закрытом контуре невозможно.

Решение. Разместите сертификат и задайте `letsencrypt['enable'] = false`, как описано в разделе «TLS», затем выполните `sudo gitlab-ctl reconfigure`.

Чтение ошибки в выводе reconfigure

Где проявляется. Вывод команды `gitlab-ctl reconfigure`, завершившейся с ошибкой.

Решение. В конце вывода найдите блок с именем ресурса и причиной:

```
Error executing action ... on resource '...'
```

Полный лог запуска сохраняется в каталоге `/var/log/gitlab/reconfigure/`.

Общая диагностика

Команды применимы к любой из перечисленных ошибок:

```
# Состояние всех сервисов.
sudo gitlab-ctl status
# Живой лог сервиса, например puma или nginx.
sudo gitlab-ctl tail <SERVICE>
# Самопроверка инстанса.
sudo gitlab-rake gitlab:check SANITIZE=true
# Каталоги логов по сервисам.
ls /var/log/gitlab/
```

Omnibus Docker

Требования

Deckhouse Code работает в одном контейнере с тем же omnibus-пакетом, что и при установке пакетом для ОС Linux. Хост предоставляет Docker Engine, ресурсы одного инстанса, каталоги под тома и сетевые порты.

Хост

Контейнер запускается на хосте с ОС Linux, архитектурой x86_64 и Docker Engine.

Требования к памяти и диску совпадают с установкой пакетом для ОС Linux и перечислены в разделе «Требования»: 8 ГБ RAM (минимум 6 ГБ и swap) и не менее 20 ГБ свободного места под данные инстанса. Образ занимает на хосте ещё 2,5–3 ГБ.

Тома

Образ объявляет тома для конфигурации, данных и логов. Подключите их все: в них хранится состояние, которое переживает контейнер, — при обновлении контейнер пересоздаётся, а тома остаются.

Путь в контейнере	Содержимое
`/etc/gitlab`	Файл конфигурации `gitlab.rb`, ключи шифрования `gitlab-secrets.json`, TLS-сертификаты, ключи хоста SSH, файл с первоначальным паролем `root`.
`/var/opt/gitlab`	Репозитории, база данных, загруженные файлы, артефакты CI, реестр пакетов, архивы резервных копий в `/var/opt/gitlab/backups`.
`/var/log/gitlab`	Логи всех сервисов и лог каждого запуска настройки в `/var/log/gitlab/reconfigure`.

В примерах на этих страницах тома — это каталоги хоста: `/srv/code/config`, `/srv/code/data` и `/srv/code/logs`.

Сетевые порты

Опубликованные порты обслуживают трафик Git по SSH, HTTP и HTTPS.

Порт	Назначение
`22`	Git по SSH. В контейнере работает собственный `sshd`, который принимает пользователя `git` и аутентификацию по открытому ключу.
`80`	Веб-интерфейс и Git по HTTP, после настройки TLS — перенаправление на HTTPS.
`443`	Веб-интерфейс и Git по HTTPS.

Порты публикуются при создании контейнера. Порт хоста может отличаться от порта контейнера: тогда в адресе инстанса указывают порт хоста, который открывают пользователи, а порт SSH, отличный от 22, указывают в конфигурации, как описано на странице «Настройка после установки».

Разделяемая память

Создавайте контейнер с параметром `--shm-size 256m`. По умолчанию Docker выделяет под `/dev/shm` 64 МБ.

Быстрый старт

Для работающего инстанса достаточно одной команды `docker run` и первого запуска, который настраивает инстанс. Сначала инстанс отвечает по HTTP, включение TLS — последний шаг этой страницы.

Перед началом

Проверьте хост по разделу «Требования» и определитесь с именем `<HOSTNAME>`: по нему пользователи открывают веб-интерфейс и обращаются к репозиториям Git. Это же имя указывается в адресе инстанса и в сертификате.

Проверьте, что имя разрешается в IP-адрес хоста:

```
getent hosts <HOSTNAME>
```

Если DNS-записи нет, добавьте имя в `/etc/hosts` на хосте и на каждой машине, с которой открывают веб-интерфейс и работают с Git.

Получение образа

Образ публикуется в вариантах по базовому дистрибутиву: название варианта подставляется вместо `<FLAVOR>` в ссылке на образ.

Вариант	Базовый дистрибутив
`ubuntu_22.04`	Ubuntu 22.04
`ubuntu_24.04`	Ubuntu 24.04
`redos_8`	RED OS 8

Каждый тег называет версию, отдельного тега для последней версии нет, поэтому и при первом запуске, и при каждом обновлении версия указывается явно:

```
docker pull <REGISTRY>/<FLAVOR>:<VERSION>
```

Запуск контейнера

Создайте контейнер с томами, опубликованными портами и адресом инстанса:

```
docker run -d --name code \
  --shm-size 256m \
  -p 80:80 -p 443:443 -p 22:22 \
  -e GITLAB_OMNIBUS_CONFIG="external_url 'http://<HOSTNAME>'" \
  -v /srv/code/config:/etc/gitlab \
  -v /srv/code/logs:/var/log/gitlab \
  -v /srv/code/data:/var/opt/gitlab \
  <REGISTRY>/<FLAVOR>:<VERSION>
```

Имя `code` используется во всех командах ниже. Левый порт в каждом `-p` — это порт хоста, он может отличаться от порта контейнера; порт 22 на хосте обычно занят системным `sshd`, а настройка, которая указывает другой порт в URL для клонирования, описана в разделе «Настройка после установки».

Порты, тома и переменные окружения задаются при создании контейнера. Чтобы изменить любое из них, удалите контейнер и создайте новый с теми же томами.

Адрес инстанса

Без настройки адресом становится `http://<CONTAINER_HOSTNAME>`, где `<CONTAINER_HOSTNAME>` — имя хоста контейнера. Docker задаёт его равным идентификатору контейнера, если не указан параметр `--hostname`. Этот идентификатор попадёт во все ссылки и URL для клонирования, поэтому задайте адрес при первом запуске.

Адрес — это настройка `external_url` в `gitlab.rb`, и образ берёт её из переменной окружения и из файла конфигурации:

- `GITLAB_OMNIBUS_CONFIG` содержит строки конфигурации и вычисляется при каждом запуске;
- `/etc/gitlab/gitlab.rb` на томе конфигурации читается после переменной, поэтому настройка из файла переопределяет ту же настройку из переменной.

Переменная `EXTERNAL_URL`, которую читает установщик пакета, в контейнере не действует.

Первый запуск

Стартовый скрипт готовит инстанс до того, как сервисы начнут принимать запросы:

- копирует `gitlab.rb` из шаблона, если на томе конфигурации файла нет;
- генерирует ключи хоста SSH в `/etc/gitlab`, если их нет;
- выполняет `gitlab-ctl reconfigure` — шаг, который при первом запуске занимает несколько минут;
- приводит базу данных к текущей версии PostgreSQL;
- выводит логи сервисов в вывод контейнера.

Следите за запуском по выводу контейнера:

```
docker logs -f code
```

В образе настроена проверка состояния, которая выполняется каждые 60 секунд. До первой успешной проверки состояние контейнера — `starting`, после пяти неуспешных проверок подряд — `unhealthy`; первый запуск доходит до этого состояния, пока настройка ещё выполняется:

```
docker inspect --format '{{.State.Health.Status}}' code
```

Запуск завершён, когда состояние контейнера — `healthy`, а все сервисы находятся в состоянии `run`:

```
docker exec code gitlab-ctl status
```

Первый вход

Первый пароль пользователя `root` записывается на том конфигурации. Прочитайте его из контейнера:

```
docker exec code cat /etc/gitlab/initial_root_password
```

Откройте `http://<HOSTNAME>` и войдите под пользователем `root` с этим паролем.

> ⚠ Файл с паролем удаляется при первой настройке, которая выполняется позже чем через 24 часа после записи файла, и шаг с TLS ниже — это такая настройка. Смените пароль `root` в веб-интерфейсе и удалите файл: `docker exec code rm -f /etc/gitlab/initial_root_password`.

По HTTP данные для входа и содержимое репозиториев передаются в открытом виде. Инстанс, с которым работают пользователи, работает по TLS.

TLS

Разместите сертификат и ключ на томе конфигурации, чтобы контейнер читал их из `/etc/gitlab/ssl`:

```
sudo mkdir -p /srv/code/config/ssl && sudo chmod 755 /srv/code/config/ssl
sudo install -m 644 <CERT_FILE> /srv/code/config/ssl/<HOSTNAME>.crt
sudo install -m 600 <KEY_FILE> /srv/code/config/ssl/<HOSTNAME>.key
```

`<CERT_FILE>` — файл сертификата в формате PEM, `<KEY_FILE>` — файл его закрытого ключа.

Выпуск сертификата, требование к `subjectAltName` и причина, по которой отключается Let's Encrypt, описаны в разделе «Быстрый старт».

Добавьте настройки в файл `/srv/code/config/gitlab.rb`, который контейнер читает как `/etc/gitlab/gitlab.rb`:

```
external_url 'https://<HOSTNAME>'
letsencrypt['enable'] = false
nginx['redirect_http_to_https'] = true
nginx['ssl_certificate'] = "/etc/gitlab/ssl/<HOSTNAME>.crt"
nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/<HOSTNAME>.key"
```

Пути в настройках — это пути внутри контейнера. Примените изменение перезапуском контейнера: настройка выполняется при каждом запуске.

```
docker restart code
```

Проверка

Проверьте сервисы, состояние контейнера и перенаправление с HTTP:

```
docker exec code gitlab-ctl status
docker exec code gitlab-healthcheck --fail --max-time 10
docker inspect --format '{{.State.Health.Status}}' code
curl -sI http://<HOSTNAME>/ | grep -i location
```

Откройте `https://<HOSTNAME>` в браузере, войдите и создайте тестовый проект, чтобы проверить доступ к Git по HTTPS и по SSH.

Настройка после установки

Настройки инстанса в контейнере — это настройки установки пакетом для ОС Linux, к которым добавляется слой контейнера: где лежит файл конфигурации, какие переменные читает стартовый скрипт и какие порты публикует хост.

Применение изменений конфигурации

Файл конфигурации — `/etc/gitlab/gitlab.rb` на томе конфигурации, то есть `/srv/code/config/gitlab.rb` на хосте. Правьте его на хосте или в контейнере:

```
docker exec -it code editor /etc/gitlab/gitlab.rb
```

Примените изменение, не останавливая контейнер:

```
docker exec code gitlab-ctl reconfigure
```

Перезапуск даёт тот же результат: настройка выполняется при каждом запуске контейнера.

```
docker restart code
```

Язык интерфейса, почта, число рабочих процессов и срок хранения резервных копий — это те же настройки, что и при установке на хост; что делает каждая из них, описано в разделе «Настройка после установки».

Переменные запуска

Стартовый скрипт читает переменные при запуске контейнера, поэтому для изменения любой из них контейнер создают заново с теми же томами.

Переменная	Назначение
`GITLAB_OMNIBUS_CONFIG`	Содержит строки конфигурации, которые вычисляются до чтения `/etc/gitlab/gitlab.rb`. Настройка из файла переопределяет ту же настройку из переменной.
`GITLAB_SKIP_RECONFIGURE`	Со значением `true` на уже настроенном инстансе запуск пропускает `gitlab-ctl reconfigure`, и изменения конфигурации остаются неприменёнными до следующей настройки.
`GITLAB_SKIP_PG_UPGRADE`	Со значением `true` запуск не выполняет `gitlab-ctl pg-upgrade`, и база данных остаётся на текущей версии PostgreSQL.
`GITLAB_DISABLE_OPENSSH`	Со значением `true` запуск не готовит сервис `sshd`, и Git по SSH недоступен.
`GITLAB_SKIP_TAIL_LOGS`	Со значением `true` логи сервисов не выводятся в вывод контейнера и остаются на томе логов.
`GITLAB_PRE_RECONFIGURE_SCRIPT`	Если переменная задана, её значение выполняется как команда оболочки до запуска сервисов.
`GITLAB_POST_RECONFIGURE_SCRIPT`	Если переменная задана, её значение выполняется как команда оболочки после шага с PostgreSQL.
`GITLAB_ALLOW_SHA1_RSA`	Со значением `true` конфигурация `sshd` принимает алгоритм ключа хоста и тип открытого ключа `ssh-rsa`.
`OPENSSL_FORCE_FIPS_MODE`	Значение переменной записывается в окружение сервиса `sshd`.

Ключи хоста SSH

Ключи хоста RSA, ECDSA и Ed25519 генерируются при первом запуске в `/etc/gitlab` и связываются символическими ссылками с `/etc/ssh`, откуда их читает инстанс. Ключи лежат на томе конфигурации, поэтому контейнер, пересозданный при обновлении, сохраняет их, и клиенты

Git видят прежний ключ хоста. Новый том конфигурации означает новые ключи, и каждый клиент при следующем подключении сообщит о смене ключа хоста.

Git по SSH работает с ключами, которые пользователи добавляют в веб-интерфейсе: `sshd` в контейнере слушает порт 22, принимает только пользователя `git`, аутентификация по паролю отключена.

Ресурсы и рабочие процессы

Контейнеру доступна память хоста, если при создании она не ограничена параметром `--memory`, и ему нужен параметр `--shm-size 256m`, как описано в разделе «Требования».

На хосте с 6–8 ГБ RAM закрепите число рабочих процессов, чтобы его не подбирала автонастройка. Настройки задаются в `/etc/gitlab/gitlab.rb` или в `GITLAB_OMNIBUS_CONFIG`, их значения описаны в разделе «Настройка после установки»:

```
puma['worker_processes'] = 2
sidekiq['concurrency'] = 10
```

Порты и файрвол хоста

Порты 80, 443 и 22 публикуются при создании контейнера. Порт хоста может отличаться от порта контейнера, например `-p 8080:80 -p 8443:443 -p 2222:22`.

В адресе инстанса указывается порт хоста, который открывают пользователи; порт SSH, который веб-интерфейс показывает в URL для клонирования, задаёт параметр `gitlab_rails['gitlab_shell_ssh_port']`, описанный в разделе «Порт SSH».

Откройте опубликованные порты на хосте для сетей пользователей; команды для `firewalld` и `ufw` приведены в разделе «Настройка после установки». Docker публикует порты собственными правилами `iptables`, поэтому после каждого изменения правил файрвола проверяйте порты с рабочей станции:

```
curl -sI http://<HOSTNAME>/ | head -n 1
ssh -T -p 22 git@<HOSTNAME>
```

Расширенный поиск (OpenSearch)

Подключение к OpenSearch — параметр `gitlab_rails['fe_search']` и правило маршрутизации Sidekiq, описанные в разделе «Расширенный поиск (OpenSearch)». Добавьте их в `/etc/gitlab/gitlab.rb` в томе с конфигурацией или в `GITLAB_OMNIBUS_CONFIG`, затем перезапустите контейнер или выполните `docker exec code gitlab-ctl reconfigure`.

Диагностика

Большинство сбоев инстанса в контейнере проявляется при запуске, а причина видна в выводе контейнера или в логах на томе логов. В каждом разделе ниже приведены симптом, причина и порядок действий.

Веб-интерфейс отвечает 502 после запуска

Причина. при каждом запуске выполняется `gitlab-ctl reconfigure`, и приложение принимает запросы только после него. Первый запуск занимает несколько минут, последующие — одну-две минуты.

Решение. следите за запуском по выводу контейнера и дождитесь окончания настройки:

```
docker logs -f code
docker exec code gitlab-ctl status
```

Если 502 держится дольше, приложение не запускается: прочитайте его лог и проверьте свободную память на хосте.

```
docker exec code gitlab-ctl tail puma
```

Состояние контейнера остаётся unhealthy

Причина. проверка состояния выполняется каждые 60 секунд и после пяти неуспешных проверок подряд переводит контейнер в состояние `unhealthy`; первый запуск доходит до этого состояния, пока настройка ещё выполняется. Если состояние `unhealthy` сохраняется и после неё, сервисы не запустились.

Решение. выполните ту же проверку вручную и посмотрите состояние сервисов и лог настройки:

```
docker exec code gitlab-healthcheck --fail --max-time 10
docker exec code gitlab-ctl status
docker exec code ls /var/log/gitlab/reconfigure
```

Сервисы не запускаются после копирования или восстановления томов

Причина. владелец и права файлов на томах не соответствуют учётным записям внутри образа, которые созданы с фиксированными идентификаторами.

Решение. восстановите права командой из образа и перезапустите контейнер:

```
docker exec -it code update-permissions
docker restart code
```

Обновление базы данных завершилось с ошибкой, контейнер остановился

Где проявляется. в выводе контейнера есть строка `Upgrading the existing database failed and was reverted.`

Причина. при запуске выполняется `gitlab-ctl pg-upgrade`, обновление завершилось с ошибкой, и база данных возвращена к предыдущей версии.

Решение. создайте контейнер заново, добавив `-e GITLAB_SKIP_PG_UPGRADE=true` к команде из раздела «Быстрый старт». Инстанс запустится на текущей версии PostgreSQL, а обновление базы данных выполняется по разделу «Обновление».

Инстанс не отвечает по HTTPS

Причина. образ объявляет порт 443, но порт доступен только тогда, когда контейнер его публикует, а порты задаются при создании контейнера.

Решение. удалите контейнер и создайте его заново командой из раздела «Быстрый старт» с параметром `-p 443:443` и теми же томами; данные остаются на томах:

```
docker stop code
docker rm code
```

В ссылках и URL для клонирования указан идентификатор контейнера

Причина. адрес инстанса не задан, и образ строит его из имени хоста контейнера, которое Docker задаёт равным идентификатору контейнера.

Решение. задайте `external_url` в `/etc/gitlab/gitlab.rb` и перезапустите контейнер либо создайте контейнер заново с адресом в `GITLAB_OMNIBUS_CONFIG`, как описано в разделе «Быстрый старт».

Контейнер останавливается на проверке версии

Где проявляется. в выводе контейнера указаны найденная версия данных и версия, на которую нужно обновиться сначала.

Причина. при запуске версия на томе данных сравнивается с версией образа, и запуск прекращается, если перейти к ней за один шаг нельзя.

Решение. запустите сначала тег промежуточной версии, затем целевой, как описано в разделе «Обновление».

Изменение конфигурации не действует

Где проявляется. в выводе контейнера есть строка `Skipped reconfigure because GITLAB_SKIP_RECONFIGURE is set.`

Причина. переменной `GITLAB_SKIP_RECONFIGURE` задано значение `true`, и запуск оставляет конфигурацию прежней. Тот же симптом без этой строки означает, что настройка задана в `GITLAB_OMNIBUS_CONFIG` и переопределена файлом `/etc/gitlab/gitlab.rb`.

Решение. выполните настройку вручную либо создайте контейнер без этой переменной:

```
docker exec code gitlab-ctl reconfigure
```

Git по SSH отклоняет подключение

Причина. порт 22 контейнера не опубликован, либо переменной `GITLAB_DISABLE_OPENSSH` задано значение `true` и сервис `sshd` не подготовлен, либо клиент подключается к порту хоста, который не указан в URL для клонирования.

Решение. проверьте опубликованные порты и переменные контейнера, затем задайте порт SSH, как описано в разделе «Настройка после установки»:

```
docker port code
docker inspect --format '{{.Config.Env}}' code
```

Если клиент сообщает о смене ключа хоста после перезапуска, контейнер работает с новым томом конфигурации: ключи хоста сгенерированы заново.

Общая диагностика

Эти команды показывают состояние работающего контейнера:

```
docker logs -f code
docker exec code gitlab-ctl status
docker exec code gitlab-ctl tail <SERVICE>
docker exec code gitlab-rake gitlab:check SANITIZE=true
docker inspect --format '{{.State.Health.Status}}' code
```

Логи всех сервисов лежат на томе логов, на хосте это `/srv/code/logs`, а лог каждого запуска настройки — в `/var/log/gitlab/reconfigure`.

Helm-чарт

Требования

Helm-чарт развёртывает Deckhouse Code напрямую в кластере Kubernetes, без модуля Deckhouse Kubernetes Platform. Кластер и перечисленные ниже сервисы должны существовать до установки чарта.

Кластер Kubernetes

Кластер работает на Kubernetes версии `<VALUE>` или новее.

Helm

Чарт устанавливается с помощью Helm 3.

Ingress-контроллер

В кластере уже запущен Ingress-контроллер. Чарт создаёт объекты Ingress для веб-интерфейса, а также для необязательных компонентов реестра контейнеров и Pages; каждый из них использует класс Ingress, указанный в `global.ingress.class`.

StorageClass по умолчанию

StorageClass по умолчанию, либо StorageClass, указанный явно в `gitaly.persistence.storageClass`, чтобы кластер мог создать PersistentVolumeClaim, который каждый под Gitaly использует для хранения Git-репозиториев.

Внешний PostgreSQL

Внешний сервер PostgreSQL, адрес которого указан в `global.psql.host`, хранит данные приложения Deckhouse Code. Чарт не разворачивает базу данных.

Внешний Redis

Внешний сервер Redis, адрес которого указан в `global.redis.host`, используется для очередей фоновых задач, кеша и хранения сессий. Чарт не разворачивает Redis.

Объектное хранилище

S3-совместимый бакет объектного хранилища, настроенный через `global.appConfig.object_store`, хранит артефакты CI/CD, загруженные файлы и архивы резервных копий. Чарт не разворачивает объектное хранилище.

DNS-имя и TLS-сертификат

DNS-имя инстанса, указанное в `global.hosts.domain`, которое разрешается в адрес Ingress-контроллера. TLS-сертификат для этого имени, переданный как Kubernetes Secret, указанный в `global.ingress.tls.secretName`, независимо от того, выпущен он вручную или центром сертификации, уже работающим в кластере.

Быстрый старт

Перед началом работы выполните проверки из раздела Требования. Шаги ниже разворачивают Deckhouse Code с минимальным набором значений, необходимым для любой установки.

Получение чарта

Добавьте репозиторий чарта и обновите его:

```
helm repo add deckhouse-code <CHART_REGISTRY>
helm repo update
```

Подготовка файла values.yaml

Создайте файл `values.yaml` с доменом, классом Ingress, Secret с TLS-сертификатом, StorageClass для хранения Git-репозиториев и подключениями к внешним PostgreSQL, Redis и объектному хранилищу:

```
global:
  hosts:
    domain: code.example.com
  ingress:
    class: <INGRESS_CLASS_NAME>
    tls:
      secretName: <TLS_SECRET_NAME>
  psql:
    host: <POSTGRESQL_HOST>
    port: 5432
    database: <POSTGRESQL_DATABASE>
    username: <POSTGRESQL_USERNAME>
    password:
      secretName: <POSTGRESQL_PASSWORD_SECRET>
      key: password
  redis:
    host: <REDIS_HOST>
    port: 6379
    password:
      secretName: <REDIS_PASSWORD_SECRET>
      key: password
  appConfig:
    object_store:
      enabled: true
      connection:
        secretName: <OBJECT_STORE_CONNECTION_SECRET>
gitaly:
  persistence:
    size: <VALUE>
    storageClass: <STORAGE_CLASS_NAME>
```

Замените каждый плейсхолдер реальными именами класса, Secret и серверов. Раздел Настройка называет каждое из этих значений и поясняет, что оно настраивает.

Установка релиза

```
helm upgrade --install code deckhouse-code \
  -n code --create-namespace \
  -f values.yaml
```

Команда создаёт пространство имён `code`, если его ещё нет, и устанавливает релиз с именем `code`.

Настройка DNS-записи на Ingress

Получите адрес, который Ingress-контроллер присвоил объектам Ingress релиза:

```
d8 k -n code get ingress
```

Настройте DNS-имя из `global.hosts.domain` на адрес из колонки `ADDRESS` — в виде записи A, AAAA или CNAME, в зависимости от того, какой адрес вернул Ingress-контроллер.

Ожидание готовности релиза

Перед запуском подов, которые зависят от схемы базы данных, один раз выполняется задача миграций. Дождитесь, пока все поды перейдут в состояние `Running`:

```
d8 k -n code get pods
```

Первый вход в систему

Чарт создаёт Secret с начальным паролем учётной записи `root`, названный по имени релиза (`code-initial-root-password`, по принятому в чарте соглашению об именовании). Получите пароль:

```
d8 k -n code get secret code-initial-root-password -o jsonpath='{.data.password}' |
base64 -d
```

Откройте `https://code.example.com` (домен из `global.hosts.domain`) и войдите под именем `root` с этим паролем.

Проверка установки

Убедитесь, что страница входа открывается по HTTPS с сертификатом из `global.ingress.tls.secretName`, и что все поды и задача миграций находятся в исправном состоянии:

```
d8 k -n code get pods,jobs
```

Настройка

Значения ниже настраивают Helm-релиз Deckhouse Code сверх минимальной установки из раздела Быстрый старт. Каждое имя значения здесь совпадает с именем в `values.yaml` чарта; раздел Расширенная настройка перечисляет все значения, которые принимает чарт.

Адреса и Ingress

`global.hosts.domain` задаёт базовый домен инстанса. `global.ingress.class` выбирает Ingress-контроллер, который его обслуживает. Чарт создаёт объект Ingress для веб-интерфейса на этом домене и ещё один для каждого включённого необязательного компонента, например реестра контейнеров или Pages.

```
global:
  hosts:
    domain: code.example.com
  ingress:
    class: <INGRESS_CLASS_NAME>
```

TLS

`global.ingress.tls.secretName` задаёт имя Kubernetes Secret с сертификатом и закрытым ключом для `global.hosts.domain`. Каждый объект Ingress, который создаёт чарт, ссылается на этот Secret.

```
global:
  ingress:
    tls:
      secretName: <TLS_SECRET_NAME>
```

Secret должен существовать в пространстве имён релиза до выполнения `helm upgrade`; чарт не создаёт и не продлевает его.

PostgreSQL

`global.psql.host` и `global.psql.port` задают адрес внешнего сервера PostgreSQL. `global.psql.database` и `global.psql.username` задают базу данных и роль, от имени которой подключается Deckhouse Code. `global.psql.password.secretName` и `global.psql.password.key` указывают на Secret с паролем.

```
global:
  psql:
    host: <POSTGRESQL_HOST>
    port: 5432
    database: <POSTGRESQL_DATABASE>
    username: <POSTGRESQL_USERNAME>
    password:
      secretName: <POSTGRESQL_PASSWORD_SECRET>
      key: password
```

К этому серверу подключается каждый компонент, который читает или записывает данные приложения; чарт не разворачивает PostgreSQL.

Redis

`global.redis.host` и `global.redis.port` задают адрес внешнего сервера Redis. `global.redis.password.secretName` и `global.redis.password.key` указывают на Secret с паролем, если сервер его требует.

```
global:
  redis:
    host: <REDIS_HOST>
    port: 6379
    password:
      secretName: <REDIS_PASSWORD_SECRET>
      key: password
```

Redis хранит очереди фоновых задач, кеш и сессии; чарт не разворачивает Redis.

Объектное хранилище

`global.appConfig.object_store.enabled` включает S3-совместимое объектное хранилище для артефактов CI/CD, загруженных файлов и архивов резервных копий. `global.appConfig.object_store.connection.secretName` задаёт имя Secret с адресом, регионом и ключами доступа.

```
global:
  appConfig:
    object_store:
      enabled: true
      connection:
        secretName: <OBJECT_STORE_CONNECTION_SECRET>
```

Хранилище Git (Gitaly persistence)

`gitaly.persistence.size` задаёт размер PersistentVolumeClaim, который монтирует каждый под Gitaly для хранения Git-репозиториев. `gitaly.persistence.storageClass` выбирает StorageClass, из которого он создаётся.

```
gitaly:
  persistence:
    size: <VALUE>
    storageClass: <STORAGE_CLASS_NAME>
```

Реестр контейнеров

`registry.enabled` включает компонент реестра контейнеров. `registry.storage.bucket` задаёт бакет объектного хранилища, в котором хранятся слои образов, используя Secret подключения из `global.appConfig.object_store.connection.secretName`.

```
registry:
  enabled: true
  storage:
    bucket: <REGISTRY_BUCKET_NAME>
```

Pages

`pages.enabled` включает компонент Pages. `pages.storage.bucket` задаёт бакет объектного хранилища, в котором хранятся опубликованные сайты.

```
pages:
  enabled: false
  storage:
    bucket: <PAGES_BUCKET_NAME>
```

SMTP

`global.smtp.enabled` включает отправку уведомлений по электронной почте.
`global.smtp.address` и `global.smtp.port` задают адрес сервера SMTP.
`global.smtp.credentialsSecret` задаёт имя Secret с именем пользователя и паролем.

```
global:
  smtp:
    enabled: true
    address: <SMTP_HOST>
    port: 587
    credentialsSecret: <SMTP_CREDENTIALS_SECRET>
```

Расширенная настройка

В таблицах ниже перечислены все значения чарта, на которые ссылаются разделы Быстрый старт и Настройка, по одной таблице на группу ключей. Имена значений совпадают с именами в `values.yaml` чарта. Значения в колонке «Значение по умолчанию» — заглушки до выхода чарта.

Адреса и Ingress

Ключ	Тип	Значение по умолчанию	Описание
`global.hosts.domain`	строка	—	Базовый домен инстанса и его объектов Ingress.
`global.ingress.class`	строка	—	Класс Ingress, который используют объекты Ingress чарта.

TLS

Ключ	Тип	Значение по умолчанию	Описание
`global.ingress.tls.secretName`	строка	—	Имя Secret с TLS-сертификатом и ключом для `global.hosts.domain`.

PostgreSQL

Ключ	Тип	Значение по умолчанию	Описание
`global.psql.host`	строка	—	Адрес внешнего сервера PostgreSQL.
`global.psql.port`	целое число	`<VALUE>`	Порт внешнего сервера PostgreSQL.
`global.psql.database`	строка	—	Имя базы данных, которую использует Deckhouse Code.
`global.psql.username`	строка	—	Роль PostgreSQL, от имени которой подключается Deckhouse Code.
`global.psql.password.secretName`	строка	—	Имя Secret с паролем PostgreSQL.
`global.psql.password.key`	строка	`password`	Ключ внутри этого Secret, в котором хранится пароль.

Redis

Ключ	Тип	Значение по умолчанию	Описание
`global.redis.host`	строка	—	Адрес внешнего сервера Redis.
`global.redis.port`	целое число	`<VALUE>`	Порт внешнего сервера Redis.
`global.redis.password.secretName`	строка	—	Имя Secret с паролем Redis, если сервер его требует.
`global.redis.password.key`	строка	`password`	Ключ внутри этого Secret, в котором хранится пароль.

Объектное хранилище

Ключ	Тип	Значение по умолчанию	Описание
`global.appConfig.object_store.enabled`	булево	`<VALUE>`	Хранит ли Deckhouse Code артефакты CI/CD, загруженные файлы и архивы резервных копий в S3-совместимом объектном хранилище.
`global.appConfig.object_store.connection.secretName`	строка	—	Имя Secret с адресом, регионом и ключами доступа объектного хранилища.

Хранилище Git (Gitaly persistence)

Ключ	Тип	Значение по умолчанию	Описание
`gitaly.persistence.size`	строка	`<VALUE>`	Размер PersistentVolumeClaim, который монтирует каждый под Gitaly для хранения Git-репозиториев.
`gitaly.persistence.storageClass`	строка	—	StorageClass, из которого создаются PersistentVolumeClaim для Gitaly.

Реестр контейнеров

Ключ	Тип	Значение по умолчанию	Описание
`registry.enabled`	булево	`<VALUE>`	Разворачивает ли чарт компонент реестра контейнеров.
`registry.storage.bucket`	строка	—	Бакет объектного хранилища, в котором реестр контейнеров хранит слои образов.

Pages

Ключ	Тип	Значение по умолчанию	Описание
`pages.enabled`	булево	`<VALUE>`	Разворачивает ли чарт компонент Pages.
`pages.storage.bucket`	строка	—	Бакет объектного хранилища, в котором хранится содержимое Pages.

SMTP

Ключ	Тип	Значение по умолчанию	Описание
`global.smtp.enabled`	булево	`<VALUE>`	Отправляются ли уведомления по электронной почте через SMTP.
`global.smtp.address`	строка	—	Адрес сервера SMTP.
`global.smtp.port`	целое число	`<VALUE>`	Порт сервера SMTP.
`global.smtp.credentialsSecret`	строка	—	Имя Secret с именем пользователя и паролем SMTP.

Диагностика

Каждый раздел ниже описывает один симптом, его причину и действие, которое его устраняет.

Поды остаются в состоянии Pending

Команда `d8 k -n code get pods` показывает один или несколько подов в состоянии `Pending`.

Причина. Ни один StorageClass не подходит под `gitaly.persistence.storageClass` (либо в кластере нет StorageClass по умолчанию), либо ни у одного узла не хватает CPU и памяти для запросов ресурсов пода.

Решение. Прочитайте события пода:

```
d8 k -n code describe pod <POD_NAME>
```

Событие `FailedScheduling` с упоминанием PersistentVolumeClaim указывает на StorageClass; проверьте его командой `d8 k get storageclass`. Событие `FailedScheduling` с упоминанием CPU или памяти указывает на ресурсы узла.

У Ingress нет адреса

Команда `d8 k -n code get ingress` не показывает значение в колонке `ADDRESS`.

Причина. `global.ingress.class` не совпадает ни с одним Ingress-контроллером, установленным в кластере, либо у Service самого контроллера ещё нет внешнего адреса.

Решение. Убедитесь, что Ingress-контроллер запущен и что имя его класса совпадает с `global.ingress.class`. Если Service контроллера имеет тип `LoadBalancer`, проверьте, что инфраструктура выделила ему адрес.

TLS-сертификат не выпущен

При открытии `global.hosts.domain` браузер сообщает, что сертификат недействителен или самоподписан.

Причина. Secret, указанный в `global.ingress.tls.secretName`, не существует в пространстве имён `code`, либо не содержит сертификат, действительный для `global.hosts.domain`.

Решение. Проверьте Secret и содержащийся в нём сертификат:

```
d8 k -n code get secret <TLS_SECRET_NAME> -o yaml
```

Убедитесь, что субъект сертификата и альтернативные имена (Subject Alternative Names) покрывают `global.hosts.domain`.

Отказ в подключении к базе данных

Поды перезапускаются, а в их логах видно сообщение `PG::ConnectionBad: could not connect to server: Connection refused`.

Причина. `global.psql.host` или `global.psql.port` не позволяют подключиться к внешнему серверу PostgreSQL, либо сетевые политики блокируют подключение из пространства имён релиза.

Решение. Проверьте доступность сервера из кластера и убедитесь, что база данных из `global.psql.database` и роль из `global.psql.username` существуют на сервере.

Учётные данные объектного хранилища отклонены

Загрузка файлов и артефактов завершается ошибкой, а в логах подов виден отказ в доступе от объектного хранилища.

Причина. Secret, указанный в `global.appConfig.object_store.connection.secretName`, содержит устаревший адрес, регион или ключ доступа, либо бакет не существует.

Решение. Сверьте содержимое Secret с консолью провайдера объектного хранилища и убедитесь, что бакет существует, а у учётных данных есть право на запись в него.

Задача миграций завершается ошибкой

Под задачи миграций завершается с ненулевым кодом, а поды, зависящие от схемы базы данных, остаются в состоянии `Pending` или перезапускаются по кругу.

Причина. PostgreSQL ещё не доступен, учётные данные в `global.psql.password.secretName` неверны, либо релиз пропустил версию Deckhouse Code, от которой зависит более поздняя миграция.

Решение. Прочитайте лог задачи:

```
d8 k -n code logs job/<MIGRATIONS_JOB_NAME>
```

Сначала устраните проблему с подключением к базе данных, затем сверьтесь с порядком версий в разделе Обновление, если причина — пропущенная версия.

Helm upgrade завершается по тайм-ауту

Команда `helm upgrade` завершается с сообщением `Error: UPGRADE FAILED: timed out waiting for the condition`.

Причина. Под или задача миграций не перешли в готовое или завершённое состояние за время ожидания Helm — обычно из-за одной из причин выше.

Решение. Проверьте поды и задачи релиза:

```
d8 k -n code get pods,jobs
```

Устраните причину, затем выполните `helm upgrade` повторно, добавив `--timeout`, если релизу действительно требуется больше времени.

Модуль Deckhouse Kubernetes Platform

Модуль Deckhouse Kubernetes Platform поставляет Deckhouse Code через компонент `code-operator`, который устанавливает и обслуживает инстанс Deckhouse Code. Каждая настройка инстанса — это поле ресурса CodeInstance; для пустого поля оператор берёт глобальную настройку платформы: шаблон публичных доменов, IngressClass или режим HTTPS. В собственном ModuleConfig модуля задаётся только параметр `logLevel`.

Веб-интерфейс модульной поставки доступен на поддомене `code` шаблона публичных доменов платформы, например `https://code.example.com`. Если в CodeInstance задан `spec.network.web.hostname`, используется указанный в нём адрес.

Требования

Для работы модуля необходимы:

- Deckhouse Kubernetes Platform версии 1.68 или новее;
- Kubernetes версии 1.29 или новее;
- включённый модуль `cert-manager`;
- PostgreSQL версии 17 или новее;

- Redis версии 7.0 или новее;
- S3-хранилище с провайдером `YCloud` или `Generic`;
- StorageClass для Git-репозиториев, указанный в поле `gitData.storageClass`;
- IngressClass для трафика к веб-интерфейсу.

Ресурсы кластера для размера инстанса приведены в разделе «Минимальные требования» документации модуля, а настройка PostgreSQL, Redis и S3-хранилища описана в разделе «Начало работы».

Установка

Включение модуля устанавливает `code-operator`; когда PostgreSQL, Redis и S3-хранилище готовы, применение ресурса CodeInstance запускает установку Deckhouse Code с заданными параметрами. Шаги установки — от включения модуля до первого входа — приведены в разделе «Начало работы» документации модуля.

> Отключение модуля удаляет конфигурацию Code, данные Git и все Secret и ConfigMap, принадлежащие оператору, включая ключи шифрования данных в PostgreSQL. Перед отключением модуля создайте резервную копию.

Настройка

Каждая настройка модульной поставки — это поле ресурса CodeInstance. Полный список полей приведён в разделе Custom Resources документации модуля.

Эксплуатация

Резервное копирование, масштабирование и настройка сети модульной поставки — это поля CodeInstance, которые описаны в документации модуля: «Обслуживание», «Масштабирование» и «Сеть». Обновление модуля выполняет платформа; резервная копия перед обновлением описана на странице «Обновление». Перенос инстанса между пакетом для ОС Linux или Omnibus Docker и модулем описан в разделе «Миграция».

Администрирование

Администратор модульной поставки использует раздел «Обзор» и остальные страницы раздела «Администрирование» для задач уровня инстанса: пользователи и доступ, аутентификация, безопасность, мониторинг, резервное копирование и обновление.

Администрирование

Обзор

Раздел «Администрирование» описывает настройки уровня инстанса Deckhouse Code: контроль доступа, аутентификацию, безопасность, мониторинг, режим обслуживания, резервное копирование и обновление.

Страницы, которые отличаются по типу установки

Deckhouse Code поставляется в виде пакета для ОС Linux, Omnibus Docker, Helm-чарта или модуля Deckhouse Kubernetes Platform. В разделе «Мониторинг и лимиты» встроенный мониторинг каждого типа установки вынесен на отдельную вкладку. «Резервное копирование и восстановление» и «Обновление» — единые страницы: сначала общая для всех типов установки часть, затем каждый шаг с различающимися командами во вкладках по типу установки.

Остальные страницы

- Пользователи и доступ — ограничение создания групп и проектов и управление доступом на уровне инстанса.
- Аутентификация — настройка входа через провайдеров OmniAuth и LDAP.
- Настройки безопасности — настройки безопасности инстанса, включая ограничения регистрации и двухфакторную аутентификацию администраторов.
- Ключи доступа — просмотр и отзыв токенов, ключей SSH и GPG, выданных в рамках инстанса.
- Сервисные аккаунты — создание сервисных аккаунтов для CI/CD-пайплайнов и интеграций.
- События аудита безопасности — просмотр событий аудита, зафиксированных для инстанса.
- Режим обслуживания — снижение количества операций записи на время регламентных работ.
- Расширенный поиск — управление полнотекстовой индексацией после подключения OpenSearch.

Операции уровня кластера для модуля Deckhouse Kubernetes Platform — расчёт размера, настройка сети, миграция с пакета для ОС Linux или Omnibus Docker в модуль и обратно — описаны в документации модуля.

Пользователи и доступ

Ограничение создания групп и проектов

1. Перейдите в «Admin» → «Настройки» → «Основные».

2. Разверните раздел «Видимость и контроль доступа» и в поле «Минимальная роль по умолчанию, необходимая для создания проектов» укажите минимальную роль, которой разрешено создавать проекты в инстансе.
3. Разверните раздел «Аккаунт и ограничения». В группе «Ограничения пользователя» снимите флажки «Разрешить новым пользователям создавать top level группы» и «Разрешить пользователям с ролью гостя создавать группы и личные проекты».
4. Нажмите «Сохранить изменения» в каждом изменённом разделе.

Флажок «Разрешить новым пользователям создавать top level группы» задаёт значение по умолчанию для учётных записей, созданных после изменения. Для уже существующей учётной записи разрешение задаётся в разделе «Admin» → «Обзор» → «Пользователи»: откройте пользователя, нажмите «Редактировать» и задайте значение «Может создавать группы верхнего уровня».

Включённые протоколы доступа к Git

Инстанс принимает операции Git по SSH и по HTTP(S). Чтобы оставить один протокол:

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните раздел «Видимость и контроль доступа».
3. В списке «Включенные протоколы доступа к Git» выберите «Only SSH» или «Only HTTP(S)».
4. Нажмите «Сохранить изменения».

Операции Git по оставшемуся за списком протоколу отклоняются.

Аутентификация по паролю для Git через HTTPS

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните раздел «Ограничения входа».
3. Снимите флажок «Разрешить аутентификацию по паролю для Git через HTTP(S)».
4. Нажмите «Сохранить изменения».

После этого Git-клиент проходит аутентификацию по HTTPS с персональным токеном доступа вместо пароля учётной записи. При настроенном LDAP принимается также пароль LDAP.

Двухфакторная аутентификация для всего инстанса

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните раздел «Ограничения входа».
3. Установите флажок «Требовать включения двухфакторной аутентификации для всех пользователей», чтобы второй фактор требовался от каждого пользователя инстанса, или флажок «Требовать двухфакторную аутентификацию для администраторов», чтобы он требовался только от администраторов.
4. В поле «Период отсрочки двухфакторной аутентификации» укажите, сколько часов пользователь может откладывать настройку. Значение `0` требует настройки при следующем входе.

5. Нажмите «Сохранить изменения».

Пользователю, который не настроил второй фактор, настройка предлагается при следующем входе. Доступные пользователю методы описаны на странице «Вход в систему».

Правила отправки изменений на уровне инстанса

Правила, заданные в разделе «Admin» → «Настройки» → «Репозиторий» → «Push-правила», действуют на все проекты инстанса: группа и проект получают их как значения по умолчанию и меняют их, только пока правило разрешает переопределение. Сами правила и порядок переопределения описаны на странице «Правила отправки изменений».

Аутентификация

Обзор

Deckhouse Code выполняет вход пользователей через внешних провайдеров аутентификации (OmniAuth) и берёт членство в группах и права доступа из каталога LDAP. Поведение провайдеров, правила синхронизации и решения о предоставлении и отзыве доступа одинаковы для всех типов установки; различается место, где записываются параметры:

- пакет для ОС Linux — файл `/etc/gitlab/gitlab.rb`;
- Omnibus Docker — переменная `GITLAB_OMNIBUS_CONFIG` или файл `/etc/gitlab/gitlab.rb` на томе конфигурации;
- Helm-чарт — значения чарта;
- модуль Deckhouse Kubernetes Platform — ресурс `CodeInstance`.

Ключи и манифесты для каждого типа приведены на странице «Настройка».

Параметры ниже названы так, как их читает продукт. Ключ, под которым записывается каждый из них, указан на странице вашего типа установки.

Ограничения входа и аутентификация по паролю

Вход локальных учётных записей настраивается в разделе «Admin» → «Настройки» → «Основные» → «Ограничения входа». Флажок «Разрешить аутентификацию по паролю через веб-интерфейс» показывает, входит ли учётная запись с паролем в веб-интерфейс наряду с внешними провайдерами; в интерфейсе флажок доступен только для чтения.

Чтобы включить аутентификацию по паролю для веб-интерфейса, откройте Rails-консоль:

Пакет для ОС Linux

```
sudo gitlab-rails console
```

Omnibus Docker

```
docker exec -it code gitlab-rails console
```

Helm-чарт

```
d8 k -n code exec -it deploy/code-toolbox -- gitlab-rails console -e production
```

Модуль Deckhouse Kubernetes Platform

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails console -e
production
```

Выполните в ней команду:

```
Gitlab::CurrentSettings.update!(password_authentication_enabled_for_web: true)
```

Аутентификация по паролю для Git через HTTPS и двухфакторная аутентификация для всего инстанса задаются в том же разделе административной панели и описаны на странице «Пользователи и доступ».

Конфигурация OmniAuth

Общие параметры OmniAuth действуют для всех провайдеров; OpenID Connect и SAML добавляют собственные параметры.

Поддерживаемые провайдеры

В параметре `name` записи провайдера указывается одно из следующих значений:

- `openid_connect` — OpenID Connect (описан ниже);
- `saml` — SAML (описан ниже);
- `oauth2_generic` — произвольный провайдер OAuth 2.0;
- `jwt` — аутентификация по JWT;
- `github` — GitHub;
- `gitlab` — GitLab.com;
- `google_oauth2` — Google;
- `azure_activedirectory_v2` — Microsoft Entra ID (Azure AD);

- `atlassian_oauth2` — Atlassian;
- `crowd` — Atlassian Crowd;
- `auth0` — Auth0;
- `alicloud` — AliCloud;
- `salesforce` — Salesforce;
- `shibboleth` — Shibboleth.

Вход через LDAP настраивается отдельно, в секции LDAP-серверов (подробнее — в разделе «Синхронизация с LDAP»).

Общие параметры OmniAuth

Следующие параметры действуют для всех провайдеров:

- `enabled` — разрешает вход через внешних провайдеров. По умолчанию — `true`.
- `providers` — список провайдеров, через которых разрешён вход. По умолчанию — `[]`.
- `allow_single_sign_on` — список провайдеров, для которых при первом входе автоматически создаётся учётная запись (например, `['openid_connect']`). Также принимает значения `true` (все провайдеры) и `false`. Если автоматическое создание отключено, пользователь должен сначала получить учётную запись в Deckhouse Code, а затем связать её с провайдером. По умолчанию — `false`.
- `block_auto_created_users` — если `true`, автоматически созданные учётные записи блокируются до одобрения администратором. По умолчанию — `true`.
- `auto_link_ldap_user` — связывает учётную запись с учётной записью LDAP при первом входе (подробнее — в разделе «Связывание учётных записей OIDC с LDAP»). По умолчанию — `false`.
- `auto_link_user` — связывает вход через провайдера с существующей учётной записью Deckhouse Code по адресу электронной почты. Принимает список провайдеров или значения `true` и `false`. По умолчанию — `false`.
- `auto_sign_in_with_provider` — имя провайдера, на страницу входа которого пользователь перенаправляется автоматически, минуя страницу входа Deckhouse Code. По умолчанию — `false`.
- `external_providers` — список провайдеров, учётные записи которых создаются как внешние. По умолчанию — `[]`.
- `allow_bypass_two_factor` — список провайдеров, вход через которых не требует двухфакторной аутентификации. Также принимает значения `true` и `false`. По умолчанию — `false`.
- `sync_profile_from_provider` — список провайдеров, из данных которых обновляется профиль пользователя при каждом входе. Также принимает значения `true` и `false`. По умолчанию — `false`.
- `sync_profile_attributes` — список атрибутов профиля, которые обновляются при синхронизации: `name`, `email`, `location`. Синхронизируемые атрибуты становятся доступными только для чтения. По умолчанию — `['email']`.

OpenID Connect (OIDC)

Провайдеры перечисляются в параметре `providers`. Для провайдера OIDC доступны следующие параметры:

- `name` — тип провайдера. Для OIDC — всегда `'openid_connect'`.
- `label` — подпись кнопки входа. По умолчанию — `'Openid Connect'`.
- `icon` — адрес изображения, которое будет показано на кнопке входа.
- `args` — параметры подключения к провайдеру:
  - `name` — имя стратегии OmniAuth, совпадает со значением параметра `name` провайдера;
  - `scope` — список запрашиваемых областей доступа, например `['openid', 'profile', 'email']`;
  - `response_type` — тип ответа OAuth 2.0. Для потока Authorization Code — `'code'`;
  - `issuer` — адрес OIDC-провайдера;
  - `discovery` — если `true`, настройки провайдера запрашиваются автоматически по адресу `<issuer>/.well-known/openid-configuration`;
  - `client_auth_method` — способ аутентификации клиента на token endpoint: `'basic'` или `'query'`;
  - `uid_field` — поле из данных пользователя, которое используется как `uid` учётной записи (например, `preferred_username`). Если параметр не задан или поле отсутствует, используется поле `sub`;
  - `send_scope_to_token_endpoint` — передавать ли параметр `scope` в запросах к token endpoint. Задайте `false`, если провайдер не принимает этот параметр. По умолчанию — `true`;
  - `pkce` — включает Proof Key for Code Exchange (PKCE);
  - `client_options`:
    - `identifier` — идентификатор клиента, зарегистрированного у провайдера;
    - `secret` — секрет клиента;
    - `redirect_uri` — адрес установки Deckhouse Code с путём `/users/auth/openid_connect/callback`. Тот же адрес должен быть указан в настройках клиента на стороне провайдера.

Дополнительно Deckhouse Code поддерживает следующие параметры. Они указываются на верхнем уровне записи провайдера, рядом с параметром `name`:

- `allowed_groups` — список групп, пользователям которых разрешён вход. Пользователи вне этих групп будут заблокированы. По умолчанию — `null` (разрешены все группы).
- `admin_groups` — список групп, пользователи которых получают административные права. По умолчанию — `null` (права администратора не выдаются ни одной группе).
- `auditor_groups` — список групп, пользователи которых получают роль аудитора: доступ только на чтение ко всем группам и проектам, без доступа к административному разделу. По умолчанию — `null` (роль аудитора не выдаётся ни одной группе).

- `groups_attribute` — имя атрибута, из которого извлекаются группы пользователя. По умолчанию — `'groups'`.

> Параметры `admin_groups` и `auditor_groups` учитываются, только если задан параметр `allowed_groups`. Если пользователь входит и в `admin_groups`, и в `auditor_groups`, ему назначаются права администратора.

Примеры записи провайдера для каждого типа установки: «Провайдер OpenID Connect» и «Провайдер SAML».

SAML

Для провайдеров SAML доступны аналогичные параметры:

- `allowed_groups` — список групп с разрешённым входом. По умолчанию — `null` (разрешены все группы).
- `admin_groups` — группы с административными правами. По умолчанию — `null` (права администратора не выдаются ни одной группе).
- `auditor_groups` — группы, пользователи которых получают роль аудитора: доступ только на чтение ко всем группам и проектам. По умолчанию — `null` (роль аудитора не выдаётся ни одной группе).
- `groups_attribute` — имя атрибута, содержащего группы. По умолчанию — `'Groups'`.

> Если пользователь входит в `admin_groups`, но не указан в `allowed_groups`, доступ будет запрещён. В этом случае административные права также не будут назначены.

Синхронизация с LDAP

Deckhouse Code поддерживает синхронизацию пользователей, групп и прав доступа с LDAP-сервером. Синхронизация выполняется автоматически раз в час либо с периодичностью, заданной расписанием задачи `ldap_sync_worker`.

Примеры записи LDAP-сервера для каждого типа установки: «LDAP-серверы» и «Расписание синхронизации».

Ограничения на стороне LDAP-сервера

Во время синхронизации выполняются LDAP-запросы ко всем пользователям и группам, указанным в конфигурации. При необходимости используется постраничная загрузка (pagination). Если на стороне LDAP установлены ограничения на число возвращаемых объектов, это может привести к ошибкам синхронизации или удалению прав доступа у пользователей.

Несколько LDAP-серверов

В секции LDAP можно описать несколько серверов. Ключ записи — имя сервера; из него формируется имя провайдера `ldap<ключ>` в нижнем регистре. Например, ключу `main` соответствует провайдер `ldapmain`. Это имя хранится в идентификаторе учётной записи и попадает в поле `server` записей о синхронизации в логах.

У каждого сервера собственные параметры подключения. Параметры синхронизации — секция `group_sync` со всем её содержимым, включая `role_mapping`, — учитываются только у сервера под ключом `main`; у остальных серверов эта секция игнорируется (раздел «Область синхронизации»).

Вход работает через любой из описанных серверов: на странице входа и на странице входа в административный раздел появляется отдельная вкладка для каждого сервера. Отдельно включать это не требуется.

Серверы задействуются по-разному в зависимости от сценария:

Сценарий	Какие серверы используются
Вход через форму LDAP в веб-интерфейсе	Все серверы, отдельной вкладкой на каждый
Поиск учётной записи при связывании с OIDC (`auto_link_ldap_user`)	Все серверы, по очереди до первого совпадения
Синхронизация пользователей, групп и прав доступа	Только сервер `ldapmain`
Периодическая перепроверка доступа	`ldapmain`, если у учётной записи есть его идентификатор; иначе — каталоги её собственных идентификаторов

Область синхронизации

Синхронизация пользователей, групп и прав доступа охватывает только один сервер — тот, чьё имя провайдера равно `ldapmain`, то есть сервер под ключом `main`. Остальные серверы остаются источником входа, но не источником групп и прав. Ограничение действует одинаково и в задании синхронизации по расписанию, и при синхронизации, которая выполняется при входе пользователя.

> ⚠ Имя `ldapmain` фиксировано и не настраивается. Если сервера под ключом `main` в секции нет, синхронизация не выполняется ни для одного из описанных серверов.

Следствия:

- пользователь, заведённый только в неглавном каталоге, входит в Deckhouse Code, но групп и прав из LDAP не получает — их назначают вручную;

- у пользователя с двумя идентификаторами (одинаковый основной адрес электронной почты в обоих каталогах) группы и права всегда приходят из `ldapmain`, независимо от того, через какой сервер он вошёл;
- секция `group_sync` неглавных серверов не читается, поэтому ошибка в их параметре `group_sync.filter` не мешает запуску приложения. Ошибка в фильтре сервера `main` по-прежнему обнаруживается сразу при старте и не даёт приложению подняться.

Дойдя до неглавного сервера, синхронизация пропускает его и пишет об этом одну запись уровня `INFO` с именем пропущенного сервера в поле `server`:

```
Server is not synchronized: users, groups and permissions come from 'ldapmain' only
```

Идентификация пользователей при нескольких серверах

Пользователь ищется сначала по идентификатору учётной записи, затем по основному адресу электронной почты:

- если основной адрес в двух каталогах совпадает, при входе через второй сервер к существующей учётной записи добавляется второй LDAP-идентификатор; новая учётная запись не создаётся;
- если адреса разные, будут созданы две разные учётные записи.

Список идентификаторов учётной записи доступен в административном разделе на странице `/admin/users/<username>/identities`.

Решение о доступе при нескольких идентификаторах

Deckhouse Code периодически, не чаще одного раза в час, перепроверяет доступ пользователя LDAP: обращается в каталог и блокирует учётную запись, если запись в каталоге не найдена, либо разблокирует, если она нашлась снова. Если у учётной записи несколько LDAP-идентификаторов, каталог для проверки выбирается так:

- есть идентификатор `ldapmain` — решает только он, остальные каталоги не опрашиваются. Пользователь есть в `ldapmain` и не отключён там — доступ есть; его там нет или он отключён — доступа нет, что бы ни происходило в остальных каталогах;
- идентификатора `ldapmain` нет, то есть пользователь заведён только в неглавных каталогах, — он проверяется по своим идентификаторам, и доступ сохраняется, пока запись найдена хотя бы в одном из соответствующих каталогов.

«Запись найдена» означает, что она существует в каталоге и не отключена средствами Active Directory. Второе проверяется только для каталогов, сконфигурированных как AD.

Правило согласовано с областью синхронизации: синхронизация блокирует пользователей по данным `ldapmain`, и перепроверка доступа блокирует ровно тех же. Поэтому вход через другой каталог не снимает блокировку, поставленную синхронизацией.

> ⚠ При переводе пользователя из каталога `ldapmain` в другой каталог удалите у его учётной записи устаревший идентификатор `ldapmain` на странице `/admin/users/<username>/identities`. Пока этот идентификатор на месте, пользователь остаётся заблокированным, даже если он заведён в другом каталоге. Автоматически идентификатор не удаляется.

Удаление или отключение пользователя в неглавном каталоге само по себе доступ не закрывает, пока пользователь остаётся в `ldapmain`. Отзывать доступ нужно в каталоге `ldapmain`.

Группы и права доступа

LDAP-группы сопоставляются с группами GitLab. При этом можно назначать роли пользователям на основе имени группы.

Обязательные параметры:

- `group_sync.base` — DN, с которого начинается поиск LDAP-групп.

Опциональные параметры:

- `group_sync.create_groups` — если `true`, группы будут создаваться в Deckhouse Code.
- `group_sync.filter` — LDAP-фильтр для поиска групп.
- `group_sync.scope` — область поиска групп (0 — Base, 1 — SingleLevel, 2 — WholeSubtree).
- `group_sync.prefix` — определяет, из какого атрибута брать имя родительской группы. Если атрибут отсутствует — используется значение по умолчанию.
- `group_sync.top_level_group` — имя группы верхнего уровня, в которую будут добавлены все синхронизированные группы.
- `group_sync.name_mask` — регулярное выражение для извлечения имени группы из атрибута CN (Common Name).
- `group_sync.owner` — имя пользователя, который будет добавлен как владелец группы (по умолчанию — `root`).

Секция role_mapping

Назначает права пользователям на основе имени группы (`cn`):

- `role_mapping.by_name` — регулярное выражение; если имя группы совпадает, пользователю назначается соответствующая роль.
- `role_mapping.gitlab_role` — имя роли, доступной на инстансе.

Список доступных ролей берётся из штатного справочника ролей — того же, по которому определяется, какие роли можно назначить участнику группы или проекта. Справочник читается заново при каждой синхронизации, поэтому роль, отключённая на уровне инстанса, в него не попадает, и указать её в `role_mapping` нельзя.

В настоящее время справочник содержит семь имён:

`gitlab_role`	Роль	Уровень доступа
`guest`	Guest	10
`planner`	Planner	15
`reporter`	Reporter	20
`security_manager`	Security Manager	25
`developer`	Developer	30
`maintainer`	Maintainer	40
`owner`	Owner	50

Имя роли записывается так, как указано в колонке `gitlab_role`: в нижнем регистре, слова разделяются подчёркиванием. Регистр и пробелы не нормализуются, поэтому, например, `Security Manager` считается нераспознанным именем.

Роль `security_manager` присутствует в справочнике, только если она включена на инстансе (переменная окружения `GITLAB_SECURITY_MANAGER_ROLE`, по умолчанию включена). Если роль отключена, её имя считается нераспознанным.

Если под имя LDAP-группы подошло несколько правил, назначается численно наибольший уровень доступа. Например, если группа подошла и под правило с ролью `security_manager` (уровень 25), и под правило с ролью `developer` (уровень 30), её участники получат роль Developer.

Проверка имён ролей

Имена ролей проверяются один раз, в начале назначения ролей пользователям по их LDAP-группам (далее — назначение членств), включая правила, под которые при текущей синхронизации не подошла ни одна LDAP-группа. Это позволяет обнаруживать и скрытые опечатки, которые проявились бы только при появлении подходящей группы.

Проверка не прерывает назначение членств:

- LDAP-группы, все подошедшие правила которых распознаны, обрабатываются полностью и получают членства;
- нераспознанное правило не участвует в вычислении уровня доступа;
- если под LDAP-группу подошли только нераспознанные правила, уровень доступа для неё не определяется, и группа целиком пропускается при текущей синхронизации: её текущие участники не пересчитываются и не удаляются.

При этом синхронизация завершается ошибкой — уже после того, как группы, пользователи и корректные членства записаны. Следствия:

- синхронизация учитывается как аварийно завершённая на странице метрик задачи синхронизации, а отметка об успешной синхронизации не обновляется (раздел «Ручной запуск синхронизации»);
- в логи попадает запись уровня `ERROR`, в которой перечислены и нераспознанные имена, и полный список допустимых:

```
Unknown gitlab_role in group_sync.role_mapping: 'security_manger'. Available
roles: guest, planner, reporter, security_manager, developer, maintainer, owner
```

Назначение членств выполняется и при входе пользователя через LDAP-провайдера. На этом пути та же ошибка только записывается в логи: вход отрабатывает штатно, остальные членства проставляются.

Определение членов группы

> ⚠ LDAP Sync не поддерживает транзитивность для вложенных групп. Подробная информация в разделе «Вложенные группы и транзитивность».

Deckhouse Code поддерживает следующие атрибуты для определения членов группы (все значения — массив DN):

- `member`;
- `uniquemember`;
- `memberof`;
- `memberuid`;
- `submember`.

Синхронизация пользователей

Во время синхронизации обновляются имена и email-адреса пользователей, а также статус блокировки.

Опциональные параметры:

- `sync_name` — если `true`, имя пользователя будет обновлено по данным LDAP.

Блокировка пользователей по данным LDAP

Если пользователь удалён из LDAP, очередная плановая синхронизация заблокирует его учётную запись. Если пользователя вернули в LDAP, следующая синхронизация разблокирует учётную запись автоматически.

Блокировка выполняется всегда и не требует дополнительных настроек: источником истины для статуса учётной записи остаётся LDAP. Вход заблокированного пользователя через OIDC-провайдера завершится отказом, даже если в самом провайдере пользователь активен.

Если в вашем каталоге пользователей не удаляют, а блокируют через атрибут, исключите заблокированных пользователей из выдачи LDAP с помощью параметра `user_filter` сервера. Укажите один фильтр, соответствующий вашему каталогу:

Каталог	**Значение `user_filter`**
OpenLDAP с ppolicy	`(!(pwdAccountLockedTime=*))`
389-DS	`(!(nsAccountLock=TRUE))`
Active Directory	`(!(userAccountControl:1.2.840.113556.1.4.803:=2))`
Собственный атрибут	`(!(employeeType=blocked))`

Связывание учётных записей OIDC с LDAP

Если пользователи входят через OIDC-провайдера (например, Keycloak), а права выдаются по LDAP-группам, включите автоматическое связывание OIDC-учётной записи с учётной записью LDAP параметром `auto_link_ldap_user`.

Как ищется учётная запись LDAP

При первом входе через OIDC Deckhouse Code ищет пользователя в LDAP. Для поиска используются два значения из данных OIDC-провайдера:

- `uid` — значение поля, указанного в параметре `uid_field` провайдера (например, `preferred_username`);
- email — адрес электронной почты пользователя.

Настроенные LDAP-серверы опрашиваются по очереди. На каждом сервере выполняется до четырёх попыток поиска, до первого совпадения:

Искомое значение	**Атрибут для поиска**
`uid`	Атрибут, указанный в параметре `uid` LDAP-сервера (например, `cn`)
`uid`	Почтовые атрибуты: `mail`, `email`, `userPrincipalName`
email	Те же почтовые атрибуты
`uid`	DN — если значение `uid` само является DN

Если пользователь найден, к его учётной записи добавляется LDAP-идентификатор с найденным DN. Если ни одна попытка не дала результата, учётная запись создаётся без привязки к LDAP.

Таким образом, связывание сработает, только если `uid` или email из OIDC-провайдера совпадает со значением соответствующего атрибута в LDAP.

Первый и последующие входы

При первом успешном входе учётная запись связывается с LDAP. При последующих входах:

- LDAP не опрашивается — используется ранее установленная связь;
- доступ и административные права переоцениваются по группам из данных OIDC-провайдера (параметры `allowed_groups` и `admin_groups`). Если пользователя убрали из разрешённой группы, учётная запись будет заблокирована;
- членства в группах и проектах, а также роли в них при входе через OIDC не меняются — их обновляет фоновая синхронизация с LDAP по расписанию.

Поэтому сразу после первого входа пользователь может войти, но членств в группах и проектах у него ещё нет: они появятся после ближайшей синхронизации. Чтобы не ждать её, запустите синхронизацию вручную (подробнее — в разделе «Ручной запуск синхронизации»).

> Синхронизация групп и членств пользователя при входе выполняется только при входе через LDAP-провайдера (имя провайдера начинается с `ldap`). Вход через OIDC-провайдера её не запускает, даже если учётная запись уже связана с LDAP.

Вход пользователей, найденных в LDAP

Чтобы пользователи, найденные в LDAP, могли входить сразу, а остальные отправлялись на одобрение администратору, сочетайте два параметра:

- `block_auto_created_users` на уровне OmniAuth со значением `true` — пользователь, не найденный в LDAP, получает учётную запись, заблокированную до одобрения администратором;
- `block_auto_created_users` у LDAP-сервера `main` со значением `false` — пользователь, найденный в LDAP, входит сразу.

Особенности связывания

- Если пользователя блокируют переименованием `cn` в каталоге, при первом входе он всё равно может быть найден по email и связан с учётной записью LDAP. Для блокировки используйте удаление пользователя из каталога или параметр `user_filter` (подробнее — в разделе «Блокировка пользователей по данным LDAP»).
- Если пользователь не был связан с LDAP и его заблокировала задача `Ldap::BlockNonLdapUsersWorker`, автоматическая разблокировка не сработает. Такого пользователя нужно разблокировать вручную и связать с учётной записью LDAP.

Устранение проблем с синхронизацией

Если предыдущее задание синхронизации завершилось некорректно, Redis может сохранить блокировку на его выполнение (по умолчанию параметр `concurrency = 1`). Это помешает запуску нового задания.

Чтобы снять блокировку:

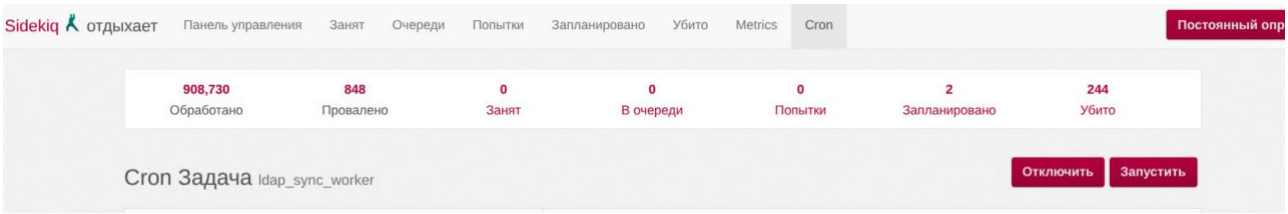
1. Подключитесь к Redis, используя базы, указанные в `config/redis.shared_state.yml` и `config/redis.queues.yml`.
2. Удалите ключ `sidekiq:concurrency_limit:throttled_jobs:{ldap/sync_worker}` следующими командами:

   ```
   keys *ldap*
   del "sidekiq:concurrency_limit:throttled_jobs:{ldap/sync_worker}"
   ```

Ручной запуск синхронизации

Чтобы синхронизировать группы сразу после их изменения на стороне LDAP, выполните следующие шаги:

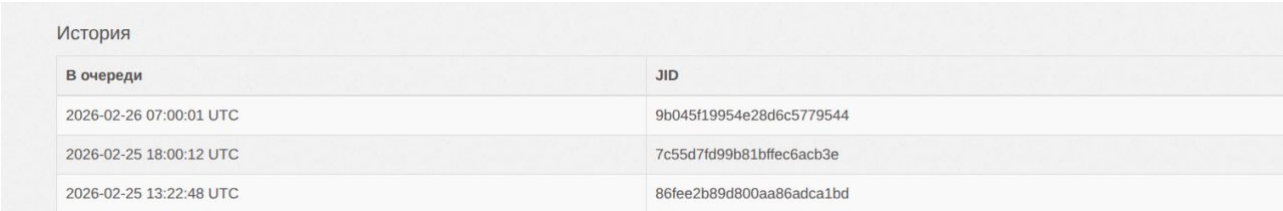
1. Зайдите на страницу worker’a синхронизации LDAP `/admin/sidekiq/cron/namespaces/default/jobs/ldap_sync_worker`.
2. В верхнем правом углу нажмите кнопку «Запустить» и подтвердите в диалоговом окне.



Чтобы посмотреть, как завершилась запущенная синхронизация, откройте страницу метрик задачи синхронизации LDAP: `/admin/sidekiq/metrics?substr=SyncWorker&period=8h`. На графике отображается статистика вызовов, в таблице ниже — количество успешно и аварийно завершённых вызовов синхронизации LDAP.

Чтобы посмотреть полные логи процесса синхронизации:

1. На странице worker’а `/admin/sidekiq/cron/namespaces/default/jobs/ldap_sync_worker` найдите таблицу событий запусков «История». Первая строка соответствует последнему запуску. Скопируйте значение в колонке JID (Job ID) — оно понадобится для поиска по логам.



2. Соберите записи логов Sidekiq для этого запуска, подставив скопированный JID:

Пакет для ОС Linux

```
sudo grep '<JID>' /var/log/gitlab/sidekiq/current
```

Omnibus Docker

```
docker exec code grep '<JID>' /var/log/gitlab/sidekiq/current
```

Helm-чарт

```
d8 k -n code logs -l app.kubernetes.io/component=sidekiq | jq 'select(.jid=="
<JID>")'
```

Модуль Deckhouse Kubernetes Platform

```
d8 k -n d8-code -l app.kubernetes.io/component=sidekiq get pod -o NAME
d8 k -n d8-code logs <POD_NAME> -c sidekiq | jq 'select(.jid=="<JID>")'
```

Со временем старые логи удаляются при ротации, и получить их будет нельзя. При необходимости запустите синхронизацию повторно и соберите актуальные логи.

Особенности работы LDAP Sync

Алгоритм синхронизации

LDAP Sync использует плоский алгоритм синхронизации:

1. Получение групп. Выполняется LDAP-запрос, который получает все группы по параметрам `base`, `filter` и `scope`.
2. Извлечение участников. Для каждой найденной группы читаются атрибуты членства: `member`, `uniquemember`, `memberof`, `memberuid`, `submember`.
3. Сопоставление пользователей. Каждый DN из атрибутов членства сопоставляется со значением `Identity.extern_uid` в базе данных.
4. Игнорирование неизвестных DN. Если DN не соответствует известному пользователю, он пропускается. Например, это может быть DN вложенной группы.

Циклические зависимости между группами

Циклические зависимости в иерархии LDAP-групп не приводят к ошибкам синхронизации. LDAP Sync обрабатывает группы плоско, по заданному фильтру, и не пытается воспроизводить древовидную структуру LDAP.

Рекурсивный обход вложенности не выполняется, поэтому циклы не влияют на результат синхронизации.

Вложенные группы и транзитивность

LDAP Sync не поддерживает транзитивность для вложенных групп.

Во время синхронизации LDAP Sync обрабатывает только прямые значения атрибутов членства группы и не выполняет рекурсивный обход вложенных групп.

Если одна LDAP-группа содержит другую как участника, пользователи вложенной группы не будут автоматически добавлены в родительскую группу.

Для корректной синхронизации все необходимые DN пользователей должны присутствовать непосредственно в атрибутах членства группы.

Если требуется учитывать вложенные группы, это нужно реализовать на стороне LDAP-сервера. Например, можно заполнять атрибут `submember` полным списком транзитивных участников.

Такой подход упрощает процесс синхронизации и исключает проблемы, связанные с рекурсивной обработкой групп.

Создание локальной учётной записи при включённой синхронизации с LDAP

Локальные учётные записи можно создавать и использовать даже при включённой синхронизации с LDAP.

Такие пользователи входят через веб-интерфейс, пока включена аутентификация по паролю для веб-интерфейса (раздел «Ограничения входа и аутентификация по паролю»).

Настройка

Назначение каждого параметра описано на странице «Обзор». Здесь для каждого типа установки указано место, в котором записывается параметр, и форма его значения.

Где хранится конфигурация

Пакет для ОС Linux. Параметры задаются ключами файла `/etc/gitlab/gitlab.rb`.

Omnibus Docker. Контейнер читает те же ключи `gitlab.rb`. Они попадают в него из переменной окружения `GITLAB_OMNIBUS_CONFIG`, которая вычисляется при каждом запуске, и из файла `/etc/gitlab/gitlab.rb` на томе конфигурации, который читается после переменной. Ключ, записанный в файле, переопределяет тот же ключ в переменной. Переменная задаётся при создании контейнера:

```
docker run -d --name code \
  --shm-size 256m \
  -p 80:80 -p 443:443 -p 22:22 \
  -e GITLAB_OMNIBUS_CONFIG="external_url 'https://<HOSTNAME>';
gitlab_rails['omniauth_enabled'] = true; gitlab_rails['omniauth_allow_single_sign_on']
= ['openid_connect']; gitlab_rails['omniauth_auto_link_ldap_user'] = true" \
  -v /srv/code/config:/etc/gitlab \
  -v /srv/code/logs:/var/log/gitlab \
  -v /srv/code/data:/var/opt/gitlab \
  <REGISTRY>/<FLAVOR>:<VERSION>
```

Переменная фиксируется при создании контейнера, поэтому её изменение означает удаление контейнера и создание нового с теми же томами. Записи провайдеров и LDAP-серверов занимают несколько строк, а том конфигурации сохраняет их при замене контейнера. С томами из быстрого старта файл `/etc/gitlab/gitlab.rb` внутри контейнера — это `/srv/code/config/gitlab.rb` на хосте, и редактируется он на хосте:

```
sudo vi /srv/code/config/gitlab.rb
```

Helm-чарт. Параметры задаются значениями чарта в файле `values.yaml` релиза.

Модуль Deckhouse Kubernetes Platform. Параметры задаются в ресурсе `CodeInstance`, описанном на страницах «Настройка OmniAuth и LDAP» и Custom Resources в документации модуля.

Общие параметры OmniAuth

Параметры и их действие описаны в разделе «Общие параметры OmniAuth».

Пакет для ОС Linux

У каждого общего параметра OmniAuth собственный ключ в `gitlab.rb`:

Параметр	Ключ в `gitlab.rb`
`enabled`	`gitlab_rails['omniauth_enabled']`
`providers`	`gitlab_rails['omniauth_providers']`
`allow_single_sign_on`	`gitlab_rails['omniauth_allow_single_sign_on']`
`block_auto_created_users`	`gitlab_rails['omniauth_block_auto_created_users']`
`auto_link_ldap_user`	`gitlab_rails['omniauth_auto_link_ldap_user']`
`auto_link_user`	`gitlab_rails['omniauth_auto_link_user']`
`auto_sign_in_with_provider`	`gitlab_rails['omniauth_auto_sign_in_with_provider']`
`external_providers`	`gitlab_rails['omniauth_external_providers']`
`allow_bypass_two_factor`	`gitlab_rails['omniauth_allow_bypass_two_factor']`
`sync_profile_from_provider`	`gitlab_rails['omniauth_sync_profile_from_provider']`
`sync_profile_attributes`	`gitlab_rails['omniauth_sync_profile_attributes']`

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_allow_single_sign_on'] = ['openid_connect']
gitlab_rails['omniauth_block_auto_created_users'] = true
gitlab_rails['omniauth_auto_link_ldap_user'] = true
gitlab_rails['omniauth_sync_profile_attributes'] = ['email']
```

Omnibus Docker

Ключи и их значения те же, что во вкладке «Пакет для ОС Linux». Однострочный ключ помещается в `GITLAB_OMNIBUS_CONFIG` и отделяется от следующего символом `;`, как в команде `docker run` выше; ключ, значение которого занимает несколько строк, записывается в файл `/srv/code/config/gitlab.rb`.

Helm-чарт

Общие параметры — это значения в секции `global.appConfig.omniauth`:

```
global:
  appConfig:
    omniauth:
      enabled: true
      allowSingleSignOn: ['openid_connect']
      blockAutoCreatedUsers: true
      autoLinkLdapUser: true
      syncProfileAttributes: ['email']
```

Модуль Deckhouse Kubernetes Platform

Общие параметры задаются в секции `spec.appConfig.omniauth`, а провайдеры перечисляются в её параметре `providers`:

```
enabled: true
allowSingleSignOn: ['openid_connect']
blockAutoCreatedUsers: true
autoLinkLdapUser: true
syncProfileAttributes: ['email']
```

Провайдер OpenID Connect

Параметры подключения провайдера и их действие описаны в разделе «OpenID Connect (OIDC)».

Пакет для ОС Linux

Провайдер — это запись массива `gitlab_rails['omniauth_providers']`. Ключи `allowed_groups`, `admin_groups`, `auditor_groups` и `groups_attribute` указываются рядом с `name`, на верхнем уровне записи; параметры подключения — внутри `args`:

```
gitlab_rails['omniauth_providers'] = [
  {
    name: 'openid_connect',   # Не изменяйте это значение.
    label: 'Keycloak',        # Подпись кнопки входа.
    allowed_groups: ['gitlab'],
    admin_groups: ['admin'],
    auditor_groups: ['audit'],
    groups_attribute: 'gitlab_group',
    args: {
      name: 'openid_connect',
      scope: ['openid', 'profile', 'email'],
      response_type: 'code',
      issuer: 'https://keycloak.example.com/realms/example',
      discovery: true,
      client_auth_method: 'query',
      uid_field: 'preferred_username',
      send_scope_to_token_endpoint: false,
      pkce: true,
      client_options: {
        identifier: '<client_id>',
        secret: '<client_secret>',
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
      }
    }
  }
]
```

Omnibus Docker

Запись та же, что во вкладке «Пакет для ОС Linux». Она занимает несколько строк, поэтому записывайте её в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

Запись провайдера хранится в объекте Secret и подключается из значений, поэтому секрет клиента остаётся вне `values.yaml`. Создайте Secret с записью провайдера:

```
d8 k -n code create secret generic code-omniauth-keycloak \
  --from-file=provider=keycloak.yaml
```

Файл содержит ту же запись, что и при остальных типах установки:

```
name: 'openid_connect'
label: 'Keycloak'
allowed_groups:
  - 'gitlab'
admin_groups:
  - 'admin'
groups_attribute: 'gitlab_group'
args:
  name: 'openid_connect'
  scope:
    - 'openid'
    - 'profile'
    - 'email'
  response_type: 'code'
  issuer: 'https://keycloak.example.com/realms/example'
  discovery: true
  client_auth_method: 'query'
  uid_field: 'preferred_username'
  send_scope_to_token_endpoint: false
  pkce: true
  client_options:
    identifier: '<client_id>'
    secret: '<client_secret>'
    redirect_uri: 'https://code.example.com/users/auth/openid_connect/callback'
```

Подключите Secret в значениях:

```
global:
  appConfig:
    omniauth:
      providers:
        - secret: code-omniauth-keycloak
          key: provider
```

Модуль Deckhouse Kubernetes Platform

Провайдер — это запись параметра `providers` в секции `spec.appConfig.omniauth`. Ключи `allowedGroups`, `adminGroups` и `groupsAttribute` указываются рядом с `name`; параметра для групп аудиторов в ресурсе CodeInstance нет. Параметры подключения указываются внутри `args`:

```
providers:
  - name: 'openid_connect'   # Не изменяйте это значение.
    label: 'Keycloak'        # Подпись кнопки входа.
    allowedGroups:
      - 'gitlab'
    adminGroups:
      - 'admin'
    groupsAttribute: 'gitlab_group'
    args:
      name: 'openid_connect'
      scope:
        - 'openid'
        - 'profile'
        - 'email'
      response_type: 'code'
      issuer: 'https://keycloak.example.com/realms/example'
      discovery: true
      client_auth_method: 'query'
      uid_field: 'preferred_username'
      send_scope_to_token_endpoint: false
      pkce: true
      client_options:
        identifier: '<client_id>'
        secret: '<client_secret>'
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
```

Провайдер SAML

Параметры провайдера SAML и их действие описаны в разделе «SAML».

Пакет для ОС Linux

Провайдер SAML записывается ещё одной записью массива `gitlab_rails['omniauth_providers']`:

```
gitlab_rails['omniauth_providers'] = [
  {
    name: 'saml',
    allowed_groups: ['gitlab'],
    admin_groups: ['admin'],
    groups_attribute: 'gitlab_group'
  }
]
```

Omnibus Docker

Запись та же, что во вкладке «Пакет для ОС Linux»; она записывается в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

Запись SAML хранится в объекте Secret и подключается в `global.appConfig.omniauth.providers` так же, как запись OpenID Connect.

Модуль Deckhouse Kubernetes Platform

Провайдер — это запись параметра `providers` в секции `spec.appConfig.omniauth`:

```
providers:
  - name: 'saml'
    allowedGroups:
      - 'gitlab'
    adminGroups:
      - 'admin'
    groupsAttribute: 'gitlab_group'
```

LDAP-серверы

Ключ записи сервера является его именем. Назначение параметров подключения описано в разделе «Синхронизация с LDAP»; секция `group_sync`, которая создаёт группы и назначает роли,

описана в разделах «Группы и права доступа» и «Секция role_mapping». Секция `group_sync` учитывается только у сервера под ключом `main` (раздел «Область синхронизации»).

Пакет для ОС Linux

Вход через LDAP включается ключом `gitlab_rails['ldap_enabled']`. Серверы описываются в ключе `gitlab_rails['ldap_servers']`, значение которого — документ YAML.

> ⚠ Значение разбирается как YAML, поэтому отступы внутри блока значимы, а символы табуляции его ломают.

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    label: 'Головной офис'
    host: ldap-main.example.com
    port: 3389
    uid: 'cn'
    bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    # Не учитывать пользователей, заблокированных в каталоге (опционально).
    user_filter: '(!(nsAccountLock=TRUE))'
    # Пользователь найден в LDAP — вход разрешён сразу.
    block_auto_created_users: false
    sync_name: true
    group_sync:
      create_groups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      prefix:
        attribute: 'businessCategory'
        default: 'default-program'
      top_level_group: 'LdapGroups'
      name_mask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      role_mapping:
        - by_name: '.*-project_manager-.*'
          gitlab_role: 'maintainer'
        - by_name: '.*-developer-.*'
          gitlab_role: 'developer'
        - by_name: '.*-participant-.*'
          gitlab_role: 'reporter'
  contractors:
    label: 'Подрядчики'
    host: ldap-contractors.example.com
    port: 3389
    uid: 'cn'
    bind_dn: 'uid=viewer,ou=People,dc=contractors,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=contractors,dc=example,dc=com'
EOS
```

Omnibus Docker

Ключи `gitlab_rails['ldap_enabled']` и `gitlab_rails['ldap_servers']` те же, что во вкладке «Пакет для ОС Linux». Значение ключа `gitlab_rails['ldap_servers']` занимает несколько строк, поэтому записывайте его в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

Серверы описываются значениями в секции `global.appConfig.ldap.servers` теми же ключами, которые читает продукт. Пароль сервисной учётной записи берётся из объекта Secret:

```
d8 k -n code create secret generic code-ldap-password \
  --from-literal=password='viewer123'
```

```
global:
  appConfig:
    ldap:
      servers:
        main:
          label: 'Головной офис'
          host: ldap-main.example.com
          port: 3389
          uid: 'cn'
          bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=example,dc=com'
          user_filter: '(!(nsAccountLock=TRUE))'
          block_auto_created_users: false
          sync_name: true
          group_sync:
            create_groups: true
            base: 'ou=Groups,dc=example,dc=com'
            filter: '(objectClass=groupOfNames)'
            top_level_group: 'LdapGroups'
            owner: 'root'
            role_mapping:
              - by_name: '.*-developer-.*'
                gitlab_role: 'developer'
        contractors:
          label: 'Подрядчики'
          host: ldap-contractors.example.com
          port: 3389
          uid: 'cn'
          bind_dn: 'uid=viewer,ou=People,dc=contractors,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=contractors,dc=example,dc=com'
```

Модуль Deckhouse Kubernetes Platform

Серверы — это ключи параметра `servers` в секции `spec.appConfig.ldap`:

```
servers:
  main:
    label: ldap
    host: 127.0.0.1
    port: 3389
    bindDn: 'uid=viewer,ou=People,dc=example,dc=com'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    password: 'viewer123'
    syncName: true
    groupSync:
      createGroups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      prefix:
        attribute: 'businessCategory'
        default: 'default-program'
      topLevelGroup: 'LdapGroups'
      nameMask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      roleMapping:
        - byName: '.*-project_manager-.*'
          gitlabRole: 'Maintainer'
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
        - byName: '.*-participant-.*'
          gitlabRole: 'Reporter'
```

Несколько LDAP-серверов

Несколько серверов описываются повторением записи сервера под другим ключом. У каждого сервера собственные параметры подключения; секция `group_sync` со всем её содержимым учитывается только у сервера под ключом `main` (раздел «Область синхронизации»). Идентификация пользователя и решение о доступе, когда один и тот же пользователь найден на нескольких серверах, описаны в разделе «Несколько LDAP-серверов».

Пакет для ОС Linux

Записи являются ключами того же документа YAML в `gitlab_rails['ldap_servers']`, как `main` и `contractors` в разделе «LDAP-серверы» выше.

Omnibus Docker

Записи те же, что во вкладке «Пакет для ОС Linux»; они записываются в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

Записи являются ключами секции `global.appConfig.ldap.servers`, как `main` и `contractors` в разделе «LDAP-серверы» выше.

Модуль Deckhouse Kubernetes Platform

Записи являются ключами параметра `servers` секции `spec.appConfig.ldap`:

```
servers:
  main:
    label: 'Головной офис'
    host: ldap-main.example.com
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    # Остальные параметры подключения.
    groupSync:
      base: 'ou=Groups,dc=example,dc=com'
      roleMapping:
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
  contractors:
    label: 'Подрядчики'
    host: ldap-contractors.example.com
    base: 'ou=People,dc=contractors,dc=example,dc=com'
    uid: 'cn'
    # Остальные параметры подключения.
```

Связывание учётных записей OIDC с LDAP

Автоматическое связывание OIDC-учётной записи с учётной записью LDAP включается параметром `auto_link_ldap_user`. Как ищется учётная запись LDAP и что происходит при первом и последующих входах, описано в разделе «Связывание учётных записей OIDC с LDAP».

Пакет для ОС Linux

```
gitlab_rails['omniauth_auto_link_ldap_user'] = true
```

Omnibus Docker

Ключ тот же, что во вкладке «Пакет для ОС Linux». Его значение занимает одну строку, поэтому помещается и в `GITLAB_OMNIBUS_CONFIG`, и в файл `/srv/code/config/gitlab.rb`.

Helm-чарт

```
global:
  appConfig:
    omniauth:
      autoLinkLdapUser: true
```

Модуль Deckhouse Kubernetes Platform

Параметр задаётся в секции `spec.appConfig.omniauth`:

```
autoLinkLdapUser: true
```

Вход пользователей, найденных в LDAP

Чтобы пользователи, найденные в LDAP, могли входить сразу, а остальные отправлялись на одобрение администратору, сочетайте параметры `auto_link_ldap_user` и `block_auto_created_users` секции OmniAuth с параметром `block_auto_created_users` LDAP-сервера `main` (раздел «Вход пользователей, найденных в LDAP»).

Пакет для ОС Linux

```
gitlab_rails['omniauth_auto_link_ldap_user'] = true
# Пользователь не найден в LDAP — учётная запись создаётся заблокированной,
# до одобрения администратором.
gitlab_rails['omniauth_block_auto_created_users'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    # Пользователь найден в LDAP — вход разрешён сразу.
    block_auto_created_users: false
    # Остальные параметры подключения.
EOS
```

Omnibus Docker

Ключи те же, что во вкладке «Пакет для ОС Linux»; они записываются в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

```
global:
  appConfig:
    omniauth:
      autoLinkLdapUser: true
      # Пользователь не найден в LDAP — учётная запись создаётся заблокированной,
      # до одобрения администратором.
      blockAutoCreatedUsers: true
    ldap:
      servers:
        main:
          # Пользователь найден в LDAP — вход разрешён сразу.
          block_auto_created_users: false
```

Модуль Deckhouse Kubernetes Platform

Параметры задаются в секции `spec.appConfig`:

```
omniauth:
  autoLinkLdapUser: true
  # Пользователь не найден в LDAP — учётная запись создаётся заблокированной,
  # до одобрения администратором.
  blockAutoCreatedUsers: true
ldap:
  servers:
    main:
      # Пользователь найден в LDAP — вход разрешён сразу.
      blockAutoCreatedUsers: false
```

Расписание синхронизации

Расписание задачи синхронизации записывается в формате cron. Что задача делает при каждом запуске, описано в разделе «Синхронизация с LDAP»; запуск вручную описан в разделе «Ручной запуск синхронизации».

Пакет для ОС Linux

```
gitlab_rails['ldap_sync_worker_cron'] = "0 * * * *"
```

Omnibus Docker

Ключ тот же, что во вкладке «Пакет для ОС Linux». Его значение занимает одну строку, поэтому помещается и в `GITLAB_OMNIBUS_CONFIG`, и в файл `/srv/code/config/gitlab.rb`.

Helm-чарт

```
global:
  appConfig:
    cron_jobs:
      ldap_sync_worker:
        cron: "0 * * * *"
```

Модуль Deckhouse Kubernetes Platform

Периодичность задаётся параметром `cronJobs` секции `spec.appConfig`:

```
cronJobs:
  ldapSyncWorker:
    cron: "0 * * * *"
```

Применение конфигурации

Пакет для ОС Linux

Каждое изменение файла `/etc/gitlab/gitlab.rb` вступает в силу после выполнения команды:

```
sudo gitlab-ctl reconfigure
```

Omnibus Docker

Примените изменённый файл в работающем контейнере:

```
docker exec code gitlab-ctl reconfigure
```

Перезапуск контейнера тоже применяет конфигурацию; так вступает в силу изменённая переменная `GITLAB_OMNIBUS_CONFIG` пересозданного контейнера:

```
docker restart code
```

Helm-чарт

Примените изменённый файл `values.yaml` к релизу:

```
helm upgrade code deckhouse-code \
  -n code \
  -f values.yaml
```

Модуль Deckhouse Kubernetes Platform
Изменённый ресурс `CodeInstance` применяет оператор.

Пример настройки: вход через OIDC и права из LDAP

Ниже приведена последовательность настройки, при которой пользователи входят через OIDC-провайдера (например, Keycloak), а группы, членства в них и роли приходят из LDAP.

Настройка включает в себя следующие шаги:

1. Настройка LDAP-провайдера.
2. Настройка OIDC-провайдера и включение связывания с LDAP.
3. Проверка связывания при первом входе.
4. Синхронизация прав.

Требования для настройки

Для настройки требуются:

- OIDC-провайдер.
- LDAP-каталог с теми же пользователями и с группами, имена которых позволяют определить роль.
- Сервисная учётная запись LDAP с правами на чтение каталога (параметры `bind_dn` и `password`).

> Значение `uid` или email в OIDC-провайдере должно совпадать со значением соответствующего атрибута в LDAP, иначе связывание не сработает.

Порядок поиска учётной записи описан в разделе «Как ищется учётная запись LDAP».

Настройка LDAP-провайдера

В записи сервера указываются каталог, база поиска и сервисная учётная запись, а её секция `group_sync` превращает имена групп в роли.

Пакет для ОС Linux

```
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-'EOS'
  main:
    label: ldap
    host: ldap.example.com
    port: 3389
    bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    sync_name: true
    # Не учитывать пользователей, заблокированных в каталоге (опционально).
    user_filter: '(!(nsAccountLock=TRUE))'
    # Пользователь найден в LDAP — вход разрешён сразу.
    block_auto_created_users: false
    group_sync:
      create_groups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      top_level_group: 'LdapGroups'
      name_mask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      role_mapping:
        - by_name: '.*-maintainer-.*'
          gitlab_role: 'maintainer'
        - by_name: '.*-developer-.*'
          gitlab_role: 'developer'
        - by_name: '.*-participant-.*'
          gitlab_role: 'reporter'
EOS
```

Omnibus Docker

Ключи `gitlab_rails['ldap_enabled']` и `gitlab_rails['ldap_servers']` те же, что во вкладке «Пакет для ОС Linux»; они записываются в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

```
global:
  appConfig:
    ldap:
      servers:
        main:
          label: ldap
          host: ldap.example.com
          port: 3389
          bind_dn: 'uid=viewer,ou=People,dc=example,dc=com'
          password:
            secret: code-ldap-password
            key: password
          base: 'ou=People,dc=example,dc=com'
          uid: 'cn'
          sync_name: true
          # Не учитывать пользователей, заблокированных в каталоге (опционально).
          user_filter: '(!(nsAccountLock=TRUE))'
          # Пользователь найден в LDAP — вход разрешён сразу.
          block_auto_created_users: false
          group_sync:
            create_groups: true
            base: 'ou=Groups,dc=example,dc=com'
            filter: '(objectClass=groupOfNames)'
            top_level_group: 'LdapGroups'
            name_mask: '(?<=-)[A-z0-9]*$'
            owner: 'root'
            role_mapping:
              - by_name: '.*-maintainer-.*'
                gitlab_role: 'maintainer'
              - by_name: '.*-developer-.*'
                gitlab_role: 'developer'
              - by_name: '.*-participant-.*'
                gitlab_role: 'reporter'
```

Модуль Deckhouse Kubernetes Platform

Настройте конфигурацию в параметре `servers` секции `spec.appConfig.ldap`:

```
servers:
  main:
    label: ldap
    host: ldap.example.com
    port: 3389
    bindDn: 'uid=viewer,ou=People,dc=example,dc=com'
    password: 'viewer123'
    base: 'ou=People,dc=example,dc=com'
    uid: 'cn'
    syncName: true
    # Не учитывать пользователей, заблокированных в каталоге (опционально).
    userFilter: '(!(nsAccountLock=TRUE))'
    # Пользователь найден в LDAP — вход разрешён сразу.
    blockAutoCreatedUsers: false
    groupSync:
      createGroups: true
      base: 'ou=Groups,dc=example,dc=com'
      filter: '(objectClass=groupOfNames)'
      topLevelGroup: 'LdapGroups'
      nameMask: '(?<=-)[A-z0-9]*$'
      owner: 'root'
      roleMapping:
        - byName: '.*-maintainer-.*'
          gitlabRole: 'Maintainer'
        - byName: '.*-developer-.*'
          gitlabRole: 'Developer'
        - byName: '.*-participant-.*'
          gitlabRole: 'Reporter'
```

Настройка OIDC-провайдера и включение связывания с LDAP

Значение `uid_field` используется при поиске пользователя в LDAP. Оно должно совпадать со значением атрибута, указанного в параметре `uid` LDAP-сервера, или с адресом электронной почты — подробнее в разделе «Как ищется учётная запись LDAP».

Остальные параметры провайдера описаны в разделе «OpenID Connect (OIDC)».

Пакет для ОС Linux

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_auto_link_ldap_user'] = true
gitlab_rails['omniauth_providers'] = [
  {
    name: 'openid_connect',   # Не изменяйте это значение.
    label: 'Keycloak',        # Подпись кнопки входа.
    groups_attribute: 'gitlab_group',
    args: {
      name: 'openid_connect',
      scope: ['openid', 'profile', 'email'],
      response_type: 'code',
      issuer: 'https://keycloak.example.com/realms/example',
      discovery: true,
      client_auth_method: 'query',
      uid_field: 'preferred_username',
      send_scope_to_token_endpoint: false,
      pkce: true,
      client_options: {
        identifier: '<client_id>',
        secret: '<client_secret>',
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
      }
    }
  }
]
```

Omnibus Docker

Ключи `gitlab_rails['omniauth_enabled']`, `gitlab_rails['omniauth_auto_link_ldap_user']` и `gitlab_rails['omniauth_providers']` те же, что во вкладке «Пакет для ОС Linux»; они записываются в файл `/srv/code/config/gitlab.rb` на томе конфигурации.

Helm-чарт

Запись провайдера из этого примера хранится в объекте Secret и подключается из значений, рядом с которыми включается связывание:

```
global:
  appConfig:
    omniauth:
      enabled: true
      autoLinkLdapUser: true
      providers:
        - secret: code-omniauth-keycloak
          key: provider
```

Модуль Deckhouse Kubernetes Platform

Настройте конфигурацию в секции `spec.appConfig.omniauth`:

```
autoLinkLdapUser: true
providers:
  - name: 'openid_connect'   # Не изменяйте это значение.
    label: 'Keycloak'        # Подпись кнопки входа.
    groupsAttribute: 'gitlab_group'
    args:
      name: 'openid_connect'
      scope:
        - 'openid'
        - 'profile'
        - 'email'
      response_type: 'code'
      issuer: 'https://keycloak.example.com/realms/example'
      discovery: true
      client_auth_method: 'query'
      uid_field: 'preferred_username'
      send_scope_to_token_endpoint: false
      pkce: true
      client_options:
        identifier: '<client_id>'
        secret: '<client_secret>'
        redirect_uri: 'https://code.example.com/users/auth/openid_connect/
callback'
```

Проверка связывания при первом входе

Для проверки связывания выполните следующие шаги:

1. Войдите под тестовым пользователем через OIDC-провайдера.
2. Откройте страницу пользователя в административном разделе (`/admin/users/<username>/identities`). У связанной учётной записи должно быть два идентификатора: `openid_connect` и `ldapmain` (имя LDAP-идентификатора складывается из префикса `ldap` и имени LDAP-сервера, в примере — `main`).

Если LDAP-идентификатора нет, проверьте следующее:

- значения `uid` или email в OIDC-провайдере и в LDAP совпадают;
- пользователь попадает в область поиска `base` и не отсекается фильтром `user_filter` ;
- параметр `auto_link_ldap_user` включён.

Синхронизация прав

Сразу после первого входа у пользователя есть учётная запись, но нет членства в группах и проектах. Дождитесь ближайшей синхронизации или запустите её вручную (раздел «Ручной запуск синхронизации»), после чего проверьте, что:

- группы созданы внутри группы, указанной в `group_sync.top_level_group` ;
- пользователь добавлен в них с ролью, соответствующей `role_mapping` .

Настройки безопасности

Учётная запись администратора

Инстанс запускается с учётной записью администратора `root` , пароль которой создаётся при установке.

1. Войдите под `root` с первоначальным паролем.
2. Откройте меню пользователя и выберите «Редактировать профиль».
3. Перейдите в «Доступ» → «Пароль и аутентификация» и задайте новый пароль.
4. Сохраните новый пароль вне инстанса.

После смены пароля удалите первоначальный пароль оттуда, где его хранит поставка:

Пакет для ОС Linux

Файл `/etc/gitlab/initial_root_password` , удаляется командой `sudo rm -f /etc/gitlab/initial_root_password` . Команда `gitlab-ctl reconfigure` , выполненная больше чем через 24 часа после установки, удаляет файл сама, значение пароля при этом не меняется.

Omnibus Docker

Файл `/etc/gitlab/initial_root_password` внутри контейнера, удаляется командой `docker exec code rm -f /etc/gitlab/initial_root_password`. Файл лежит в томе с конфигурацией: с томами из раздела «Быстрый старт» это `/srv/code/config/initial_root_password` на хосте.

Helm-чарт

Секрет `code-initial-root-password` в пространстве имён релиза, читается командой `d8 k -n code get secret code-initial-root-password -o jsonpath='{.data.password}' | base64 -d`. После смены пароля значение из секрета больше не открывает учётную запись.

Модуль Deckhouse Kubernetes Platform

Секрет `initial-root-password` в пространстве имён `d8-code`, читается командой `d8 k -n d8-code get secret initial-root-password -o jsonpath='{.data.password}' | base64 -d`. После смены пароля значение из секрета больше не открывает учётную запись.

Регистрация и вход

Новые учётные записи

После установки пакета для ОС Linux, Omnibus Docker или Helm-чарта любой, у кого есть сетевой доступ к инстансу, заводит себе учётную запись. Чтобы закрыть регистрацию:

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните раздел «Ограничения аккаунтов новых пользователей».
3. Снимите флажок «Разрешить новые учётные записи пользователей».
4. Нажмите «Сохранить изменения».

В модуле Deckhouse Kubernetes Platform регистрация закрыта после установки; оператор задаёт её из поля `appConfig.signUpEnabled` ресурса CodeInstance.

В остальных типах установки то же самое делается из консоли:

Пакет для ОС Linux

```
sudo gitlab-rails runner 'ApplicationSetting.current.update!(signup_enabled:
false)'
sudo gitlab-rails runner 'puts ApplicationSetting.current.signup_enabled'
```

Omnibus Docker

```
docker exec code gitlab-rails runner 'ApplicationSetting.current.update!
(signup_enabled: false)'
docker exec code gitlab-rails runner 'puts
ApplicationSetting.current.signup_enabled'
```

Helm-чарт

```
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner 'ApplicationSetting.c
urrent.update!(signup_enabled: false)'
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner 'puts
ApplicationSetting.current.signup_enabled'
```

Вторая команда выводит `false`, пока регистрация закрыта.

Пока регистрация открыта, тот же раздел административной панели ограничивает круг тех, кто получает учётную запись:

- «Требовать одобрение администратора для новых учётных записей» — учётная запись ждёт администратора до первого входа;
- «Настройки подтверждения электронной почты» — в режиме «Жесткий режим» адрес подтверждается до первого входа;
- «Разрешённые домены для новых пользователей» и «Список запрещённых доменов» — почтовые домены, с которыми регистрация разрешена и запрещена;
- «Минимальная длина пароля (количество символов)» — действует для учётных записей, которые входят по паролю, по умолчанию 8 символов.

Ни одна из этих настроек не действует на учётную запись, созданную через внешнего провайдера.

Ограничения входа

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните раздел «Ограничения входа».
3. Установите флажок «Требовать двухфакторную аутентификацию для администраторов».
4. Нажмите «Сохранить изменения».

В том же разделе находятся:

- «Включить режим администратора» — администратор проходит аутентификацию ещё раз перед открытием административной панели;

- «Уведомление по email о неизвестных входах» — пользователь получает письмо, когда вход выполнен с устройства или IP-адреса, которые раньше не использовались;
- «Требовать подтверждение email при блокировке учётной записи» — заблокированная учётная запись подтверждает почтовый ящик перед следующим входом.

Двухфакторная аутентификация для всех пользователей инстанса, включённые протоколы доступа к Git и аутентификация по паролю для Git через HTTPS задаются в том же разделе и описаны на странице «Пользователи и доступ». Аутентификация по паролю для веб-интерфейса описана на странице «Аутентификация».

Внешний провайдер аутентификации

При использовании LDAP, SAML или OIDC учётные записи и членство в группах хранятся в каталоге, и закрытая там учётная запись теряет доступ к Deckhouse Code без действий внутри инстанса. Провайдеры, синхронизация и связывание учётных записей описаны на странице «Аутентификация».

Видимость и контроль доступа

Настройки находятся в разделе «Admin» → «Настройки» → «Основные» → «Видимость и контроль доступа»:

- «Уровень видимости проектов по умолчанию», «Уровень видимости сниппетов по умолчанию» и «Уровень видимости групп по умолчанию» — задайте значение «Приватный», чтобы новый проект, сниппет или группа были закрыты, пока их не откроют;
- «Ограниченные уровни видимости» — уровни, которые не может выбрать неадминистратор. Если отметить здесь публичный и внутренний уровни, обычный пользователь создаёт только приватные проекты;
- срок хранения удалённой группы или проекта — время, в течение которого удаление ещё обратимо;
- «Минимальная роль по умолчанию, необходимая для создания проектов» и «Включенные протоколы доступа к Git» — описаны на странице «Пользователи и доступ»;
- «RSA SSH keys», «DSA SSH keys», «ECDSA SSH keys» и остальные списки ключей — каждый из них запрещает свой алгоритм или задаёт минимальную длину ключа, которую инстанс принимает от пользователя.

Запретите ключи DSA, задайте минимальную длину ключа RSA 3072 бита, ключа ECDSA — 256 бит. Списки `ECDSA_SK` и `ED25519_SK` оставьте включёнными, пока используются аппаратные ключи FIDO2 или U2F.

Ограничения учётных записей

Настройки находятся в разделе «Admin» → «Настройки» → «Основные» → «Аккаунт и ограничения»:

- «Лимит проектов по умолчанию» — максимальное количество проектов, которые создаёт один пользователь;
- «Максимальная продолжительность сессии» — длительность сессии в минутах, по умолчанию 7 дней; изменение применяется после перезапуска инстанса;
- «Запомнить меня» → «Разрешить пользователям использовать функцию „Запомнить меня“» — со снятым флажком сессия заканчивается по истечении срока независимо от действий пользователя;
- «Требовать дату истечения» — новый персональный, групповой или проектный токен получает дату истечения;
- «Пользовательские OAuth-приложения» — со снятым флажком пользователь не регистрирует в инстансе собственный клиент OAuth;
- группа «Ограничения пользователя» — создание групп и проектов, описано на странице «Пользователи и доступ».

Перезапустите инстанс после изменения длительности сессии:

Пакет для ОС Linux

```
sudo gitlab-ctl restart
```

Omnibus Docker

```
docker restart code
```

Helm-чарт

```
d8 k -n code rollout restart deploy/<WEBSERVICE_DEPLOYMENT>
```

Модуль Deckhouse Kubernetes Platform

```
d8 k -n d8-code rollout restart deploy -l 'app.kubernetes.io/component in
(webservice, sidekiq)'
```

Импорт и экспорт

Настройки находятся в разделе «Admin» → «Настройки» → «Основные» → «Настройки импорта и экспорта»:

- «Источники импорта» — системы, из которых импортируется проект; по умолчанию выключены все источники, а источник «Repository by URL» принимает любой репозиторий Git по адресу;
- «Экспорт проекта» — со снятым флажком пользователь не собирает архив `.tar.gz` с данными проекта для переноса на другой инстанс;
- «Разрешить перенос групп и проектов GitLab путем прямого переноса» — выключено по умолчанию;
- «Экспорты администраторами без аудита» — пока флажок установлен, экспорт, выполненный администратором, не попадает в журнал аудита.

Настройки репозиториев

Ветка по умолчанию

1. Перейдите в «Admin» → «Настройки» → «Репозиторий».
2. Разверните раздел «Ветка по умолчанию».
3. В поле «Имя начальной ветки по умолчанию» укажите имя ветки, с которой начинается каждый новый репозиторий.
4. В группе «Защита начальной ветки по умолчанию» выберите защиту, с которой создаётся ветка по умолчанию, и задайте в полях «Разрешен push» и «Разрешен merge» роль, от которой ветка принимает изменения.
5. Снимите флажки «Разрешен force push» и «Разрешить разработчикам отправлять начальный коммит».
6. Нажмите «Сохранить изменения».

Настройки действуют на репозитории, созданные после изменения; в существующем репозитории правила веток остаются прежними.

Правила отправки изменений

Раздел «Push-правила» на той же странице задаёт правила, с которыми создаётся каждый проект инстанса: регулярные выражения для сообщения коммита, адреса электронной почты автора,

имени файла и имени ветки, проверки коммитера и запрет на коммит секретов, например файлов `pem` или `id_rsa`. Правило со снятым разрешением на переопределение нельзя изменить в группе или проекте. Сами правила описаны на странице «Правила отправки изменений», уровень инстанса — на странице «Пользователи и доступ».

Настройки CI/CD

Токены аутентификации раннеров

Раздел «Admin» → «Настройки» → «CI/CD» → «Непрерывная интеграция и деплой» задаёт срок жизни токенов аутентификации раннеров уровня инстанса, группы и проекта; пока значение не задано, срок не ограничен. Раннер в сети сам ротирует токен по мере приближения срока истечения; раннер, который в этот момент был офлайн, регистрируется заново.

Раннеры

Раздел «Admin» → «Настройки» → «CI/CD» → «Раннеры» содержит:

- «Получить данные о версиях релизов раннеров с GitLab.com» — пока флажок установлен, инстанс запрашивает данные о версиях раннеров с GitLab.com; снимите его в установке без доступа в интернет;
- «Разрешить токен регистрации раннера» — со снятым флажком раннер регистрируется собственным токеном аутентификации вместо общего токена регистрации инстанса, группы или проекта.

Права токена заданий

Раздел «Admin» → «Настройки» → «CI/CD» → «Права доступа токена заданий в CI/CD» содержит настройку «Включить и применять белый список токенов заданий для всех проектов». Пока она включена, задание обращается к другому проекту только тогда, когда тот перечислил проект задания в своём списке, а вариант с доступом от всех групп и проектов скрыт.

Защита от спама и ботов

Раздел «Admin» → «Настройки» → «Отчетность» → «Защита от спама и ботов» содержит reCAPTCHA и ограничение количества учётных записей на один IP-адрес. Раздел применяется для инстанса, доступного из интернета.

Статистика использования

В разделе «Admin» → «Настройки» → «Метрики и профилирование» подраздел «Статистика использования» содержит проверку обновлений и Service Ping, а подраздел «Отслеживание событий» — отправку событий. Отслеживание событий выключено, его флажок доступен только для чтения; выключенную проверку обновлений нельзя включить обратно из интерфейса.

Сетевые ограничения

Раздел «Admin» → «Настройки» → «Сеть» содержит ограничения частоты запросов: «Ограничения скорости пользователей и IP-адресов» для инстанса в целом, «Ограничения на частоту запросов Git HTTP», «Ограничения на частоту запросов Git LFS», «Ограничения скорости поиска», «Ограничения скорости создания пайплайна» и «Ограничения скорости импорта и экспорта» для своего типа запросов, каждое из которых переопределяет общее ограничение. Настройка «Исходящие запросы» ограничивает адреса, к которым обращаются хуки и интеграции инстанса.

Защита на уровне платформы

В модуле Deckhouse Kubernetes Platform трафик между компонентами, доступ к пространству имён `d8-code`, сетевые политики и экспорт событий аудита настраиваются в кластере. Они описаны на странице «Настройки безопасности» в документации модуля.

Ключи доступа

Раздел ключей доступа предоставляет полный обзор всех токенов (персональные, групповые, проектные) и ключей (SSH, GPG) на платформе Code. Администраторы могут переключать вкладки с различными видами ключей, применять фильтры и опции сортировки, и удалять или отзывать ключи, чтобы ограничить доступ пользователей и повысить безопасность.

Цели

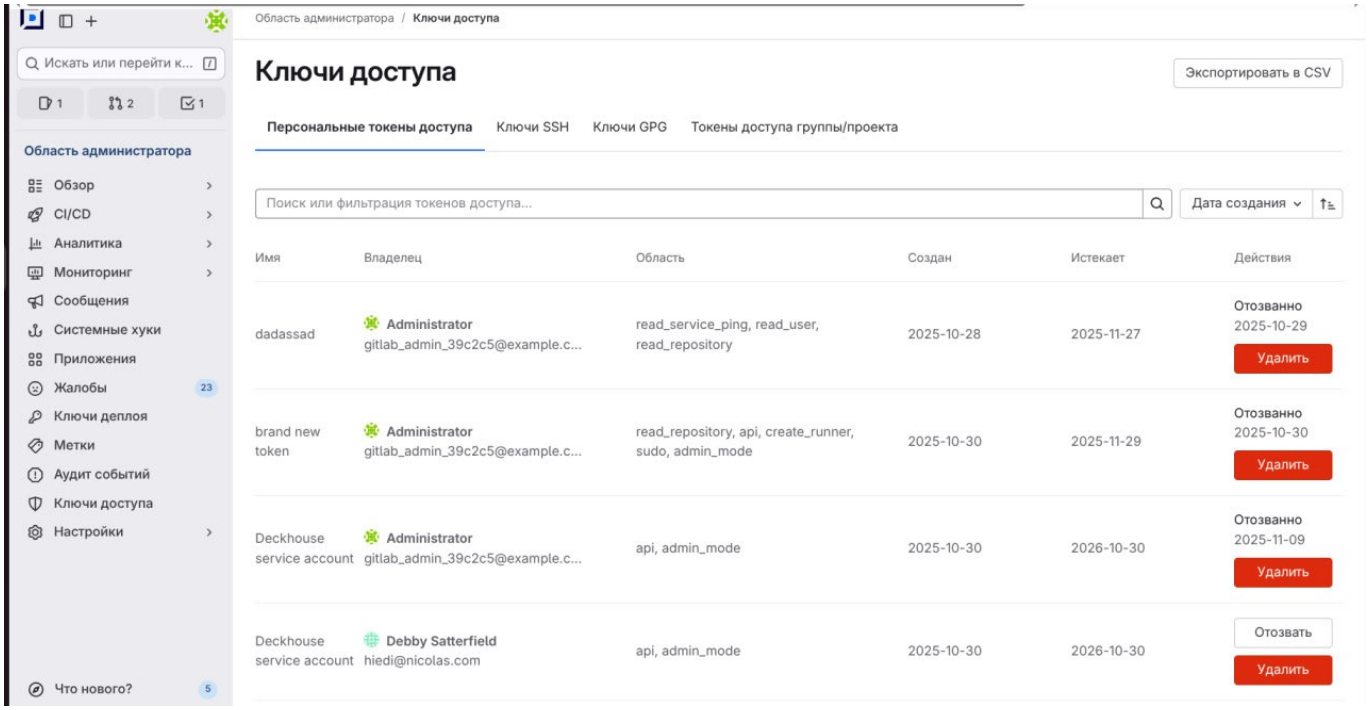
Раздел ключей доступа используется в следующих целях:

- Предоставление полной информации обо всех ключах доступа (владелец, область использования, даты создания/истечения срока/отзыва).
- Фильтрация и сортировка ключей доступа.
- Управление жизненным циклом ключей: отзыв, удаление.

Работа с разделом ключей доступа через UI

Чтобы начать работу с ключами доступа, перейдите в режим администратора и в боковом меню выберите пункт «Ключи доступа». Откроется таблица ключей доступа. Раздел состоит из 4 вкладок:

- **Персональные токены доступа** — список токенов, принадлежащих реальным пользователям
- **Ключи SSH** — список ключей SSH, принадлежащих реальным пользователям.
- **Ключи GPG** — список ключей GPG, принадлежащих реальным пользователям.
- **Токены доступа группы/проекта** — список токенов, принадлежащих группам верхнего уровня или проектам.



Наборы действий над ключами доступа

Набор действий, которые можно совершить над ключом доступа, зависит от его типа. В таблице представлена зависимость действий от типа ключа.

Операция\Тип ключа	Персональный токен доступа	Ключ SSH	Ключ GPG	Токен доступа группы/ проекта
Удаление	Да	Да	Да	Да
Отзыв	Да	Нет	Нет	Да

Сервисные аккаунты

Сервисный аккаунт — это учётная запись, предназначенная для использования в автоматизированных скриптах. Такие аккаунты применяются в CI/CD-пайплайнах и интеграциях. Сервисный аккаунт нельзя использовать для аутентификации в веб-интерфейсе, а также для выполнения действий от его имени через имперсонацию.

Создание сервисного аккаунта

Rails-консоль

Сервисный аккаунт создаётся из Rails-консоли. Откройте консоль для используемого типа установки:

Пакет для ОС Linux

```
sudo gitlab-rails console
```

Omnibus Docker

```
docker exec -it code gitlab-rails console
```

`code` — имя контейнера из команды `docker run` в разделе «Быстрый старт».

Helm-чарт

```
d8 k -n code exec -it deploy/code-toolbox -- gitlab-rails console
```

Модуль Deckhouse Kubernetes Platform

Консоль входит в набор служебных инструментов Toolbox:

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails console -e
production
```

Создание аккаунта

1. Используя Rails-консоль, подготовьте параметры с описанием создаваемого аккаунта. Заполните поля `name`, `username`, `email` и `admin` и задайте остальные параметры на основе следующего примера:

```
user_args = {
name: 'kaiten_sa',
username: 'kaiten_sa',
email: 'kaiten_sa@flant.com',
admin: false,
user_type: :service_account,
organization_id: Organizations::Organization.default_organization.id,
password_automatically_set: true,
force_random_password: true,
skip_confirmation: true
}
```

2. Выберите пользователя, от имени которого будет создан сервисный аккаунт, после чего выполните создание пользователя:

```
user = User.find_by_username('root')
Users::CreateService.new(user, user_args).execute
```

Генерация токена доступа

Чтобы сгенерировать токен доступа, используйте Personal access tokens API от GitLab.

События аудита безопасности

Аудит безопасности — это детальный анализ вашей инфраструктуры, направленный на выявление потенциальных уязвимостей и небезопасных практик. Code помогает упростить процесс аудита с помощью событий аудита, которые позволяют отслеживать широкий спектр действий, происходящих в системе.

Назначение и область применения

Механизм регистрации событий безопасности (события аудита) используется для:

- фиксации действий пользователей и администраторов, связанных с изменением конфигурации системы;
- регистрации инцидентов информационной безопасности;
- обеспечения возможности расследования происшествий и восстановления полной картины действий в системе.

Область применения — все уровни платформы Code, от отдельных проектов и групп до конфигурации инстанса. События фиксируются независимо от прав пользователя (при условии, что у него есть доступ к совершению действия) и сохраняются централизованно.

Технические меры регистрации событий

Система аудита реализует следующие меры:

- **Централизованная регистрация событий** — все события фиксируются в едином журнале аудита.
- **Непрерывный сбор данных** — событие фиксируется в реальном времени по ходу выполнения бизнес-операции. Например, при входе в систему, смене адреса электронной почты пользователя или включении возможности выполнять `force push` в защищённой ветке.
- **Защита целостности** — журнал событий доступен только в режиме чтения для администраторов. Удалить или изменить события в журнале невозможно.
- **Доступ через UI и API** — просмотр и фильтрация событий возможны как в интерфейсе администратора, так и через специализированный API.

Сценарии использования

События аудита безопасности позволяют отследить:

- кто и когда изменил уровень доступа пользователя в проекте Code;
- случаи создания и удаления пользователей;
- признаки подозрительной активности — например, массовая смена адресов электронной почты сотрудников или удаление репозиториев;
- случаи изменения переменных окружения в CI/CD;
- случаи изменения уровня видимости проектов и групп.

События аудита помогают:

- оценивать риски и усиливать меры безопасности;
- своевременно реагировать на инциденты.

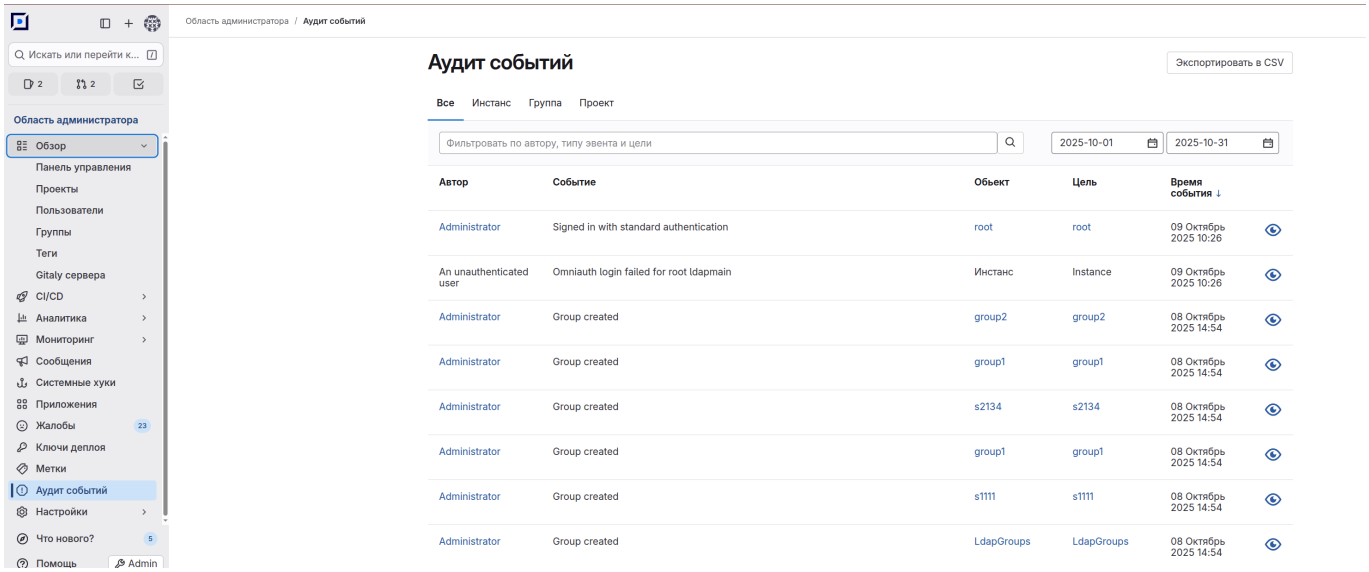
Доступ к событиям аудита

Доступ через UI

Чтобы получить доступ к событиям аудита, войдите в режим администратора и в боковом меню выберите пункт «Аудит событий». Откроется таблица событий аудита безопасности.

Описание столбцов в таблице:

- **Автор** — имя пользователя, запустившего событие.
- **Событие** — системное сообщение с информацией о событии.
- **Объект** — область, к которой относится событие (название инстанса, группы, проекта, имя пользователя).
- **Цель** — сущность, которая была изменена (название проекта, защищённой ветки, CI-переменной, имя пользователя, токена, и т. д.).
- **Время события** — дата и время наступления события.



Доступ через API

Deckhouse Code предоставляет следующий API-метод для получения списка событий аудита:

`POST /api/v4/admin/audit_events/search`

Допускается фильтрация по датам, текстовому поиску и типам сущностей.

> ⚠ Диапазон дат должен находиться в пределах одного календарного месяца. Если значения `created_after` и `created_before` относятся к разным месяцам, параметр `created_before` будет автоматически приведён к последнему дню месяца, в который входит `created_after`.

Параметры запроса

Параметр	Тип	Обязательный	Описание
`created_after`	Строка	Нет	Начальная дата (включительно) в формате ISO8601. По умолчанию — начало текущего месяца
`created_before`	Строка	Нет	Конечная дата (включительно) в формате ISO8601. По умолчанию — конец текущего месяца
`q`	Строка	Нет	Полнотекстовый поиск по сообщению события
`sort`	Строка	Нет	Сортировка по дате создания. Возможные значения: `created_asc`, `created_desc` (по умолчанию)
`entity_types`	Массив строк	Нет	Список типов сущностей для фильтрации: `User`, `Project`, `Group`, `Gitlab::Audit::InstanceScope`

Пример запроса:

```
curl --request POST "https://example.com/api/v4/admin/audit_events/search" \
     --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
     --header "Content-Type: application/json" \
     --data '{
       "created_after": "2025-08-01",
       "created_before": "2025-08-31",
       "q": "repository",
       "sort": "created_desc",
       "entity_types": ["Project"]
     }'
```

Содержимое событий аудита

Каждое событие аудита содержит дату, время, данные об IP-адресе и учётной записи пользователя, а также всю необходимую информацию об области изменения, измененном объекте и о том, что именно было изменено.

Перечень событий аудита

В таблице приведены примеры системных сообщений. В production-среде события аудита содержат полную информацию в самом сообщении или в дополнительном JSON-поле с данными.

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`2fa_login_failed`	User 2fa login failed	Обнаружена неудачная попытка входа с двухфакторной аутентификацией.	
`access_approved`	User access was approved	Пользователю одобрен запрос на доступ в инстанс.	
`access_token_created`	Project/Group access token created	Создан токен доступа для проекта или группы.	
`access_token_revoked`	Project/Group access token revoked	Токен доступа был отозван.	
`added_gpg_key`	Added new gpg key to user	Пользователь добавил новый GPG-ключ.	
`added_ssh_key`	User added new ssh key	Пользователь добавил новый SSH-ключ.	
`application_created`	Application was created	Создано приложение (OAuth или интеграция).	
`application_deleted`	Application deleted	Приложение было удалено.	
`application_secret_renew`	Application secret renew	Обновлён секрет приложения.	
`application_updated`	Application Updated	Изменены параметры приложения.	
`ci_cd_job_token_removed_from_allowlist`	Disallow group to use job token	В проекте ограничено использование CI/CD Job Token	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
		определённой группой.	
`ci_cd_job_token_added_to_allowlist`	Allow group to use job token	В проекте разрешено использование CI/CD Job Token определённой группой.	
`ci_variable_created`	Ci variable `#{key}` created	Создана новая переменная CI/CD.	
`ci_variable_deleted`	Ci variable `#{key}` deleted	Удалена переменная CI/CD.	
`ci_variable_updated`	Ci variable updated (Value, Protected)	Изменено значение или параметры защиты переменной CI/CD.	
`deploy_key_created`	Deploy key added	Создан новый ключ развёртывания (deploy key) для проекта/инстанса.	
`deploy_key_deleted`	Deploy key was deleted	Удалён ключ развертывания.	
`deploy_key_disabled`	Deploy key disabled	Ключ развёртывания отключён.	
`deploy_key_enabled`	Deploy key enabled	Ключ развёртывания включён.	
`deploy_token_created`	Deploy token created	Создан токен развёртывания (deploy token) для доступа к данным.	
`deploy_token_deleted`	Deploy token deleted	Удалён токен развёртывания.	
	Deploy token revoked	Токен развёртывания был отозван	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`deploy_token_revoked`		пользователем или системой.	
`feature_flag_created`	Created feature flag with description	Создан новый флаг функций (feature flag).	
`feature_flag_deleted`	Feature flag was deleted	Флаг функций был удалён.	
`feature_flag_updated`	Feature flag was updated	Обновлены параметры флага функций.	
`group_created`	Group was created	Создана новая группа.	
`group_export_created`	Group file export was created	Создан файл экспорта группы.	
`group_invite_via_group_link_created`	Invited group to group	В группу приглашена другая группа через групповую ссылку.	
`group_invite_via_group_link_deleted`	Revoked group from group	Доступ группы, приглашенной по ссылке, был отозван.	
`group_invite_via_group_link_updated`	Group access changed	Изменены параметры доступа группы через групповую ссылку.	
`group_invite_via_project_group_link_created`	Invited group to project	В проект приглашена группа через групповую ссылку.	
`group_invite_via_project_group_link_deleted`	Revoked group from project	Доступ группы к проекту был отозван.	
	Group access for project changed		

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`group_invite_via_project_group_link_updated`		Изменены параметры доступа группы к проекту.	
`group_updated`	Group updated (visibility, 2FA grace period)	Изменения в настройках группы (видимость, безопасность, лимиты, политика доступа).	• `repository_size_limit` • `two_factor_grace_period` • `lfs_enabled` • `membership_lock` • `path` • `require_two_factor_authentication` • `request_access_enabled` • `shared_runners_minutes_limit` • `share_with_group_lock` • `mentions_disabled` • `max_personal_access_token_lifetime` • `visibility_level` • `name` • `description` • `project_creation_level` • `default_branch_protected` • `seat_control` • `duo_features_enabled`

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
			• `prevent_forking_outside_group` • `allow_mfa_for_subgroups` • `default_branch_name` • `resource_access_token_creation_allowed` • `new_user_signups_cap` • `show_diff_preview_in_email` • `enabled_git_access_protocol` • `runner_registration_enabled` • `allow_runner_registration_token` • `emails_enabled` • `service_access_tokens_expiration_enforced` • `enforce_ssh_certificates` • `disable_personal_access_tokens` • `remove_dormant_members` • `remove_dormant_members_period` • `prevent_sharing_groups_outside_hierarchy`

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
			• `default_branch_protection_defaults` • `wiki_access_level`
`impersonation_initiated`	User root impersonated another user	Администратор начал сессию от имени другого пользователя.	
`impersonation_stopped`	User root stopped impersonation	Администратор завершил сессию от имени другого пользователя.	
`instance_settings_updated`	Instance settings updated: Signup enabled turned on	Изменены глобальные настройки инстанса.	Все поля с настройками инстанса, кроме зашифрованных.
`login_failed`	Attempt to login failed	Неудачная попытка входа в систему.	
`manually_trigger_housekeeping`	Housekeeping task	Запущена задача обслуживания репозитория вручную.	
`member_permissions_created`	New member access granted	Пользователю предоставлен доступ (роль) к группе или проекту.	
`member_permissions_destroyed`	Member access revoked	Доступ пользователя к проекту или группе отозван.	
`member_permissions_updated`	Member access updated	Изменены права или срок действия доступа пользователя.	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`merge_request_closed_by_project_bot`	Merge request `#{merge_request.title}` closed by project bot	Запрос на merge закрыт системным ботом проекта.	
`merge_request_created_by_project_bot`	Merge request `#{merge_request.title}` created by project bot	Запрос на merge создан системным ботом проекта.	
`merge_request_merged_by_project_bot`	Merge request `#{merge_request.title}` merged by project bot	Запрос на merge обработан системным ботом проекта.	
`merge_request_reopened_by_project_bot`	Merge request `#{merge_request.title}` reopened by project bot	Запрос на merge переоткрыт системным ботом проекта.	
`omniauth_login_failed`	Omniauth login failed for `#{user}` `#{provider}`	Ошибка входа через внешний OAuth/ Omniauth-провайдер.	
`password_reset_failed`	Password reset failed	Неудачная попытка сброса пароля пользователем.	
`personal_access_token_issued`	Personal access token issued	Выпущен новый токен доступа (personal access token).	
`personal_access_token_revoked`	Personal access token revoked	Токен доступа был отозван.	
`pipeline_deleted`	Pipeline deleted	Конвейер CI/CD был удалён.	
`project_blobs_removal`	Project blobs removed	Массовое удаление объектов (blobs) из проекта.	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`project_created`	Project was created	Создан новый проект.	
`project_default_branch_changed`	Project default branch updated	Изменена ветка по умолчанию в проекте.	
`project_export_created`	Project export created	Создан файл экспорта проекта.	
`project_feature_updated`	Project features updated	Изменены уровни доступа к функциям проекта (issues, wiki и т. д.).	
`project_setting_updated`	Project settings updated	Изменены шаблоны merge commit и squash commit.	
`project_text_replacement`	Project text replaced	В проекте выполнена массовая замена текста.	
`project_topic_changed`	Project topic changed	Изменена тема проекта.	
`project_updated`	Project updated (name, namespace)	Изменены настройки проекта (имя, неймспейс, политики).	• `name` • `packages_enabled` • `reset_approvals_on_push` • `path` • `merge_requests_author_approval` • `merge_requests_disable_committers_approval` • `only_allow_merge_if_all_disc`

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
			`ussions_are_resolved` • `only_allow_merge_if_pipeline_succeeds` • `require_password_to_approve` • `disable_overriding_approvers_per_merge_request` • `repository_size_limit` • `project_namespace_id` • `namespace_id` • `printing_merge_request_link_enabled` • `resolve_outdated_diff_discussions` • `merge_requests_ff_only_enabled` • `merge_requests_rebase_enabled` • `remove_source_branch_after_merge` • `merge_requests_template` • `visibility_level` • `builds_access_level`

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
			• `container_registry_access_level` • `environments_access_level` • `feature_flags_access_level` • `forking_access_level` • `infrastructure_access_level` • `issues_access_level` • `merge_requests_access_level` • `metrics_dashboard_access_level` • `monitor_access_level` • `operations_access_level` • `package_registry_access_level` • `pages_access_level` • `releases_access_level` • `repository_access_level` • `requirements_access_level` • `security_and_compliance_access_level` • `snippets_access_level`

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
			• `wiki_access_level` • `merge_commit_template` • `squash_commit_template` • `runner_registration_enabled` • `show_diff_preview_in_email` • `selective_code_owner_removals`
`protected_branch_created`	Protected branch created	Создана защищённая ветка.	
`protected_branch_deleted`	Protected branch was deleted	Удалена защищённая ветка.	
`protected_branch_updated`	Protected branch was updated:	Обновлены правила защищённой ветки.	
`protected_tag_created`	Protected tag created	Создан защищённый тег.	
`protected_tag_deleted`	Protected tag was deleted	Удалён защищённый тег.	
`protected_tag_updated`	Protected tag updated:	Обновлены правила защищённого тега.	
`removed_gpg_key`	Removed gpg key from user	Удалён GPG-ключ пользователя.	
`removed_ssh_key`	User removed ssh key	Удалён SSH-ключ пользователя.	
`requested_password_reset`	User requested password change	Пользователь запросил сброс пароля.	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`revoked_gpg_key`	Revoked gpg key from user	GPG-ключ пользователя был отозван.	
`unban_user`	User was unban	Пользователь разблокирован (unban).	
`unblock_user`	User was unblocked	С пользователя снята блокировка (unblock).	
`user_access_locked`	User access locked	Учётная запись пользователя заблокирована.	
`user_access_unlocked`	User access unlocked	Учётная запись пользователя разблокирована.	
`user_activated`	User was activated	Учётная запись пользователя активирована.	
`user_banned`	User was banned	Пользователь забанен.	
`user_blocked`	User was blocked	Учётная запись пользователя заблокирована.	
`user_created`	User was created	Создан новый пользователь.	
`user_deactivated`	User was deactivated	Учётная запись пользователя деактивирована.	
`user_destroyed`	User was destroyed	Учётная запись пользователя удалена.	
`user_email_updated`	User email updated		

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
		Изменён адрес электронной почты пользователя.	
`user_logged_in`	User logged in	Успешный вход пользователя.	
`user_password_updated`	Password updated	Пароль пользователя изменён.	
`user_rejected`	User was rejected	Учётная запись пользователя отклонена (например, при регистрации).	
`user_removed_two_factor`	Two factor disabled	Пользователь отключил двухфакторную аутентификацию.	
`user_settings_updated`	User settings updated	Обновлены настройки профиля пользователя.	• `name` • `public_email` • `otp_secret` • `otp_required_for_login` • `admin` • `private_profile`
`user_signup`	User was registered	Пользователь зарегистрирован.	
`user_switched_to_admin_mode`	User switched to admin mode	Пользователь включил режим администратора.	
`user_username_updated`	Username updated	Изменено имя пользователя (username).	
`webhook_created`	Webhook was created		

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
		Создан вебхук для проекта, группы или инстанса.	
`webhook_destroyed`	System hook removed	Вебхук удалён.	
`group_deleted`	Group was deleted	Группа удалена.	
`project_deleted`	Project was deleted	Проект удалён.	
`logout`	User logged out	Пользователь вышел из системы.	
`unauthenticated_session`	Redirected to login	Система перенаправила неаутентифицированного пользователя на страницу входа.	
`ci_runners_bulk_deleted`	CI runner bulk deleted: Errors:	Массовое удаление CI runners.	
`ci_runner_registered`	CI runner created via API	Регистрация CI runner через API.	
`ci_runner_unregistered`	CI runner unregistered	Отмена регистрации CI runner.	
`ci_runner_token_reset`	CI runner registration token reset	Сброшен токен CI runner.	
`ci_runner_assigned_to_project`	CI runner assigned to project	Runner был привязан к проекту.	
`ci_runner_unassigned_from_project`	CI runner unassigned from project	Runner был отвязан от проекта.	
`ci_runner_created`	CI runner created via UI	Runner был создан через графический интерфейс.	
`package_registry_package_published`	`#{name}` package version `#{version}` has been published	В реестре пакетов опубликован новый пакет.	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
`package_registry_package_deleted`	package version `#{package.version}` has been deleted	Пакет удалён из реестра пакетов.	
`group_access_token_destroyed`	Group access token destroyed	Токен доступа группы был удалён.	
`group_access_token_revoked`	Group access token revoked	Токен доступа группы был отозван.	
`personal_access_token_destroyed`	Personal access token destroyed	Персональный токен доступа был удалён.	
`project_access_token_destroyed`	Project access token destroyed	Токен доступа проекта был удалён.	
`project_access_token_revoked`	Project access token revoked	Токен доступа проекта был отозван.	
`approval_rule_created`	Merge request\Project\Group approval rule created	Правило апрува для запроса на слияние, проекта или группы было создано.	
`approval_rule_updated`	Merge request\Project\Group approval rule updated	Правило апрува для запроса на слияние, проекта или группы было обновлено.	
`approval_rule_deleted`	Merge request\Project\Group approval rule deleted	Правило апрува для запроса на слияние, проекта или группы было удалено.	
`approval_rule_propagated`	Approval rules `#{rule_names}` propagated to `#{project_and_group_names}`	Правила апрува были распространены в эти проекты и группы.	
`variable_viewed_api`	User viewed all CI/CD variables via API for this scope or User	Все CI/CD переменные были просмотрены через	

Название	Системное сообщение	Назначение	Отслеживаемые атрибуты
	viewed CI/CD variable `#{variable.key}` via API	API для данной области или CI/CD переменная `#{variable.key}` была просмотрена через API.	
`group_unarchived`	User unarchived group	Группа была разархивирована.	
`group_archived`	User archived group	Группа была архивирована.	
`comment_deleted`	Comment deleted	Комментарий был удалён.	
`comment_created`	Comment created	Комментарий был создан.	
`comment_updated`	Comment updated	Комментарий был обновлён.	
`user_records_migrated_to_ghost`	User records migrated to ghost	Записи пользователя были перенесены ghost-пользователю.	
`user_enable_passkey`	User enable new passkey	Новый passkey был включён.	
`user_disable_passkey`	User disable passkey	Passkey был отключён.	
`push_rule_propagated`	Push rule propagated	Push-правило было распространено.	
`push_rule_updated`	Push rule updated	Существующее push-правило было обновлено.	

Мониторинг и лимиты

Мониторинг инстанса Deckhouse Code складывается из лимитов, заданных в веб-интерфейсе, и встроенного Prometheus каждого типа установки.

Управление ресурсами

Лимиты и настройки производительности инстанса задаются администратором в веб-интерфейсе:

1. Настройте лимиты на репозитории и артефакты:
    - Перейдите в «Admin» → «Настройки» → «Аккаунт и ограничения».
2. Настройте производительность системы:
    - Перейдите в «Admin» → «Настройки» → «Сеть».

Встроенный мониторинг

У каждого типа установки свой источник метрик или оповещений; вкладка называет тип.

Пакет для ОС Linux

Prometheus и экспортёры (node, postgres, redis и другие) устанавливаются вместе с пакетом и слушают только адрес `127.0.0.1`, наружу порты не открыты. Проверьте их состояние:

```
sudo gitlab-ctl status | grep -E "prometheus|exporter"
# Готовность встроенного Prometheus.
curl -s http://127.0.0.1:9090/-/ready
```

Чтобы метрики забирала внешняя система мониторинга, задайте адрес прослушивания Prometheus в файле `/etc/gitlab/gitlab.rb`:

```
prometheus['listen_address'] = '0.0.0.0:9090'
```

Примените настройку:

```
sudo gitlab-ctl reconfigure
```

Ограничьте доступ к порту `9090` на файрволе адресами системы мониторинга.

firewalld (РЕД ОС)

```
sudo firewall-cmd --permanent --new-zone=monitoring
sudo firewall-cmd --permanent --zone=monitoring --add-source=<MONITORING_IP>/32
sudo firewall-cmd --permanent --zone=monitoring --add-port=9090/tcp
# Зона принимает весь трафик с этого адреса, поэтому открываются и базовые
сервисы.
sudo firewall-cmd --permanent --zone=monitoring --add-service={ssh,http,https}
sudo firewall-cmd --reload
```

ufw (Ubuntu)

```
sudo ufw allow from <MONITORING_IP> to any port 9090 proto tcp
```

Omnibus Docker

В контейнере работают тот же Prometheus и те же экспортёры, кроме `node_exporter`: образ его отключает. Образ объявляет только порты `80`, `443` и `22`, поэтому порт Prometheus нужно опубликовать при создании контейнера: добавьте `-p 9090:9090` в команду `docker run`, у запущенного контейнера список опубликованных портов не меняется.

Задайте `prometheus['listen_address']` в файле `/etc/gitlab/gitlab.rb` в томе с конфигурацией; значение, его смысл и ограничение доступа к порту описаны на вкладке «Пакет для ОС Linux». Настройка применится при следующем старте контейнера: команда `gitlab-ctl reconfigure` выполняется при каждом запуске.

Helm-чарт

Helm-чарт создаёт объект ServiceMonitor (кастомный ресурс Prometheus Operator) для каждого компонента, который отдаёт метрики Prometheus, если `metrics.serviceMonitor.enabled` равно `true`. `metrics.serviceMonitor.labels` задаёт лейблы, по которым его находит Prometheus.

```
metrics:
  serviceMonitor:
    enabled: true
    labels:
      release: <PROMETHEUS_RELEASE_LABEL>
```

Экземпляр Prometheus, настроенный на отслеживание объектов ServiceMonitor с такими лейблами — в пространстве имён `code` или во всём кластере, в зависимости от его собственной настройки, — обнаруживает их и собирает метрики без дополнительной настройки со стороны релиза.

Модуль Deckhouse Kubernetes Platform

Модуль поставляет собственные правила оповещений: список оповещений и действия по каждому из них приведены в разделе «Алерты» документации модуля.

Режим обслуживания

Режим обслуживания позволяет администраторам свести операции записи к минимуму на время проведения регламентных работ. Его основная цель — заблокировать все внешние действия, изменяющие внутреннее состояние базы данных PostgreSQL, а также файлов, Git-репозиториев и хранилищ образов контейнеров.

Когда режим обслуживания включён, уже запущенные действия завершаются за короткое время, поскольку новые действия перестают поступать, а изменения внутреннего состояния минимальны. В таком состоянии проще выполнять различные регламентные работы.

Режим обслуживания разрешает большинство внешних действий, которые не изменяют внутреннее состояние. На уровне HTTP блокируются запросы с методами `POST`, `PUT`, `PATCH` и `DELETE`.

Включение режима обслуживания

Включите режим обслуживания от имени администратора одним из следующих способов:

- **Через веб-интерфейс**:
  1. Перейдите в раздел «Admin» → «Настройки» → «Основные».
  2. Разверните раздел «Режим обслуживания» и установите флажок «Включить режим обслуживания».
  3. При необходимости добавьте текст сообщения, которое будет отображаться в баннере.
  4. Нажмите «Сохранить изменения».
- **Через API**:

  Выполните следующий запрос:

  ```
  curl --request PUT --header "PRIVATE-TOKEN: $ADMIN_TOKEN" \
    "<INSTANCE_URL>/api/v4/application/settings?maintenance_mode=true"
  ```

Отключение режима обслуживания

Отключите режим обслуживания одним из следующих способов:

- **Через Веб-интерфейс**:
  1. Перейдите в раздел «Admin» → «Настройки» → «Основные».
  2. Разверните раздел «Режим обслуживания» и снимите флажок «Включить режим обслуживания».
  3. Нажмите «Сохранить изменения».
- **Через API**:

  Выполните следующий запрос:

  ```
  curl --request PUT --header "PRIVATE-TOKEN: $ADMIN_TOKEN" \
    "<INSTANCE_URL>/api/v4/application/settings?maintenance_mode=false"
  ```

Особенности работы в режиме обслуживания

Когда режим обслуживания включён, в верхней части интерфейса отображается баннер. При необходимости для него можно задать собственное сообщение.

При попытке выполнить недопустимую операцию записи пользователь получит сообщение об ошибке.

> В некоторых случаях визуальная реакция на действие может вводить в заблуждение. Например, при добавлении проекта в избранное кнопка «В избранное» меняется на «Убрать из избранного». Это связано с тем, что интерфейс обновляется до получения результата `POST`-запроса.

Действия администратора

Администраторы могут по-прежнему редактировать настройки приложения. Это позволяет отключить режим обслуживания после его включения.

Аутентификация

Все пользователи могут входить в систему и выходить из неё, но создание новых пользователей блокируется.

Если на время обслуживания запланирована синхронизация LDAP, она будет завершена с ошибкой, поскольку создание пользователей отключено. По той же причине будет завершено с ошибкой создание пользователей на основе SAML и других провайдеров OmniAuth.

> Когда создание пользователя блокируется при входе через LDAP, пользователь видит сообщение о том, что инстанс находится в режиме обслуживания (только для чтения), а не стандартную ошибку запрета доступа.

Операции с Git

Все операции чтения Git продолжают работать, включая `git clone` и `git pull`.

Все операции записи завершаются с ошибкой — как через командную строку, так и через web-редактор. При этом отображается следующее сообщение:

```
Git push is not allowed because this instance is currently in (read-only) maintenance mode.
```

Запросы на слияние и задачи

Все операции записи, не перечисленные в исключениях, завершаются с ошибкой. Например, пользователь не может изменить запрос на слияние или задачу.

Входящая почта

Создание ответов на задачи, новых задач (включая задачи Service Desk) и запросов на слияние по электронной почте завершается с ошибкой. Обработчики входящей почты пропускают обработку, пока включён режим обслуживания.

Исходящая почта

Уведомления по электронной почте продолжают приходить, но письма, для отправки которых требуется запись в базу данных (например, для сброса пароля), не доставляются.

REST API

Для большинства JSON-запросов методы `POST`, `PUT`, `PATCH` и `DELETE` блокируются. API возвращает ответ `503` с сообщением об ошибке `system is in maintenance mode` и заголовком `Retry-After`.

Разрешены только следующие запросы:

HTTP-запрос	Разрешённые маршруты	Примечания
`POST`	`/admin/application_settings/general`	Обновление настроек приложения в интерфейсе администратора
`PUT`	`/api/v4/application/settings`	Обновление настроек приложения через API
`POST`	`/users/sign_in`	Вход пользователей в систему
`POST`	`/users/sign_out`	Выход пользователей из системы
`POST`	`/oauth/token`	Получение OAuth-токенов
`POST`	`/admin/session`, `/admin/session/destroy`	Использование режима администратора (Admin Mode)
`POST`	Пути, оканчивающиеся на `/compare`	Сравнение ревизий Git
`POST`	`*.git/git-upload-pack`	Выполнение `git pull` и `git clone`
`POST`	`/api/v4/internal`	Маршруты внутреннего API
`POST`	`/admin/sidekiq`	Управление фоновыми задачами в разделе «Admin»

GraphQL API

Запросы `POST /api/graphql` разрешены, но GraphQL-мутации блокируются со следующим сообщением об ошибке: `You cannot perform write operations on a read-only instance`.

Непрерывная интеграция

Во время обслуживания действуют следующие ограничения:

- новые задачи и конвейеры не запускаются ни по расписанию, ни вручную;
- задачи, которые уже выполнялись на момент включения режима обслуживания, продолжают отображаться в интерфейсе со статусом `running`, даже если они завершились на раннере;
- задачи в состоянии `running` не завершаются по тайм-ауту после превышения лимита времени проекта;

- конвейеры нельзя запустить, перезапустить или отменить. Новые задачи в существующих конвейерах также не создаются;
- статус раннеров в разделе «Admin» → «Раннеры» не обновляется.

После отключения режима обслуживания новые задачи снова начинают выполняться.

Задачи, которые были в состоянии `running` до включения режима обслуживания, возобновляются, а их логи снова обновляются.

> После отключения режима обслуживания рекомендуется перезапустить конвейеры, которые были в состоянии `running` на момент включения режима обслуживания.

Развёртывания

Развёртывания не выполняются, поскольку связанные с ними конвейеры не завершаются. Отключите автоматические развёртывания на время режима обслуживания и снова включите их после завершения работ.

Хранилище образов контейнеров

Команда `docker push` завершается следующей ошибкой:

```
denied: requested access to the resource is denied
```

При этом операция `docker pull` продолжает работать.

Реестр пакетов

Реестр пакетов позволяет устанавливать пакеты, но не публиковать их.

Фоновые задачи

Фоновые задачи (включая cron-задачи и задачи Sidekiq) продолжают выполняться как обычно, поскольку они не отключаются автоматически.

Так как фоновые задачи выполняют операции, которые могут изменить внутреннее состояние инстанса, на время режима обслуживания может потребоваться отключить часть из них или все задачи целиком.

Чтобы отслеживать очереди и отключать задачи:

1. Перейдите в раздел «Admin» → «Мониторинг» → «Фоновые задачи».
2. В панели Sidekiq выберите «Cron» и отключите задачи по отдельности или все сразу, нажав «Disable All».

Управление инцидентами

Функции управления инцидентами ограничены. Создание алертов и инцидентов полностью приостановлено, а связанные с ними уведомления и оповещения не отправляются.

Флаги функций

- Флаги функций разработки нельзя включать или отключать через API, но можно переключать через консоль Rails.
- Сервис флагов функций продолжает отвечать на запросы проверки состояния флагов, но переключать сами флаги нельзя.

Расширенный поиск

Документация для администратора по расширенному поиску в Deckhouse Code: эксплуатация индексации после подключения OpenSearch. В модуле Deckhouse Kubernetes Platform OpenSearch подключается в ресурсе CodeInstance, это описано в разделе «Расширенный поиск» документации модуля. Инструкции для пользователей — в руководстве пользователя.

Эксплуатация

Управление индексацией, мониторинг и устранение неполадок.

Управление на уровне инстанса

Чтобы управлять индексацией на уровне инстанса, перейдите в «Admin» → «Настройки» → «Поиск».

Раздел доступен после подключения OpenSearch.

Приостановка индексации

Установите флаг «Приостановить индексирование OpenSearch», чтобы приостановить фоновые задачи индексации и переиндексации. Чтобы снова включить индексацию, отключите флаг «Приостановить индексирование OpenSearch». После этого Sidekiq pause control автоматически возобновит работу в течение нескольких минут.

Режим индексации веток

В проектах могут использоваться следующие режимы индексации веток:

Режим	Описание
Только ветка по умолчанию	Индексируется только ветка по умолчанию для всех проектов
Разрешить регулярное выражение для веток на уровне проекта	Для проектов можно задать regex для индексации дополнительных веток

⚠ После смены режима индексации веток выполните ручную переиндексацию кода. Результаты поиска могут быть неполными до завершения индексации.

Статус индексов OpenSearch

На странице отображается таблица индексов:

Суффикс индекса	Область поиска
`code`	Код (`blobs`)
`commits`	Коммиты
`wiki`	Wiki-страницы
`notes`	Комментарии
`milestones`	Этапы (milestones)
`merge-requests`	Запросы на слияние
`work-items`	Задачи (work items)

Имя индекса в OpenSearch состоит из общего префикса инстанса и суффикса из таблицы, например `deckhouse-development-code`.

Для каждого индекса показаны: имя в OpenSearch, наличие, количество документов, состояние индекса.

Операции над индексами

На той же странице доступны следующие операции:

- **Переиндексировать** — переиндексация одного индекса (удаляет существующие документы и ставит фоновые задачи).
- **Переиндексировать все индексы** — переиндексация всех индексов.

⚠ Операция «Переиндексировать» удаляет существующие документы. Результаты поиска могут быть неполными до завершения фоновой индексации.

Индекс `commits` переиндексируется вместе с индексом `code`: отдельной операции для коммитов нет. По этой же причине у коммитов нет отдельного значения `schema_class`.

Операции над индексами также можно выполнять с помощью запросов к API OpenSearch.

Мониторинг

Метрики

Обращения к OpenSearch учитываются в Prometheus отдельно для HTTP-запросов (поиск в UI и API) и для фоновых задач Sidekiq (индексация). Имена метрик содержат `elasticsearch` — это историческое название в GitLab, метрики относятся к OpenSearch.

HTTP-запросы (поиск пользователей):

Метрика	Описание
`http_elasticsearch_requests_total`	Число обращений к OpenSearch за один HTTP-запрос
`http_elasticsearch_requests_duration_seconds`	Суммарное время обращений к OpenSearch за один HTTP-запрос
`http_elasticsearch_requests_failed_total`	Число неудачных обращений за один HTTP-запрос (ошибки подключения или авторизации) — **добавлено в Deckhouse Code**

Sidekiq (фоновая индексация):

Метрика	Описание
`sidekiq_elasticsearch_requests_total`	Число обращений к OpenSearch за выполнение одного Sidekiq-задания
`sidekiq_elasticsearch_requests_duration_seconds`	Суммарное время обращений к OpenSearch за выполнение одного Sidekiq-задания
`sidekiq_elasticsearch_requests_failed_total`	Число неудачных обращений за выполнение одного Sidekiq-задания (ошибки подключения или авторизации) — **добавлено в Deckhouse Code**

Индексатор репозиториев (`Search::RepositoryIndexerWorker` — код, коммиты, wiki):

Метрика	Лейблы	Описание
`search_repository_indexer_starts_total`	`indexer_class`	Число запусков индексации после прохождения проверки `advanced_search_enabled`
`search_repository_indexer_runs_total`	`outcome`, `indexer_class`	Число завершённых запусков после получения exclusive lock (`outcome`: `success` или `error`)
`search_repository_indexer_duration_seconds`	`outcome`, `indexer_class`	Длительность фазы индексации под exclusive lock
`search_repository_indexer_lock_contention_total`	—	Число случаев, когда lock не получен и задание перенесено

Значение `indexer_class` — тип выполняемой индексации:

`indexer_class`	Когда используется
`Search::RepositoryIndexer::IncrementalIndexService`	Инкрементальная индексация после изменений в репозитории
`Search::RepositoryIndexer::FullIndexService`	Полная переиндексация (force)
`Search::RepositoryIndexer::MaintainsService`	Обновление индекса по событию
`Search::RepositoryIndexer::DeleteService`	Удаление документов из индекса (пустой или удалённый репозиторий/wiki)

Рост `search_repository_indexer_lock_contention_total` — признак конкуренции за lock между заданиями одного проекта. Рост `search_repository_indexer_runs_total{outcome="error"}` — ошибки go-indexer или сервисов индексации; детали в логах Sidekiq.

Метрики `*_failed_total` увеличиваются при ошибках подключения к OpenSearch или ошибках авторизации. Рост `*_failed_total` указывает на недоступность OpenSearch или на использование неверных данных для доступа. Рост `*_duration_seconds` при стабильном `*_total` — на медленные ответы OpenSearch.

Для мониторинга индексации репозиториев ориентируйтесь на `search_repository_indexer_*`, для обращений к OpenSearch из Sidekiq — на `sidekiq_elasticsearch_*`. Для мониторинга пользовательского поиска — на `http_elasticsearch_*`.

На странице «Admin» → «Настройки» → «Поиск» виджет прогресса индексации показывает число оставшихся задач переиндексации. Те же данные доступны через эндпоинт `indexing_queue_stats`.

Очередь Sidekiq

Задачи индексации OpenSearch выполняются в отдельной очереди `global-search-indexing`, а не в общей очереди `default`. Маршрутизация настраивается правилом Sidekiq: все воркеры с категорией `fe_global_search` попадают в эту очередь. Отдельная очередь изолирует нагрузку индексации от остальных фоновых задач Deckhouse Code.

Cron-задачи

Для автоматизации индексации регулярно выполняются следующие cron-задачи:

Расписание	Назначение
Каждую минуту	Индексация комментариев — обрабатывает накопившуюся очередь изменений notes
Ежедневно в 03:00	Запускает индексацию проектов

Cron-задачи не выполняют индексацию напрямую: они запускают или возобновляют соответствующие воркеры в очереди `global-search-indexing`.

Логи

Задачи индексации OpenSearch пишутся в логи Sidekiq. Для фильтрации используйте имя очереди `global-search-indexing`.

Устранение неполадок

OpenSearch недоступен

- На странице «Admin» → «Настройки» → «Поиск» появится сообщение о невозможности подключения.
- Поиск вернёт ошибку.

Адрес и учётные данные OpenSearch задаются в конфигурации типа установки, а не в интерфейсе:

Пакет для ОС Linux

Ключ `gitlab_rails['fe_search']` в файле `/etc/gitlab/gitlab.rb`, применяется командой `sudo gitlab-ctl reconfigure`.

Omnibus Docker

Тот же ключ в файле `/etc/gitlab/gitlab.rb` в томе с конфигурацией, применяется командой `docker exec code gitlab-ctl reconfigure`.

Helm-чарт

Значения `global.appConfig.search` релиза, применяются командой `helm upgrade`.

Модуль Deckhouse Kubernetes Platform

Ресурс CodeInstance, описан в разделе «Расширенный поиск» документации модуля.

Неполные результаты поиска

- Дождитесь завершения фоновой индексации (виджет прогресса на странице «Admin» → «Настройки» → «Поиск»).
- Запустите переиндексацию на уровне проекта или операцию «Переиндексировать» для нужного индекса в «Admin» → «Настройки» → «Поиск».

Задачи индексации не появляются

Если новые задачи не ставятся в очередь `global-search-indexing`:

1. Проверьте, не включена ли пауза (включён флаг «Приостановить индексирование OpenSearch») в «Admin» → «Настройки» → «Поиск». Снимите флаг и дождитесь возобновления (cron-задача выполняется каждые 5 минут).
2. Если пауза снята, а задачи по-прежнему не появляются, выполните очистку Redis — возможны зависшие lease или duplicate-ключи Sidekiq:

   Пакет для ОС Linux

   ```
   sudo gitlab-rails runner /opt/gitlab/embedded/service/gitlab-rails/fe/
   scripts/clear_search_opensearch_worker_redis.rb
   ```

   Omnibus Docker

   ```
   docker exec code gitlab-rails runner /opt/gitlab/embedded/service/gitlab-
   rails/fe/scripts/clear_search_opensearch_worker_redis.rb
   ```

Helm-чарт

```
d8 k -n code exec deploy/code-toolbox -- gitlab-rails runner /srv/gitlab/fe/
scripts/clear_search_opensearch_worker_redis.rb
```

Модуль Deckhouse Kubernetes Platform

```
d8 k -n d8-code exec -it -c toolbox deploy/toolbox -- gitlab-rails runner /
srv/gitlab/fe/scripts/clear_search_opensearch_worker_redis.rb
```

Скрипт снимает exclusive lease для `Search::RepositoryIndexerWorker`, concurrency limit и dedup-ключи очереди `global-search-indexing`.

API OpenSearch

В этом разделе описаны административные OpenSearch-эндпоинты Deckhouse Code. Параметры пользовательского поиска — [в разделе «API поиска»](#).

POST /api/v4/admin/opensearch/recreate_indices

Синхронно пересоздаёт индекс(ы) OpenSearch и ставит фоновые задачи повторной индексации. Чтобы пересоздать (переиндексировать) все индексы, выполните запрос без тела. Чтобы пересоздать конкретный индекс, укажите его в теле запроса.

Права доступа: только администратор (`authenticated_as_admin!`).

Тело запроса

Поле	Тип	Обязательное	Допустимые значения
`schema_class`	string	Да	`recreate_all`, `Search::Opensearch::IndicesSchema::Code`, `Search::Opensearch::IndicesSchema::Wiki`, `Search::Opensearch::IndicesSchema::Note`, `Search::Opensearch::IndicesSchema::Milestone`, `Search::Opensearch::IndicesSchema::WorkItem`, `Search::Opensearch::IndicesSchema::MergeRequest`

Ответы

Тексты полей `message` в примерах ниже возвращаются API на английском языке.

- `202 Accepted`

```
{
  "message": "OpenSearch indices were reset; reindex jobs were enqueued."
}
```

- `400 Bad Request` (например, OpenSearch выключен или сервис вернул ошибку)

```
{
  "message": "OpenSearch is disabled"
}
```

Пример запроса

Этот запрос пересоздает все индексы:

```
curl --request POST \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --header "Content-Type: application/json" \
  --data '{"schema_class":"recreate_all"}' \
  --url "https://gitlab.example.com/api/v4/admin/opensearch/recreate_indices"
```

GET /api/v4/admin/opensearch/indexing_queue_stats

Возвращает статистику Sidekiq-очереди индексации OpenSearch.

Права доступа: пользователь с правом `read_admin_search_indexing_queue_stats` на `:global`.

Ответ (200 OK)

```
{
  "total": 42,
  "updated_at": "2026-07-01T12:34:56.789Z"
}
```

Поля ответа:

- `total` — общее количество задач индексации в очереди;
- `updated_at` — timestamp ISO8601 с миллисекундами (или `null`).

Пример запроса

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://gitlab.example.com/api/v4/admin/opensearch/indexing_queue_stats"
```

Резервное копирование и восстановление

Резервная копия Deckhouse Code состоит из архива с данными инстанса и файлов, которые в архив не попадают. Состав архива одинаков для пакета для ОС Linux и Omnibus Docker: обе поставки работают на одном движке, и копию в них создаёт команда `gitlab-backup`. В контейнере те же команды выполняются через `docker exec`, а архив записывается на том данных.

Что входит в резервную копию

Команда `gitlab-backup create` собирает в один архив:

- репозитории;
- базу данных;
- загруженные файлы (uploads);

- артефакты CI;
- LFS-объекты;
- реестр пакетов.

Что хранится отдельно

Эти файлы в архив не попадают, копируйте их вместе с каждым архивом:

- `/etc/gitlab/gitlab-secrets.json` — ключи шифрования базы данных, без которых при восстановлении теряются настройки двухфакторной аутентификации, токены и зашифрованные переменные CI;
- `/etc/gitlab/gitlab.rb` — конфигурация инстанса;
- сертификаты и ключи из каталога `/etc/gitlab/ssl/`.

Файлы секретов храните с тем же уровнем защиты, что и пароли.

Пакет для ОС Linux

Файлы лежат в каталоге `/etc/gitlab/` на машине инстанса. Копируйте их вместе с каждым архивом.

Omnibus Docker

Файлы находятся на томе конфигурации. Архивируйте их отдельной командой:

```
docker exec -t code gitlab-ctl backup-etc
```

Команда записывает файл `/etc/gitlab/config_backup/gitlab_config_<TIMESTAMP>_<DATE>.tar` с содержимым каталога `/etc/gitlab`: `gitlab.rb`, `gitlab-secrets.json` и TLS-сертификатами. Каталог находится на томе конфигурации, на хосте это `/srv/code/config/config_backup`.

Создание резервной копии

Пакет для ОС Linux

Копия снимается одной командой:

```
sudo gitlab-backup create
```

Архив появится в каталоге `/var/opt/gitlab/backups/` под именем вида `<TIMESTAMP>_<DATE>_<ENGINE_VERSION>_gitlab_backup.tar`. Версия в имени — версия встроенного движка, она не совпадает с версией пакета `deckhouse-code`, поэтому при снятии копии записывайте версию пакета рядом с архивом:

```
# РЕД ОС.
rpm -q deckhouse-code
# Ubuntu.
dpkg -l deckhouse-code
```

Omnibus Docker

Выполните создание копии в работающем контейнере:

```
docker exec -t code gitlab-backup create
```

Архив появится в каталоге `/var/opt/gitlab/backups` внутри контейнера, с томами из раздела «Быстрый старт» на хосте это `/srv/code/data/backups`. Копируйте его за пределы хоста вместе с архивом конфигурации и секретов.

Helm-чарт

Утилита `backup-utility` запускается внутри пода toolbox, который разворачивает чарт:

```
d8 k -n code exec deploy/code-toolbox -- backup-utility
```

Команда создаёт архив резервной копии и загружает его в бакет объектного хранилища, настроенный через `global.appConfig.object_store` (раздел «Объектное хранилище»).

Срок хранения архивов и расписание

Архивы старше `backup_keep_time` удаляются при создании следующей резервной копии.

Пакет для ОС Linux

Срок хранения задаётся в файле `/etc/gitlab/gitlab.rb` в секундах:

```
# Семь дней.
gitlab_rails['backup_keep_time'] = 604800
```

Изменение вступает в силу после `sudo gitlab-ctl reconfigure`.

Регулярное снятие копии задаётся заданием cron, например ежедневно в 02:00:

```
echo '0 2 * * * root /opt/gitlab/bin/gitlab-backup create CRON=1' \
  | sudo tee /etc/cron.d/deckhouse-code-backup
```

Omnibus Docker

Срок хранения архивов задаёт параметр `gitlab_rails['backup_keep_time']` в конфигурации на томе конфигурации; его значение и поведение описаны на вкладке «Пакет для ОС Linux».

Для расписания выполняйте ту же команду из `cron` на хосте:

```
echo '0 2 * * * root /usr/bin/docker exec -t code gitlab-backup create CRON=1' \
  | sudo tee /etc/cron.d/deckhouse-code-backup
```

Копирование за пределы машины

Регулярно копируйте свежие архивы на отдельный сервер.

Пакет для ОС Linux

Забирайте архивы из каталога `/var/opt/gitlab/backups/`, например командой `rsync`. Вместе с архивом копируйте файлы `/etc/gitlab/gitlab-secrets.json` и `/etc/gitlab/gitlab.rb`: без них архив восстанавливается не полностью.

Omnibus Docker

Забирайте архивы из каталога `/srv/code/data/backups` на хосте вместе с архивом конфигурации из каталога `/srv/code/config/config_backup`: без конфигурации и секретов архив восстанавливается не полностью.

Восстановление

Восстановление заменяет данные инстанса содержимым архива. До запуска проверьте, что выполнены условия:

- установлена ровно та версия, на которой сделана резервная копия;
- файлы `gitlab.rb` и `gitlab-secrets.json` исходного инстанса лежат на своих местах. При переносе на новый сервер скопируйте их до восстановления и выполните настройку;
- сертификаты и ключи из каталога `/etc/gitlab/ssl/` лежат на своих местах: `gitlab.rb` ссылается на них по путям, и без этих файлов настройка завершается ошибкой запуска nginx;
- на диске достаточно свободного места для распакованного архива.

В командах для пакета для ОС Linux и Omnibus Docker `<BACKUP_NAME>` — имя файла архива без суффикса `_gitlab_backup.tar` : для архива `1784800000_2026_07_23_19.1.2_gitlab_backup.tar` это `1784800000_2026_07_23_19.1.2` .

Пакет для ОС Linux

Проверьте установленную версию пакета командой `rpm -q deckhouse-code` на РЕД ОС или `dpkg -l deckhouse-code` на Ubuntu. Если конфигурация и секреты перенесены с другой машины, после копирования выполните `sudo gitlab-ctl reconfigure`.

1. Скопируйте архив в каталог резервных копий и передайте файл пользователю `git`:

   ```
   sudo cp <BACKUP_NAME>_gitlab_backup.tar /var/opt/gitlab/backups/
   sudo chown git:git /var/opt/gitlab/backups/<BACKUP_NAME>_gitlab_backup.tar
   ```

2. Остановите сервисы, которые пишут в базу данных; остальные сервисы продолжают работать:

   ```
   sudo gitlab-ctl stop puma
   sudo gitlab-ctl stop sidekiq
   ```

3. Восстановите данные из архива:

   ```
   sudo gitlab-backup restore BACKUP=<BACKUP_NAME>
   ```

   Команда дважды запросит подтверждение: перед пересозданием файла `authorized_keys` и перед удалением текущих таблиц базы данных. Оба раза ответьте `yes`.

4. Перезапустите инстанс и выполните самопроверку:

   ```
   sudo gitlab-ctl reconfigure
   sudo gitlab-ctl restart
   sudo gitlab-rake gitlab:check SANITIZE=true
   ```

Omnibus Docker

Проверьте, что контейнер запущен с тегом образа той версии, на которой сделана копия, и что файлы конфигурации и секретов исходного инстанса лежат на томе конфигурации.

1. Скопируйте архив в каталог резервных копий на томе данных и назначьте ему владельца, под которым работает инстанс:

   ```
   sudo cp <BACKUP_NAME>_gitlab_backup.tar /srv/code/data/backups/
   docker exec code chown git:git /var/opt/gitlab/backups/
   <BACKUP_NAME>_gitlab_backup.tar
   ```

2. Остановите сервисы, которые пишут в базу данных:

   ```
   docker exec code gitlab-ctl stop puma
   docker exec code gitlab-ctl stop sidekiq
   ```

3. Запустите восстановление. Команда дважды запросит подтверждение — перед пересозданием файла `authorized_keys` и перед удалением текущих таблиц базы данных; оба раза ответьте `yes`:

   ```
   docker exec -it code gitlab-backup restore BACKUP=<BACKUP_NAME>
   ```

4. Примените конфигурацию и перезапустите контейнер:

   ```
   docker exec code gitlab-ctl reconfigure
   docker restart code
   ```

5. Проверьте инстанс:

   ```
   docker exec code gitlab-ctl status
   docker exec code gitlab-rake gitlab:check SANITIZE=true
   ```

Helm-чарт

1. Переведите инстанс в режим обслуживания.
2. Убедитесь, что бакет объектного хранилища, настроенный через `global.appConfig.object_store`, содержит архив, который нужно восстановить.
3. Выполните команду восстановления внутри пода toolbox:

   ```
   d8 k -n code exec deploy/code-toolbox -- backup-utility restore --backup-id <BACKUP_ID>
   ```

4. Дождитесь завершения команды и убедитесь, что все поды находятся в состоянии `Running`:

   ```
   d8 k -n code get pods
   ```

5. Выведите инстанс из режима обслуживания.

После восстановления войдите в веб-интерфейс и склонируйте репозиторий, чтобы убедиться, что инстанс работает со своими данными.

Модуль Deckhouse Kubernetes Platform

В поставке модулем резервные копии создаёт оператор по расписанию из ресурса CodeInstance и выгружает их в объектное хранилище: состав архива, предусловия восстановления и матрица команд приведены в разделе «Бекапы и восстановление» документации модуля.

Обновление

Обновление Deckhouse Code устанавливает новую версию поверх текущей: конфигурация и данные сохраняются, настройка и миграции базы данных запускаются автоматически. Правила ниже действуют для всех типов установки.

Порядок версий

Обновляйтесь последовательно, без пропуска мажорных версий: миграции базы данных рассчитаны на переход с предыдущей мажорной версии. Если между текущей и целевой версией несколько мажорных релизов, устанавливайте их по очереди и дожидайтесь между шагами завершения фоновых миграций. Путь обновления для вашей пары версий уточните у вендора при получении новой версии.

Фоновые миграции

Обновление применяет основные миграции базы данных и не дожидается фоновых: они продолжают выполняться на уже работающем инстансе. Перед следующим обновлением все фоновые миграции должны завершиться, иначе обновление прервётся с ошибкой.

Чтобы проверить состояние миграций, в правом верхнем углу выберите «Admin», затем в левой панели перейдите в «Мониторинг» → «Фоновые миграции». Все миграции должны быть в конечном статусе: в списке не осталось задач в очереди и в процессе завершения, а задач, завершившихся с ошибкой, нет. На типовом инстансе это занимает минуты, на больших объёмах данных — дольше.

Перед обновлением

Пакет для ОС Linux

Создайте резервную копию по разделу «Резервное копирование и восстановление» и проверьте, что есть свежие копии файлов `/etc/gitlab/gitlab-secrets.json` и `/etc/gitlab/gitlab.rb`:

```
sudo gitlab-backup create
```

Omnibus Docker

Создайте резервную копию данных и конфигурации по разделу «Резервное копирование и восстановление» и убедитесь, что архивы хранятся за пределами хоста.

Запишите тег, с которым запущен контейнер: к нему возвращает откат.

```
docker inspect --format '{{.Config.Image}}' code
```

Helm-чарт

Каждый релиз чарта указывает версию Deckhouse Code, которую он разворачивает. Проверьте её перед обновлением:

```
helm show chart deckhouse-code --version <CHART_VERSION>
```

Создайте резервную копию по разделу «Резервное копирование и восстановление»: она нужна для отката изменения схемы базы данных.

Обновление инстанса

Пакет для ОС Linux

Конфигурация в файле `/etc/gitlab/gitlab.rb` и данные сохраняются.

1. Установите новый пакет; настройка и миграции запустятся автоматически:

   ```
   # РЕД ОС.
   sudo rpm -Uvh ./deckhouse-code-<NEW_VERSION>.el8.x86_64.rpm
   # Ubuntu.
   sudo apt install -y ./deckhouse-code_<NEW_VERSION>_amd64.deb
   ```

2. Проверьте состояние инстанса и установленную версию:

   ```
   # Все сервисы в состоянии run.
   sudo gitlab-ctl status
   # Установленная версия на РЕД ОС.
   rpm -q deckhouse-code
   # Установленная версия на Ubuntu.
   dpkg -l deckhouse-code
   # Ожидается health=200.
   curl --cacert /etc/gitlab/ssl/<HOSTNAME>.crt \
        --resolve '<HOSTNAME>:443:127.0.0.1' -o /dev/null \
        -w "health=%{http_code}\n" https://<HOSTNAME>/-/health
   ```

На время миграций инстанс отвечает медленнее; на типовом инстансе обновление занимает минуты.

Omnibus Docker

Контейнер заменяется контейнером из нового тега образа; тома остаются, а данные и конфигурация инстанса на них переносятся при первом запуске. Каждый тег называет версию, отдельного тега для последней версии нет, поэтому целевая версия указывается явно.

1. Получите образ целевой версии:

   ```
   docker pull <REGISTRY>/<FLAVOR>:<VERSION>
   ```

2. Остановите и удалите контейнер. Тома — это каталоги хоста, они остаются на месте:

   ```
   docker stop code
   docker rm code
   ```

3. Создайте контейнер из нового тега с тем же именем, портами, томами и переменными окружения, что и в разделе «Быстрый старт»:

   ```
   docker run -d --name code \
     --shm-size 256m \
     -p 80:80 -p 443:443 -p 22:22 \
     -e GITLAB_OMNIBUS_CONFIG="external_url 'http://<HOSTNAME>'" \
     -v /srv/code/config:/etc/gitlab \
     -v /srv/code/logs:/var/log/gitlab \
     -v /srv/code/data:/var/opt/gitlab \
     <REGISTRY>/<FLAVOR>:<VERSION>
   ```

4. Следите за запуском. Миграции базы данных выполняются во время настройки, и до её окончания инстанс отвечает 502:

   ```
   docker logs -f code
   ```

5. Проверьте версию и сервисы:

   ```
   docker logs code | grep 'Current version'
   docker exec code gitlab-ctl status
   docker inspect --format '{{.State.Health.Status}}' code
   ```

Стартовый скрипт читает версию данных на томе и сравнивает её с версией образа. Если перейти к новой версии за один шаг нельзя, в выводе указывается версия, которую нужно установить сначала, и запуск прекращается: запустите тег этой версии, дождитесь завершения фоновых миграций и затем запустите целевой тег.

После настройки скрипт приводит базу данных к версии PostgreSQL из нового образа. Неудачное обновление базы отменяется, и контейнер завершает работу с сообщением `Upgrading the existing database failed and was reverted.` Создайте контейнер с параметром `-e GITLAB_SKIP_PG_UPGRADE=true`, чтобы запустить инстанс на текущей версии PostgreSQL, и обновите базу данных отдельно.

Helm-чарт

Обновите релиз:

```
helm repo update
helm upgrade code deckhouse-code \
  -n code \
  -f values.yaml \
  --version <CHART_VERSION>
```

Команда переиспользует существующий `values.yaml`. Если по таблице соответствия версий нужно пройти через несколько версий Deckhouse Code, соблюдайте порядок версий: пропуск версии может привести к тому, что более поздняя миграция не сможет выполниться.

При каждом обновлении один раз выполняется задача миграций — перед перезапуском компонентов, которые зависят от новой схемы базы данных. Проверьте её ход:

```
d8 k -n code get jobs
d8 k -n code logs job/<MIGRATIONS_JOB_NAME>
```

Фоновые миграции продолжаются на работающем инстансе после обновления и должны завершиться до следующего обновления.

Откат

Откат возможен только из резервной копии, снятой перед обновлением: новая версия перенесла данные.

Пакет для ОС Linux

Установите пакет той версии, на которой сделана копия, и восстановите данные по разделу «Восстановление». Точную процедуру понижения версии для вашей конфигурации уточните у вендора.

Omnibus Docker

1. Остановите и удалите контейнер:

   ```
   docker stop code
   docker rm code
   ```

2. Создайте контейнер из предыдущего тега с теми же томами.
3. Восстановите данные из резервной копии, сделанной перед обновлением, по разделу «Восстановление».

Helm-чарт

Откатите релиз на предыдущую ревизию:

```
helm rollback code <REVISION>
```

> ⚠ Команда `helm rollback` откатывает только объекты Kubernetes релиза. Она не отменяет изменение схемы базы данных, сделанное миграциями обновления.

Если обновление, которое вы откатываете, изменило схему, после завершения отката восстановите резервную копию, созданную до обновления, по разделу «Восстановление».

Переход между пакетом, Docker и модулем

Перенос инстанса с пакета для ОС Linux или Omnibus Docker в модуль Deckhouse Kubernetes Platform и обратно описан в разделе «Миграция» документации модуля.

Модуль Deckhouse Kubernetes Platform

В поставке модулем обновление запускает платформа. Если в ресурсе CodeInstance заданы `backup.enabled` и `backup.backupBeforeUpdate`, оператор сначала создаёт резервную копию и обновляет остальные компоненты после успешного завершения задания копирования; неуспешное задание откладывает обновление и вызывает оповещение `D8CodeOperatorUpdatePostpone`. Порядок описан в разделе «Автоматическое создание бэкапов перед обновлениями модуля» документации модуля.

Использование

Обзор

Deckhouse Code хранит исходный код команды, обеспечивает ревью изменений и запускает CI/CD-пайплайны. Раздел «Использование» описывает повседневную работу участника проекта и владельца группы — от первого входа в систему до поиска по инстансу.

Войдите в веб-интерфейс с учётными данными своей учётной записи или через внешнего провайдера аутентификации, затем склонируйте репозиторий и отправляйте изменения по SSH или HTTPS. Подробнее — в разделе Вход в систему.

Группа объединяет проекты общей ролевой моделью и общими политиками, такими как защищённые ветки и правила отправки изменений; проект хранит репозиторий, его задачи и wiki. Подробнее — в разделе Группы и участники.

Репозиторий хранит историю Git проекта: ветки, теги и файлы, которые можно редактировать через веб-интерфейс или Git LFS. За то, как изменения попадают в репозиторий, отвечают защищённые ветки, правила отправки изменений и pull-зеркалирование. Подробнее — в разделе Управление репозиториями Git.

Запрос на слияние объединяет изменения одной ветки с другой через ревью: комментарии, правила апрувов и внешние проверки статуса определяют, когда его можно слить. Подробнее — в разделе Ревью кода.

Пайплайн, описанный в `.gitlab-ci.yml`, собирает, тестирует и разворачивает код проекта; секреты для него доступны из внешнего Vault или Deckhouse Stronghold. Подробнее — в разделе Непрерывная сборка и поставка ПО.

Владельцы групп и мейнтейнеры проектов настраивают уровни видимости, применение правил отправки изменений и проверки при слиянии, чтобы защитить кодовую базу. Подробнее — в разделе Настройки безопасности группы и проекта.

Вебхуки уведомляют внешние системы о событиях в группе, проекте или подгруппе. Подробнее — в разделе Вебхуки.

Расширенный поиск находит код, коммиты, задачи и запросы на слияние в доступных вам проектах. Подробнее — в разделе Расширенный поиск.

Начало работы

Вход в систему

1. Откройте веб-интерфейс Deckhouse Code.

Пакет для ОС Linux

Адрес — значение параметра `external_url`, заданное при установке. Подробнее — в разделе Быстрый старт.

Omnibus Docker

Адрес — значение параметра `external_url`, заданное при установке. Подробнее — в разделе Быстрый старт.

Модуль Deckhouse Kubernetes Platform

Введите в адресной строке браузера адрес веб-интерфейса. Это имя хоста, заданное для веб-интерфейса в ресурсе CodeInstance, или, если оно не задано, глобальный параметр платформы `publicDomainTemplate`, в котором `%s` заменён на `code`: для шаблона `%s.example.com` адрес — `code.example.com`.

Helm-чарт

Адрес задан в файле `values.yaml` Helm-релиза при установке. Подробнее — в разделе Быстрый старт.

2. Авторизуйтесь, используя учётные данные своей учётной записи.

   Если администратор настроил вход через внешнего провайдера аутентификации, войдите через него. Подробнее — в разделе Аутентификация.

Двухфакторная аутентификация

Вы можете включить двухфакторную аутентификацию для своей учётной записи: тогда при входе, помимо пароля, потребуется дополнительное подтверждение. Deckhouse Code поддерживает приложения-аутентификаторы (например, Google Authenticator или Authy) и аппаратные ключи безопасности (U2F, WebAuthn).

Чтобы включить двухфакторную аутентификацию, перейдите в «Профиль» → «Аккаунт» → «Двухфакторная аутентификация» и выполните настройку выбранным способом.

Работа с Git

Способы доступа

Операции Git из командной строки аутентифицируются SSH-ключом или личным токеном доступа по HTTPS.

- SSH-ключи — сгенерируйте пару ключей на компьютере и добавьте открытый ключ в «Профиль» → «SSH-ключи». Используйте SSH-ключ, чтобы клонировать репозиторий и отправлять изменения по URL вида `git@...` без ввода пароля при каждом запросе.
- HTTPS с личным токеном доступа — создайте токен в «Профиль» → «Токены доступа» и используйте его как пароль, когда Git запрашивает учётные данные для URL вида `https://...`.

Администратор может ограничить, какой из двух протоколов доступен на инстансе. Подробнее — в разделе Пользователи и доступ.

Клонирование репозитория

Клонирование позволяет скопировать удалённый Git-репозиторий на локальный компьютер. Для клонирования выполните команду в терминале:

```
git clone <REPOSITORY_URL>
```

Замените `<REPOSITORY_URL>` на HTTPS или SSH-URL вашего репозитория.

Получение изменений

Чтобы получить последние изменения из удалённого репозитория, перейдите в директорию репозитория на локальном компьютере и выполните команду:

```
git pull origin <BRANCH_NAME>
```

Замените `<BRANCH_NAME>` на имя ветки, например, `main` или `feature/my-new-feature`.

Отправка изменений (Push)

1. Внесите необходимые исправления в кодовую базу.
2. Зафиксируйте изменения командой:

   ```
   git commit -am "Описание ваших изменений"
   ```

3. Отправьте изменения в удалённый репозиторий:

   ```
   git push origin <BRANCH_NAME>
   ```

Замените `<BRANCH_NAME>` на имя ветки, в которую отправляются изменения.

Создание feature-ветки

Feature-ветки используются для разработки новых функций или внесения исправлений. Ниже приведены шаги по созданию feature-ветки и отправке её в Deckhouse Code:

1. Склонируйте репозиторий:

   ```
   git clone <REPOSITORY_URL>
   ```

   Замените `<REPOSITORY_URL>` на актуальный URL репозитория.

2. Перейдите в директорию репозитория:

   ```
   cd <REPOSITORY_PATH>
   ```

3. Создайте новую feature-ветку:

   ```
   git checkout -b <BRANCH_NAME>
   ```

   Замените `<BRANCH_NAME>` на имя новой ветки. Флаг `-b` указывает Git создать новую ветку и сразу на неё переключиться.

4. Отправьте новую ветку в удалённый репозиторий:

   ```
   git push origin <BRANCH_NAME>
   ```

   Убедитесь, что ветка появилась в удалённом репозитории. Для проверки откройте проект в Deckhouse Code и найдите новую ветку в меню выбора веток.

Запрос на слияние (Merge Request)

Запрос на слияние используется для объединения изменений из одной ветки в другую.

Преимущества использования Merge Request:

1. Контроль качества кода:
   - Позволяет проводить ревью перед слиянием в основную ветку.
   - Помогает обнаруживать ошибки и поддерживать единый стиль кодирования.
2. Организация совместной работы:
   - Разработчики могут комментировать код, предлагать исправления и обсуждать изменения.

3. Автоматизация тестирования и CI/CD:
    - MR может интегрироваться с CI/CD-пайплайнами для автоматического тестирования изменений.
4. История изменений и документация:
    - MR содержит описание изменений, комментарии ревьюеров и обсуждения.
5. Гибкость разработки и контроль версий:
    - Позволяет легко откатить или внести исправления в код до слияния.
6. Повышение безопасности:
    - Предотвращает внедрение уязвимостей благодаря проверкам и ревью.

Создание запроса на слияние

1. Переход в раздел Merge Requests:
    - Откройте проект в Deckhouse Code.
    - В боковой панели выберите раздел: «Код» → «Запросы на слияние».
2. Создание нового запроса:
    - Нажмите кнопку «Новый запрос на слияние».
    - Выберите исходную ветку (с изменениями) и целевую ветку (обычно main).
    - Нажмите «Сравнить ветки и продолжить».
3. Заполнение деталей запроса:
    - Название: опишите кратко суть изменений.
    - Описание: укажите подробности, при необходимости добавьте скриншоты.
    - Ответственный: назначьте пользователя, который будет проводить ревью.
4. Отправка запроса:
    - Нажмите «Создать запрос на слияние».

Ревью и слияние запроса

Слияние происходит после выполнения следующих условий:

- Запрос на слияние прошёл ревью и получил необходимое количество одобрений.
- Отсутствуют конфликты между ветками.
- Все CI/CD пайплайны успешно завершены.

Чтобы завершить процесс, нажмите кнопку «Слияние» на странице запроса.

Проекты и задачи

Группы и участники

Каждый участник группы или проекта имеет одну из предустановленных ролей: Гость, Наблюдатель, Разработчик, Мейнтейнер или Владелец. Роль определяет, что участник может видеть и изменять — как в самой группе, так и в каждом проекте внутри неё.

Группа также несёт настройки, которые применяются ко всем её проектам: правила защиты веток и тегов ограничивают, кто может отправлять изменения напрямую или удалять их, а правила отправки изменений и другие групповые политики задают значения по умолчанию, которые проект наследует, если группа не разрешила ему переопределять их.

1. Создайте группу:
    - Перейдите в раздел «Группы» → «Новая группа».
    - Укажите название группы (например, «Development Team»).
    - Настройте уровень видимости:
        - «Приватный» — виден только членам группы.
        - «Публичный» — виден всем без исключения.
        - «Внутренний» — виден всем зарегистрированным пользователям.

    Создание группы требует разрешения на создание групп, которое по умолчанию есть у каждого зарегистрированного пользователя. Администратор может ограничить это разрешение. Подробнее — в разделе Пользователи и доступ.

2. Пригласите пользователей:
    - Откройте созданную группу.
    - Перейдите в раздел «Управлять доступом».
    - Нажмите «Пригласить участников», укажите e-mail или имя пользователя и задайте роль (например, «Developer»).

    > ⚠ При использовании SSO-интеграций процесс организации доступа может отличаться.

3. Назначьте пользователям группы роли:
    - Перейдите в нужную группу.
    - Выберите «Управление» → «Участники».
    - Назначьте каждому приглашённому пользователю необходимую роль.

В зависимости от назначенной роли пользователю доступен различный набор действий:

Гость:

- Группа: Может просматривать публичные элементы. Нет доступа к конфиденциальным данным.
- Проект: Может просматривать issues в публичных репозиториях, но без права комментирования и управления.

Наблюдатель:

- Группа: Видит всю информацию, включая issues и публичные проекты.
- Проект: Может читать код, CI/CD pipeline и отчёты без права внесения изменений.

Разработчик:

- Группа: Может создавать проекты и вносить изменения в репозитории.
- Проект: Доступ к веткам, коммитам, Merge Requests, CI/CD и задачам разработки.

Мейнтейнер:

- Группа: Полный контроль над проектами и управление членами группы.
- Проект: Управление настройками проекта, ветками, тегами и Merge Requests.

Владелец:

- Группа: Полный контроль, включая удаление и управление ролями.
- Проект: Доступ на уровне группы с полным управлением проектами.

Создание проекта

1. Перейдите в раздел «Проекты» в боковой панели.
2. Нажмите кнопку «Новый проект».
3. Выберите один из доступных способов создания:
    - «Создать пустой проект» — создаёт новый репозиторий с нуля.
    - «Импортировать проект» — перенос существующего проекта из другой системы. При использовании способа «Репозиторий по URL» можно установить флажок «Зеркалировать репозиторий», чтобы новый проект автоматически синхронизировался с исходным репозиторием в режиме pull-зеркалирования.
    - «Создать из шаблона» — проект на основе предварительно подготовленного шаблона.
4. В открывшейся форме укажите параметры проекта:
    - «Имя проекта» — задайте понятное и уникальное имя, например: `my-project`.
    - «Уровень доступа» — определите, кто сможет просматривать проект:
        - «Приватный» — доступ только у участников группы или проекта.
        - «Внутренний» — доступ для всех зарегистрированных пользователей.
5. Нажмите «Создать проект».

6. После создания:
    - Перейдите в созданный проект.
    - Нажмите кнопку «Код» в правом верхнем углу.
    - Скопируйте URL для клонирования — можно выбрать HTTPS или SSH.

Уровни доступа к проекту

Уровень	Описание
«Приватный»	Проект виден только участникам группы или самого проекта
«Внутренний»	Проект доступен всем зарегистрированным пользователям
«Публичный»	Проект доступен всем, включая незарегистрированных пользователей

Импорт проекта

Если исходный инстанс GitLab не имеет сетевого доступа к Deckhouse Code, импорт проекта по URL невозможен. Выгрузите проект в файл на исходном инстансе, а затем загрузите этот файл в Deckhouse Code.

Требования

- Версия GitLab на исходном инстансе совпадает с версией GitLab, на которой работает Deckhouse Code, либо старше её не более чем на две минорные версии. Например, если Deckhouse Code работает на GitLab 19.1, совместимы экспорты из GitLab 19.1, 19.0 и 18.11.
- У вас есть роль `Maintainer` или `Owner` на проекте, который нужно экспортировать.
- У вас есть роль `Maintainer` или `Owner` на целевой группе в Deckhouse Code.

Шаги импорта

Шаг 1. Включите экспорт проектов на исходном инстансе

Администратор исходного инстанса GitLab включает функцию экспорта:

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните «Настройки импорта и экспорта».
3. В разделе «Экспорт проекта» выберите «Включен».
4. Нажмите «Сохранить изменения».

> Если экспорт проектов на исходном инстансе уже включён, пропустите этот шаг.

Шаг 2. Экспортируйте проект

1. Откройте проект, который нужно перенести.
2. Перейдите в «Настройки» → «Основные».
3. Разверните «Расширенные».
4. Нажмите «Экспорт проекта».
5. После формирования экспорта скачайте файл (архив `.tar.gz`) по ссылке из письма или нажав «Скачать экспортируемые данные» на той же странице.

Сохраните экспортированный файл на рабочем компьютере: он понадобится на следующем шаге для загрузки в Deckhouse Code.

Шаг 3. Включите источник импорта «Экспорт в Deckhouse Code»

Администратор инстанса Deckhouse Code включает источник импорта:

1. Перейдите в «Admin» → «Настройки» → «Основные».
2. Разверните «Настройки импорта и экспорта».
3. В разделе «Источники импорта» установите флажок «Экспорт в Deckhouse Code».
4. Нажмите «Сохранить изменения».

Шаг 4. Импортируйте проект

1. В правом верхнем углу выберите «Создать новый» (+) → «Новый проект/репозиторий».
2. Выберите «Импортировать проект».
3. В поле «Импортировать проект из» выберите «Экспорт в Deckhouse Code».
4. Введите имя проекта и URL, затем выберите экспортированный файл.
5. Нажмите «Импортировать проект».

Проверить статус импорта можно через API импорта проекта.

> Для крупных проектов вместо веб-интерфейса используйте Rake-задачи для импорта проекта.

Что переносится, а что нет

Переносится: репозиторий, wiki, задачи, запросы на слияние, метки, вехи, сниппеты, релизы, прямые участники проекта, LFS-объекты, доски задач, архивные CI/CD-пайплайны, защищённые ветки и теги, правила отправки изменений.

Не переносится: CI/CD-переменные, вебхуки, логи и артефакты CI/CD-задач, образы package и container registry, зашифрованные токены, deploy keys, триггеры пайплайнов, политики безопасности.

Полный список экспортируемых и неэкспортируемых данных приведён в справке GitLab по экспорту и импорту проектов.

Управление задачами

Deckhouse Code предоставляет встроенные средства для управления задачами в рамках проекта.

Возможности:

- Создание задач с описанием, лейблами и назначением исполнителей.
- Установка приоритетов и сроков выполнения.
- Поддержка чек-листов (task lists) для разбивки задач на подэтапы.
- Автоматическое закрытие задач по коммитам (например, `Fixes #123`).

Этапы разработки (milestones)

Milestones позволяют объединять задачи и merge-запросы в релизы или спринты.

Возможности:

- Группировка задач и запросов по срокам реализации.
- Визуальное отображение прогресса выполнения.
- Автоматическое закрытие milestone при завершении всех связанных задач.

Kanban-доска (Issue board)

Инструмент для визуального управления задачами по методологии Kanban.

Возможности:

- Настраиваемые колонки (например, To Do, In Progress, Done).
- Перетаскивание задач между колонками для изменения статуса.
- Фильтрация по лейблам, исполнителям и состояниям задач.

Проектная документация (Wiki)

Встроенная Wiki с поддержкой Markdown для хранения и ведения документации проекта.

Возможности:

- Организация документации по страницам и разделам.
- Импорт и экспорт страниц для обмена и резервного копирования.

Wiki группы

Включение Wiki на уровне группы

Чтобы включить или настроить доступ к Wiki, откройте страницу нужной группы и перейдите в: «Настройки» → «Основные» → «Права доступа и возможности группы» → «Уровень доступа к Wiki».

Доступные варианты:

- Включено — Wiki доступна всем (для публичных групп) или только зарегистрированным пользователям (для внутренних групп).
- Приватные — Wiki доступна только участникам группы.
- Отключено — Wiki полностью отключена и недоступна.

Значение по умолчанию — Включено.

Доступ к Wiki

Чтобы открыть Wiki группы, на странице группы выберите «Планирование» → «Wiki».

Роли и Wiki

Уровень доступа определяется членством пользователя в группе:

Роль	Действия
Guest	Чтение Wiki
Reporter	Чтение, загрузка кода
Developer	Создание Wiki
Maintainer	Администрирование Wiki
Planner	Администрирование Wiki
Аноним / внешний пользователь	Доступ, если группа публичная или internal, и пользователь не external — только загрузка кода

Возможности

1. Структура и вложенность — создаёт древовидную навигацию и помогает организовать контент:
    - Создание иерархической структуры страниц с помощью символа `/` в названиях. Пример:

```
devops/ci-pipelines
devops/kubernetes
product/design
```

2. Markdown, Rich Text, вложения и диаграммы:
    - Markdown (GitLab Flavored):
        - Упоминания (`@username`), ссылки на задачи/merge-реквесты (`#123` , `!456`), чек-листы (`- [ ]`), таблицы, блоки кода.
    - Rich Text Editor:
        - WYSIWYG-редактор для пользователей, предпочитающих визуальное форматирование.
        - Поддержка Mermaid и Draw.io без дополнительной настройки.
    - Вложения:
        - Загрузка и вставка изображений, PDF и других файлов.
        - Хранятся внутри Git-репозитория Wiki.
3. История изменений:
    - Полная история изменений для каждой страницы:
        - Кто вносил изменения.
        - Когда они были сделаны.
    - Просмотр diff-ов, откат изменений, отслеживание всех правок.
4. Обсуждения и комментарии на страницах.
5. Доступ через Git:
    - Каждая Wiki — это отдельный Git-репозиторий.
    - Клонирование через SSH или HTTPS:

      ```
      git clone git@code.example.com:groupname/wiki.git
      ```

    - Полный доступ к `.md` -файлам, веткам, истории для локального редактирования, резервных копий или автоматизации.
6. Экспорт в PDF:
    - Экспорт любой страницы Wiki в PDF через веб-интерфейс.
    - Полезно для оффлайн-доступа, обмена или печати.
7. Шаблоны страниц:
    - Многоразовые шаблоны, хранящиеся в директории `templates/` .
    - Применяются при создании или редактировании страниц для стандартизации контента.

8. Настраиваемое боковое меню
    - Боковое меню показывает страницы в виде вложенного дерева.
    - Полностью настраивается с помощью специальной страницы `_sidebar`.
    - Можно добавлять ссылки, разделы и улучшать навигацию.

Аналитика задач

Отчёт «Аналитика задач» показывает, сколько задач открыто и закрыто в каждом месяце указанного периода, и перечисляет задачи этого периода в порядке убывания. Отчёт доступен для группы и для проекта.

Доступность отчёта

Отчёт доступен участникам группы или проекта с ролью «Гость» и выше, в том числе тем, кто получил роль в родительской группе, и пользователям с типом «Аудитор».

В группе или проекте с уровнем видимости «Публичный» отчёт открыт любому посетителю, включая не выполнившего вход.

С уровнем видимости «Внутренний» отчёт открыт любому пользователю, даже если он не состоит в этой группе или проекте.

Просмотр отчёта

Чтобы просмотреть отчёт группы:

1. В верхней панели нажмите «Искать или перейти к…» и выберите группу.
2. В меню группы перейдите в «Аналитика» → «Аналитика задач».

Чтобы просмотреть отчёт проекта:

1. В верхней панели нажмите «Искать или перейти к…» и выберите проект.
2. В меню проекта перейдите в «Аналитика» → «Аналитика задач».

Также отчёт можно открыть по прямому адресу:

Отчёт	Адрес
Группа	`https://<HOST>/groups/<GROUP>/-/issues_analytics`
Проект	`https://<HOST>/<GROUP>/<PROJECT>/-/analytics/issues_analytics`

Отчёт группы охватывает задачи её проектов и проектов её подгрупп, кроме архивных.

Обзор в виде графика

График «Обзор» содержит по столбцу на каждый месяц периода, разделённому на серии:

- «Открыто» — задачи по месяцу создания;
- «Закрыто» — задачи по месяцу закрытия.

Задача, созданная в одном месяце и закрытая в другом, попадает в обе серии, поэтому в месяце может быть закрыто больше задач, чем открыто. Месяцы отсчитываются по UTC, а месяц без задач показан пустым столбцом.

Строка над графиком подводит итог периода: `Открыто: всего 128, 10.7 в месяц. Закрыто: всего 94, 7.8 в месяц.` Среднее — это итог, делённый на число месяцев периода.

Период отчёта

Период составляет 12 месяцев, включая текущий. Чтобы задать другую длительность, добавьте к адресу страницы параметр `months_back`:

```
https://<HOST>/groups/<GROUP>/-/issues_analytics?months_back=3
```

Параметр принимает значение от 1 до 36 месяцев, включая текущий. Значение вне диапазона приводится к ближайшей границе, а значение, не являющееся числом, — к 12.

Подпись под горизонтальной осью называет период: `За 3 месяца (Июл. 2026 – Сент. 2026)`, а для периода в один месяц — «Текущий месяц».

Панель фильтров

Панель фильтров над графиком ограничивает график и таблицу одновременно:

- «Автор» — один автор;
- «Ответственный» — один или несколько ответственных;
- «Этап» — один этап;
- «Метка» — одна или несколько меток;
- «Моя реакция» — эмодзи, которым отреагировал текущий пользователь;
- строка поиска — поиск по задачам.

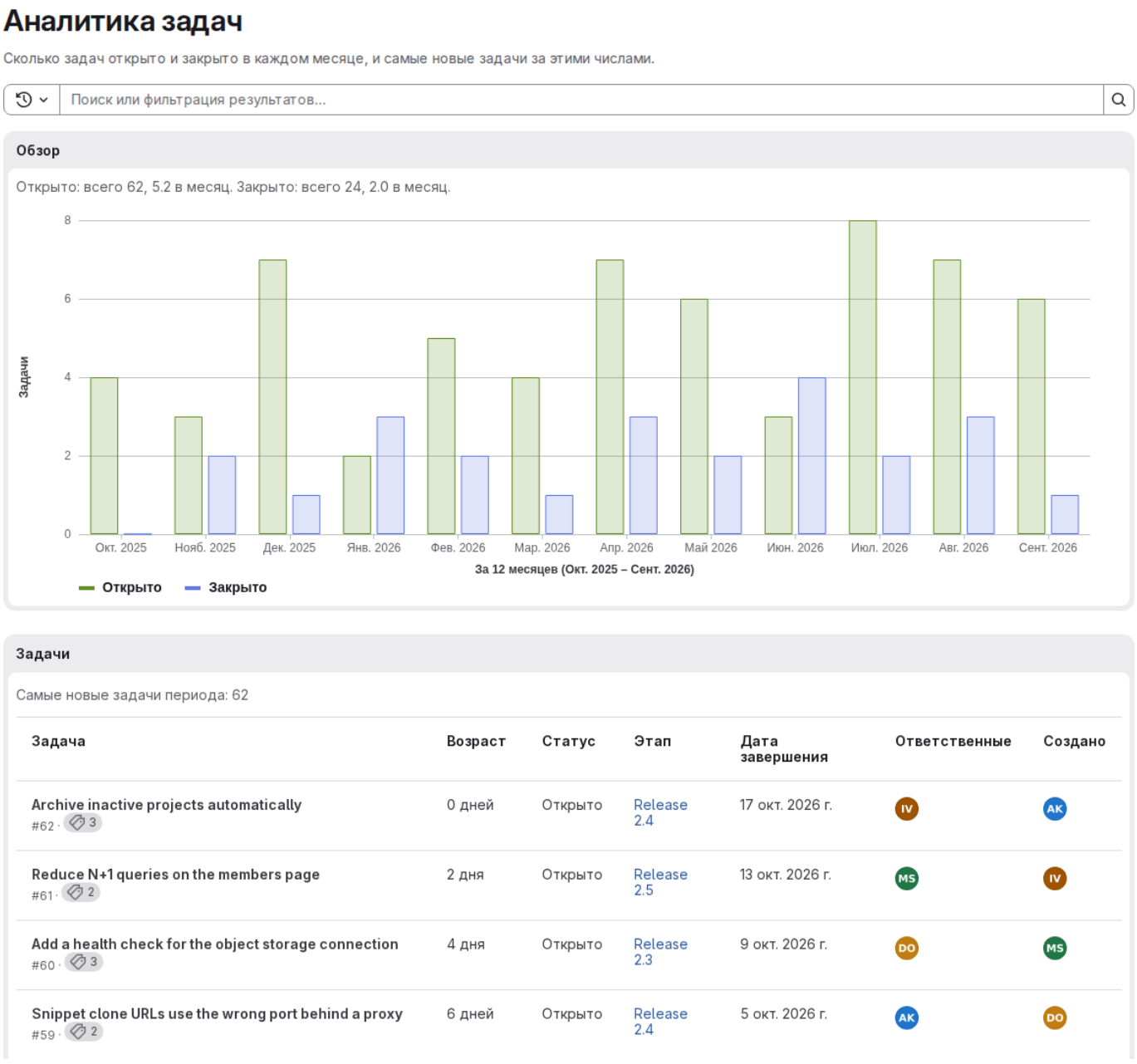
Все фильтры, кроме «Моя реакция», принимают также оператор `!=`, который исключает выбранное значение из отчёта.

Фильтры и период сохраняются в адресе страницы, поэтому настроенный отчёт можно передать ссылкой.

Таблица задач

Таблица «Задачи» перечисляет до 100 задач, созданных в периоде, от новых к старым. Строка над таблицей показывает, сколько задач отображено и сколько соответствует фильтрам: `Самые`

`новые задачи периода: 100 из 240`. Итог считается до 1000; далее строка принимает вид `Самые новые задачи периода: 100 из более чем 1000`.



Колонка	Содержимое
«Задача»	Заголовок со ссылкой на задачу, её номер и число меток. При наведении на число меток показываются сами метки, а выбор метки открывает список задач с фильтром по ней
«Возраст»	Число дней с момента создания задачи
«Статус»	«Открыто» или «Закрыто»
«Этап»	Этап задачи со ссылкой на него
«Дата завершения»	Дата завершения задачи
«Ответственные»	Пользователи, назначенные на задачу
«Создано»	Автор задачи

Репозитории

Управление репозиториями Git

Хранение и версионирование кода

Deckhouse Code предоставляет инструменты для работы с репозиториями Git. Каждый проект включает репозиторий для хранения исходного кода и отслеживания истории изменений:

- Полноценная поддержка распределённой системы контроля версий Git.
- Выполнение стандартных команд: клонирование (`git clone`), коммит (`git commit`), отправка (`git push`), получение (`git pull`).
- Автоматическое отслеживание и сохранение истории изменений.

Поддержка форков (Forks)

Форк — это независимая копия репозитория, предназначенная для разработки без риска повлиять на основной код:

1. Разработчик создаёт форк основного репозитория.
2. Вносит изменения в своём форке.
3. Отправляет merge request (MR) в оригинальный репозиторий.
4. После проверки изменения могут быть слиты.

Подход широко используется в open source-проектах и при ограниченном доступе к основному репозиторию.

Работа с ветками (branching) и слиянием (merging)

Ветки позволяют разрабатывать новые функции и исправления изолированно от основной кодовой базы:

- Создание веток: `git branch <BRANCH_NAME>`, `git checkout -b <BRANCH_NAME>`.
- Переключение между ветками: `git checkout <BRANCH_NAME>`.
- Слияние изменений: `git merge <BRANCH_NAME>`.
- Поддержка защищённых веток (protected branches) — для внесения изменений требуется merge request.
- Автоматическое удаление веток после слияния (если включено).

Встроенный web-редактор

Позволяет вносить изменения в код прямо через браузер без необходимости клонировать репозиторий:

- Создание и редактирование файлов через веб-интерфейс.
- Автоматическое создание коммитов при сохранении изменений.
- Поддержка Markdown с возможностью предпросмотра.

Поддержка Git LFS (Large File Storage)

Git LFS позволяет эффективно управлять большими файлами, такими как:

- Изображения;
- Видео;
- Аудиофайлы;
- Датасеты для машинного обучения.

Git LFS заменяет содержимое больших файлов ссылками, а сами файлы хранятся вне Git-репозитория. Это снижает нагрузку на историю коммитов и ускоряет работу с репозиторием.

Защищённые ветки

Защищённые ветки (protected branches) ограничивают изменение важных веток репозитория: отправлять изменения (push) и выполнять слияние в такую ветку могут только те, кому это явно разрешено. Обычно защита используется для веток `main`, `master`, релизных и стабильных веток — чтобы изменения попадали в них только через merge request и код-ревью.

Ветка проекта, назначенная веткой по умолчанию, защищается автоматически при его создании.

Ограничения защищённых веток

Для защищённой ветки действуют следующие ограничения:

- Прямая отправка изменений (`git push`) запрещена для всех, кроме пользователей, указанных в списке «Разрешено выполнять push и слияние». Остальные участники вносят изменения через merge request.
- Принудительная отправка (`git push --force`) запрещена для всех, пока в правиле ветки не включена настройка «Разрешить force push».
- Ветку нельзя удалить командами Git. Удаление доступно только через веб-интерфейс пользователям с ролью `Maintainer` и выше.

Защита ветки по умолчанию

При создании проекта его ветка по умолчанию автоматически становится защищённой. Начальный уровень защиты задаётся настройкой «Защита начальной ветки по умолчанию»:

- **На уровне инстанса** — раздел «Область администратора» → «Настройки» → «Репозиторий», секция «Ветка по умолчанию». Настройка доступна администраторам инстанса.
- **На уровне группы** — на странице группы перейдите в раздел «Настройки» → «Репозиторий», секция «Ветка по умолчанию». Настройка доступна пользователям с ролью `Owner` в группе.

Настройка группы имеет приоритет над настройкой инстанса: если в группе она не задана, действует настройка инстанса. Для проектов в личном пространстве пользователя всегда действует настройка инстанса.

По умолчанию применяется полная защита: push и слияние разрешены только участникам с ролью `Maintainer` и выше, force push запрещён.

Правила веток

Защита настраивается через правила веток. Для этого откройте страницу проекта и перейдите в раздел «Настройки» → «Репозиторий» → «Правила веток». Настройка доступна пользователям с ролью `Maintainer` и `Owner`.

Чтобы создать правило:

1. Нажмите «Добавить правило ветки».
2. Выберите «Имя или шаблон ветки».
3. Укажите имя конкретной ветки (например, `main`) или шаблон с подстановочным символом `*` (например, `release/*`). Правило с шаблоном защищает все подходящие ветки, включая те, которые будут созданы позже.
4. Нажмите «Создать правило ветвления».

Чтобы открыть настройки существующего правила, нажмите «Просмотреть детали» напротив него в списке.

Настройка доступа к защищённой ветке

На странице правила в секции «Защитить ветку» настраиваются два списка доступа:

- **«Разрешено слияние»** — кто может выполнять слияние merge request в эту ветку.
- **«Разрешено выполнять push и слияние»** — кто может отправлять изменения в ветку напрямую (`git push`), а также выполнять слияние.

Чтобы изменить соответствующий список, нажмите «Изменить права на слияние» или «Изменить права на push и слияние».

В каждом списке можно комбинировать роли, отдельных пользователей и группы. В списке «Разрешено выполнять push и слияние» дополнительно доступны ключи развёртывания. Доступ получает каждый, кто подходит хотя бы под один из выбранных пунктов.

Доступные варианты:

Пункт списка	Кому предоставляет доступ
«Мейнтейнеры»	Участникам проекта с ролью `Maintainer` и выше
«Разработчики и Мейнтейнеры»	Участникам проекта с ролью `Developer` и выше
«Никто»	Никому. Доступ по ролям полностью закрыт
«Администраторы»	Администраторам инстанса
«Пользователи»	Перечисленным участникам проекта с правом записи в репозиторий (роль `Developer` и выше). Подробнее — в разделе «Доступ для отдельных пользователей и групп»
«Группы»	Прямым участникам перечисленных групп с ролью в группе `Developer` и выше. Группа должна быть приглашена в проект с ролью не ниже `Developer`. Подробнее — в разделе «Доступ для отдельных пользователей и групп»
«Ключи развёртывания»	Владельцу выбранного ключа развёртывания, в том числе при отправке изменений по этому ключу. Ключ должен быть включён в проекте с правом записи, а владелец ключа — быть участником проекта. Доступно только для списка «Разрешено выполнять push и слияние»

Доступ для отдельных пользователей и групп

Помимо ролей, доступ к защищённой ветке можно выдать точечно — конкретным пользователям и группам. Это позволяет, например, закрыть ветку для всех (выбрав «Никто») и разрешить слияние только релиз-менеджеру, не повышая его роль в проекте.

Пользователи

В селекторе «Пользователи» можно выбрать участников проекта с правом записи в репозиторий — пользователей с ролью `Developer` и выше, включая участников, унаследованных из родительской группы.

Добавление пользователя в список не повышает его права в проекте. Участник без роли `Developer` и выше не сможет отправлять изменения в защищённую ветку, даже если он указан в списке.

Эти списки позволяют сузить круг тех, кому право записи уже выдано ролью. Например, выберите «Никто» в ролях и перечислите конкретных пользователей — тогда работать с этой веткой смогут только они.

Группы

В селекторе «Группы» отображаются только группы, приглашённые в проект с ролью `Developer` и выше. Приглашение группы настраивается в разделе «Управление» → «Участники» проекта.

Родительские группы проекта и группы, приглашённые с меньшей ролью, выбрать нельзя.

Доступ через группу получают только прямые участники этой группы с ролью в ней `Developer` и выше. Участники вложенных подгрупп доступа не получают.

Прекращение действия доступа

Права проверяются в момент каждой операции, поэтому изменения членства применяются сразу:

- если пользователь перестал быть участником проекта, его доступ к защищённой ветке перестаёт действовать;
- если приглашение группы в проект отозвано или её роль понижена ниже `Developer`, доступ участников группы перестаёт действовать.

Запись при этом остаётся в списке доступа и снова начнёт действовать, если членство или приглашение будет восстановлено. Чтобы окончательно отозвать доступ, удалите пользователя или группу из списка в правиле ветки.

Разрешить force push

Переключатель «Разрешить force push» на странице правила разрешает принудительную отправку изменений (`git push --force`) в защищённую ветку. Пользователи, которым разрешён force push, определяются списком «Разрешено выполнять push и слияние». По умолчанию переключатель выключен, и force push запрещён для всех, включая мейнтейнеров и администраторов.

Особенности и ограничения

- Доступ, выданный на защищённую ветку, не меняет роль пользователя в проекте и не предоставляет ему дополнительные права. Он действует только на операции push и слияния в ветки, подпадающие под правило.
- Правило с шаблоном (например, `release/*`) применяется ко всем существующим и будущим веткам, имена которых соответствуют шаблону.
- Выбор пункта «Никто» отключает доступ по ролям, но не очищает списки пользователей, групп и ключей развёртывания. Перечисленные в них сущности сохраняют доступ.

- Настраивать правила веток могут только пользователи с ролью `Maintainer` и выше.

Правила отправки изменений

В Deckhouse Code представлен набор правил для контроля за соблюдением определённых политик при отправке кода в Git. Эти правила помогают поддерживать целостность репозитория, повышать качество кода и обеспечивать соответствие стандартам безопасности вашей организации.

Доступные правила

- ***Проверять email коммитера***

  Адрес электронной почты коммитера должен быть подтвержден в профиле Deckhouse Code.

- ***Запретить удаление тегов***

  Тег нельзя удалить через git push. Удаление через веб-интерфейс доступно.

- ***Проверка зарегистрированного пользователя***

  Автор коммита должен быть пользователем Deckhouse Code.

- ***Блокировать утечки секретов***

  Попытка закоммитить файлы, подпадающие под следующие шаблоны, будет отклонена:

  - `aws/credentials`,
  - `ssh/personal_rsa`, `ssh/personal_dsa`, `ssh/personal_ed25519`, `ssh/personal_ecdsa`, `ssh/personal_ecdsa_sk`, `ssh/personal_ed25519_sk`,
  - `ssh/server_rsa`, `ssh/server_dsa`, `ssh/server_ed25519`, `ssh/server_ecdsa`, `ssh/server_ecdsa_sk`, `ssh/server_ed25519_sk`,
  - `id_rsa`, `id_dsa`, `id_ed25519`, `id_ecdsa`, `id_ecdsa_sk`, `id_ed25519_sk`,
  - файлы с расширением `.pem` или `.key`,
  - файлы `.history` или `_history`,
  - файлы с расширением `.keystore` или `.jks`,
  - `keystore.p12`, `keystore.pfx`.

- ***Коммит должен быть подписан GPG***

  Разрешены только коммиты, подписанные GPG-ключом. Изменения через веб-интерфейс будут отклоняться, если для веб-UI не настроена подпись GPG.

- ***В сообщении коммита должен присутствовать DCO (Сертификат происхождения разработчика)***

  Коммиты, включая merge-коммиты и squash-коммиты, должны содержать строку DCO «Signed-off-by». При включении этого правила в веб-интерфейсе будет недоступна опция

отката запроса на слияние, поскольку автоматически сгенерированное сообщение коммита не содержит подписи DCO.

- ***Проверять имя коммитера***

  Имя коммитера должно совпадать с именем профиля в Deckhouse Code.

- ***Регулярное выражение для сообщения коммита***

  Коммиты должны соответствовать указанному шаблону регулярного выражения. Пустое поле отключает правило.

- ***Регулярное выражение для email автора коммита***

  Email автора должен совпадать с указанным регулярным выражением. Пустое поле отключает правило.

- ***Регулярное выражение для имени файла***

  Имена файлов должны соответствовать указанному шаблону регулярного выражения. Пустое поле отключает правило.

- ***Регулярное выражение для имени ветки***

  Имя ветки должно совпадать с указанным регулярным выражением. Пустое поле отключает правило.

- ***Регулярное выражение для недопустимых сообщений коммитов***

  Если сообщение соответствует указанному шаблону регулярного выражения, коммиты будут отклонены. Пустое поле отключает правило.

- ***Максимальный размер файла (МБ)***

  Файлы, размер которых превышает указанный предел (в мегабайтах), будут отклонены. Чтобы отключить правило, установите значение `0`.

Настройка правил отправки изменений

Чтобы настроить правила отправки изменений, откройте страницу проекта и перейдите в раздел «Настройки» → «Репозиторий» → «Правила отправки изменений».

Настройка правил доступна пользователям с ролью `Maintainer` и `Owner`.

Настройка правил на уровне группы или инстанса

Чтобы получить доступ к настройке правил на уровне группы, откройте страницу группы и выберите «Настройки» → «Репозиторий» → «Правила отправки изменений». Настройка на уровне группы доступна пользователям с ролью `Owner`.

Правила уровня инстанса применяются ко всем проектам инстанса и задаются администратором. Подробнее — в разделе Пользователи и доступ.

При изменении правила на уровне инстанса новые значения применяются ко всем группам и проектам.

При изменении правила на уровне группы новые значения будут применены ко всем нижестоящим группам и их проектам.

Правила, заданные на уровне группы или инстанса, используются как настройки по умолчанию для проектов. При создании проекта будут включены те правила, которые заданы в его группе или на уровне инстанса (если проект создан в пользовательском пространстве).

Запрет на переопределение правил

Можно запретить переопределение правил в нижестоящих проектах и группах. Для этого снимите галочку «Разрешить переопределять на уровне группы/проекта». Такое правило нельзя изменять в нижестоящих группах и проектах. На уровне инстанса эта настройка применяется ко всем группам и проектам.

Примеры:

1. Если на уровне группы включить правило *«Проверять email коммитера»* и запретить его переопределение, это правило будет включено во всех проектах группы и подгрупп, без возможности его отключить.
2. Если на уровне группы выключить это же правило и запретить его переопределение, оно будет отключено во всех проектах группы и подгрупп, также без возможности изменения.
3. Если разрешено переопределение правила, оно будет автоматически обновляться при изменениях на уровне предка, но в нижестоящих группах и проектах его можно будет изменить.

Иерархия применения выглядит так:
Инстанс → Группа → Подгруппа → Проект

Pull-зеркалирование

Чтобы настроить зеркалирование репозитория, на странице проекта перейдите в «Настройки» → «Репозиторий» → «Зеркалирование репозиториев».

Если репозиторий пуст, сначала выполните его импорт. При зеркалировании запускаются все хуки, и загрузка большого репозитория может существенно нагрузить систему.

Настройка pull-зеркалирования репозитория

> ⚠ Для одного проекта можно настроить только одну задачу pull-зеркалирования. Задач push-зеркалирования может быть несколько.

Чтобы настроить pull-зеркалирование репозитория, выполните следующие шаги:

1. Перейдите на страницу проекта:
    - Откройте проект в интерфейсе Deckhouse Code.
    - В левом меню выберите «Настройки» → «Репозиторий».
    - Прокрутите до секции «Зеркалирование репозиториев».
2. Укажите URL-адрес репозитория. Учётные данные в URL-адресе игнорируются. Для авторизации используйте поля в блоке «Метод аутентификации» ниже.
3. Настройте аутентификацию:
    - Если подключение осуществляется по HTTP(S), в поле «Метод аутентификации» выберите «Имя пользователя и пароль» и укажите:
        - «Имя пользователя» — имя пользователя;
        - «Пароль» — пароль или токен доступа.
    - Если используется SSH-зеркалирование, укажите имя пользователя (обычно `git`). После сохранения конфигурации Deckhouse Code сгенерирует SSH-ключ, который будет использоваться для доступа.

Фильтр веток

Блок «Фильтр веток» в настройках зеркалирования определяет, какие ветки загружаются из исходного репозитория:

- «Зеркалировать все ветки» — зеркалируются все ветки исходного репозитория.
- «Зеркалировать только защищённые ветки» — зеркалируются только ветки, защищённые в этом проекте. Параметр недоступен, пока в проекте нет ни одной защищённой ветки (учитываются также защищённые ветки родительской группы). Подробнее — в разделе «Защищённые ветки».
- «Зеркалировать совпадающие ветки» — зеркалируются только ветки, имена которых соответствуют указанному регулярному выражению (синтаксис RE2). Выражение сопоставляется с именем ветки целиком. При вводе выражения форма показывает, сколько веток этого репозитория ему соответствует.

Влияние фильтра веток на синхронизацию тегов

> ⚠ Фильтр веток действует и на теги. Если выбран параметр «Зеркалировать только защищённые ветки» или «Зеркалировать совпадающие ветки», тег исходного репозитория синхронизируется только в том случае, если его коммит есть в одной из веток этого репозитория. Остальные теги пропускаются.

Учитывайте следующие особенности:

- При выборе параметра «Зеркалировать все ветки» синхронизируются все теги исходного репозитория.
- Тег, пропущенный из-за фильтра, не теряется: он проверяется заново при каждой синхронизации, и как только его коммит попадает в одну из зеркалируемых веток, тег создаётся в зеркале.
- Проверка выполняется по всем веткам репозитория-зеркала, включая ветки, созданные непосредственно в Deckhouse Code, а не только по веткам, попадающим под фильтр.
- Уже синхронизированные теги остаются в репозитории, даже если фильтр веток впоследствии изменён так, что больше их не охватывает.

Импорт репозитория по URL с настройкой pull-зеркалирования

Вы можете настроить pull-зеркалирование во время импорта репозитория по URL: проект будет создан и сразу настроен как pull-зеркало исходного репозитория. При этом не требуется настраивать зеркалирование отдельно после импорта.

> Зеркалирование репозиториев должно быть включено администратором на уровне инстанса. В «Области администратора» перейдите в «Настройки» → «Репозиторий» → «Зеркалирование репозитория» (настройка `mirror_available`). Если зеркалирование не включено на уровне инстанса, флажок «Зеркалировать репозиторий» отображается неактивным (недоступен для выбора).

Чтобы импортировать репозиторий по URL с настройкой pull-зеркалирования, выполните следующие шаги:

1. Создайте новый проект:
    - Перейдите в «Новый проект» → «Импортировать проект» → «Репозиторий по URL».
    - Укажите URL-адрес исходного репозитория.
    - Если для доступа к исходному репозиторию требуется аутентификация, укажите учётные данные.
2. Установите флажок «Зеркалировать репозиторий».
3. Создайте проект.

При первоначальном импорте репозиторий создаётся и наполняется данными из источника. После этого проект автоматически поддерживается в актуальном состоянии по расписанию pull-зеркалирования (подробнее — в разделе «Работа расписания и обработка ошибок» ниже).

Чтобы ограничить зеркалирование защищёнными ветками или ветками, подходящими под шаблон, после импорта настройте фильтр веток на странице проекта «Настройки» → «Репозиторий» → «Зеркалирование репозиториев» (подробнее — в разделе «Фильтр веток» выше).

Также вы можете превратить существующий проект в pull-зеркало, выполнив в него импорт по URL; для этого требуется роль Maintainer. В обоих случаях зеркалирование выполняется от имени пользователя, который его настроил.

Работа расписания и обработка ошибок

- Назначение задач pull-зеркалирования выполняется один раз в час (`Projects::PullMirrorScheduleWorker`).
- Каждое зеркало обновляется не чаще одного раза в 6 часов.
- Если задача зеркалирования завершилась с ошибкой, следующая попытка будет выполнена при следующем запуске `Projects::PullMirrorScheduleWorker`. Максимальное количество попыток при ошибках — **5**. Нажатие кнопки «Обновить сейчас» сбрасывает счётчик неуспешных попыток.
- Если задача Sidekiq завершилась аварийно (например, при перезапуске Sidekiq), её статус будет обновлён через 3 часа, после чего будет добавлена новая попытка синхронизации.

Особенности работы с LFS (Large File Storage)

При pull-зеркалировании LFS-объекты загружаются **только в том случае**, если LFS включён в целевом проекте Deckhouse Code.

CODEOWNERS

Функция **CODEOWNERS** позволяет определить ответственных за отдельные части репозитория. Ответственными могут быть пользователи и группы. Изменения, затрагивающие файлы, указанные в `CODEOWNERS`, должны получить апрув от владельцев кода.

Использование CODEOWNERS помогает:

- требовать апрувы от доменных экспертов для важных директорий;
- упростить процесс поиска «ответственных» за конкретный участок кода.

CODEOWNERS дополняет механизм правил апрува, но не заменяет их. В отличие от правил апрува, которые задаются вручную в UI, CODEOWNERS работает через файл `CODEOWNERS` в репозитории.

Как работает CODEOWNERS

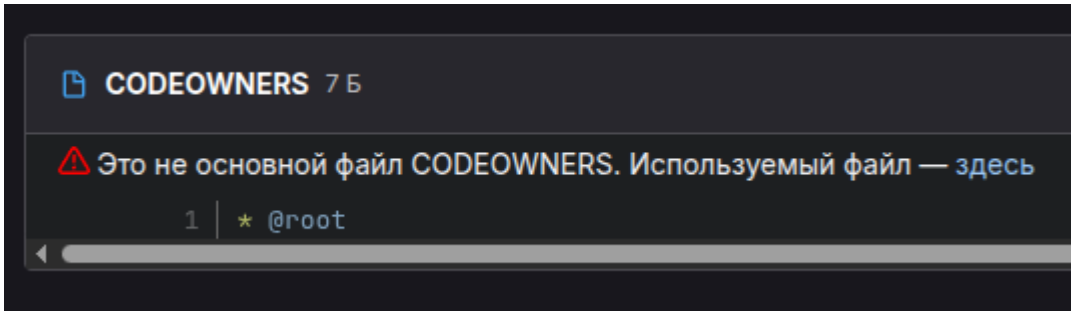
`CODEOWNERS` — текстовый файл в репозитории. Deckhouse Code использует его для определения владельцев файлов, участвующих в запросе на слияние.

Ищется в трёх местах (по приоритету):

1. `./CODEOWNERS`.
2. `./docs/CODEOWNERS`.
3. `./.gitlab/CODEOWNERS`.

Используется **только первый найденный файл**.

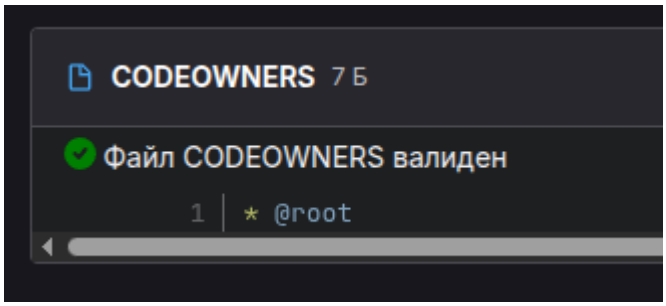
Если в проекте существует несколько файлов и вы сейчас просматриваете неприоритетный, то увидите соответствующее сообщение.



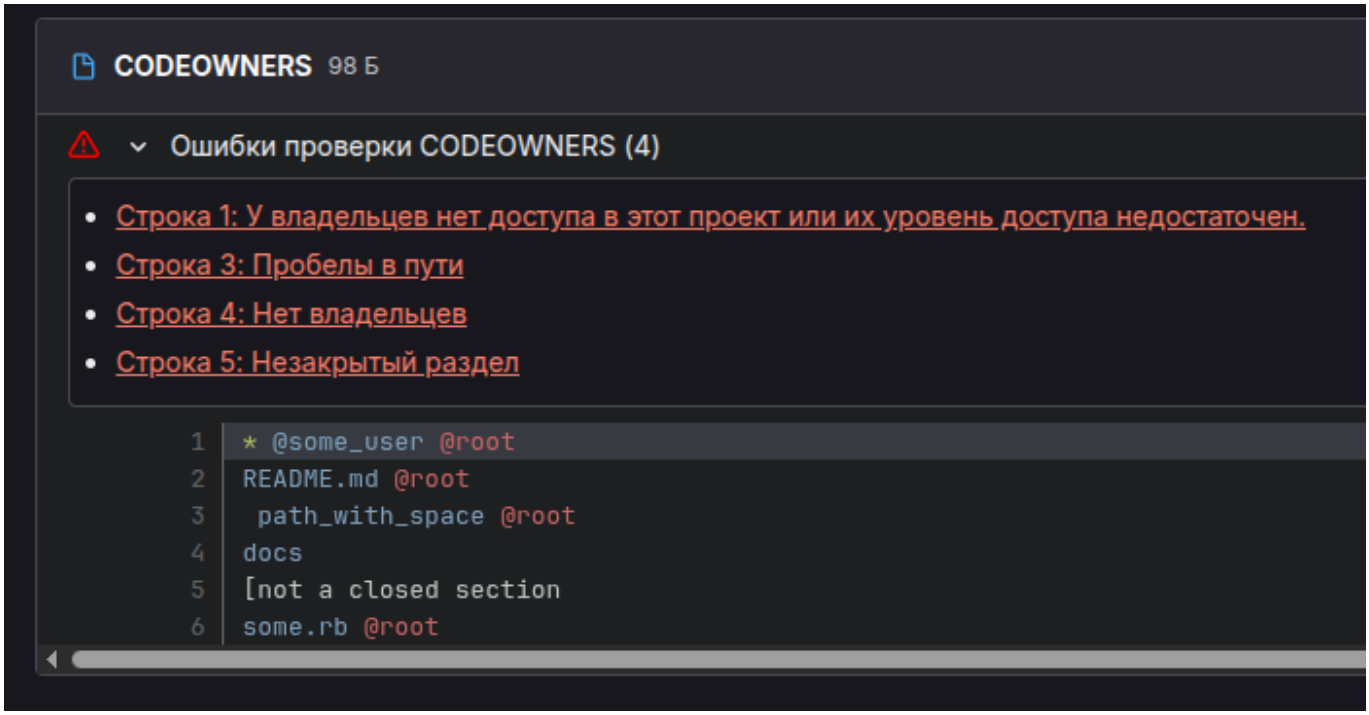
Валидация файла CODEOWNERS

Валидация помогает увидеть ошибки в файле на этапе редактирования.

Если файл валиден, отображается соответствующее сообщение:



Если файл невалиден, будут отображены проблемные строки и тип ошибки:



Валидация сообщает о следующих типах ошибок:

- *У владельцев нет доступа в этот проект или их уровень доступа недостаточен* — как минимум один из владельцев не имеет достаточных прав в проекте.
- *Пробелы в пути* — если путь содержит пробелы, нужно их экранировать символом `\`.
- *Нет владельцев* — не указано ни одного владельца.
- *Незакрытый раздел* — не хватает символа закрытия раздела `]`.

Где применяется CODEOWNERS

Если MR, изменяющий файл, попадает под правила CODEOWNERS, то необходимо получить апрувы от владельцев.

Формат CODEOWNERS

Строка правила:

```
<путь или паттерн> <список владельцев>
```

Владельцы:

- `@user`;
- `@group`;
- `@group/subgroup`;
- роли: `@@developer`, `@@maintainer`, `@@owner`.

Примеры

```
# Владелец всех файлов
* @default-team

# README.md только в корне
/README.md @docs-team

# Все Ruby-файлы
*.rb @backend-team

# Вся директория config
/config/ @devops

# README.md в любом месте
README.md @docs
```

Паттерны путей

Deckhouse Code использует подстановочные символы:

- `*` — любые символы кроме `/`.

- `**` — матч нескольких уровней директорий.
- `/dir/` — только указанная директория.
- `*.md` — все Markdown-файлы.
- `/config/**/*.rb` — все Ruby-файлы внутри `config/*` на всех уровнях.

Примеры паттернов

```
/docs/*       @docs-one      # один уровень
/docs/**      @docs-two      # рекурсивно
/app/**/*.rb  @ruby-team
```

Секции CODEOWNERS

В файле `CODEOWNERS` секции — это именованные области, которые анализируются отдельно и всегда применяются. Пока вы не определите ни одной секции, Deckhouse Code рассматривает весь файл `CODEOWNERS` как одну единственную секцию.

Особенности использования `CODEOWNERS`:

- Deckhouse Code рассматривает записи без секций, включая правила, определённые до первой секции, как отдельную, безымянную секцию.
- Каждая секция обрабатывает свои правила отдельно.
- Если путь к файлу соответствует нескольким записям внутри одной секции, используется только последняя подходящая запись в этой секции.
- Если путь к файлу соответствует записям в нескольких секциях, используется последняя подходящая запись в каждой секции.

Например, в файле `CODEOWNERS` со следующими секциями, определяющими владельцев для README:

```
* @root

[README Owners]
README.md @user1 @user2
internal/README.md @user4

[README other owners]
README.md @user3
```

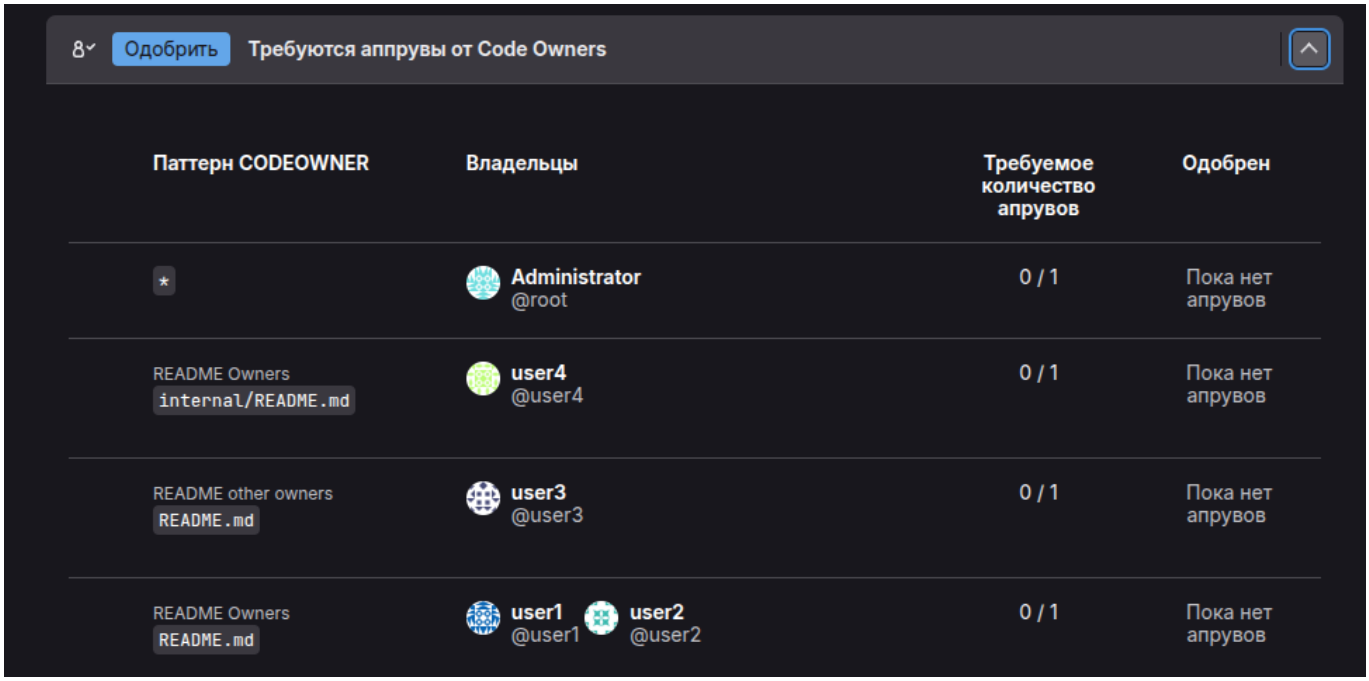
Владельцы для `README.md` в корневой директории:

- `@root` — из безымянной секции.
- `@user1` и `@user2` — из секции `[README Owners]`.
- `@user3` — из секции `[README other owners]`.

Владельцы для `internal/README.md`:

- `@root` — из безымянной секции.
- `@user4` — из секции `[README Owners]`. (И `README.md`, и `internal/README.md` подходят под правила секции, но используется только последняя подходящая запись в этой секции).
- `@user3` — из секции `[README other owners]`.

В виджете запроса на слияние каждый владелец кода отображается под своим ярлыком.



Детальный пример: сложная комбинация секций

Ниже — пример реального промышленного файла, показывающий:

- секции с разным количеством апрувов;
- переопределение владельцев;
- опциональные секции;
- исключения;
- наследование секций с одинаковыми именами.

```
# Глобальные настройки
* @fallback-team
!*.lock                    # lock-файлы не требуют апрувов
!**/generated/**           # автоматическая генерация

[Backend][2] @backend-core
# Требуются 2 апрува от backend-core для любого Ruby-кода
app/**/*.rb

# Но для критических моделей определяем другой порядок
app/models/**/*.rb @backend-core @security-team

# Эта модель — исключение, владелец другой
app/models/legacy/**/*.rb @migration-team

[Backend]
# Дополнение секции: теперь Backend включает также SQL
db/**/*.sql @db-team

[Ruby Optional]
# Дополнительная подсветка владельцев, апрувы НЕ обязательны
^[Ruby Optional]
*.rb @ruby-advisors

[Frontend][3]
# Для UI требуется 3 апрува
*.vue @frontend-team
*.js  @frontend-team

# переопределение: конкретный файл → конкретный человек
frontend/critical_entry.vue @frontend-lead

[Docs]
*.md @technical-writers

[Docs]
# переопределение: README всегда принадлежит docs-lead
README.md @docs-lead
```

Особенности примера:

- секция `[Backend]` объявлена дважды → Deckhouse Code объединит её;
- `[Backend][2] @backend-core` задаёт **обязательные 2 апрува**;
- `[Frontend][3]` требует **3 апрува** от фронтенда;
- `[Ruby Optional]` отмечена как опциональная — владельцы видны, но их апрув не обязателен;

- исключения `!*.lock` и `!**/generated/**` означают, что эти файлы вообще не требуют апрувов.

Исключения (!)

Используются для исключения файлов внутри секции:

```
* @default
!package-lock.json
!**/generated/**
```

После исключения файл **не может быть снова включён** в ту же секцию.

Порядок обработки правил

Правила обрабатываются в соответствии со следующим порядком:

- Deckhouse Code читает файл сверху вниз.
- В пределах одной секции правила с одинаковым путём **переопределяют** предыдущие.
- Если файл подходит под несколько секций — владельцы из всех этих секций добавляются.
- Секции не переопределяют друг друга — они *независимы*.

Кто может быть владельцем (eligible owners)

1. Пользователи (через @username)

Пользователь считается валидным владельцем, если:

- Имеет **прямое** членство в проекте (Developer, Maintainer или Owner).
- Имеет членство в группе проекта (включая наследуемое: группа проекта, родительские группы, предки родительских групп).
- Является прямым или унаследованным участником группы, которая напрямую приглашена в проект.
- Является прямым но не унаследованным участником группы которая приглашена в группу проекта.
- Является прямым но не унаследованным участником группы которая приглашена в предка группы проекта.

Если подытожить, то подходят все пользователи которые отображаются на странице участников проекта с ролью developer или выше.

Не считается владельцем, если:

- заблокирован,
- доступ отозван,
- роль ниже Developer.

2. Группы (через @group или @group/subgroup)

Группа может быть владельцем **только если:**

- Она напрямую приглашена в проект с ролью Developer+ (через «Пригласить группу»).
- Владельцами считаются только *прямые участники* этой группы.

Наследуемые участники родительских групп владельцами не считаются.

3. Роли (через @@role)

Формат:

```
@@developer
@@maintainer
@@owner
```

Особенности:

- учитываются **только прямые участники проекта**;
- члены групп в расчёт не входят;
- роли не наследуют друг друга (`@@developer` НЕ включает maintainer'ов).

Можно указывать несколько ролей в одной строке:

```
file.md @@developer @@maintainer
```

Взаимодействие с правилами апрува

CODEOWNERS и правила апрувов работают независимо, но их требования суммируются.

Пример:

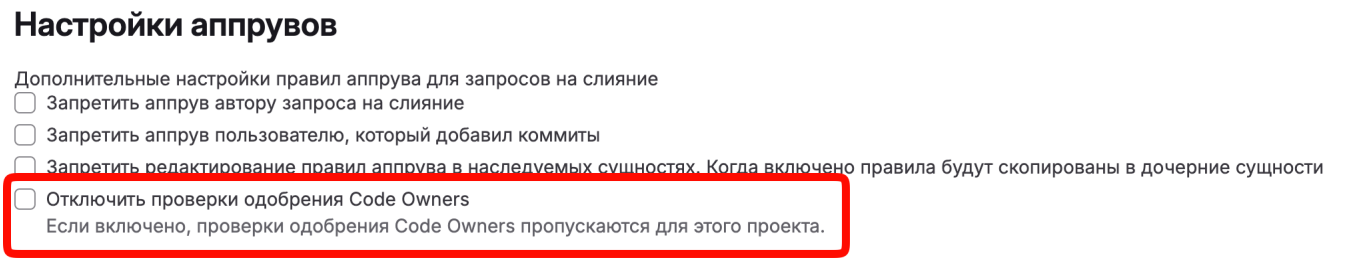
- правило `Security` требует 1 апрув;
- секция `[Backend][2]` требует 2 апрува;
- MR меняет backend-файл.

Итог: необходимо **3 апрува**.

Отключение проверок CODEOWNERS

Проверки CODEOWNERS, которые включены в Deckhouse Code по умолчанию, можно отключить на уровне проекта. Это удобно, когда в проекте есть файл `CODEOWNERS`, но требовать апрувы от владельцев кода не нужно.

Чтобы отключить проверки, перейдите в раздел «Проект» → «Настройки» → «Запросы на слияние» → раздел дополнительных настроек правил апрува и выберите настройку «Отключить проверки одобрения Code Owners».



При отключении проверки CODEOWNERS пропускаются во всём проекте:

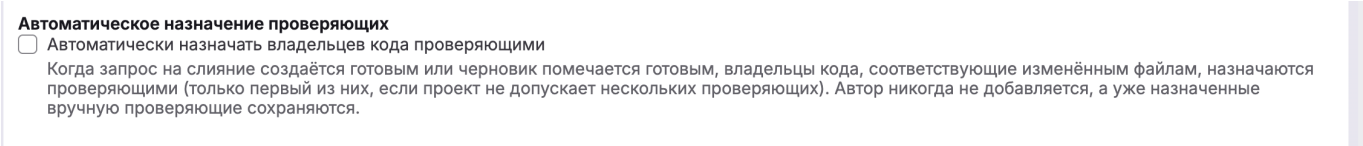
- запросы на слияние не блокируются из-за отсутствующих апрувов от владельцев кода;
- владельцы кода не определяются для изменённых файлов — список применимых записей пуст;
- изменение настройки сразу применяется к уже открытым запросам на слияние.

Проверка синтаксиса файла `CODEOWNERS` при этом продолжает работать. Файл можно валидировать даже при отключённых проверках.

Автоматическое назначение владельцев кода проверяющими

Владельцев кода, соответствующих изменённым файлам, можно автоматически назначать проверяющими запроса на слияние.

Для этого перейдите в раздел «Проект» → «Настройки» → «Запросы на слияние» → «Автоматическое назначение проверяющих» и выберите настройку «Автоматически назначать владельцев кода проверяющими».



По умолчанию настройка выключена. При её включении владельцы кода назначаются проверяющими:

- при создании запроса на слияние сразу в готовом статусе (не в статусе черновика);
- при снятии статуса черновика с запроса на слияние.

При автоматическом назначении действуют следующие ограничения:

- для черновиков назначение не выполняется;
- если у запроса на слияние уже есть проверяющие, автоматическое назначение не выполняется — назначенные вручную проверяющие сохраняются;
- автор запроса на слияние не может быть назначен проверяющим;
- назначаются только пользователи, у которых есть доступ на чтение запроса на слияние;
- если в проекте запрещено назначать нескольких проверяющих, назначается только первый владелец кода.

Рекомендации по оформлению

При оформлении файла `CODEOWNERS` следуйте рекомендациям:

- Сначала задавайте широкие правила (`* @team`). Далее уточняющие (`app/**/*.rb @backend-team`).
- Используйте секции для лучшей структуры и наглядности.
- Избегайте дубликатов — последнее правило в секции имеет приоритет.
- Опциональные секции полезны для подсветки экспертов без обязательных апрувов.

Пример финального файла

```
# Глобальные владельцы
* @default-team

# Исключения
!yarn.lock
!**/generated/**

[Backend][2]
app/**/*.rb @backend-core

[Frontend][3]
*.vue @frontend-team
*.js @frontend-team

^[Docs]
*.md @docs-team

[Critical Overrides]
/config/production.yml @ops-lead
```

Групповая аналитика репозитория

Отчёт «Аналитика репозитория» доступен на уровне группы. Он показывает покрытие кода тестами в проектах группы и её подгрупп. Данные появляются после завершения в ветке по умолчанию пайплайна, который публикует покрытие.

Доступность отчёта

Отчёт доступен участникам группы с ролью «Наблюдатель» и выше, а также пользователям с типом «Аудитор». Гости и пользователи, не выполнившие вход, не могут открыть отчёт, включая отчёт публичной группы.

Просмотр отчёта

1. В верхней панели нажмите «Искать или перейти к…» и выберите группу.

2. В меню группы перейдите в «Аналитика» → «Аналитика репозитория».

Также отчёт можно открыть по прямому адресу:

```
https://<HOST>/groups/<GROUP>/-/analytics/repository_analytics
```

Покрытие кода тестами

Раздел «Покрытие кода тестами» показывает данные за последний день, в котором есть данные о покрытии:

- «Проекты с покрытием» — число проектов с данными о покрытии;
- «Среднее покрытие по заданиям» — среднее покрытие по заданиям, которые его публикуют;
- «Задания с покрытием» — число заданий, публикующих покрытие.

Подпись «Обновлено» указывает этот день.

Среднее покрытие тестами

График «Среднее покрытие тестами» показывает среднее покрытие всех заданий по дням за последние 30 дней. Дни без данных о покрытии на графике не отмечены.

Наведите указатель на точку графика, чтобы увидеть дату, процент покрытия и число заданий с покрытием.

Последние результаты покрытия тестами

Таблица «Последние результаты покрытия тестами» по умолчанию показывает до 20 проектов с покрытием. Строки отсортированы в алфавитном порядке. Чтобы заменить этот набор, выберите один или несколько проектов в списке «Проекты». В список входят проекты подгрупп.

Колонка	Содержимое
«Проект»	Название проекта со ссылкой на его график покрытия
«Покрытие»	Среднее покрытие по заданиям проекта
«Задания с покрытием»	Число заданий, публикующих покрытие
«Последнее обновление»	Дата последних данных о покрытии

Скачивание данных о покрытии

Отчёт позволяет скачать исторические данные о покрытии для всех проектов группы и её подгрупп или для выбранных проектов. CSV-файл содержит до 1000 последних записей в порядке от новых

к старым. Каждая строка содержит дату, имя группы заданий, название проекта и процент покрытия.

Чтобы скачать CSV-файл:

1. Нажмите «Скачать исторические данные покрытия тестами (.csv)».
2. В списке «Проекты» выберите проекты или оставьте «Все проекты».
3. В поле «Диапазон дат» выберите 7, 14, 30, 60 или 90 дней. По умолчанию выбрано 30 дней.
4. Нажмите «Скачать данные покрытия тестами (.csv)».

Аналитика репозитория проекта

Отчёт «Аналитика репозитория» доступен на уровне проекта. Он показывает языки программирования в ветке репозитория по умолчанию, а также покрытие кода тестами и статистику коммитов для выбранной ветки или тега.

Доступность отчёта

Отчёт доступен пользователям, которые могут скачать код проекта. В проекте должен быть инициализированный репозиторий и хотя бы один коммит в ветке по умолчанию. Коммиты вики в отчёт не входят.

Просмотр отчёта

1. В верхней панели нажмите «Искать или перейти к…» и выберите проект.
2. В меню проекта перейдите в «Аналитика» → «Аналитика репозитория».

Также отчёт можно открыть по прямому адресу:

```
https://<HOST>/<GROUP>/<PROJECT>/-/graphs/<REF>/charts
```

Чтобы посмотреть покрытие кода тестами и статистику коммитов для другой ветки или тега, выберите её или его в списке рядом с «Статистика коммитов». График языков программирования всегда использует ветку по умолчанию.

Языки программирования

График «Языки программирования, используемые в этом репозитории» показывает долю кода каждого языка программирования в ветке проекта по умолчанию. Расчёт использует размер кода в байтах и исключает сгенерированный и сторонний код.

Покрытие кода тестами

График «Статистика покрытия кода» показывает данные о покрытии выбранной ветки или тега за последние 90 дней. Данные появляются после завершения пайплайна с данными о покрытии.

Если покрытие публикуют несколько групп заданий, выберите их название в списке над графиком. Наведите указатель на точку графика, чтобы увидеть дату и процент покрытия. Если на графике есть данные, нажмите «Скачать исходные данные (.csv)», чтобы скачать их.

Данные о покрытии доступны только пользователям, которые могут просматривать пайплайны и результаты заданий проекта. Для публичного проекта пользователь, не выполнивший вход, может просматривать покрытие при включённых публичных пайплайнах.

Статистика коммитов

Раздел «Статистика коммитов» использует до 2000 коммитов без коммитов слияния из выбранной ветки или тега. Он показывает общее число коммитов, среднее число за день и число авторов, а затем графики коммитов:

- по дням месяца;
- по дням недели;
- по часам в UTC.

Запросы на слияние

Ревью кода

Merge Requests (MR) — запросы на слияние

Ключевой инструмент обеспечения качества кода перед его объединением в основную ветку.

Основные возможности:

- Создание MR после коммита в функциональную ветку.
- Автоматическое сравнение изменений с целевой веткой.
- Встроенные обсуждения и ревью кода.
- Поддержка авто-слияния после успешных проверок (CI/CD, ревью).
- Визуальное отображение конфликтов и средства для их разрешения.

Просмотр изменений (Diff View)

Позволяет визуально сравнивать версии кода.

Поддерживаемые режимы:

- Стандартный diff — отображение изменений по коммитам.
- Side-by-side diff — сравнение строк кода построчно.
- Inline diff — подсветка изменений внутри строк..

Комментарии и обсуждения

Инструменты совместной работы над изменениями в коде.

Возможности:

- Комментирование отдельных строк кода.
- Ведение обсуждений с упоминанием участников.
- Пометка обсуждений как разрешённых (Mark as resolved).
- Автоматическое формирование задач по комментариям.

Уведомления о событиях

Система оповещений о событиях в проекте.

Поддержка:

- Email-уведомлений о новых коммитах, MR и комментариях.
- Интеграция с внешними мессенджерами: Slack, Mattermost, Telegram (настраиваемые уведомления).

Подтверждение Merge Request’ов (Approvals)

Механизм контроля качества через обязательное одобрение MR перед слиянием.

Возможности:

- Настройка минимального числа одобрений.
- Назначение обязательных ревьюеров.
- Поддержка auto-approval по заданным условиям.
- Блокировка слияния при отсутствии требуемых подтверждений.
- Интеграция с механизмом Codeowners для обязательного ревью владельцами кода.
- Автоматический сброс подтверждений при изменении содержимого MR.
- Логирование всех действий, связанных с процессом одобрения.

Codeowners

Механизм назначения ответственных за участки кодовой базы.

Функциональность:

- Определение владельцев для файлов и директорий проекта.
- Использование файла CODEOWNERS для описания ответственности.
- Автоматическое назначение владельцев в ревьюеры для MR.
- Интеграция с системой Approvals: владельцы становятся обязательными ревьюерами.
- Поддержка шаблонов путей (`*`, `**`) и групп владельцев.
- Возможность ограничить изменения в защищённых ветках только для владельцев.

Правила апрувов запросов на слияние

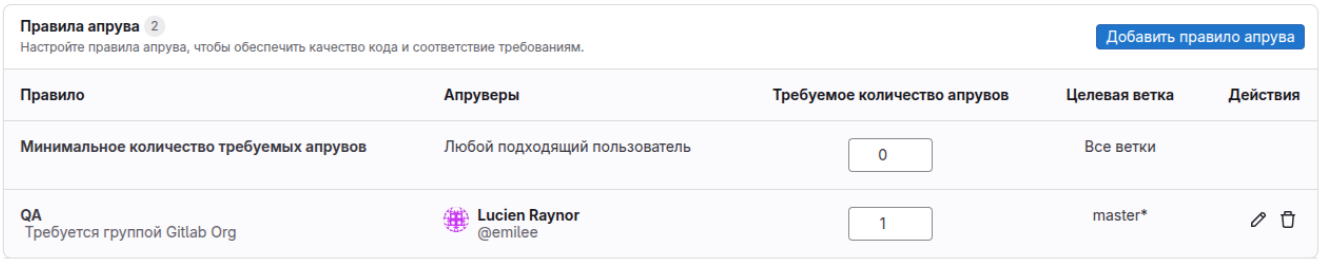
Правила апрувов помогают контролировать качество кода и обеспечивать обязательное ревью перед слиянием. С их помощью можно определить, сколько апрувов нужно получить, кто именно должен участвовать в проверке, и в каких случаях правило должно срабатывать.

Эти правила работают на уровнях группы, проекта и запросов на слияние, наследуются вниз по иерархии и позволяют централизованно управлять процессом ревью. Вместе с CODEOWNERS они формируют гибкую и надёжную систему контроля изменений, защищая важные части репозитория от нежелательных или непросмотренных правок.

Определение правил

Правила можно определить на уровне:

- Группы (доступно для роли **Maintainer**). Чтобы определить правила для группы, перейдите в «Группа» → «Настройки» → «Запросы на слияние» → «Правила апрува».
- Проекта (доступно для роли **Maintainer**). Чтобы определить правила для проекта, перейдите в «Настройки» → «Запросы на слияние» → «Правила апрува».
- Запроса на слияние (доступно для роли **Developer**). Чтобы определить правила для запроса на слияние, на странице создания или редактирования MR раскройте секцию **«Правила апрува»**.



Описание столбцов таблицы правил апрува

- **Правило** — содержит название правила и признак наследования. Запись «Требуется группой Example Org» означает, что правило создано в группе *Example Org*.
- **Апруверы** — содержит список пользователей и групп, чьи апрувы учитываются при проверке выполнения правила.
- **Требуемое количество апрувов** — количество апрувов, которое необходимо получить от пользователей или участников групп, указанных в столбце «Апруверы». Значение `Любой подходящий пользователь` означает, что будет засчитан апрув от любого пользователя, который имеет право его выполнить.
- **Целевая ветка** — правило применяется только если запрос на слияние направлен в указанную ветку. В качестве названия ветки можно использовать wildcard (`*`).

Наследование правил

Правила апрувов наследуются каскадно: группа → подгруппа → проект → запрос на слияние. Механизм наследования имеет следующие особенности:

- Когда создаётся подгруппа, проект или запрос на слияние, правила родительской сущности автоматически появляются в новой сущности как унаследованные. В таблице правил рядом с названием отображается, на каком уровне правило было изначально создано.
- При создании новых правил на уровне группы или проекта они автоматически добавляются во все существующие нижестоящие подгруппы и проекты. **В уже созданные запросы на слияние новые правила не добавляются**.
- Если изменить правило на родительском уровне, изменения автоматически применяются ко всем унаследованным правилам на нижестоящих уровнях. Например, если проект унаследовал правило из группы и в группе изменить количество требуемых апрувов, то это изменение отразится и в проекте.
- Если же изменить унаследованное правило в дочерней сущности, **оно превращается в самостоятельное правило**. То есть правило перестаёт быть связанным с родительским: на дочернем уровне остаётся собственная копия с новыми параметрами, и дальнейшие изменения правила в группе уже не затронут это правило в проекте.

Правила для группы

Правила, созданные на уровне группы, автоматически добавляются в новые проекты и подгруппы как унаследованные. Чтобы открыть их, перейдите в «Группа» → «Настройки» → «Запросы на слияние» → «Правила апрува».

Управлять правилами группы могут участники группы с ролью **Maintainer**.

В разделе «Правила апрува» отображается список правил, настроенных на уровне группы.

Создание правила в группе

Создать новое правило можно, нажав кнопку **«Добавить правило апрува»** и указав:

- Название правила.
- Целевую ветку (доступны пресеты «Все ветки», «Все защищённые ветки», «Ввести название ветки», либо wildcard `*`).
- Требуемое количество апрувов.
- Апруверов — прямых участников группы или группы на инстансе. При добавлении группы учитываются апрувы только от прямых участников добавленной группы.

Правила для проекта

Правила, созданные на уровне проекта, автоматически добавляются в новые запросы на слияние как унаследованные. Чтобы открыть их, перейдите в «Проект» → «Настройки» → «Запросы на слияние» → «Правила апрува».

Управлять правилами проекта могут участники проекта с ролью **Maintainer**.

Создание правила в проекте

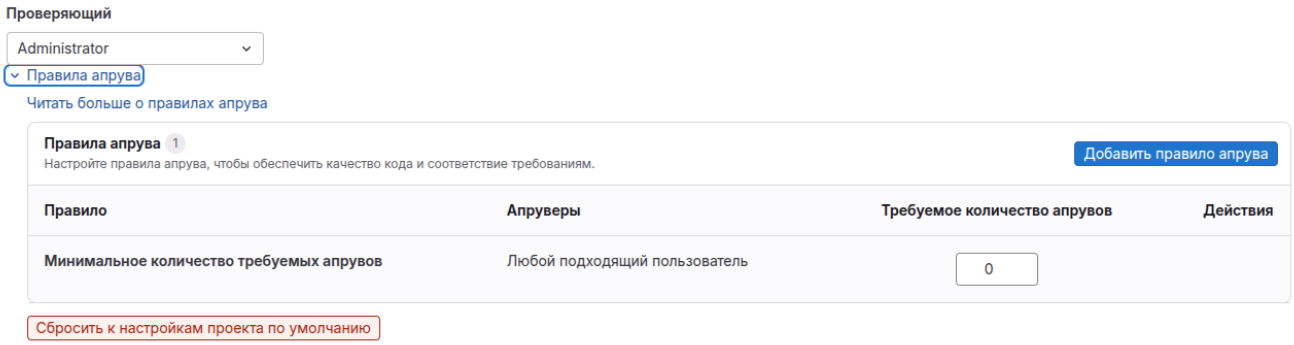
При создании нового правила можно указать:

- Название правила.
- Целевую ветку (можно выбрать пресет, указать название ветки или регулярное выражение; также можно выбрать защищённую ветку проекта).
- Требуемое количество апрувов.
- Апруверов — прямых или унаследованных участников проекта, а также группы, приглашённые в проект с ролью **Reporter**. Со стороны добавленной группы учитываются апрувы только от прямых участников группы.

Правила для запроса на слияние

Чтобы открыть правила запроса на слияние, на странице его создания или редактирования раскройте секцию **«Правила апрува»**.

Правилами запроса на слияние могут управлять пользователи с ролью **Developer**.



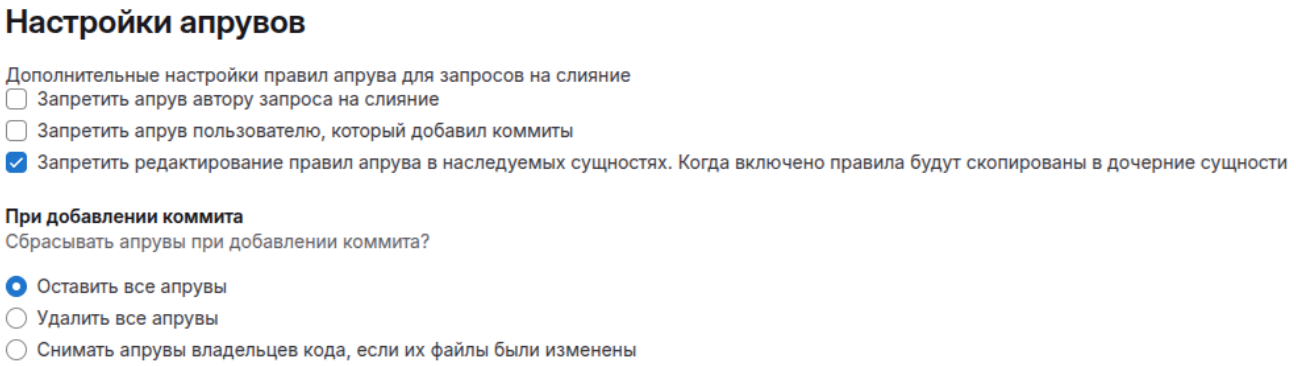
На странице MR доступна кнопка **«Сбросить к настройкам проекта по умолчанию»** — она удаляет текущие правила и добавляет все правила проекта, которые применимы для целевой ветки MR.

Создание правила в MR

При создании правила можно указать:

- Название правила.
- Требуемое количество апрувов.
- Апруверов — прямых или унаследованных участников проекта, а также группы с ролью **Reporter** или выше. Со стороны группы учитываются апрувы только от прямых участников группы.

Дополнительные настройки апрувов



На уровне проекта или группы можно настроить дополнительные параметры:

- **Запретить апрув автору запроса на слияние** — автор MR не может делать апрув.
- **Запретить апрув пользователю, который добавил коммиты** — коммиттеры не могут делать апрувы.
- **Запретить редактирование правил в наследуемых сущностях** — при включении:
  - правила копируются в дочерние сущности и становятся недоступны для редактирования или удаления;
  - нижестоящие группы и проекты могут создавать только собственные правила;
  - в существующие MR правила автоматически не добавляются;
  - текущие настройки апрувов также распространяются на все дочерние сущности.
- **При добавлении коммита** — секция определяет, должны ли сбрасываться существующие апрувы при появлении нового коммита.

Кто может делать апрувы

Если настройки апрувов этого не запрещают, апрувы могут делать:

- все участники проекта с ролью **Developer** и выше;
- участники с ролью **Planner** и выше, если они назначены ответственными в запросе на слияние.

Внешние проверки статуса в запросах на слияние

Внешние проверки статуса позволяют мейнтейнерам проекта отправлять данные запроса на слияние во внешний сервис и использовать ответ сервиса как дополнительное условие слияния. Используйте их, если перед слиянием запрос должен пройти внешнюю проверку соответствия требованиям, контроль качества или проверку безопасности.

Внешние проверки статуса настраиваются отдельно для каждого проекта и не наследуются другими проектами.

Как работают внешние проверки статуса

Когда происходит событие запроса на слияние, Deckhouse Code отправляет информацию о запросе во все внешние сервисы, проверки статуса через которые применяются к целевой ветке. Внешний сервис проверяет запрос на слияние и отправляет результат обратно в Deckhouse Code.

Результат проверки всегда относится к текущему `HEAD` SHA исходной ветки. Если в запрос на слияние добавлены новые коммиты, предыдущий результат больше не относится к новому состоянию, и Deckhouse Code ожидает новый результат.

Внешние проверки статуса влияют на возможность слияния только если в проекте включена настройка **Проверки статуса должны быть успешными**. Если настройка выключена, виджет продолжает показывать результаты проверок, но проверки в состоянии `с ошибкой` или `ожидает выполнения` не блокируют слияние.

Настройка внешних проверок статуса

Чтобы настроить внешние проверки статуса, откройте проект и перейдите в «Настройки» → «Запросы на слияние».

На странице используются две связанные области настроек:

- «Проверки слияния» — настройки проекта, влияющие на возможность слияния.
- «Внешние проверки статуса» — таблица для создания, обновления и удаления внешних сервисов проверки статуса.

Пользователю нужно право на управление настройками запросов на слияние в проекте. В стандартных ролях оно доступно пользователям с ролью **Мейнтейнер** или **Владелец**.

Настройки проверок слияния

Проверки статуса должны быть успешными

Флажок «Проверки статуса должны быть успешными» блокирует запросы на слияние до тех пор, пока все применимые внешние проверки статуса не перейдут в состояние `пройдено`.

Если флажок выключен, внешние проверки статуса продолжают отображаться в запросах на слияние, но не блокируют слияние.

Тайм-аут проверок статуса

Поле «Тайм-аут проверок статуса» задает тайм-аут по умолчанию для проверок статуса в проекте.

Поведение:

- Значение по умолчанию — `5` минут.
- Значение должно быть `1` минута или больше.
- Значение применяется к проверкам без индивидуального тайм-аута.
- Если внешний сервис не ответил до истечения тайм-аута, ответ проверки переходит в состояние `с ошибкой`.

Таблица внешних проверок статуса

Таблица «Внешние проверки статуса» показывает все сервисы проверки статуса, настроенные в проекте.

В таблице отображаются:

- Название проверки статуса.
- URL внешнего сервиса.
- Область применения по целевым веткам.
- Тайм-аут проверки или тайм-аут проекта по умолчанию.
- Состояние общего секрета HMAC.
- Действия «Обновить» и «Удалить».

Если у проверки не задан индивидуальный тайм-аут, в таблице отображается тайм-аут проекта с пометкой `(по умолчанию)`.

Добавление внешней проверки статуса

Чтобы добавить внешнюю проверку статуса:

1. Откройте проект.
2. Перейдите в «Настройки» → «Запросы на слияние».
3. В блоке «Внешние проверки статуса» нажмите «Добавить внешнюю проверку статуса».
4. Заполните поля.
5. Нажмите «Новая внешняя проверка статуса».

Поле	Описание
Название сервиса	Название внешнего сервиса. Значение обязательно, должно быть уникальным в проекте и не должно превышать 255 символов
API для проверки	URL эндпоинта внешнего сервиса. Значение обязательно, должно быть уникальным в проекте и должно использовать протокол `http` или `https`
Целевая ветка	Область, определяющая целевые ветки запросов на слияние, для которых применяется проверка
Тайм-аут в минутах	Индивидуальный тайм-аут проверки. Если задан, переопределяет значение проекта **Тайм-аут проверок статуса**
Общий секрет HMAC	Необязательный секрет для подписи запросов, отправляемых из Deckhouse Code во внешний сервис

После создания проверки Deckhouse Code создает ответы в состоянии `ожидает выполнения` для подходящих открытых запросов на слияние. Запросы во внешний сервис для таких уже существующих запросов на слияние не отправляются. Эти ответы перейдут в состояние `с ошибкой` после истечения тайм-аута, если пользователь не перезапустит проверку.

Для новых событий запросов на слияние Deckhouse Code автоматически отправляет запросы в подходящие внешние сервисы проверки статуса.

Область применения по веткам

Поле «Целевая ветка» определяет, для каких запросов на слияние используется проверка статуса.

Значение поля «Целевая ветка»	Описание
Все ветки	Проверка применяется к запросам на слияние в любую целевую ветку
Все защищённые ветки	Проверка применяется к запросам на слияние в защищённые ветки
Выбранные защищённые ветки	Проверка применяется только к запросам на слияние в выбранные защищённые ветки. Поддерживаются защищённые ветки с wildcard-шаблонами

Если целевая ветка запроса на слияние не соответствует области применения проверки, проверка не отображается в виджете запроса на слияние.

Когда целевая ветка меняется, Deckhouse Code пересчитывает применимые проверки. Проверки, которые больше не применяются, удаляются из запроса на слияние, а новые применимые проверки добавляются в состоянии `ожидает выполнения`.

Общий секрет HMAC

Если задан **Общий секрет HMAC**, Deckhouse Code добавляет заголовок `X-Gitlab-Signature` к запросам во внешний сервис.

Значение заголовка — HMAC-SHA256 от тела запроса, сформированный с использованием общего секрета.

Секрет хранится в зашифрованном виде. После сохранения интерфейс показывает только факт наличия секрета. Чтобы заменить секрет:

1. В строке проверки нажмите «Обновить».
2. Нажмите «Изменить секрет».
3. Введите новое значение.
4. Нажмите «Обновить проверку статуса».

Жизненный цикл проверки

Когда происходит событие запроса на слияние, Deckhouse Code отправляет данные запроса на слияние во все применимые внешние сервисы проверки статуса. Данные содержат объект `external_approval_rule` с `id`, `name` и `external_url` проверки.

Ответ проверки может иметь один из следующих статусов:

Статус	Описание
`ожидает выполнения`	Deckhouse Code ожидает ответ внешнего сервиса
`пройдено`	Внешний сервис подтвердил состояние запроса на слияние
`с ошибкой`	Внешний сервис отклонил состояние запроса на слияние, запрос к внешнему URL завершился ошибкой или истек тайм-аут

Внешний сервис обновляет ответ через API-эндпоинт:

```
POST /projects/:id/merge_requests/:merge_request_iid/status_check_responses
```

Запрос должен содержать:

- `external_status_check_id` — идентификатор проверки статуса.
- `sha` — текущий `HEAD` SHA исходной ветки запроса на слияние.
- `status` — `пройдено` или `с ошибкой`.

Deckhouse Code не обновляет ответ, если:

- `sha` не совпадает с текущим `HEAD` исходной ветки.
- Ответ больше не находится в состоянии `ожидает выполнения`.
- Тайм-аут уже истек.

Тайм-ауты

Отсчёт тайм-аута начинается в момент отправки запроса во внешний сервис. Если пользователь перезапускает проверку в состоянии `с ошибкой`, отсчёт тайм-аута начинается с момента перезапуска.

Значение тайм-аута выбирается в следующем порядке:

1. Значение «Тайм-аут в минутах» у проверки статуса.
2. Значение проекта «Тайм-аут проверок статуса».

Фоновый обработчик проверяет ответы в состоянии `ожидает выполнения` каждую минуту. Когда ответ превышает свой тайм-аут, Deckhouse Code переводит его в состояние `с ошибкой` и записывает причину `Automatically closed after timeout`.

Если запрос во внешний сервис завершился ошибкой, Deckhouse Code переводит ответ в состояние `с ошибкой` и сохраняет причину ошибки. Пользователи могут увидеть причину ошибки в виджете запроса на слияние.

Виджет внешних проверок статуса

Виджет «Внешние проверки статуса» отображается на странице запроса на слияние, если для запроса есть применимые внешние проверки.

Виджет показывает:

- Название проверки.
- Статус проверки.
- URL внешнего сервиса, если ваша роль позволяет его видеть.
- Подробности ошибки для проверок в состоянии `с ошибкой`, если Deckhouse Code сохранил причину ошибки.
- Действие «Повторить» для проверок в состоянии `с ошибкой`, если ваша роль позволяет их перезапускать.

Виджет помогает авторам и ревьюерам понять, какие внешние системы ещё должны ответить до слияния запроса.

Проверки в состоянии «ожидает выполнения»

Проверка остаётся в состоянии `ожидает выполнения`, пока Deckhouse Code ожидает ответ внешнего сервиса.

Пока хотя бы одна проверка находится в состоянии `ожидает выполнения`, виджет запроса на слияние обновляет внешние проверки статуса примерно каждые 10 секунд. Опрос останавливается, когда проверок в состоянии `ожидает выполнения` больше нет.

Проверка в состоянии `ожидает выполнения` может автоматически перейти в состояние `с ошибкой`, если внешний сервис не ответит до истечения настроенного тайм-аута. Тайм-аут настраивается на уровне проекта или для конкретной внешней проверки статуса.

Проверки в состоянии «с ошибкой»

Проверка может перейти в состояние `с ошибкой`, если:

- Внешний сервис вернул `с ошибкой` для текущего `HEAD` SHA запроса на слияние.
- Deckhouse Code не смог отправить запрос во внешний сервис.
- URL внешнего сервиса заблокирован или недоступен.
- Внешний сервис не ответил до истечения тайм-аута.

Если в проекте включена настройка **Проверки статуса должны быть успешными**, проверка в состоянии `с ошибкой` блокирует слияние до тех пор, пока проверка не перейдет в `пройдено` или пока настройки проекта не будут изменены.

Перезапуск проверки в состоянии «с ошибкой»

Вы можете перезапустить проверку в состоянии `с ошибкой`, если у вас есть роль **Разработчик** или выше в проекте.

Перезапуск доступен только для проверок в состоянии `с ошибкой`, которые относятся к текущему `HEAD` SHA исходной ветки запроса на слияние.

Чтобы перезапустить проверку в состоянии `с ошибкой`:

1. Откройте запрос на слияние.
2. Найдите виджет «Внешние проверки статуса».
3. В строке проверки в состоянии `с ошибкой` нажмите «Повторить».

После перезапуска Deckhouse Code переводит проверку обратно в состояние `ожидает выполнения` и повторно отправляет текущие данные запроса на слияние во внешний сервис.

Используйте перезапуск после исправления проблемы во внешнем сервисе или если сбой был временным.

URL проверок

Виджет может показывать URL внешнего сервиса для проверки. URL виден только пользователям, роль которых позволяет просматривать URL внешних проверок статуса. В стандартных ролях проекта это доступно пользователям с ролью **Разработчик** или выше.

Если вы не видите URL, вы все равно можете видеть статус проверки, если ваша роль позволяет смотреть ответы проверок статуса.

Доступ к результатам проверок

Доступ к информации о внешних проверках статуса зависит от роли в проекте:

Действие	Минимальная роль
Создать, обновить, удалить или посмотреть сервисы проверки статуса проекта	**Мейнтейнер** или **Владелец**
Отправить обратный вызов со статусом проверки	**Разработчик**
Смотреть ответы проверок в виджете	**Наблюдатель**
Смотреть URL внешнего сервиса	**Разработчик**
Перезапустить проверку в состоянии `с ошибкой`	**Разработчик**

Для проектов с видимостью `внутренний` авторизованный пользователь без признака внешнего пользователя может читать ответы внешних проверок статуса в виджете на странице запроса на слияние, если у него есть доступ на чтение данного запроса.

URL внешней проверки при этом отображается только пользователям с правом просмотра URL внешних проверок (в стандартных ролях — `Разработчик` или выше), поэтому не отображается обычному пользователю, который не является участником проекта.

Удаление внешней проверки статуса

Чтобы удалить проверку:

1. Откройте проект.
2. Перейдите в «Настройки» → «Запросы на слияние».
3. В блоке «Внешние проверки статуса» нажмите «Удалить» для нужной проверки.
4. Подтвердите удаление.

После удаления проверка больше не применяется к запросам на слияние проекта.

События аудита

Deckhouse Code записывает события аудита для управления проверками статуса и изменения ответов.

Событие аудита	Описание
`external_status_check_created`	Создана внешняя проверка статуса
`external_status_check_updated`	Обновлена внешняя проверка статуса
`external_status_check_destroyed`	Удалена внешняя проверка статуса
`external_status_check_response_updated`	Ответ изменен через обратный вызов внешнего сервиса, перезапуск, ошибку запроса или тайм-аут

Устранение неполадок

Запрос на слияние заблокирован внешней проверкой статуса

Проверьте:

- Включен ли флажок «Проверки статуса должны быть успешными».
- Применяется ли проверка к целевой ветке запроса на слияние.
- Отправил ли внешний сервис `пройдено` для текущего `HEAD` SHA.
- Не истек ли тайм-аут до обратного вызова внешнего сервиса.

Проверка завершилась в статусе «с ошибкой» по тайм-ауту

Проверьте:

- Значение проекта «Тайм-аут проверок статуса».
- Индивидуальное значение «Тайм-аут в минутах» у проверки.

- Время ответа внешнего сервиса.
- Нужно ли перезапустить проверку после исправления внешнего сервиса.

Проверка остаётся в состоянии «ожидает выполнения»

Проверка может оставаться в состоянии `ожидает выполнения`, пока Deckhouse Code ожидает внешний сервис.

Проверьте:

- Доступен ли внешний сервис.
- Может ли сервис обработать данные запроса на слияние.
- Отправляет ли сервис ответ для текущего `HEAD` SHA.
- Достаточен ли настроенный тайм-аут для завершения обработки.

Если тайм-аут истечет, проверка перейдет в состояние `с ошибкой`.

Запрос во внешний сервис завершился ошибкой

Проверьте:

- URL в поле «API для проверки».
- Сетевой доступ от Deckhouse Code до внешнего сервиса.
- Использует ли URL протокол `http` или `https`.
- Текст ошибки в виджете запроса на слияние.

Действие Повторить недоступно

Действие «Повторить» отображается только если выполняются все условия:

- Проверка находится в состоянии `с ошибкой`.
- Проверка относится к текущему `HEAD` SHA запроса на слияние.
- У вас есть роль **Разработчик** или выше в проекте.

Если действие «Повторить» недоступно, попросите мейнтейнера проекта проверить настройки проекта и вашу роль.

URL внешнего сервиса не отображается

URL скрыт, если ваша роль не позволяет просматривать URL внешних проверок статуса. Если вам нужен доступ к URL внешнего сервиса, обратитесь к мейнтейнеру проекта.

Ошибка дублирования имени или URL

В проекте не может быть двух внешних проверок статуса с одинаковым «Название сервиса» или «API для проверки». Используйте уникальные значения для каждой проверки.

Цепочки слияния

Цепочка слияния (merge train) формирует очередь из запросов на слияние в целевую ветку. Перед слиянием каждый запрос проверяется с учётом запросов, стоящих перед ним в очереди. Благодаря этому в целевую ветку попадают только изменения, проверенные с учётом порядка их слияния.

Для проверки используются пайплайны для результатов слияния (merged results pipelines). Для первого запроса в очереди такой пайплайн проверяет его изменения вместе с текущим состоянием целевой ветки. Для каждого следующего запроса также учитываются изменения из предыдущих запросов в очереди. Поэтому если запросы проходят проверки по отдельности, но их изменения несовместимы, проблема будет обнаружена до слияния в целевую ветку.

Цепочки слияния настраиваются отдельно для каждого проекта. Для каждой целевой ветки проекта используется отдельная цепочка слияния.

Принцип работы цепочки слияния

Запросы на слияние располагаются в очереди в порядке добавления. Добавленные первыми запросы находятся в начале очереди.

Обработка запросов в очереди выполняется следующим образом:

1. Запрос на слияние, для которого включено автослияние (auto-merge), добавляется в конец очереди.
2. Deckhouse Code создаёт для запроса временную Git-ссылку (Git-ref) и запускает для него пайплайн. Git-ref содержит состояние целевой ветки, изменения из предыдущих запросов в очереди и изменения из текущего запроса.
3. Если пайплайн первого запроса в очереди завершается успешно, запрос сливается в целевую ветку и покидает очередь.
4. Первым в очереди становится следующий запрос. Если после слияния предыдущего запроса состояние, для которого запускался пайплайн, стало неактуальным, Deckhouse Code пересоздаёт пайплайн для нового состояния. Пайплайны последующих запросов также пересоздаются.

Цепочка не хранится как отдельный объект. Она формируется из запросов на слияние, добавленных в очередь для одной целевой ветки проекта. Когда запрос сливается или удаляется из очереди, позиции остальных запросов пересчитываются.

Предварительные требования

Перед настройкой цепочек слияния убедитесь, что выполнены следующие требования:

- в проекте включён механизм CI/CD;
- конфигурационный файл CI/CD настроен на создание пайплайнов для запросов на слияние;
- в проекте включены пайплайны для результатов слияния;
- для изменения настроек проекта у пользователя есть роль **Мейнтейнер** или **Владелец**.

Включение цепочек слияния

Чтобы включить цепочки слияния:

1. Откройте проект.
2. Перейдите в «Настройки» → «Запросы на слияние».
3. В разделе «Параметры слияния» установите флажок «Включить пайплайны с результатами слияния».
4. Установите флажок «Включить цепочки слияния».
5. Нажмите «Сохранить изменения».

После включения цепочек слияния становятся доступны дополнительные настройки:

Настройка	Описание
Сливать немедленно без перезапуска цепочки слияния	Добавляет варианты немедленного слияния, описанные в разделе Немедленное слияние. Настройка недоступна, если проект требует слияния в режиме fast-forward или semi-linear, а также пока установлено требование слияния через цепочку
Требовать слияния запросов только через цепочку слияния	Отклоняет все прямые слияния в проекте. Подробнее — в разделе Обязательное слияние через цепочку
Максимальное количество параллельных пайплайнов на цепочку слияния	Ограничивает количество одновременно выполняемых пайплайнов цепочки. Подробнее — в разделе Ограничение параллельных пайплайнов

Снятие флажка «Включить цепочки слияния» не очищает существующую очередь сразу. Очередь освобождается по мере того, как цикл обновления доходит до каждого запроса, поэтому некоторое время после сохранения настройки они могут по-прежнему отображаться в очереди.

Работа с очередью

Запросы на слияние можно добавлять в очередь, просматривать, удалять из неё или сливать немедленно.

Добавление запроса на слияние в цепочку

Добавить запрос на слияние в цепочку можно через веб-интерфейс или API.

Добавление через веб-интерфейс

Чтобы добавить запрос на слияние в цепочку, нажмите «Установить автоматическое слияние» на странице запроса. Если в проекте включены цепочки слияния, запрос будет добавлен в цепочку вместо прямого слияния.

В зависимости от состояния запроса доступен один из следующих вариантов:

Вариант	Описание
Добавить в цепочку слияния	Запрос сразу добавляется в очередь. Вариант доступен, если запрос готов к слиянию и пайплайн для текущего `HEAD` SHA завершён
Добавить в цепочку слияния после прохождения всех проверок слияния	Запрос добавляется в очередь после успешного прохождения проверок

Вариант, который будет использован, отображается рядом с кнопкой слияния. Если запрос уже проверялся пайплайном цепочки слияния, отображается текст «Повторно добавить в цепочку слияния».

Если последний пайплайн запроса завершился с ошибкой, перед добавлением запроса в очередь Deckhouse Code запрашивает подтверждение.

Добавление через API

Чтобы добавить запрос на слияние в цепочку через API, выполните запрос:

```
curl --request POST --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/merge_requests/
<MERGE_REQUEST_IID>"
```

Где:

- `<TOKEN>` — токен для аутентификации в Deckhouse Code;
- `<HOST>` — адрес экземпляра Deckhouse Code;
- `<PROJECT_ID>` — идентификатор проекта;
- `<MERGE_REQUEST_IID>` — внутренний идентификатор запроса на слияние в проекте.

В запросе можно указать следующие параметры:

Параметр	Описание
`sha`	SHA, который вы ожидаете увидеть в исходной ветке. Если текущий SHA не совпадает с указанным, запрос отклоняется
`squash`	Объединяет коммиты исходной ветки в один при слиянии запроса. При слиянии через цепочку также учитывается настройка squash самого запроса на слияние
`auto_merge`	Добавляет запрос в очередь только после успешного прохождения проверок слияния

Возможные коды ответа:

Код	Описание
`201`	Запрос на слияние добавлен в очередь
`202`	Запрос ожидает прохождения проверок слияния и ещё не добавлен в очередь
`400`	Цепочки слияния отключены в проекте или текущее состояние запроса не позволяет добавить его в очередь
`401`	Запрос выполнен без аутентификации
`403`	Роль пользователя не позволяет просматривать цепочку или сливать этот запрос
`404`	Проект или запрос на слияние не найден
`409`	Для запроса на слияние уже включено автослияние

Просмотр цепочки слияния

Получить информацию о цепочке можно через веб-интерфейс или API.

Просмотр через веб-интерфейс

Открыть страницу цепочек слияния проекта можно одним из следующих способов:

- на странице запроса на слияние, добавленного в очередь, перейдите по ссылке «Просмотреть цепочку слияния»;
- на странице пайплайна цепочки слияния перейдите по ссылке «Просмотр деталей цепочки слияния» в заголовке страницы;
- откройте страницу напрямую по адресу `https://<HOST>/<NAMESPACE>/<PROJECT>/-/merge_trains`.

На странице цепочек слияния отображаются:

- фильтр для выбора целевой ветки. Изначально выбрана ветка проекта по умолчанию.
- вкладка «Активные» с запросами на слияние, которые находятся в очереди, и вкладка «Слитые» с уже слитыми запросами. На каждой вкладке указано количество запросов;
- информация о каждом запросе на слияние: статус пайплайна, заголовок, время добавления в очередь или слияния, а также пользователь, который добавил или слил запрос.

Пока запрос на слияние находится в очереди, его позиция отображается на странице запроса:

- для первого запроса — «Запущена новая цепочка слияния, и этот запрос на слияние стоит первым в очереди»;
- в остальных случаях отображается позиция в очереди, например, «Этот запрос на слияние занимает позицию 2 из 5 в очереди».

Просмотр через API

Чтобы получить информацию о цепочках слияния через API, выполните один из следующих запросов:

```
# Все цепочки проекта.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains"

# Цепочка одной целевой ветки.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/<TARGET_BRANCH>"

# Статус одного запроса на слияние в очереди.
curl --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_trains/merge_requests/
<MERGE_REQUEST_IID>"
```

Где:

- `<TOKEN>` — токен для аутентификации в Deckhouse Code;
- `<HOST>` — адрес экземпляра Deckhouse Code;
- `<PROJECT_ID>` — идентификатор проекта;
- `<TARGET_BRANCH>` — имя целевой ветки;
- `<MERGE_REQUEST_IID>` — внутренний идентификатор запроса на слияние в проекте.

Первые два эндпоинта принимают следующие параметры:

Параметр	Описание
`scope`	Ограничивает ответ запросами, которые ещё стоят в очереди (`active`), или уже покинули её (`complete`)
`sort`	Задаёт порядок запросов по их позиции в очереди: `asc` — от самых старых к новым, `desc` — от самых новых к старым. Значение по умолчанию — `desc`

Оба эндпоинта поддерживают постраничный вывод с помощью параметров `page` и `per_page`. Первый эндпоинт возвращает запросы на слияние для всех целевых веток. Чтобы получить очередь только для определённой ветки, используйте второй эндпоинт и укажите её имя.

Получить информацию о цепочках слияния и очередях проекта также можно через GraphQL API.

Удаление запроса на слияние из цепочки

Чтобы удалить запрос на слияние из цепочки, используйте один из следующих способов:

- отмените автослияние на странице запроса;
- на странице цепочек слияния нажмите значок удаления в строке запроса, затем нажмите «Удалить из цепочки слияния».

При удалении запроса из очереди пайплайны для следующих за ним запросов пересоздаются, поскольку состояние цепочки изменилось.

Запрос нельзя удалить из цепочки, если его слияние уже началось. В этом случае Deckhouse Code отклоняет удаление и завершает слияние запроса.

Deckhouse Code также автоматически удаляет запрос из цепочки, если он больше не может быть слит. Подробнее — в разделе Запрос на слияние удалён из цепочки.

Немедленное слияние

Запрос на слияние можно слить немедленно, не дожидаясь его очереди в цепочке. Немедленное слияние доступно через веб-интерфейс и API.

Слияние через веб-интерфейс

Если в проекте включена настройка «Сливать немедленно без перезапуска цепочки слияния», на странице запроса в очереди рядом с кнопкой слияния доступны два дополнительных варианта:

Вариант	Описание
Слить сейчас и перезапустить цепочку	Запрос сливается немедленно, а пайплайны следующих за ним запросов пересоздаются с учётом слитых изменений
Слить сейчас без перезапуска цепочки	Запрос сливается немедленно, а существующие пайплайны продолжают выполняться без перезапуска. В результате запросы, уже находящиеся в очереди, не проверяются вместе со слитыми изменениями

⚠ Используйте вариант «Слить сейчас без перезапуска цепочки» только если слитые изменения не могут повлиять на запросы, находящиеся в очереди.

Оба варианта требуют подтверждения и позволяют выполнить слияние вне установленного порядка очереди.

Варианты немедленного слияния недоступны, если в проекте включено обязательное слияние через цепочку. Настройка «Сливать немедленно без перезапуска цепочки слияния» также недоступна, если для проекта требуется слияние в режиме fast-forward или semi-linear.

Результат немедленного слияния зависит от выбранного варианта:

- при слиянии без перезапуска цепочки запрос отображается на вкладке «Слитые» со статусом `skip_merged`;
- при слиянии с перезапуском цепочки запрос сливается напрямую и не отображается на вкладке «Слитые». После слияния он удаляется из очереди, а удаление записывается в историю активности запроса.

Слияние через API

Для немедленного слияния через API используйте эндпоинт слияния с параметром `skip_merge_train`:

```
curl --request PUT --header "PRIVATE-TOKEN: <TOKEN>" \
  "https://<HOST>/api/v4/projects/<PROJECT_ID>/merge_requests/<MERGE_REQUEST_IID>/
merge?skip_merge_train=true"
```

Где:

- `<TOKEN>` — токен для аутентификации в Deckhouse Code;
- `<HOST>` — адрес экземпляра Deckhouse Code;
- `<PROJECT_ID>` — идентификатор проекта;
- `<MERGE_REQUEST_IID>` — внутренний идентификатор запроса на слияние в проекте.

Параметр `skip_merge_train` принимает следующие значения:

Значение	Описание
`true`	Запрос сливается немедленно без перезапуска цепочки
`false`	Значение по умолчанию. Запрос сливается немедленно, а пайплайны следующих за ним запросов пересоздаются

Параметр действует только при включённой настройке «Сливать немедленно без перезапуска цепочки слияния». В остальных случаях параметр игнорируется. Если в проекте включено обязательное слияние через цепочку, запрос на слияние отклоняется.

Настройка цепочки слияния

В настройках проекта можно сделать слияние через цепочку обязательным и ограничить количество одновременно выполняемых пайплайнов.

Обязательное слияние через цепочку

Чтобы запретить прямое слияние запросов в проекте, установите флажок «Требовать слияния запросов только через цепочку слияния».

После включения этой настройки:

- варианты немедленного слияния становятся недоступны;
- запрос на слияние через API отклоняется, если для него не включено автослияние и он не добавлен в цепочку;

- слияния, которые выполняет сама цепочка, продолжают работать без изменений.

При включении обязательного слияния настройка «Сливать немедленно без перезапуска цепочки слияния» автоматически отключается и становится недоступна. После сохранения настроек с включённым обязательным слиянием эта настройка остаётся отключённой, в том числе после отключения обязательного слияния. Чтобы снова использовать её, включите настройку заново.

Обязательное слияние действует только при включённых цепочках слияния. Если цепочки отключены, запросы можно сливать напрямую.

Ограничение распространяется на всех пользователей, включая владельцев проекта и администраторов.

Ограничение параллельных пайплайнов

Для нескольких запросов в цепочке пайплайны могут выполняться одновременно. Количество одновременно выполняемых пайплайнов определяется двумя ограничениями:

- ограничением экземпляра Deckhouse Code, которое задаёт администратор. По умолчанию — `20`;
- настройкой проекта «Максимальное количество параллельных пайплайнов на цепочку слияния».

Используется меньшее из этих двух значений. Например, если для экземпляра установлено ограничение `20`, а для проекта — `10`, одновременно могут выполняться не более 10 пайплайнов.

Чтобы использовать ограничение экземпляра, оставьте настройку проекта пустой. Для проекта можно указать значение от `1` до `500`, но оно не может увеличить ограничение, установленное для экземпляра.

Пайплайны цепочки слияния

Для каждого запроса в очереди Deckhouse Code запускает пайплайн для Git ref. Такой пайплайн выполняется не для исходной ветки запроса. В списке пайплайнов он помечается как `merge train`, что позволяет отличить его от обычного пайплайна для результатов слияния.

Завершённый пайплайн цепочки слияния нельзя перезапустить, поскольку состояние репозитория, для которого он выполнялся, могло измениться. Чтобы запустить новый пайплайн, повторно добавьте запрос на слияние в цепочку.

Пайплайн пересоздаётся в следующих случаях:

- запрос, находящийся перед ним в очереди, слит, удалён из очереди или получил новый пайплайн;
- пайплайн предыдущего запроса в очереди завершился с ошибкой;
- изменения отправлены напрямую в целевую ветку. В этом случае пересоздаются пайплайны для всех запросов в очереди, начиная с первого.

Запуск заданий только в пайплайнах цепочки слияния

В пайплайне цепочки слияния переменная `CI_MERGE_REQUEST_EVENT_TYPE` имеет значение `merge_train`. С её помощью можно запускать задание только в пайплайнах цепочки слияния:

```
integration-tests:
  script: ./run-integration-tests.sh
  rules:
    - if: $CI_MERGE_REQUEST_EVENT_TYPE == "merge_train"
```

Чтобы, наоборот, пропускать задание в пайплайнах цепочки, используйте обратное условие:

```
quick-lint:
  script: ./lint.sh
  rules:
    - if: $CI_MERGE_REQUEST_EVENT_TYPE != "merge_train"
```

Статусы запросов в очереди

Для каждого запроса на слияние в цепочке определяется статус, который можно получить через REST API или GraphQL API. В веб-интерфейсе статусы напрямую не отображаются: запросы распределяются между вкладками «Активные» и «Слитые».

Статус	Описание
`idle`	Запрос находится в очереди, пайплайн ещё не запущен
`fresh`	Пайплайн соответствует текущему результату слияния
`stale`	Состояние цепочки изменилось, пайплайн пересоздаётся
`merging`	Выполняется слияние запроса
`merged`	Запрос слит через цепочку и удалён из очереди
`skip_merged`	Запрос слит немедленно без перезапуска цепочки и удалён из очереди

Запросы на слияние со статусами `idle`, `fresh` и `stale` отображаются на вкладке «Активные», их можно удалить из очереди. Запросы со статусами `merged` и `skip_merged` отображаются на вкладке «Слитые». Запрос со статусом `merging` не отображается ни на одной из вкладок до завершения слияния.

Активность запроса на слияние

Deckhouse Code записывает, что происходит с запросом на слияние в цепочке, в виде системных заметок в разделе «Активность» запроса. Фиксируются следующие события:

- запрос запустил новую цепочку или был добавлен в существующую цепочку на определённую позицию в очереди;
- запрос удалён из цепочки пользователем;
- запрос удалён из цепочки системой с указанием причины;
- автоматическое добавление в цепочку после прохождения проверок включено, отменено или прервано с указанием причины.

Если запрос на слияние удалён из цепочки системой, его участники также получают соответствующую задачу в списке задач.

Доступ к цепочкам слияния

Доступные действия с цепочками слияния зависят от роли пользователя и настроек проекта:

Действие	Требования
Просмотр страницы цепочек слияния и просмотр информации через API	В проекте включены цепочки слияния, и у пользователя есть доступ на чтение запросов на слияние и пайплайнов. В стандартных ролях это доступно пользователям с ролью **Наблюдатель** или выше
Получение статуса цепочки для отдельного запроса на слияние	Те же требования, что и для просмотра цепочки
Добавление запроса на слияние в цепочку	У пользователя есть право на слияние этого запроса. В стандартных ролях это доступно пользователям с ролью **Разработчик** или выше
Удаление запроса на слияние из очереди на странице цепочек	У пользователя есть право на отмену пайплайнов, изменение запроса на слияние и слияние в целевую ветку. В стандартных ролях это доступно пользователям с ролью **Разработчик** или выше
Изменение настроек цепочек слияния в проекте	Роль **Мейнтейнер** или **Владелец**

Для просмотра цепочек слияния через веб-интерфейс или API требуется аутентификация. Анонимным пользователям цепочки недоступны даже в публичных проектах.

События аудита

Deckhouse Code записывает следующие события аудита при изменении настроек цепочек слияния в проекте:

Событие аудита	Описание
`project_cicd_merge_trains_enabled_updated`	Изменена настройка включения цепочек слияния в проекте
`project_cicd_merge_train_enforced_updated`	Изменена настройка обязательного слияния через цепочку

Устранение проблем

Ниже описаны распространённые проблемы при работе с цепочками слияния и способы их устранения.

Запрос на слияние удалён из цепочки

Если во время выполнения пайплайна запрос больше не может быть слит, Deckhouse Code удаляет его из очереди, чтобы не блокировать остальные запросы. Причина удаления указывается в системной заметке в разделе «Активность».

Наиболее частые причины удаления:

- цепочки слияния отключены в проекте;
- в исходную ветку запроса на слияние добавлены новые коммиты;
- запрос на слияние закрыт;
- запрос на слияние помечен как черновик;
- изменена целевая ветка запроса на слияние;
- запрос на слияние стал непригодным для слияния, например, из-за конфликта;
- для запроса на слияние отменено автослияние;
- пайплайн цепочки слияния для этого запроса завершился с ошибкой;
- удалена учётная запись пользователя, добавившего запрос в очередь.

Если запрос ожидает прохождения проверок перед добавлением в цепочку, процесс также прерывается, если запрос становится черновиком или его пайплайн завершается с ошибкой.

Дискуссии и апрувы проверяются при добавлении запроса в очередь. Новая дискуссия, открытая после этого, не блокирует слияние, даже если в проекте требуется закрытие всех дискуссий. Это позволяет сохранить уже проверенное пайплайном состояние и не пересоздавать пайплайны последующих запросов.

Если обязательный апрув отозван, слияние блокируется. Запрос сохраняет позицию в очереди, но при достижении её начала удаляется из цепочки.

Устраните причину, указанную в системной заметке, и повторно добавьте запрос в цепочку.

Кнопка слияния не предлагает варианты цепочки слияния

Проверьте следующее:

- в настройках проекта включены пайплайны с результатами слияния и цепочки слияния;
- конфигурационный файл CI/CD создаёт пайплайны для запросов на слияние;
- у запроса на слияние есть пайплайн для текущего `HEAD` SHA исходной ветки, и этот пайплайн завершён;
- роль пользователя позволяет слить этот запрос.

Слияние отклонено в проекте, требующем цепочку слияния

Если включена настройка «Требовать слияния запросов только через цепочку слияния», прямое слияние отклоняется, в том числе через API. Вместо этого добавьте запрос на слияние в цепочку.

Ограничение действует для всех пользователей, включая владельца проекта. Чтобы снова разрешить прямые слияния, отключите эту настройку.

Пайплайн цепочки слияния нельзя перезапустить

Завершённый пайплайн цепочки слияния нельзя перезапустить, поскольку состояние, для которого он выполнялся, могло измениться. Чтобы создать пайплайн для текущего состояния, повторно добавьте запрос на слияние в цепочку.

Слияние выглядит зависшим

Если процесс слияния прерывается, Deckhouse Code автоматически возвращает запрос в очередь и продолжает работу цепочки. Дополнительные действия не требуются.

Цепочки слияния отключены, но очередь всё ещё отображается

После отключения цепочек запросы удаляются из существующей очереди не сразу. Некоторое время они могут по-прежнему отображаться на странице, пока Deckhouse Code постепенно очищает очередь.

Страница цепочек слияния пуста

Проверьте следующее:

- в фильтре выбрана целевая ветка, для которой нужно просмотреть очередь;
- для запросов на слияние включено автослияние;
- в проекте включены цепочки слияния. Если они отключены, страница недоступна, даже если очередь осталась с прежнего состояния.

Действие удаления недоступно

Удалить запрос из цепочки можно только на вкладке «Активные» и при наличии прав на отмену пайплайнов, изменение запроса на слияние и слияние в целевую ветку.

Запрос нельзя удалить из цепочки, если его слияние уже началось.

Аналитика ревью кода

Отчёт показывает, сколько времени открытые запросы на слияние проекта ждут ревью.

Что попадает в отчёт

В отчёт попадают открытые запросы на слияние проекта, у которых есть хотя бы один комментарий от ревьювера запроса (любого пользователя, кроме автора).

Слитые, закрытые и запросы на слияние без комментариев в отчёте отсутствуют.

Комментарии ботов и системные заметки (смена метки, назначение проверяющего) не считаются началом ревью, поэтому запрос на слияние, у которого есть только такие комментарии, в отчёт не попадает.

Просмотр отчёта

Откройте проект и выберите в боковом меню «Аналитика» → «Аналитика ревью кода».

Подсчёт времени, затраченного на ревью

Время ревью отсчитывается от первого комментария ревьювера до текущего момента.

Значение приблизительное и отображается в минутах, часах или днях. Время начала ревью фиксируется в момент сохранения комментария. Запрос, который ждал ревью восемь месяцев, имел системные комментарии и получил комментарий ревьювера час назад, покажет время ревью «примерно час».

Фильтры

Панель фильтров над таблицей позволяет настроить фильтрацию отчёта по этапу и по метке. Фильтр «Этап» принимает одно значение, фильтр «Метка» принимает несколько значений.

Перечень запросов на слияние

Таблица «Запросы на слияние на ревью» содержит запросы, которые находятся на ревью, начиная с самого раннего по дате создания. Счётчик рядом с заголовком таблицы показывает, сколько запросов на слияние остаётся на ревью с учётом фильтров.

Одна страница таблицы содержит не более 20 запросов на слиянии, при превышении общего количества под таблицей отображаются элементы постраничной навигации. Смена фильтра возвращает таблицу на первую страницу.

Таблица содержит следующие колонки:

Колонка	Содержимое
«Запрос на слияние»	Заголовок со ссылкой на запрос на слияние, его номер, количество меток и комментариев, а также количество одобрений, если они есть
«Время ревью»	Время, прошедшее с первого комментария ревьювера
«Автор»	Автор запроса на слияние
«Апруверы»	Пользователи, одобрившие запрос на слияние
«Комментарии»	Количество комментариев
«Коммиты»	Количество коммитов
«Изменения строк»	Количество добавленных и удалённых строк

Если с учётом фильтров на ревью нет ни одного запроса на слияние, вместо таблицы отображается сообщение «Нет запросов на слияние на ревью».

Доступ к отчёту

Отчёт доступен пользователям с ролью «Наблюдатель» или выше. В публичном проекте он доступен всем, кто видит проект, включая анонимных пользователей.

Если в настройках проекта отключена функция «Аналитика», отчёт и пункт меню недоступны.

Аналитика запросов на слияние

Отчёт показывает, сколько запросов на слияние было слито за выбранный период и сколько времени занимало слияние. Информация представлена в карточке «Среднее время до слияния», графике «Пропускная способность» и таблице «Запросы на слияние».

Что попадает в отчёт

В отчёт попадают слитые запросы на слияние проекта в пределах выбранного диапазона дат. Открытые запросы и запросы, закрытые без слияния, в отчёте отсутствуют.

Запрос, созданный до начала диапазона и слитый внутри него, учитывается полностью: в среднее время до слияния попадает весь интервал от создания до слияния.

Просмотр отчёта

Откройте проект и выберите в боковом меню «Аналитика» → «Аналитика запросов на слияние».

Диапазоны дат

Для просмотра можно выбрать один из диапазонов дат, начиная с текущего момента:

- «Последние 7 дней»;
- «Последние 30 дней»;
- «Последние 90 дней»;
- «Последние 180 дней»;
- «Последние 365 дней» (вариант по умолчанию);

В варианте «Произвольный диапазон» даты задаются в полях «От» и «До». Максимальная ширина диапазона ограничена 365 днями.

Выбранный диапазон отображается рядом с заголовками «Среднее время до слияния», «Пропускная способность» и «Запросы на слияние».

График пропускной способности

График «Пропускная способность» показывает количество слитых запросов на слияние по месяцам. В первом и последнем месяце диапазона учитываются только дни, которые входят в диапазон: например, у диапазона, начинающегося 15 марта, первая точка графика охватывает 15–31 марта.

Если в диапазоне с учётом фильтров нет слитых запросов на слияние, вместо графика отображается сообщение «В этом диапазоне дат не слито ни одного запроса на слияние».

Среднее время до слияния

Карточка «Среднее время до слияния» показывает среднее время от создания запроса на слияние до его слияния за весь диапазон в днях.

В среднее значение попадают запросы, у которых момент слияния позже момента создания. Запрос, у которого момент слияния совпадает с моментом создания или предшествует ему, например импортированный с уже заполненным `merged_at`, учитывается на графике и в таблице. В расчёт среднего он не входит. Если в диапазоне нет ни одного запроса с моментом слияния позже создания, в карточке отображается прочерк (`-`).

Фильтры

Панель фильтров позволяет настроить фильтрацию для всех элементов отчёта одновременно. Доступны следующие фильтры:

- «Исходная ветка»;
- «Целевая ветка»;
- «Этап»;
- «Метка»;
- «Автор»;
- «Ответственный».

Фильтр «Метка» принимает несколько значений, и в отчёте остаются запросы на слияние, у которых есть все выбранные метки.

Таблица запросов на слияние

Таблица «Запросы на слияние» содержит слитые запросы на слияние, входящие в диапазон дат, начиная с последнего слитого. Одна страница таблицы содержит не более 20 запросов на слияние, при превышении общего количества под таблицей отображаются элементы постраничной навигации. Смена фильтра возвращает таблицу на первую страницу.

Таблица содержит следующие колонки:

Колонка	Содержимое
«Запрос на слияние»	Заголовок со ссылкой на запрос на слияние, его номер, статус пайплайна, количество меток и комментариев, а также количество одобрений, если они есть
«Дата слияния»	Дата, когда запрос на слияние был слит
«Время до слияния»	Приблизительный интервал между созданием запроса на слияние и его слиянием, в минутах, часах или днях
«Этап»	Этап запроса на слияние со ссылкой на него
«Коммиты»	Количество коммитов в запросе на слияние
«Пайплайны»	Количество пайплайнов запроса на слияние
«Изменения строк»	Количество добавленных и удалённых строк
«Ответственные»	Аватары ответственных

Если в диапазоне с учётом фильтров нет слитых запросов на слияние, вместо таблицы отображается сообщение «В этом диапазоне дат не слито ни одного запроса на слияние» и предложение расширить диапазон или очистить фильтры.

Доступ к отчёту

Отчёт доступен пользователям с ролью «Наблюдатель» или выше. Пользователям с ролью «Гость» и анонимным пользователям отчёт недоступен даже в публичном проекте.

Если в настройках проекта отключена функция «Аналитика», отчёт и пункт меню недоступны.

CI/CD

Непрерывная сборка и поставка ПО

Deckhouse Code использует файл `.gitlab-ci.yml` для автоматизации процессов тестирования, сборки и деплоя с помощью CI/CD-пайплайнов. Это позволяет реализовать следующий функционал:

- Автоматическую проверку кода при каждом коммите.
- Гибкую настройку этапов выполнения (stages) и задач (jobs).
- Параллельное выполнение задач для ускорения сборки и тестирования.
- Добавление произвольных шагов тестирования с учётом специфики проекта.
- Запуск пайплайнов по различным событиям: коммиты, merge requests, создание тегов и др.

Интеграция с внешним Vault

Эта функция позволяет настроить интеграцию с Vault-сервером и использовать секреты в CI-пайплайнах. Перед началом работы необходимо настроить Vault-сервер и подготовить соответствующие роли и политики.

Настройка Vault

1. Включите аутентификацию через JWTT:

   ```
   vault auth enable jwt

   vault write auth/jwt/config \
     oidc_discovery_url="https://code.example.com" \
     bound_issuer="https://code.example.com" \
     default_role="code-role"
   ```

2. Создайте роль:

   ```
   vault write auth/jwt/role/code-role - <<EOF
   {
     "role_type": "jwt",
     "user_claim": "sub",
     "bound_audiences": ["vault"],
     "bound_claims": {
       "project_id": "23"
     },
     "policies": ["code-policy"],
     "ttl": "1h"
   }
   EOF
   ```

> Всегда используйте `bound_claims`, чтобы ограничить доступ к роли. Без этого любой JWT, выданный платформой, сможет получить доступ с этой ролью.

3. Настройте политики:

```
vault policy write code-policy - <<EOF
path "kv/data/code/vault-demo" {
  capabilities = ["read"]
}
EOF
```

Конфигурация CI

Переменные окружения

Для корректной работы с Vault в пайплайне CI/CD необходимо задать следующие переменные окружения:

- `VAULT_SERVER_URL` — **обязательно**. URL-адрес Vault-сервера (например, `https://vault.example.com`).
- `VAULT_AUTH_ROLE` — *опционально*. Название роли в Vault. Если не указано, будет использована роль по умолчанию, заданная в конфигурации метода аутентификации.
- `VAULT_AUTH_PATH` — *опционально*. Путь до метода аутентификации в Vault. По умолчанию используется `jwt`.
- `VAULT_NAMESPACE` — *опционально*. Namespace Vault, если используется многоуровневая иерархия.

Альтернативные имена переменных

У каждой переменной подключения к Vault есть альтернативное имя с префиксом `STRONGHOLD_`. Проект, который хранит секреты для CI/CD в Deckhouse Stronghold, может настроить подключение через эти имена:

Переменная	Альтернативное имя
`VAULT_SERVER_URL`	`STRONGHOLD_SERVER_URL`
`VAULT_AUTH_ROLE`	`STRONGHOLD_AUTH_ROLE`
`VAULT_AUTH_PATH`	`STRONGHOLD_AUTH_PATH`
`VAULT_NAMESPACE`	`STRONGHOLD_NAMESPACE`

Подключение настраивают только имена из таблицы. Например, переменная `STRONGHOLD_TOKEN` не читается.

Значения определяются по следующим правилам:

- значение каждой настройки определяется независимо от остальных, поэтому адрес сервера можно задать в `STRONGHOLD_SERVER_URL`, а роль — в `VAULT_AUTH_ROLE`;
- если для одной настройки заданы оба имени, используется значение переменной с префиксом `VAULT_`;
- пустое значение переменной с префиксом `STRONGHOLD_` равнозначно незаданному, поэтому при пустом `STRONGHOLD_AUTH_PATH` сохраняется путь по умолчанию `jwt`.

Альтернативные имена влияют только на подключение к Vault. Скрипт задания CI, который читает `$VAULT_SERVER_URL`, получает значение только тогда, когда переменная с таким именем задана.

Использование секретов в CI

Для получения секретов из Vault можно использовать следующий шаблон job'а:

```
stages:
  - test
vault-login:
  stage: test
  image: ruby:3.2
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DATABASE_PASSWORD:
      vault: code/vault-demo/DATABASE_PASSWORD@kv
      token: $VAULT_ID_TOKEN
  script: echo $DATABASE_PASSWORD
```

Параметры секрета

Пример:

```
DATABASE_PASSWORD:
  vault: code/vault-demo/DATABASE_PASSWORD@kv
  token: $VAULT_ID_TOKEN
  file: false
```

Описание параметров:

1. `vault` (обязательно) — путь к секрету в формате строки `path/to/secret/KEY@ENGINE`, где:
   - `code/vault-demo/` — путь до секрета в Vault;
   - `DATABASE_PASSWORD` — имя поля внутри секрета;
   - `kv` — точка монтирования Secret Engine (по умолчанию — `secret`).

По умолчанию используется `engine kv-v2`. Если необходимо использовать другой engine, можно указать объект вместо строки:

```
DATABASE_PASSWORD:
  vault:
    path: code/vault-demo
    field: DATABASE_PASSWORD
    engine:
      name: 'kv-v1'
      path: 'kv1'
  token: $VAULT_ID_TOKEN
  file: false
```

1. `token` (обязательно) — JWT-токен из секции `id_tokens`, используемый для аутентификации в Vault.
2. `file` (опционально, по умолчанию `true`) — определяет способ предоставления секрета:
    - `true` — секрет сохраняется во временный файл;
    - `false` — секрет передаётся как строка в переменную окружения.

Поля, включённые в JWT

Следующие поля автоматически включаются в JWT-токен и могут использоваться Vault для проверки прав доступа:

Поле	Условие появления	Описание
`jti`	всегда	Уникальный идентификатор токена
`iss`	всегда	Издатель токена (обычно URL Deckhouse Code)
`iat`	всегда	Время выпуска токена (Issued At)
`nbf`	всегда	Время, до которого токен считается недействительным
`exp`	всегда	Время истечения срока действия токена
`sub`	всегда	Subject токена (обычно ID задания CI)
`namespace_id`	всегда	ID пространства (группы или пользователя)
`namespace_path`	всегда	Путь до пространства (например, `groups/dev`)
`project_id`	всегда	ID проекта
`project_path`	всегда	Путь до проекта
`user_id`	всегда	ID пользователя
`user_login`	всегда	Логин пользователя
`user_email`	всегда	Email пользователя
`pipeline_id`	всегда	ID CI-пайплайна
`pipeline_source`	всегда	Источник запуска пайплайна (push, schedule, MR и т.д.)
`job_id`	всегда	ID задания CI
`ref`	всегда	Ссылка Git (Git reference)
`ref_type`	всегда	Тип ссылки Git (Git reference) (`branch` или `tag`)
`ref_path`	всегда	

Поле	Условие появления	Описание
		Полный путь ссылки Git (Git reference) (например, `refs/heads/main`)
`ref_protected`	всегда	Признак того, что объект по ссылке Git защищён
`environment`	при наличии	Название окружения (если используется)
`groups_direct`	при наличии (<200 групп)	Пути до групп, в которых состоит пользователь
`environment_protected`	при наличии	Является ли окружение защищённым
`deployment_tier`	при наличии	Тип окружения (`production`, `staging` и т.п.)
`environment_action`	при наличии	Тип действия над окружением (например, `deploy`)

Быстрый старт

В этом разделе приведён пример минимально необходимой настройки интеграции HashiCorp Vault с Deckhouse Code и проверки того, что CI job может получать секреты из Vault.

> ⚠ Данный пример предназначен только для ознакомления. Он не отражает лучшие практики безопасности и использует упрощённую конфигурацию, позволяющую быстро проверить работоспособность интеграции.

Шаг 1. Установка переменных окружения

Установите переменные окружения для Vault и Deckhouse Code.
Некоторые параметры можно оставить как есть, но `VAULT_ADDR`, `VAULT_TOKEN`, `CODE_URL` и `PROJECT_PATH` должны быть заданы вручную.

```
export VAULT_ADDR="https://vault.example.com"
export VAULT_TOKEN="<TOKEN>"

# URL-адрес Deckhouse Code.
export CODE_URL="https://code.example.com"

# Имя роли и политики Vault.
export VAULT_ROLE="code-role"
export VAULT_POLICY="code-policy"

# Путь и данные секрета.
export VAULT_SECRET_PATH="code/vault-demo"
export VAULT_SECRET_FIELD="DATABASE_PASSWORD"
export VAULT_SECRET_VALUE="super-secret-password"

# Значение claim project_path, которое будет проверять Vault.
export PROJECT_PATH="root/my-pr"
```

Шаг 2. Включение метода аутентификации JWT

Включите в Vault метод аутентификации JWT. Без этого Vault не сможет принимать ID-токены, которые Deckhouse Code передаёт в CI jobs.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/sys/auth/jwt" \
  -d '{"type":"jwt"}'
```

Шаг 3. Настройка JWT и OIDC

Следующий запрос:

- задаёт адрес OIDC discovery (`$CODE_URL`);
- указывает ожидаемого issuer;
- определяет *роль по умолчанию*, которую Vault выдаёт при аутентификации.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/auth/jwt/config" \
  --data @- <<EOF
{
  "oidc_discovery_url": "$CODE_URL",
  "bound_issuer": "$CODE_URL",
  "default_role": "$VAULT_ROLE"
}
EOF
```

Шаг 4. Монтирование Secret Engine KV v2

KV v2 — это наиболее распространённый движок секретов Vault. Следующий запрос включает его по пути `/kv`:

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/sys/mounts/kv" \
  -d '{"type": "kv-v2"}'
```

Шаг 5. Создание тестового секрета

Создайте секрет, который затем должна будет прочитать CI job. Секрет размещается по пути `code/vault-demo` и содержит одно поле.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/kv/data/$VAULT_SECRET_PATH" \
  --data @- <<EOF
{
  "data": {
    "$VAULT_SECRET_FIELD": "$VAULT_SECRET_VALUE"
  }
}
EOF
```

Шаг 6. Создание ACL-политики

Политика определяет, к каким путям в Vault разрешён доступ. В следующем примере политика предоставляет только право на чтение указанного секрета.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X PUT "$VAULT_ADDR/v1/sys/policies/acl/$VAULT_POLICY" \
  --data @- <<EOF
{
  "policy": "path \"kv/data/$VAULT_SECRET_PATH\" { capabilities = [\"read\"] }"
}
EOF
```

Шаг 7. Создание роли Vault

Роль определяет:

- тип аутентификации;
- обязательные claims в токене (`project_path`);
- политики, которые получит аутентифицированный субъект;

- допустимые аудитории (`aud`);
- TTL токена.

Deckhouse Code будет выпускать ID-токен с `aud=vault` , а Vault проверит, что значение `project_path` соответствует указанному.

```
curl \
  -H "X-Vault-Token: $VAULT_TOKEN" \
  -X POST "$VAULT_ADDR/v1/auth/jwt/role/$VAULT_ROLE" \
  --data @- <<EOF
{
  "role_type":   "jwt",
  "user_claim":  "sub",
  "bound_audiences": ["vault"],

  "bound_claims": {
    "project_path": "$PROJECT_PATH"
  },

  "policies": ["$VAULT_POLICY"],
  "ttl": "1h"
}
EOF
```

На этом настройка Vault завершена.

Тестирование интеграции в Deckhouse Code

1. Откройте проект, указанный в `PROJECT_PATH` . CI-токен проекта должен совпадать с claim `project_path` . В противном случае Vault отклонит запрос на доступ к секретам.
2. В настройках CI/CD проекта добавьте переменную `VAULT_SERVER_URL` со значением `$VAULT_ADDR` , использованным ранее. Эта переменная сообщает Deckhouse Code, куда следует направлять запросы Vault API.
3. Создайте файл `.gitlab-ci.yml` . Файл запускает тестовую CI job, которая:
   - получает ID-токен с `aud=vault` ;
   - передаёт его в Vault;
   - загружает секрет из KV;
   - выводит значение секрета.

```
stages:
  - test

vault-demo:
  stage: test
  image: alpine
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DATABASE_PASSWORD:
      vault: code/vault-demo/DATABASE_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - echo "Raw value (masked by Deckhouse Code):"
    - echo "$DATABASE_PASSWORD"

    - echo
    - echo "Value with spaces (not masked):"
    - printf '%s\n' "$DATABASE_PASSWORD" | sed 's/./& /g'
```

Результат

Если интеграция настроена правильно:

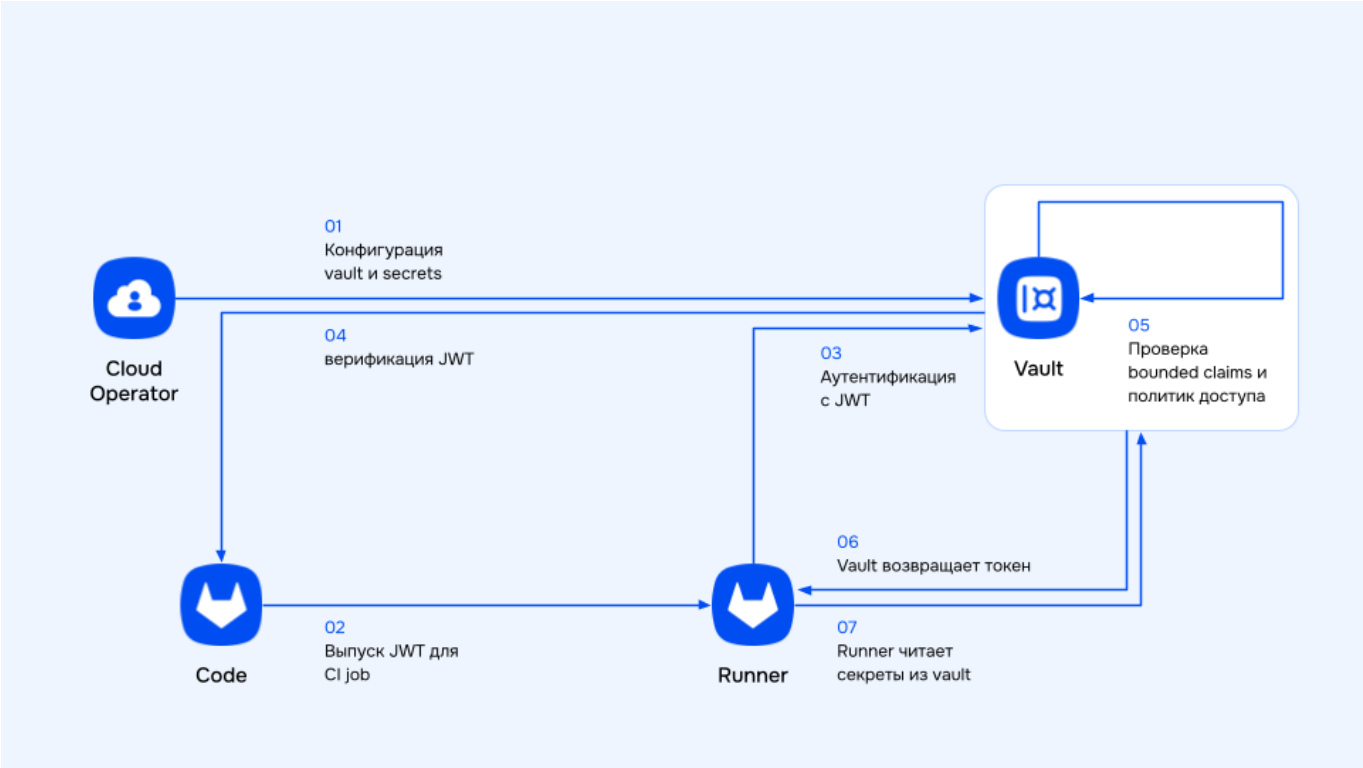
- CI job успешно получит ID-токен;
- Vault проверит значение `project_path` и выдаст доступ;
- секрет будет считан и выведен в лог.

В выводе вы увидите значение поля `DATABASE_PASSWORD`, загруженное напрямую из Vault.

Интеграция с Deckhouse Stronghold

Deckhouse Stronghold — это enterprise-решение для управления секретами, построенное на базе HashiCorp Vault. Интеграция Stronghold с Deckhouse Code позволяет безопасно использовать секреты в CI/CD-пайплайнах без необходимости хранить их в переменных проекта.

Схема взаимодействия



Преимущества интеграции:

- **Централизованное управление секретами** — все секреты хранятся в одном месте с аудитом доступа.
- **Автоматическая ротация** — секреты могут обновляться автоматически без необходимости изменения CI/CD-пайплайнов.
- **Гранулярный контроль доступа** — доступ к секретам ограничивается на основе проекта, ветки, окружения и других параметров.
- **Отсутствие секретов в репозитории** — секреты не хранятся в коде и не передаются через переменные CI.

Примеры использования

В этом разделе приведены типовые сценарии использования секретов из Vault в CI/CD-пайплайнах.

Развёртывание приложения с учётными данными базы данных

Пример получения данных для аутентификации (credentials) для подключения к PostgreSQL во время развёртывания:

```
stages:
  - deploy

deploy-production:
  stage: deploy
  image: alpine/k8s:1.28.0
  environment:
    name: production
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    DB_HOST:
      vault: apps/myapp/production/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_USER:
      vault: apps/myapp/production/DB_USER@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/production/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - kubectl create secret generic db-credentials \
        --from-literal=host=$DB_HOST \
        --from-literal=username=$DB_USER \
        --from-literal=password=$DB_PASSWORD \
        --dry-run=client -o yaml | kubectl apply -f -
    - kubectl rollout restart deployment/myapp
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

Использование API-ключей для внешних сервисов

Пример интеграции с внешними API (Slack, Telegram, S3):

```
stages:
  - notify
  - backup

notify-slack:
  stage: notify
  image: curlimages/curl:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    SLACK_WEBHOOK_URL:
      vault: integrations/slack/WEBHOOK_URL@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - |
      curl -X POST "$SLACK_WEBHOOK_URL" \
        -H "Content-Type: application/json" \
        -d "{\"text\": \"Deployment completed for $CI_PROJECT_NAME\"}"

backup-to-s3:
  stage: backup
  image: amazon/aws-cli:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    AWS_ACCESS_KEY_ID:
      vault: cloud/aws/backup-user/ACCESS_KEY_ID@kv
      token: $VAULT_ID_TOKEN
      file: false
    AWS_SECRET_ACCESS_KEY:
      vault: cloud/aws/backup-user/SECRET_ACCESS_KEY@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - aws s3 sync ./artifacts s3://my-backup-bucket/$CI_PROJECT_NAME/$CI_COMMIT_SHA/
```

Подписание Docker-образов с помощью Cosign

Пример использования приватного ключа из Vault для подписания образов:

```
stages:
  - build
  - sign

build-image:
  stage: build
  image: docker:24.0.5
  services:
    - docker:24.0.5-dind
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

sign-image:
  stage: sign
  image: bitnami/cosign:latest
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  secrets:
    COSIGN_PRIVATE_KEY:
      vault: signing/cosign/PRIVATE_KEY@kv
      token: $VAULT_ID_TOKEN
      file: true
  script:
    - cosign sign --key $COSIGN_PRIVATE_KEY $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
```

Разграничение доступа по веткам

Пример настройки различных секретов для веток `develop` и `main` с использованием bound claims по ветке (`ref`):

```
stages:
  - deploy

.deploy-template:
  image: alpine/k8s:1.28.0
  id_tokens:
    VAULT_ID_TOKEN:
      aud: vault
  script:
    - echo "Deploying from branch $CI_COMMIT_BRANCH with database $DB_HOST"
    - kubectl set env deployment/myapp DB_HOST=$DB_HOST DB_PASSWORD=$DB_PASSWORD

deploy-develop:
  extends: .deploy-template
  stage: deploy
  variables:
    VAULT_AUTH_ROLE: myapp-develop
  secrets:
    DB_HOST:
      vault: apps/myapp/develop/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/develop/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  rules:
    - if: $CI_COMMIT_BRANCH == "develop"

deploy-main:
  extends: .deploy-template
  stage: deploy
  variables:
    VAULT_AUTH_ROLE: myapp-main
  secrets:
    DB_HOST:
      vault: apps/myapp/main/DB_HOST@kv
      token: $VAULT_ID_TOKEN
      file: false
    DB_PASSWORD:
      vault: apps/myapp/main/DB_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

Пример настройки роли в Vault с bound claims по ветке для разграничения доступа:

```
# Роль для `develop` — доступ только с ветки `develop`.
vault write auth/jwt/role/myapp-develop - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_path": "myteam/myapp",
    "ref": "develop",
    "ref_type": "branch"
  },
  "policies": ["myapp-develop-policy"],
  "ttl": "1h"
}
EOF

# Роль для `main` — доступ только с ветки `main` (защищённой).
vault write auth/jwt/role/myapp-main - <<EOF
{
  "role_type": "jwt",
  "user_claim": "sub",
  "bound_audiences": ["vault"],
  "bound_claims": {
    "project_path": "myteam/myapp",
    "ref": "main",
    "ref_type": "branch",
    "ref_protected": "true"
  },
  "policies": ["myapp-main-policy"],
  "ttl": "1h"
}
EOF
```

Аналитика CI/CD

Отчёт «Аналитика CI/CD» показывает, сколько пайплайнов, созданных группой или проектом за выбранный период, завершилось, сколько они выполнялись и как распределились их статусы.

Доступность отчёта

Отчёт проекта доступен участникам с ролью «Наблюдатель» и выше, в том числе тем, кто получил роль в родительской группе.

Настройка проекта «Видимость пайплайнов на уровне проекта» открывает отчёт всем, кому виден сам проект: пользователям с типом «Аудитор», участникам с ролью «Гость», в публичном проекте — любому посетителю, включая не выполнившего вход, во внутреннем — любому вошедшему

пользователю. При выключенной настройке отчёт остаётся доступен участникам с ролью «Наблюдатель» и выше.

Для доступа к отчёту проекта читателю нужен также доступ к пайплайнам этого проекта. Если видимость функции «CI/CD» сужена до состава участников, отчёт закрывается для всех, кто не входит в проект, — в том числе для посетителей публичного проекта. Пользователи с типом «Аудитор» сохраняют доступ независимо от этой настройки.

Сужение видимости функции «Аналитика» до состава участников закрывает отчёт так же, и пользователи с типом «Аудитор» так же его сохраняют. Выключение функции «Аналитика» закрывает отчёт для всех, включая участников проекта.

Выключение функции «CI/CD» отчёт не закрывает: у проекта сохраняется уже накопленная история пайплайнов, и отчёт продолжает её отображать. При этом пункт «Аналитика CI/CD» пропадает из меню «Аналитика» — то же происходит и при пустом репозитории проекта. В обоих случаях отчёт остаётся доступен по прямой ссылке.

Отчёт группы доступен участникам с ролью «Наблюдатель» и выше, в том числе тем, кто получил роль в родительской группе, и пользователям с типом «Аудитор».

Просмотр отчёта

Чтобы просмотреть отчёт группы:

1. В верхней панели нажмите «Искать или перейти к…» и выберите группу.
2. В меню группы перейдите в «Аналитика» → «Аналитика CI/CD».

Чтобы просмотреть отчёт проекта:

1. В верхней панели нажмите «Искать или перейти к…» и выберите проект.
2. В меню проекта перейдите в «Аналитика» → «Аналитика CI/CD».

Также отчёт можно открыть по прямому адресу:

Отчёт	Адрес
Группа	`https://<HOST>/groups/<GROUP>/-/analytics/ci_cd_analytics`
Проект	`https://<HOST>/<GROUP>/<PROJECT>/-/analytics/ci_cd_analytics`

Отчёт группы охватывает пайплайны её проектов и проектов её подгрупп, включая архивные проекты.

Метрики

Строка над графиками подводит итог периода:

Метрика	Содержимое
«Общее количество запусков пайплайнов»	Число завершившихся пайплайнов, созданных в периоде
«Медианная продолжительность»	Медианная продолжительность (p50) запусков, которые выполнялись
«Доля сбоев»	Доля неуспешных запусков среди тех, что завершились успехом или неудачей
«Доля успехов»	Доля успешных запусков среди тех, что завершились успехом или неудачей
«Доля остальных»	Доля отменённых и пропущенных запусков среди всех запусков периода

«Доля сбоев» и «Доля успехов» считаются от запусков, завершившихся успехом или неудачей. Каждая из них округляется до целого отдельно, поэтому в сумме они дают около ста: период с 5 успешными и 3 неуспешными запусками показывает долю сбоев 38 % и долю успехов 63 %. «Доля остальных» считается от всех запусков периода и добавляется к ним сверху. Значок «Как это рассчитывается?» рядом с «Долей сбоев» и «Долей остальных» приводит формулу.

Доля, для знаменателя которой в периоде нет ни одного запуска, показана прочерком. Поэтому период с одними отменёнными и пропущенными запусками даёт прочерк в «Доле сбоев» и «Доле успехов» и сто процентов в «Доле остальных».

Когда в проекте есть неуспешные запуски, ссылка «Посмотреть все» под «Долей сбоев» открывает список пайплайнов проекта, отфильтрованный по неуспешным запускам. Выбранная в отчёте ветка переносится в этот список фильтром; со значением «Все ветки» список открывается без фильтра по ветке.

Графики

График «Продолжительность» содержит по линии на каждый перцентиль продолжительности пайплайна:

- «Медиана (50-й процентиль)»;
- «95-й перцентиль».

Вертикальная ось графика размечена в минутах.

График «Статус» содержит по столбцу на каждый день, разделённому на серии:

- «Успешные»;
- «Ошибка»;
- «Другие (отменённые, пропущенные)».

Вертикальная ось графика считает пайплайны.

Фильтры

Фильтры над графиками ограничивают метрики и оба графика:

Фильтр	Значения
«Источник»	Событие, создавшее пайплайн: «Push», «Расписание», «Событие запроса на слияние» и остальные. По умолчанию — «Любой источник»
«Ветвь»	Ветка пайплайна. Изначально выбрана ветка по умолчанию проекта, а значение «Все ветки» снимает фильтр. В отчёте группы такого фильтра нет
«Диапазон дат»	«Прошлая неделя», «Последние 30 дней», «Последние 90 дней», «Последние 180 дней». По умолчанию — «Прошлая неделя»

Фильтр «Ветвь» учитывает также пайплайны запросов на слияние, для которых эта ветка — исходная. Они попадают в отчёт, пока фильтр «Источник» оставлен в значении «Любой источник» или в нём выбрано «Событие запроса на слияние»; с любым другим источником в отчёт попадают запуски самой ветки.

Выбранные фильтры сохраняются в адресе страницы в параметрах `source` и `time`, поэтому настроенный отчёт можно передать ссылкой. В отчёте проекта туда же попадает ветка, в параметре `branch`.

Как считаются пайплайны

Отчёт учитывает пайплайны, созданные в периоде и дошедшие до завершённого статуса: успешные, неуспешные, отменённые и пропущенные. Пайплайн относится к дню своего создания, отсчитанному по UTC, поэтому пайплайн, созданный в один день и завершившийся на следующий, учитывается в дне создания.

Период заканчивается началом текущего дня, поэтому пайплайны, созданные сегодня, в отчёт не попадают, а на графиках нет точки за сегодня.

Перцентили графика «Продолжительность» и метрика «Медианная продолжительность» не учитывают запуски без продолжительности и запуски с нулевой продолжительностью. За день без таких запусков точки на графике нет, а за период без таких запусков вместо медианы показан прочерк.

Отчёт группы с теми же фильтрами и периодом обновляется не чаще раза в минуту, поэтому при повторном просмотре он может отставать от текущего состояния до минуты. Отчёт проекта формируется по каждому запросу.

Если отчёт строится слишком долго, вместо графиков страница показывает сообщение об этом и предлагает сузить отчётный период. Выберите меньшее значение «Диапазона дат» и откройте отчёт заново.

Безопасность

Настройки безопасности группы и проекта

Владельцы групп и мейнтейнеры проектов применяют эти настройки для повседневной защиты проектов. Откройте страницу «Настройки» группы или проекта в боковом меню слева, а затем вкладку, указанную в каждом разделе ниже.

Настройки уровня группы

Настройки, которые владелец группы применяет ко всем проектам группы.

Основные

Установите уровень видимости группы «Приватный», где это возможно, в разделе «Настройки» → «Основные»: тогда группа и её проекты видны только участникам, что снижает риск непреднамеренного доступа к исходному коду и задачам.

Права доступа и возможности группы

Настройте права доступа и возможности группы в разделе «Настройки» → «Основные»:

- Могут ли участники приглашать группы за пределами этой группы и её подгрупп.
- Можно ли передавать проекты этой группы другим группам.
- Уровень доступа к wiki группы.
- Протоколы доступа к Git (SSH, HTTP(S)), включённые для группы.
- Минимальная роль, необходимая для создания проектов в группе.
- Минимальная роль, необходимая для создания подгрупп.
- Могут ли участники создавать токены доступа группы и проекта в этой группе. Токен доступа группы может создать только участник с ролью `Owner`; токен доступа проекта, созданный участником группы, ограничен ролью этого участника в проекте.
- Обязательна ли двухфакторная аутентификация для каждого участника группы, и какова продолжительность периода до её принудительного применения. После включения этой настройки подгруппы не могут задавать собственные правила двухфакторной аутентификации.
- Ограничен ли доступ к содержимому Pages участниками проекта для всех проектов группы.

Репозиторий

Настройки, которые применяются к Git-репозиторию каждого проекта группы.

Токены деплоя

Токены деплоя предоставляют доступ к репозиториям и пакетам группы без привязки к личной учётной записи. Регулярно проверяйте токены деплоя, удаляйте неиспользуемые и следите, чтобы у оставшихся были чёткое описание, ограниченная область действия и срок действия.

Ветка по умолчанию

Настройте защиту ветки по умолчанию для всех проектов группы в разделе «Настройки» → «Репозиторий». Проект наследует эту настройку от группы, если группа не разрешила её переопределять.

Правила отправки изменений

Настройте правила отправки изменений для всех проектов группы в разделе «Настройки» → «Репозиторий». Если переопределение правил уровня инстанса разрешено, переопределите их здесь для политики, специфичной для группы. Список доступных правил — в разделе Правила отправки изменений.

Запретите переопределение этих правил на уровне проекта, если проекту не нужна собственная политика.

CI/CD

Настройки, которые применяются к CI/CD-пайплайнам и переменным во всех проектах группы.

Пайплайны

Включите опцию «Включить формат JWT для токенов заданий CI/CD» в разделе «Настройки» → «CI/CD».

Переменные

Маскируйте переменные CI/CD, которые содержат чувствительные значения, в разделе «Настройки» → «CI/CD». Значение замаскированной переменной не отображается в логах заданий и событиях аудита.

Настройки уровня проекта

Настройки, которые мейнтейнер или владелец проекта применяет к отдельному проекту.

Основные

Основные настройки проекта определяют, кто может просматривать проект и что может делать без назначенной роли.

Видимость, функции и разрешения проекта

Установите видимость проекта «Приватный», где это возможно, в разделе «Настройки» → «Основные». Включите опцию «Требуйте аутентификацию для просмотра медиафайлов», чтобы запретить неавторизованный доступ к медиафайлам, встроенным в задачи и запросы на слияние, и ограничьте доступ к объектам LFS, запросам на слияние, веткам, реестрам пакетов и контейнеров, задачам, окружениям и релизам в соответствии с требованиями вашей организации. Скачивание из реестра пакетов по умолчанию запрещено для всех пользователей.

Репозиторий

Настройки, которые применяются к Git-репозиторию проекта.

Правила отправки изменений

Настройте правила отправки изменений для проекта в разделе «Настройки» → «Репозиторий». Если переопределение правил уровня группы разрешено, переопределите их здесь для политики, специфичной для проекта. Список доступных правил — в разделе Правила отправки изменений.

Защищённые ветки

Используйте защищённые ветки, в разделе «Настройки» → «Репозиторий», чтобы определить, какие роли могут сливать изменения в основную и стабильные ветки или отправлять изменения в них напрямую. По умолчанию отправлять изменения в защищённую ветку напрямую могут только участники с ролью `Maintainer` или `Owner`. Подробнее — в разделе Защищённые ветки.

Защищённые теги

Если теги отмечают релизы или другие значимые точки в истории проекта, ограничьте, какая роль может создавать теги по заданному шаблону, в разделе «Настройки» → «Репозиторий». Например, ограничьте создание тегов версий по шаблону `*_v*` ролью `Maintainer`.

Токены деплоя

Токены деплоя предоставляют доступ к репозиторию и пакетам проекта без привязки к личной учётной записи. Регулярно проверяйте токены деплоя, удаляйте неиспользуемые и следите, чтобы у оставшихся были чёткое описание, ограниченная область действия и срок действия.

Ключи деплоя

Ключи деплоя — альтернатива токенам деплоя для предоставления доступа к репозиторию без привязки к личной учётной записи, в разделе «Настройки» → «Репозиторий». К ним применима та же проверка: область действия, описание и срок действия.

Запросы на слияние

Настройки, которые определяют, как запрос на слияние проходит ревью и сливается в основную или стабильные ветки проекта, в разделе «Настройки» → «Запросы на слияние».

Проверки слияния

- Включите опцию «Пайплайны должны завершиться успешно» и добавьте в пайплайн проекта необходимые проверки безопасности: сканирование секретов, статический и динамический анализ кода, сканирование зависимостей и образов контейнеров на уязвимости.
- Включите опцию «Все дискуссии должны быть закрыты», чтобы код, который ещё проходит ревью, не мог попасть в основную или стабильные ветки.

Назначайте ответственных за части кода с помощью файла `CODEOWNERS` и настраивайте правила апрувов запросов на слияние, чтобы задать минимальное число апруверов, запретить автору одобрять собственный запрос на слияние и ограничить, кто может изменять сами правила апрувов, например подключая файл с политикой ревью из другого проекта через механизм `include`. Подробнее — в разделах CODEOWNERS и Правила апрувов запросов на слияние.

CI/CD

Настройки, которые применяются к CI/CD-пайплайнам, переменным и токенам проекта, в разделе «Настройки» → «CI/CD».

Пайплайны

Включите опцию «Используйте отдельные кеши для защищённых ветвей». Назначьте на файл `.gitlab-ci.yml` ответственного через `CODEOWNERS`, чтобы контролировать, кто может изменять конфигурацию пайплайна, и требуйте ручного подтверждения перед запуском критичного пайплайна или задания.

Переменные

Отмечайте переменные CI/CD с чувствительными значениями как защищённые и маскируемые и храните секреты в переменных CI/CD либо во внешнем Vault или Deckhouse Stronghold вместо репозитория. Подробнее — в разделе Интеграция с внешним Vault. Включите опцию «Показывать переменные конвейера», чтобы значения переменных пайплайна отображались в логах заданий: это упрощает диагностику сбоев пайплайна и обнаружение неожиданных значений.

Токены запуска конвейера

Токен запуска конвейера запускает пайплайн с правами участника, который его выпустил. Проверяйте такие токены с той же тщательностью, что и разрешения токена заданий CI/CD (ниже).

Разрешения токена заданий CI/CD

Ограничьте список групп и проектов, которым разрешён доступ к данным этого проекта из их собственных пайплайнов, в разделе «Настройки» → «CI/CD», с помощью белого списка.

Пакеты и реестры

Защищайте пакеты, опубликованные в реестре пакетов проекта, с помощью механизма защищённых пакетов, в разделе «Настройки» → «Пакеты и реестры», ограничивая доступ участникам выше определённой роли. То же применимо к реестру контейнеров.

SAST-сканирование с помощью Semgrep

Статический анализ кода (SAST, Static Application Security Testing) в Deckhouse Code ищет уязвимости в исходном коде проекта. За ним стоит сканер Semgrep, а само сканирование называется `sast`.

Сканирование обязательное: его подмешивает в пайплайн проекта политика исполнения сканирования, которая лежит в отдельном проекте политик безопасности. Что сканируется и что приводит к ошибке пайплайна, решает политика — то есть команда безопасности. Файл `.gitlab-ci.yml` и настройки CI/CD проверяемого проекта на это решение не влияют.

> ⚠ Сканирование безопасности включается функциональным флагом уровня инстанса `fe_security_scan_policies`, который **по умолчанию отключён**. Пока он выключен, страницы политик и страницы настроек интеграций сканеров скрыты, а задачи сканирования в пайплайны не подмешиваются. Попросите администратора инстанса включить флаг.

Что делает сканирование

Сканирование `sast` добавляет в пайплайн задачи `semgrep_sast_scan` и `semgrep_sast_junit`, обе на стадии `fe-security-scanner`:

Задача	Что делает
`semgrep_sast_scan`	Запускает сканер по репозиторию и публикует его собственный JSON-вывод как артефакт.
`semgrep_sast_junit`	Читает этот JSON, строит из него отчёт для сводки тестов и отчёт безопасности и решает исход: задача завершается успешно, если ни одна находка не достигла порога блокировки, и с ошибкой в противном случае. Выполняется в образе Ruby.

Обе задачи выгружают артефакты с `when: always`, поэтому заблокированный пайплайн сохраняет находки, которые его заблокировали.

Находки попадают туда же, куда и находки остальных сканеров политики:

Отчёт	Где смотреть
Тесты (JUnit)	Вкладка «Tests» пайплайна и сводка тестов в merge request
Отчёт безопасности (SAST)	Страница отчёта об уязвимостях
Находки, переданные в DefectDojo	DefectDojo, если эта интеграция настроена

Предварительные требования

Чтобы сканирование могло запуститься, убедитесь, что:

- администратор инстанса включил функциональный флаг `fe_security_scan_policies`;
- проект политик безопасности привязан к проекту или к группе выше него;
- доступен GitLab Runner с executor `docker`, и он может загрузить оба образа, которые использует сканирование: образ сканера (раздел «Откуда берётся образ сканера») и образ Ruby, в котором выполняется вторая задача, — по умолчанию `ruby:3.3.10-slim`, его можно перенаправить переменной инстанса `FE_SCANS_REPORT_CONVERTER_IMAGE`.

Включение сканирования

Добавьте действие `sast` в файл `.gitlab/security-policies/policy.yml` проекта политик безопасности:

```
scan_execution_policy:
  - name: SAST everywhere
    enabled: true
    rules:
      - type: pipeline
        branches: ["*"]
    actions:
      - scan: sast
```

Затем привяжите проект политик к нужному проекту или группе в разделе «Настройки» → «Политика безопасности». После этого каждый пайплайн, попадающий под правила политики, получает две задачи, описанные выше.

Параметры сканирования можно задавать и в редакторе политик — он формирует форму, описанную ниже.

Что задаёт политика

Форма действия `sast` собрана из групп «Образ сканера», «Правила», «Исключаемые категории», «Блокировка», «Пути», «Уровни правил» и «Дополнительные параметры». Все параметры принадлежат политике: значения записываются в задачу сервером при формировании пайплайна, поэтому собственные переменные CI/CD проверяемого проекта их не меняют.

Каждая группа раскрывается и сворачивается по своему заголовку, и все они изначально свёрнуты. Свёрнутое состояние само по себе содержательно: свёрнутая группа означает, что политика в неё ничего не записала, поэтому эта часть сканирования остаётся с тем ответом, который сервер даёт без политики: для образа сканера — тот, который называет интеграция Semgrep, если она его называет, и поставочное значение во всём остальном. Раскройте группу и заполните поле — и политика забирает это решение себе для всех проектов, которые она покрывает. Сохраните политику и откройте её заново — группы, в которых что-то задано, откроются сами, поэтому сохранённая политика показывает всё, что задаёт.

Группа названа по тому выбору, вокруг которого собрана, и этот выбор нарисован без повторения собственного названия внутри группы: «Образ сканера» раскрывается прямо на источниках образа, «Правила» — на источниках правил, «Пути» — на источниках фильтров, «Блокировка» — на пороге, «Уровни правил» — на трёх уровнях. Поля, которые открывает выбранный вариант, сохраняют свои названия под ним, а подсказка ведущего выбора («?») стоит рядом с заголовком группы.

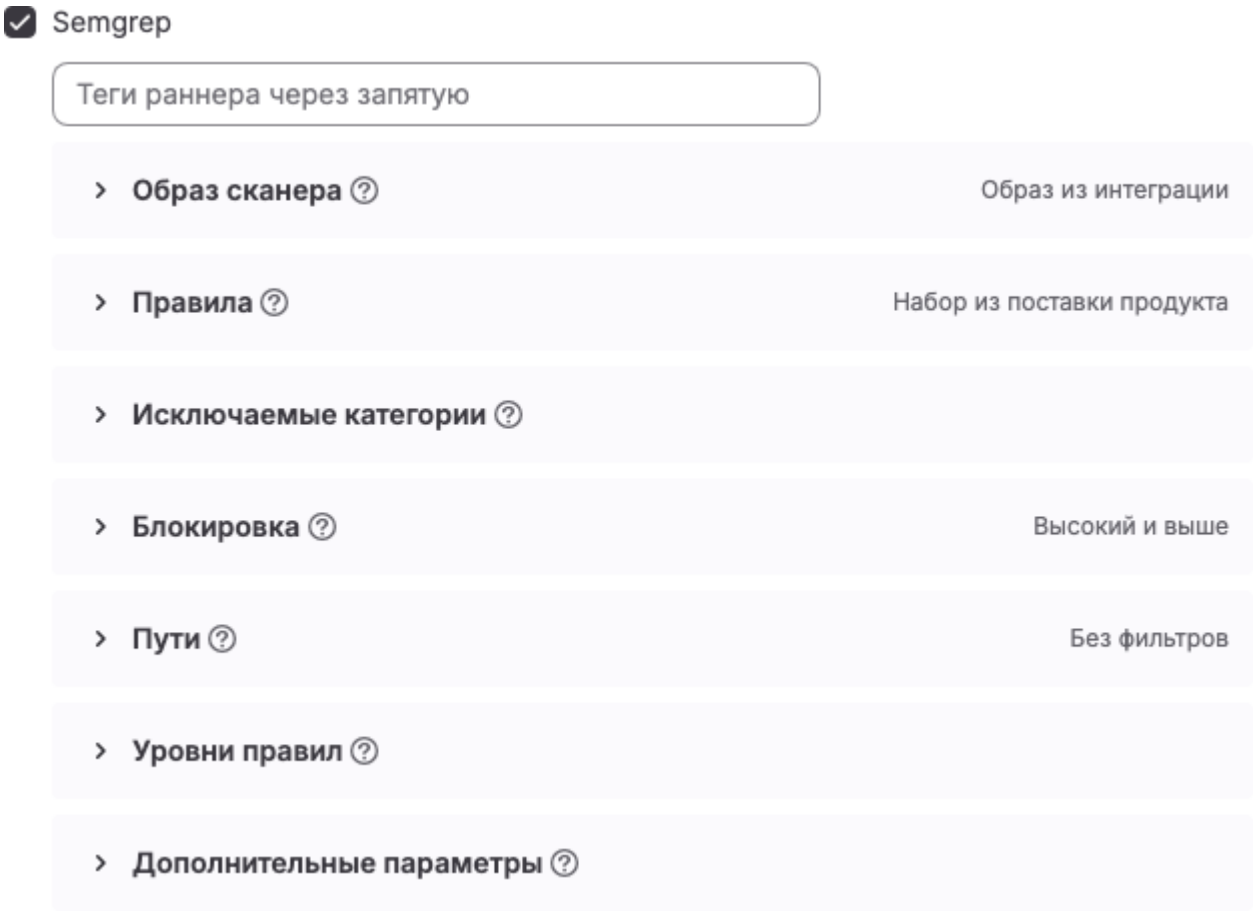
То, что решила свёрнутая группа, написано в конце строки её заголовка. Именно это делает свёрнутую форму читаемой: порог и источник правил видно, ничего не раскрывая.

Группа	**Что написано в её свёрнутой строке**
«Образ сканера»	Какой из трёх источников даёт образ — «Образ из интеграции», если интеграция называет образ, и «Образ из поставки продукта» во всех остальных случаях
«Правила»	Откуда берутся правила — «Набор из поставки продукта», пока политика не выберет другой источник
«Исключаемые категории»	Ничего. В группе стоят флажки, а строка подводит выбор из списка
«Блокировка»	Порог — «Высокий и выше», пока политика не выберет другой
«Пути»	Откуда берутся фильтры путей — «Без фильтров», пока политика не выберет другое
«Уровни правил»	Уровни, включённые политикой, например «Высокий, Средний». Пока политика не задала ни одного, строка пуста
«Дополнительные параметры»	Ничего. В свободном тексте выбора из списка нет, поэтому подводить нечего

Строки «Образ сканера», «Правила», «Блокировка» и «Пути» всегда содержат значение: их элемент управления — набор переключателей, а тот всегда показывает ответ, заданный политикой либо тот, который дал бы сервер, пока политика молчит. Строка и элемент управления поэтому согласованы. «Уровни правил» — набор из нескольких значений, и пока политика ничего не задала, называть нечего: вместо этого все три флажка стоят установленными, а это то же самое состояние.

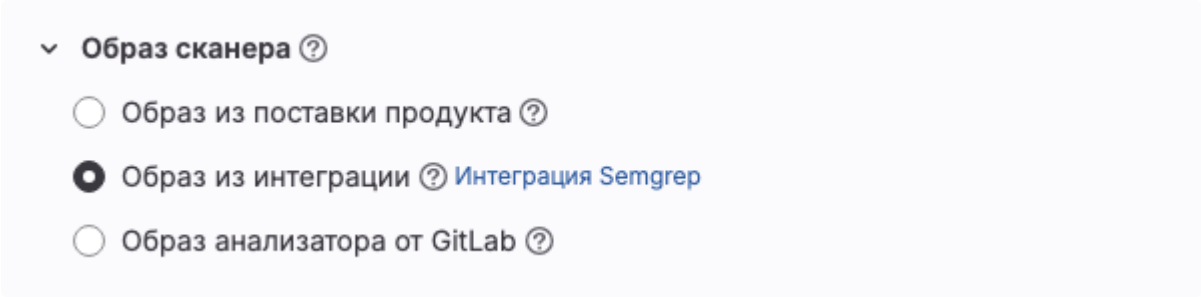
Ни у одной здешней группы нет флажка рядом с заголовком. Там, где флажок есть у группы другого сканера, эта группа — модуль, который можно выключить; здешние группы — само сканирование. Выключает его удаление действия `sast` из политики. Флажки внутри групп «Исключаемые категории» и «Пути» несут каждый своё отдельное значение.

Поле, которое отвечает одному варианту выбора, показывается под этим вариантом и больше нигде. Выбранный источник правил «Файл в проекте политик безопасности» помещает путь к файлу правил прямо под этим вариантом; выбор другого источника убирает поле с экрана.



Образ сканера

Группа задаёт, в каком образе выполняется сканирование. Она раскрывается на поле «Источник образа» — выборе из трёх вариантов:



Источник	Какой образ выполняет сканирование
«Образ из поставки продукта»	Образ, который фиксирует этот релиз, либо тот, что администратор задал для всей установки переменной инстанса `FE_SCANS_SEMGREP_IMAGE`. Образ несёт свою версию, поэтому поля версии у этого источника нет.
«Образ из интеграции»	Реестр, в который интеграция Semgrep зеркалирует сканер, либо единственный подготовленный образ, который она называет (раздел «Интеграция Semgrep»).
«Образ анализатора от GitLab»	`registry.gitlab.com/security-products/semgrep` на теге, который называет политика. Нужна установка, у которой есть доступ к этому реестру.

Поле	Что задаёт
«Версия образа»	Тег, по которому загружается образ анализатора, — так, как его указывает реестр: три числа, например `6.25.0`, либо дайджест. Показывается только под вариантом «Образ анализатора от GitLab», потому что два остальных источника несут свою версию. Если поле пустое, используется тег из поставки этого релиза.

Тег образа и версия semgrep — разные нумерации, и больший тег образа означал более старый semgrep. Подсказка поля «Версия образа» называет версию semgrep из поставки продукта и тег образа, который её несёт, а список подсказок в поле называет то же самое для каждого тега, известного этому релизу. Подсказка рядом с заголовком группы сообщает, какой образ выполнит эта установка: подготовленный образ, названный интеграцией, реестр, в который она зеркалирует сканер, либо образ, заданный для самой установки.

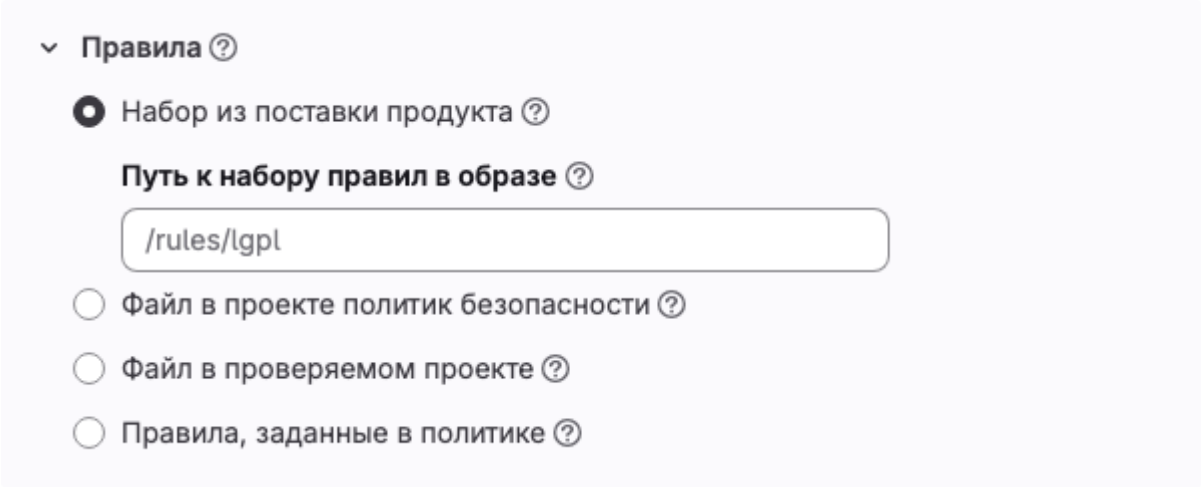
Рядом с вариантом «Образ из интеграции» стоит ссылка на страницу интеграции Semgrep — в любом состоянии этого варианта и только для политики, которой владеет группа. Политику, которой владеет проект, настраивает группа выше него, а доступа к её настройкам у мейнтейнера проекта может не быть, поэтому там у этого варианта ссылки нет.

«Образ из интеграции» — единственный источник, который может быть недоступен, и причина у этого одна: интеграция не называет образ, который сканированию забирать. Так выходит, если для группы не настроена интеграция Semgrep либо если она настроена, но не называет ни реестр, ни подготовленный образ. Вариант при этом остаётся в списке, но серым, и причина написана на самом варианте. Два остальных источника доступны всегда.

Политика, в которой источник не указан, показывает тот ответ, который сервер дал бы без неё: «Образ из интеграции», если интеграция называет образ, и «Образ из поставки продукта» во всех остальных случаях (раздел «Откуда берётся образ сканера»).

Правила

Группа задаёт, какие правила выполняет сканирование.



Поле	Что задаёт
«Набор правил»	Откуда берутся правила. Источники перечислены ниже. По умолчанию — «Набор из поставки продукта».
«Путь к набору правил в образе»	Какой из наборов, лежащих в образе сканера, выполнять (раздел «Наборы правил в образе»). По умолчанию — `/rules/lgpl`. Если по этому пути набора нет, сканирование останавливается.
«Правила»	Сами правила, в YAML сканера. Показывается, когда источник — «Правила, заданные в политике».
«Путь к файлу правил»	Путь к файлу правил внутри проекта, из которого они читаются. По умолчанию — `.semgrep.yml`.
«Заменить поставленный набор»	По умолчанию выключено: ваши правила выполняются **вместе** с поставленным набором. Если установить флажок, они выполняются **вместо** него, и всё, что покрывал поставленный набор, перестаёт проверяться.

У каждого поля есть подсказка за знаком «?» рядом с названием; подсказка поля «Набор правил», по которому названа группа, стоит рядом с её заголовком.

Источники правил отличаются тем, кто владеет файлом:

Источник	Где лежат правила	Кто ими распоряжается
«Набор из поставки продукта»	В образе сканера, по пути, указанному выше	Поставка продукта
«Файл в проекте политик безопасности»	Проект, где лежит политика, на его ветке по умолчанию	Команда безопасности
«Файл в проверяемом проекте»	Сканируемый репозиторий, на строящемся коммите	**Проверяемый проект**
«Правила, заданные в политике»	Сама политика	Автор политики

Источники «Набор из поставки продукта», «Файл в проекте политик безопасности» и «Правила, заданные в политике» читает сервер и записывает в задачу. «Файл в проверяемом проекте» читает сама задача, из своей рабочей копии, на строящемся коммите. Это же единственный источник, который пишет проверяемая сторона: коммит, переписывающий этот файл, переписывает правила сканирования, которое проверяет этот же проект. Снятый флажок «Заменить поставленный набор» сохраняет базовую линию даже тогда, когда файл в проекте опустошён.

Файл правил, названный политикой, но не доставленный сканированию, останавливает задачу: она печатает причину и завершается с ошибкой, а поставленный набор пропавший файл не подменяет.

Исключаемые категории

Группа задаёт, какие категории поставленного набора сканирование не выполняет.

Поле	Что задаёт
«Исключить правила для секретов»	По умолчанию выключено. Если установить флажок, правила, которые ищут учётные данные, попавшие в репозиторий, в прогон не идут.
«Исключить правила для конфигурации»	По умолчанию выключено. Если установить флажок, правила, которые читают конфигурацию и инфраструктурный код — Terraform, Dockerfile, манифесты Kubernetes, конфигурацию CI, — в прогон не идут.
«Исключить правила для вредоносного кода»	По умолчанию выключено. Если установить флажок, правила, которые ищут признаки намеренно скрытого кода, в прогон не идут.

Флажки показываются, пока правила берутся из поставленного набора: у своих правил категорий нет. Установленный флажок исключает категорию, снятый оставляет её в прогоне — и так же продолжает вести себя политика, написанная до появления этих полей. Подсказка группы говорит, зачем этот выбор нужен: если одну из этих категорий уже покрывает другое обязательное сканирование, о ней сообщат оба, и одна и та же строка станет находкой дважды.

Что лежит в каждой категории и какой набор правил нужен для исключения — в разделе «Категории внутри поставленного набора».

Блокировка

Группа задаёт, что находка делает с пайплайном.

Поле	Что задаёт
«Порог блокировки»	Что приводит к ошибке пайплайна. Находки ниже порога всё равно попадают в отчёт и в отчёт об уязвимостях: порог решает только исход пайплайна. По умолчанию — «Высокий и выше».

Значения порога:

Значение	Что приводит к ошибке пайплайна
«Высокий и выше» (по умолчанию)	Находка высокого уровня серьёзности. Средний и информационный уровни только попадают в отчёт.
«Средний и выше»	Средний и высокий уровни серьёзности. Информационный только попадает в отчёт.
«Любая находка»	Любая находка, какого бы уровня серьёзности она ни была.
«Только сообщать, ничего не блокировать»	Ничто. Находки всё равно попадают в отчёт, в отчёт об уязвимостях и в DefectDojo.

Semgrep сообщает об уровнях ERROR, WARNING и INFO, которые сканирование показывает как высокий, средний и информационный. Критического и низкого у него нет, поэтому порога с таким именем не достигала бы ни одна находка: «критический» блокировал бы ничто, выглядя при этом строгим, а «низкий» повторял бы «средний». Значение «Только сообщать, ничего не блокировать» вносит отказ от блокировки в политику отдельным значением.

Пути

Группа задаёт, что сканирование осматривает и что оставляет за рамками.

Поле	Что задаёт
«Фильтры путей»	Откуда берутся списки путей, которые не нужно осматривать. По умолчанию — «Без фильтров».
«Путь к файлу фильтров»	Путь к файлу фильтров внутри проекта, из которого они читаются. По умолчанию — `.semgrepignore`.
«Исключить пути»	По одному пути или шаблону в строке: сторонний код, фикстуры, сгенерированные файлы.
«Только эти пути»	По одному пути или шаблону в строке. Всё остальное в сканирование не попадает.
«Учитывать .gitignore репозитория»	По умолчанию выключено.

При любом источнике фильтров задача добавляет к тому, что задала политика, свои встроенные исключения — `node_modules/`, `vendor/`, `dist/`, `build/`, `.venv/` и `.git/`, — поэтому деревья зависимостей остаются вне сканирования и тогда, когда фильтры политики о них молчат.

У фильтров тот же выбор источника, что и у правил:

Источник	Где лежит список	Кто им распоряжается
«Без фильтров»	—	—
«Файл в проекте политик безопасности»	Рядом с политикой, применяется ко всем проектам, которые она покрывает	Команда безопасности
«Файл в проверяемом проекте»	Сканируемый репозиторий	**Проверяемый проект**
«Списки, заданные в политике»	Два списка в форме	Автор политики

По умолчанию задача убирает любой `.semgrepignore`, найденный в сканируемом репозитории, — включая вложенные в подкаталогах, — перед началом сканирования. Выбор «Файл в проверяемом проекте» это отключает:

⚠ «Файл в проверяемом проекте» и «Учитывать .gitignore репозитория» одинаково отдают проверяемому проекту право решать, что осматривает обязательное сканирование. После этого коммит в проверяемом проекте может сузить сканирование, которое его же и проверяет, и в политике это сужение не видно. Команда безопасности вполне может решить, что каждый проект лучше знает свои каталоги стороннего кода; тогда делегирование фиксируется в политике, потому что оба варианта остаются выключенными, пока эту группу не раскроют и один из них не выберут.

Ещё про эти списки:

- «Только эти пути» сильнее, чем «Исключить пути». Исключение убирает то, что названо; включение убирает всё остальное. Один шаблон `src/**` незаметно исключает из сканирования и `lib`, и `config`, и миграции.
- Применяются они в этом же порядке — сначала включения, потом исключения, — поэтому исключение действует и внутри того, что пропустило включение.

Шаблон, не совпавший ни с чем, проходит проверку формы. Ловит его задача: она печатает, сколько файлов сканер прочитал, и завершается с ошибкой, когда это число равно нулю.

Уровни правил

Группа задаёт, правила каких уровней вообще выполняются.

Поле	Что задаёт
«Уровни правил»	«Высокий» (ERROR), «Средний» (WARNING), «Информационный» (INFO). Все три флажка изначально установлены — именно это сканирование и выполняет, пока политика не задала своих уровней.

Уровень — это ровно те правила, которые сканер помечает им: «Высокий» — правила ERROR, «Средний» — WARNING, «Информационный» — INFO; находка попадает в отчёт с уровнем сработавшего правила. Снятый флажок убирает из прогона ровно правила этого уровня.

Последний оставшийся установленный флажок снять нельзя: опустевший список удаляется из политики, а политика, которая не называет уровней, — это то состояние, в котором выполняются все уровни.

«Уровни правил» и «Порог блокировки» выглядят одинаково, а делают разное:

Поле	На что влияет	Находка ниже уровня
«Уровни правил»	Какие правила **выполняются**	Её не существует: правило не выполнялось
«Порог блокировки»	Какие находки **приводят к ошибке пайплайна**	Она есть в отчёте, но не блокирует

Порог покрывает свой уровень и все уровни выше — это и означает «и выше»; уровень правил покрывает правила, помеченные сканером этим уровнем, и никакие другие.

Поэтому исключённый уровень убирает свои находки и из отчёта: отчёт об уязвимостях и DefectDojo становятся меньше, и в логе задачи об этом ничего не сказано. Порог, которого не достигает ни один установленный уровень, не сработает никогда; форма говорит об этом, когда так получилось.

Дополнительные параметры

Группа задаёт флаги, которых в остальной форме нет.

Поле	Что задаёт
«Дополнительные параметры»	Флаги, передаваемые сканированию как есть.

«Дополнительные параметры» настраивают ресурсы запуска: таймауты, лимиты памяти, максимальный размер файла. Флаги, перенаправляющие или подавляющие отчёт, отклоняются, как и флаги, которыми форма распоряжается сама: форматы вывода и отчётов (`--json` , `--sarif` , `--junit-xml` , `--gitlab-sast` , `--gitlab-secrets` , `--output` , `--text` и парные к ним `*-output`), а также `--config` , `--metrics` и `--baseline-commit` .

Подсказка этого поля сообщает и то, куда попадают находки сканирования: они публикуются как отчёт безопасности, а значит, доходят до отчёта об уязвимостях и до всего, что администратор к нему подключил.

Наборы правил в образе

По умолчанию сканирование выполняет набор правил, лежащий внутри образа сканера, по пути `/rules/lgpl` . В образе лежит несколько наборов, и поле «Путь к набору правил в образе» выбирает между ними:

Путь в образе	Правил	Правил по языкам
`/rules/lgpl` (по умолчанию)	152	`javascript` 83, `kotlin` 58, `swift` 5, `java` 4, `typescript` 4, `generic` 2
`/rules/lgpl-cc`	97	`ruby` 40, `java` 39, `php` 9, `python` 6, `javascript` 1, `typescript` 1, `generic` 1, `yaml` 1
`/rules/gitlab`	5	`java` 2, `generic` 1, `javascript` 1, `kotlin` 1
`/rules`	592	наборы выше плюс файлы правил, лежащие рядом с ними: `scala` 86, `python` 79, `c` 62, `cpp` 62, `go` 27, `csharp` 22 и другие

Правило может называть несколько языков, поэтому суммы по языкам больше числа правил. Строка `/rules` — это каталог целиком: если направить сканирование на него, загрузятся и все наборы внутри, и файлы рядом с ними.

Значения измерены в образе `registry.gitlab.com/security-products/semgrep:6.25.0` и относятся к нему; в более новом образе состав может быть другим. У каждого каталога в образе свой файл лицензии, а выполняемый набор называют в поле «Путь к набору правил в образе».

> ⚠ **Набор по умолчанию покрывает `javascript`, `kotlin`, `swift`, `java`, `typescript` и `generic`. Python, Go, Ruby, PHP, C, C++, C# и Scala остаются за его пределами.**
> Репозиторий на одном из этих языков, просканированный по пути по умолчанию, завершится успешно и без находок: ни одно правило к нему не подошло. Покрытия SAST у такого репозитория нет, и зелёный пайплайн здесь означает только то, что сканеру нечего было применить.
> Укажите в поле «Путь к набору правил в образе» набор, покрывающий его язык, добавьте собственные правила через поле «Набор правил» или попросите администратора указать в интеграции Semgrep образ с подходящим набором и выберите в политике источник «Образ из интеграции». Тот, кто приносит свой набор, отвечает за его состав и за условия, на которых тот распространяется.

Собственные правила задаются полем «Набор правил» и, если флажок «Заменить поставленный набор» остался снятым, выполняются вместе с тем набором, который даёт образ.

Категории внутри поставленного набора

Правила в наборе размечены по категориям: категория говорит о том, о чём судит правило. Измерено на наборе из 2826 правил:

Категория	Правил	О чём судят её правила
Код	1988	Код самой программы
Конфигурация	522	Конфигурация и инфраструктурный код: Terraform, Dockerfile, манифесты Kubernetes, конфигурация CI
Секреты	225	Учётные данные, попавшие в репозиторий
Вредоносный код	91	Признаки намеренно скрытого кода: обфускация, код, собираемый во время выполнения, вызовы через промежуточные слои

В более новом образе состав может быть другим. Конфигурацию, секреты и вредоносный код политика может исключить — в группе «Исключаемые категории»; правила о коде выполняются в каждом сканировании.

Для исключения нужен набор правил, который несёт индекс своих категорий. У набора из образа, публикуемого GitLab, такого индекса нет, поэтому установленный флажок останавливает задачу: она называет категории, которые её просили исключить, и набор правил, у которого нет индекса для этого; это сообщение разобрано в разделе «Диагностика». У правил, заданных в политике, категорий тоже нет, поэтому при таком источнике флажки скрыты.

⚠ Правила для секретов и сканирование `secret_detection` смотрят на одно и то же, и тогда одна и та же строка приезжает двумя находками на двух уровнях. Измерено на одном прогоне: semgrep положил попавший в репозиторий токен в SAST-отчёт с уровнем Info, а `secret_detection` — тот же токен в свой отчёт с уровнем Critical. При этом наборы находок не вложены друг в друга: ключ AWS нашёл только semgrep, а URL вебхука Slack — только `secret_detection`, поэтому исключение правил для секретов передаёт категорию целиком `secret_detection` и вместе с ней убирает из SAST-отчёта те находки, которые были только там.
То же рассуждение годится для правил конфигурации, если инфраструктуру уже проверяет отдельный сканер, и для правил вредоносного кода: они читают намерение, поэтому репозиторий, который обфусцирует или собирает код во время выполнения по своим причинам, увидит их срабатывания.

Откуда берётся образ сканера

Образ, в котором выполняется сканирование, разрешается сервером один раз, до старта задачи, и записывается в неё готовым значением. Переменной CI/CD он не становится, поэтому настройки проверяемого проекта не могут направить сканирование на другой сканер.

Какой это образ, следует из источника, названного политикой в группе «Образ сканера»:

Источник в политике	Что загружает сканирование
«Образ из поставки продукта»	Образ, заданный администратором для всей установки переменной окружения инстанса `FE_SCANS_SEMGREP_IMAGE`, либо образ, зафиксированный этим релизом, если переменная не задана.
«Образ из интеграции»	Подготовленный образ, названный интеграцией Semgrep, — ровно так, как он указан; если интеграция называет реестр, то сканер из этого реестра, в версии, которую несёт политика, либо в той, которая задана для этой установки.
«Образ анализатора от GitLab»	`registry.gitlab.com/security-products/semgrep` в версии из поля «Версия образа», а пока это поле пустое — в версии из поставки этого релиза.

Подготовленный образ, названный в интеграции, отвечает за один источник и ни за какой другой: администратор, который его указал, задал то, что загружает вариант «Образ из интеграции», вместе с версией, поэтому версия из политики к нему не применяется. Политика, которая

запрашивает образ из поставки продукта или образ анализатора от GitLab, получает запрошенный образ.

Политика, в которой источник не указан вовсе, написана до появления этого поля и сохраняет прежнее поведение: образ, названный интеграцией, затем образ, заданный для инстанса, затем образ из поставки релиза. Версия, которую несёт такая политика, по-прежнему применяется — кроме случая, когда интеграция называет подготовленный образ.

Запись интеграции самого проверяемого проекта в расчёт не идёт: образ приходит со стороны, которая обязывает к сканированию.

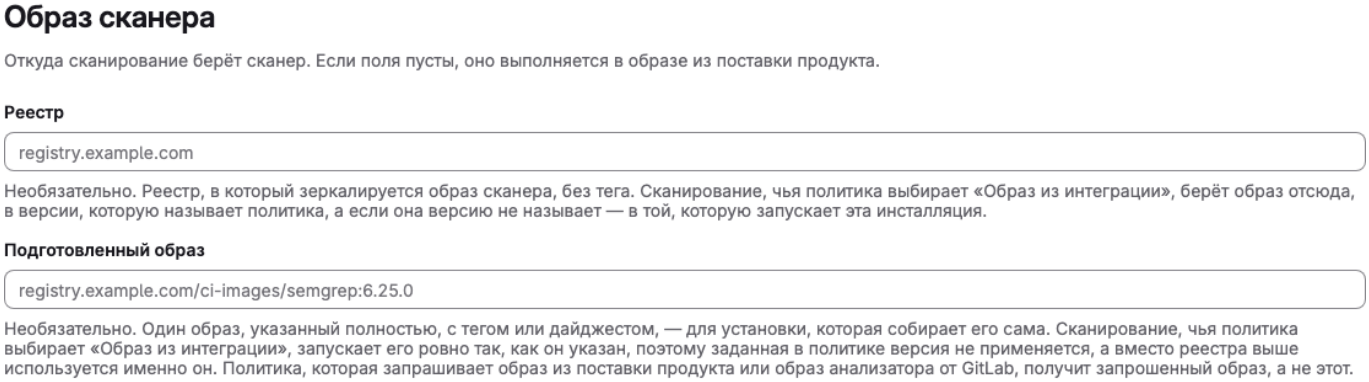
Интеграция Semgrep

Интеграция хранит то, откуда читается образ сканера. Настраивается она на **группе**, поэтому установка называет реестр один раз для всех политик под этой группой. В проекте интеграция видна без полей — так читатель узнаёт, что сканирование настраивается выше него.

Откройте на группе «Настройки» → «Интеграции» → «Semgrep» и заполните любое из полей — оба необязательны:

Поле	Что задаёт
«Реестр»	Реестр, куда скопирован образ сканера, без тега. Например, `registry.example.com` или `registry.example.com/mirror`. Сканирование, политика которого берёт образ из интеграции, загружает его отсюда — в версии, которую несёт политика, либо в той, которая задана для этой установки.
«Подготовленный образ»	Один образ, указанный полностью и с тегом или дайджестом, — для установки, которая собирает его сама. Например, `registry.example.com/ci-images/semgrep:6.25.0`. Сканирование, политика которого берёт образ из интеграции, выполняется в нём ровно так, как он написан, поэтому версия из политики не применяется. Имеет приоритет над полем «Реестр».

Если оба поля пусты, интеграция не называет образ: вариант «Образ из интеграции» в политике выбрать нельзя, и сканирование выполняется в образе из поставки продукта. То, что сканирование ищет, задаётся в политике.



Версия из поставки продукта

Этот релиз поставляет `registry.gitlab.com/security-products/semgrep:6.25.0`, зафиксированный на этой версии. Один тег фиксирует сразу обе половины сканера — движок и наборы правил, вложенные в образ, — потому что обе лежат в одном образе.

Поле «Версия образа» в политике принимает обе формы фиксации версии:

- номер версии, например `6.25.0`, — его читает человек и с ним можно сравнивать;
- дайджест, например `sha256:aafbcaff…`, — он называет одну сборку, и подменить её нельзя.

Плавающие теги (`latest`, `stable`, тег из одного мажорного номера — например, `6`) и ссылки с адресом реестра отклоняются: реестр называет интеграция, а политика называет только версию.

Версия, поставляемая с релизом, пересматривается при обновлении Deckhouse Code.

Закрытый контур

Установка без доступа в интернет обеспечивает оба образа, которые использует сканирование:

Образ	По умолчанию	Чем перенаправляется
Сканер	`registry.gitlab.com/security-products/semgrep:6.25.0`	Переменная инстанса `FE_SCANS_SEMGREP_IMAGE` — она перенаправляет вариант «Образ из поставки продукта»; либо поле «Реестр» или «Подготовленный образ» интеграции Semgrep — для политики, которая берёт образ из интеграции
Сборщик отчётов	`ruby:3.3.10-slim`, с Docker Hub	Переменная инстанса `FE_SCANS_REPORT_CONVERTER_IMAGE`

Скопируйте оба образа в свой реестр — так, как предписывает офлайн-инструкция GitLab для его аналитических образов, — а затем укажите этот реестр в соответствующей настройке. Deckhouse Code называет оба образа и загружает их извне.

> Образ сканера остаётся вне досягаемости dependency proxy. Прокси кеширует образы, лежащие на Docker Hub, а образ сканера публикуется в `registry.gitlab.com`, поэтому из двух образов прокси кеширует образ Ruby. Установка, которая хочет перестать загружать образ сканера извне при каждом запуске — в закрытом контуре или нет, — копирует образ в свой реестр и указывает эту копию в переменной `FE_SCANS_SEMGREP_IMAGE` или в поле «Реестр».

Платные возможности Semgrep

Всё описанное на этой странице работает на открытой версии сканера. Возможности Semgrep, остающиеся за её пределами:

Возможность	В открытой версии	Чем закрывается в Deckhouse Code
Анализ потока данных между файлами	Нет, только внутри одного файла	Ничем; ограничение принимается
Правила Pro	Нет	Собственными правилами, через поле «Набор правил»
Анализ зависимостей (Supply Chain)	Нет	Интеграцией CodeScoring или сканированием `dependency_scanning`
Проверка валидности найденных секретов	Нет	Ничем

> ⚠ Платные возможности работают через облако вендора. Их включение означает, что сканирование выгружает находки — и контекст кода вокруг них — на серверы Semgrep, за пределы вашей установки. В закрытом контуре до облака вендора нет сети, поэтому платные возможности там недоступны.

Обязательное сканирование оставляет всё внутри задачи. Оно выполняет `semgrep scan` с флагами `--metrics=off` и `--disable-version-check`, поэтому находки и телеметрия использования остаются там, куда их положила задача. Прежде чем сканер вообще будет найден, задача очищает все унаследованные переменные `SEMGREP_*` и `PYTHON*`: `SEMGREP_RULES` и `SEMGREP_BASELINE_COMMIT` среди них документированы как эквиваленты `--config` и `--baseline-commit`, и оставленные на месте они позволили бы настройкам CI/CD проверяемого проекта подменить набор правил или базу сравнения того сканирования, которое его проверяет.

Диагностика

Сканирование не появляется в пайплайне

Проверьте, что:

- функциональный флаг `fe_security_scan_policies` включён на инстансе;
- проект политик безопасности привязан к проекту, а `policy.yml` содержит действие `- scan: sast`;
- правила политики совпадают с пайплайном (ветка, тип пайплайна);
- доступен GitLab Runner с executor `docker`.

Задача завершается с ошибкой «no rule set at … in this image»

Пути, указанного в поле «Путь к набору правил в образе», в разрешённом образе нет. Сверьте путь с таблицей выше и проверьте, какой образ используется на самом деле: если выбран

источник «Образ из интеграции», у «Подготовленного образа», заданного на интеграции, раскладка может быть другой.

Задача завершается с ошибкой «carries no class index to switch them off by»

В группе «Исключаемые категории» установлен флажок, а разрешённый набор правил не несёт индекса своих категорий — у набора из образа, публикуемого GitLab, его нет. Снимите флажки или направьте сканирование на набор правил с индексом. В логе задачи напечатаны категории, которые её просили исключить, и прочитанный набор правил.

Соседнее сообщение, о том что индекс не содержит ни одного правила ни в одной из названных категорий, лечится так же.

Задача завершается с ошибкой «semgrep read no files at all»

Сканирование не осмотрело ничего и поэтому отказывается отчитываться об успехе. Обычные причины — шаблон в поле «Только эти пути», не совпавший ни с чем; фильтры, убравшие весь репозиторий; либо пустая рабочая копия. В логе задачи напечатано, сколько файлов она обработала и сколько из них убрали фильтры.

Сканирование завершается успешно, но ничего не находит

Чаще всего набор правил не покрывает язык репозитория, об этом предупреждает раздел «Наборы правил в образе». Посмотрите лог задачи: в нём напечатан используемый путь к набору правил и число загруженных правил. Тот же результат по другой причине даёт сужение поля «Уровни правил»: уровень, у которого политика сняла флажок, не выполняется, поэтому его находок в отчёте нет.

Список зависимостей и соответствие лицензий

Сканирование безопасности загружает в пайплайн Software Bill of Materials (SBOM) в формате CycloneDX. Две страницы проекта читают этот артефакт и показывают его содержимое:

Страница	Адрес	Что показывает
«Список зависимостей»	`/-/security/dependencies`	Компоненты SBOM: имя, версия, тип, лицензия, идентификатор пакета
«Соответствие лицензий»	`/-/security/licenses`	Лицензии этих компонентов и количество компонентов под каждой

Обе страницы доступны только для чтения и открываются из раздела «Безопасность» в боковом меню проекта.

Условия доступности

Обе страницы есть в разделе «Безопасность», пока выполнены все условия; иначе их адреса отвечают 404:

- в настройках проекта включена функция «Безопасность и соблюдение норм»;
- у пользователя в проекте роль Developer, Maintainer или Owner;
- на инстансе включён фича-флаг `fe_security_scan_policies`;
- лицензия установки включает сканирование безопасности.

> ⚠ Фича-флаг `fe_security_scan_policies` по умолчанию выключен. Пока он выключен, обе страницы скрыты вместе со страницами политик и интеграций со сканерами. Попросите администратора инстанса включить его.

Откуда берутся данные

Обе страницы читают последний пайплайн **ветки по умолчанию** проекта в любом его статусе: у выполняющегося пайплайна артефакта ещё нет, а у упавшего остаются отчёты, которые задания загрузили с `artifacts:when: always`.

Из этого пайплайна берутся все артефакты заданий с типом отчёта `cyclonedx`. Такой отчёт пишет сканирование `codescoring`: задание `codescoring_sca` публикует `gl-sbom.cdx.json` как `artifacts:reports:cyclonedx`. Задание из собственного `.gitlab-ci.yml` проекта, которое загружает отчёт того же типа, попадает на эти страницы так же.

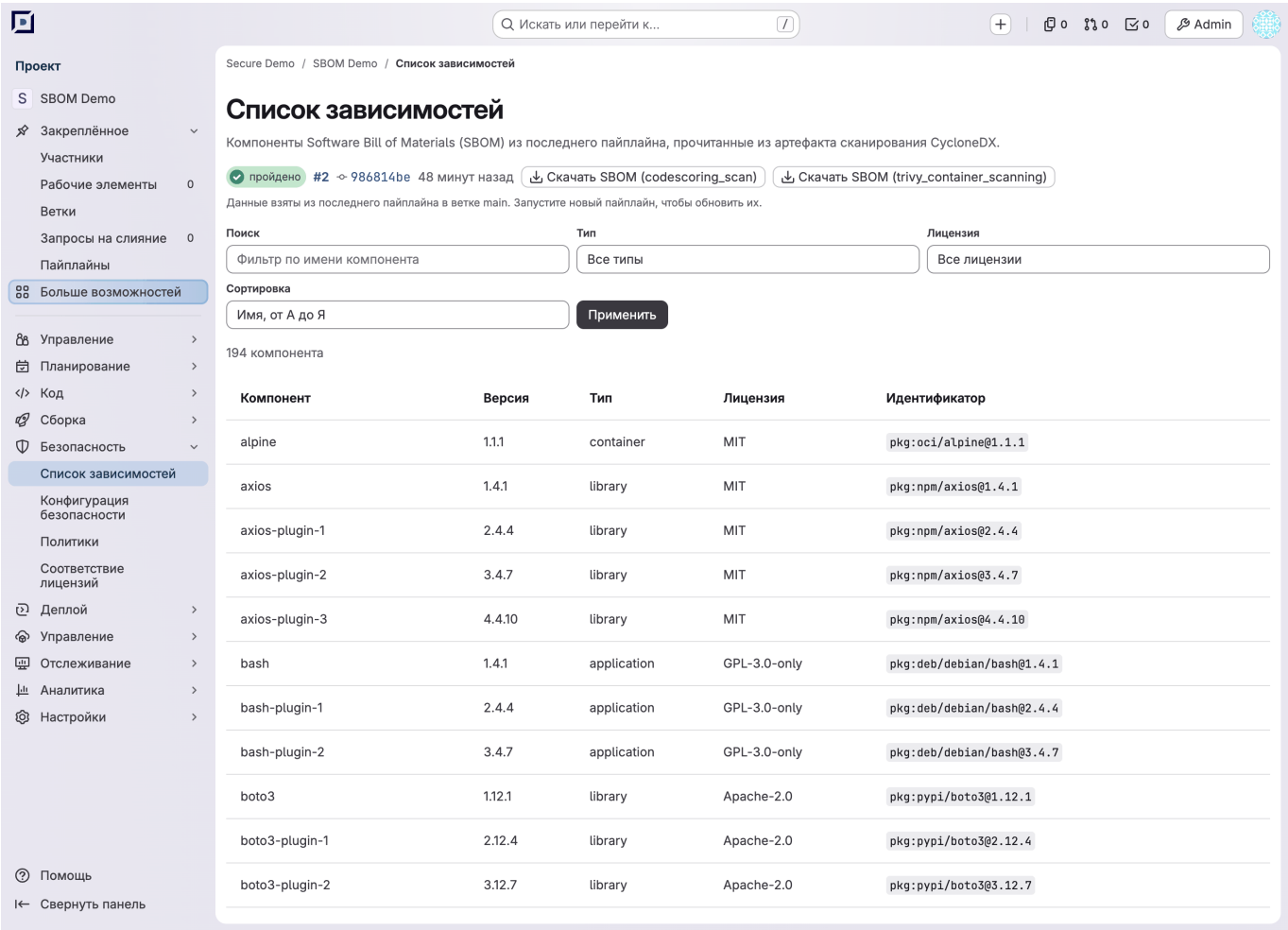
Компоненты из всех таких артефактов объединяются в одну таблицу. Компонент остаётся в ней один раз и опознаётся по идентификатору пакета, а если идентификатора нет — по имени и версии. Поэтому обе страницы показывают всё, что сообщили анализаторы пайплайна вместе.

В базе данных ничего не хранится: каждый запрос заново читает и разбирает артефакты. Чтобы обновить данные, запустите новый пайплайн в ветке по умолчанию.

Над таблицей обе страницы показывают, откуда взят SBOM: статус пайплайна, его номер, коммит, время запуска и кнопку скачивания для каждого артефакта CycloneDX — «Скачать SBOM» или «Скачать SBOM (`<имя задания>`)», если в пайплайне их несколько. Пользователю без доступа к артефактам заданий проекта кнопка не показывается. Строка ниже называет ветку: «Данные взяты из последнего пайплайна в ветке main. Запустите новый пайплайн, чтобы обновить их.»

Для проекта, в ветке по умолчанию которого нет ни одного пайплайна, этого блока нет.

Список зависимостей



Столбцы таблицы:

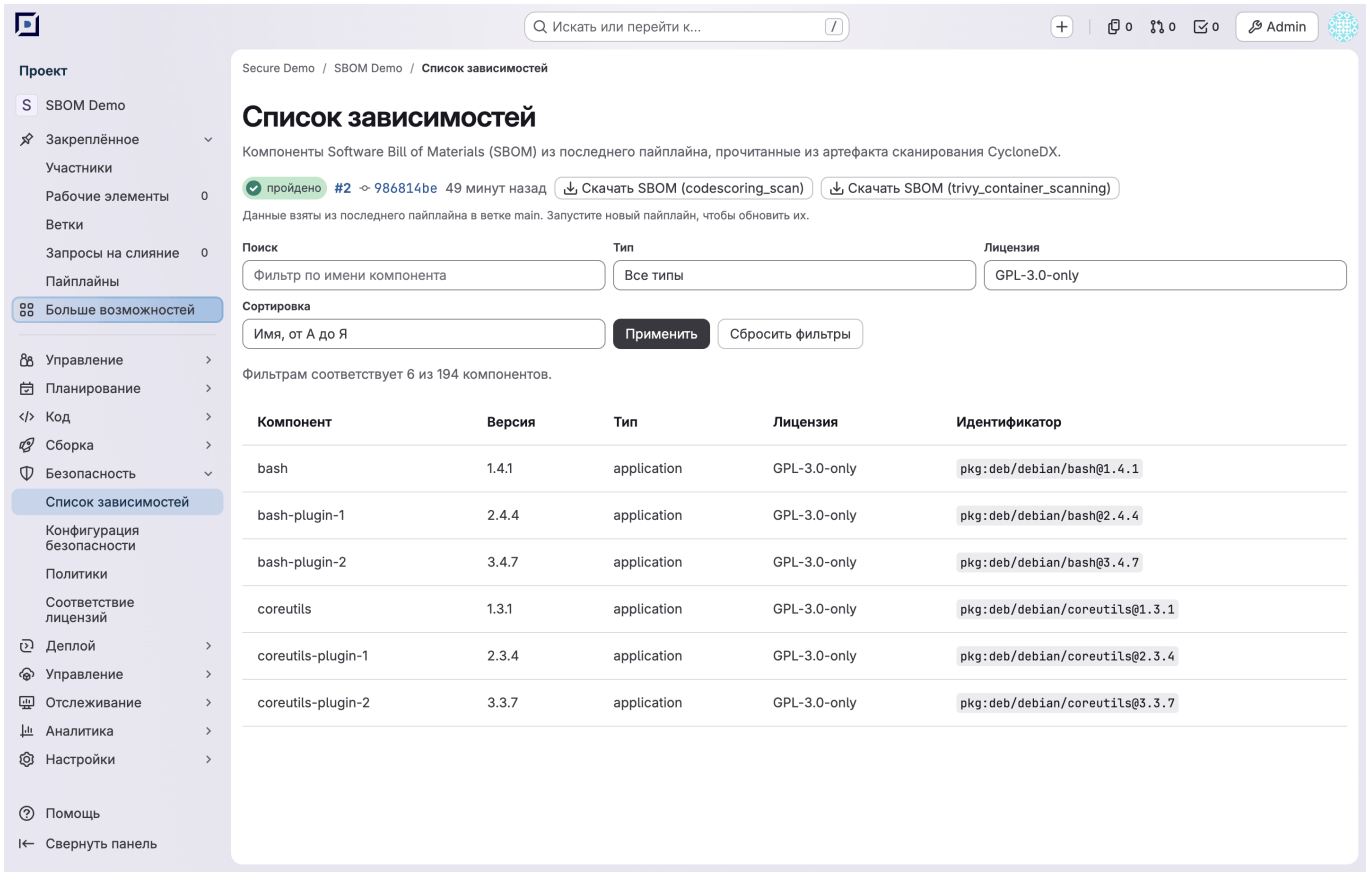
Столбец	Что содержит
«Компонент»	Имя компонента из SBOM
«Версия»	Версия компонента
«Тип»	Тип компонента по CycloneDX: `library`, `application`, `container` и другие
«Лицензия»	Лицензии, объявленные компонентом, через запятую; «Неизвестно», если он не объявил ни одной
«Идентификатор»	Идентификатор пакета в формате purl

На страницу выводится 50 строк. Над таблицей стоит количество компонентов в SBOM, а при заданном фильтре — «Фильтрам соответствует 6 из 194 компонентов.»

Элементы управления над таблицей:

Элемент	Что делает
«Поиск»	Оставляет компоненты, в имени которых есть введённый текст, без учёта регистра
«Тип»	Оставляет компоненты одного типа
«Лицензия»	Оставляет компоненты под одной лицензией; «Неизвестно» оставляет те, что не объявили ни одной
«Сортировка»	Задаёт порядок строк: «Имя, от А до Я» (по умолчанию), «Имя, от Я до А», «Тип», «Лицензия»

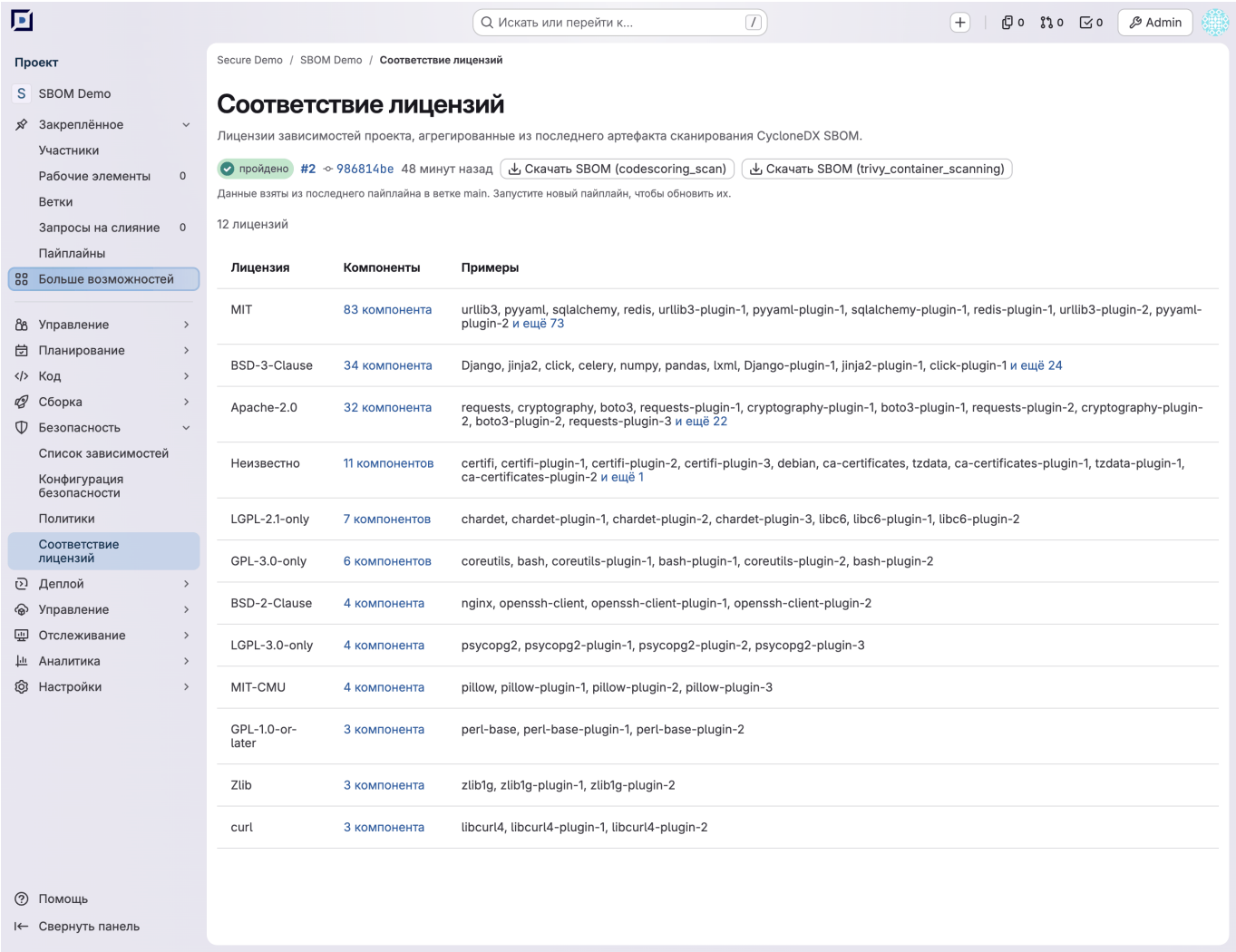
Кнопка «Применить» применяет выбранные значения. Кнопка «Сбросить фильтры» появляется, когда задан хотя бы один фильтр, и возвращает полный список. Списки «Тип» и «Лицензия» строятся по всему SBOM, поэтому сужение одного из них сохраняет варианты другого.



Выбранные значения хранятся в адресе, поэтому отфильтрованный список можно сохранить в закладки или отправить коллеге: `/-/security/dependencies?license=GPL-3.0-only&sort=name_desc`.

Фильтр, которому ничего не соответствует, показывает «Нет зависимостей, соответствующих фильтрам» и подсказку «Измените или сбросьте фильтры, чтобы увидеть больше компонентов.»

Соответствие лицензий



Столбцы таблицы:

Столбец	Что содержит
«Лицензия»	Идентификатор лицензии из SBOM; «Неизвестно» собирает компоненты, которые не объявили лицензию
«Компоненты»	Сколько компонентов под этой лицензией. Число ведёт в список зависимостей, отфильтрованный по ней
«Примеры»	До десяти имён компонентов; ссылка «и ещё N» ведёт в тот же отфильтрованный список

Строки упорядочены по количеству компонентов, от большего к меньшему, а при равном количестве — по алфавиту. Компонент, объявивший несколько лицензий, посчитан под каждой из них, поэтому сумма по строкам больше количества компонентов в SBOM.

«Неизвестно» на обеих страницах содержит одни и те же компоненты, поэтому число в строке и открываемый им отфильтрованный список совпадают.

Состояния отчёта

Что в пайплайне	Что показывают страницы
Артефакта CycloneDX нет	«Пока нет результатов SBOM» и «Запустите пайплайн с включённым сканером CodeScoring, чтобы увидеть здесь зависимости проекта.»
Артефакты есть, но ни один не удалось прочитать	«Не удалось прочитать отчёт SBOM» и «Не удалось прочитать последний артефакт сканирования. Проверьте логи задания CodeScoring и перезапустите пайплайн.»
Хотя бы один артефакт прочитан	Таблицы. Для SBOM без компонентов — «Зависимости не найдены» или «Лицензии не найдены»

Один нечитаемый артефакт оставляет остальные на месте: страницы показывают то, что сообщили прочитанные. Блок с пайплайном и кнопками скачивания показывается во всех трёх состояниях, поэтому нечитаемый артефакт можно скачать и разобрать.

Диагностика

Страниц нет в разделе «Безопасность»

Проверьте условия из раздела «Условия доступности»: настройку проекта, свою роль в проекте, фича-флаг и лицензию установки.

«Пока нет результатов SBOM» после успешного пайплайна

Последний пайплайн ветки по умолчанию не загрузил ни одного артефакта с типом отчёта `cyclonedx`. Проверьте, что:

- пайплайн со сканированием запущен в ветке по умолчанию: пайплайн другой ветки или мерж-реквеста оставляет эти страницы пустыми;
- задание сканирования завершилось и загрузило артефакты — это видно на вкладке «Задания» пайплайна;
- путь в ключе `artifacts:reports:cyclonedx` задания указывает на файл, который задание записало.

«Не удалось прочитать отчёт SBOM»

Скачайте артефакт из блока с пайплайном и откройте его: страницы читают документ CycloneDX в формате JSON. Задание `codescoring_sca` пишет формат, заданный переменной `FE_SCANS_CODESCORING_BOM_FORMAT`, по умолчанию `cyclonedx_v1_6_json`; другой формат загружается в тот же артефакт и здесь не читается.

Интеграции

Вебхуки

Вебхуки — это событийно-ориентированный механизм интеграции с внешними системами. Они позволяют автоматически отправлять HTTP-запросы при возникновении определённых событий в Deckhouse Code:

- Поддержка широкого спектра событий: Push, Merge Request, Issue, Pipeline, Release и др.
- Настройка запросов: выбор метода (POST, PUT), формат JSON, настройка заголовков.
- Безопасность: Secret Token, SSL/TLS, фильтрация по событиям.
- Поддержка на уровне проектов, групп и всей инстанции.
- Интеграция с CI/CD, мониторингом, мессенджерами и системами трекинга.
- Механизм повторных попыток при сбоях соединения (Retry).

> Вебхуки проектов доступны в Deckhouse Code.

Вебхуки групп

Чтобы добавить вебхук на уровне группы, откройте страницу группы и перейдите в «Настройки» → «Вебхуки». Далее выберите интересующие события. Вебхуки групп поддерживают все события из проектов, а также:

- События участников;
- События проектов;
- События подгрупп.

События участников срабатывают при создании, изменении или удалении участников группы или проекта.

> Если у пользователя не указан публичный email, в теле запроса email будет отображаться как `"[УДАЛЕНО]"`.

Создание вебхуков групп

Заголовок запроса:

```
X-Gitlab-Event: Member Hook
```

Пример запроса:

```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:35:51Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[УДАЛЕНО]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": "2025-07-09T00:00:00Z",
  "event_name": "user_add_to_group"
}
```

Изменение вебхуков групп

Заголовок запроса:

```
X-Gitlab-Event: Member Hook
```

Пример запроса:

```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:36:21Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[УДАЛЕНО]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": null,
  "event_name": "user_update_for_group"
}
```

Удаление вебхуков групп

Заголовок запроса:

```
X-Gitlab-Event: Member Hook
```

Пример запроса:


```
{
  "created_at": "2025-07-02T15:23:25Z",
  "updated_at": "2025-07-02T15:36:21Z",
  "group_name": "agriculture",
  "group_path": "agriculture",
  "group_id": 1130,
  "user_username": "reported_user_barabara",
  "user_name": "Estella Gleason",
  "user_email": "[УДАЛЕНО]",
  "user_id": 58,
  "group_access": "Guest",
  "expires_at": null,
  "event_name": "user_remove_from_group"
}
```


События проекта

Срабатывает при создании или удалении проектов в группе и подгруппах.

Создание вебхуков проекта

Заголовок запроса:

```
X-Gitlab-Event: Project Hook
```

Пример запроса:

```
{
  "event_name": "project_create",
  "created_at": "2025-07-02T15:40:09Z",
  "updated_at": "2025-07-02T15:40:09Z",
  "name": "rspec",
  "path": "rspec",
  "path_with_namespace": "flant-development/agriculture/rspec",
  "project_id": 28,
  "project_namespace_id": 1130,
  "owners": [
    {
      "name": "Administrator",
      "email": "[УДАЛЕНО]"
    }
  ],
  "project_visibility": "private"
}
```


Удаление вебхуков проекта

Заголовок запроса:

```
X-Gitlab-Event: Project Hook
```

Пример запроса:


```
{
  "event_name": "project_destroy",
  "created_at": "2025-07-02T15:40:09Z",
  "updated_at": "2025-07-02T15:42:04Z",
  "name": "rspec",
  "path": "rspec",
  "path_with_namespace": "flant-development/agriculture/rspec",
  "project_id": 28,
  "project_namespace_id": 1130,
  "owners": [
    {
      "name": "Administrator",
      "email": "[REDACTED]"
    }
  ],
  "project_visibility": "private"
}
```


События подгрупп

Срабатывает при создании или удалении подгруппы.

Создание вебхуков подгрупп

Заголовок запроса:

```
X-Gitlab-Event: Subgroup Hook
```

Пример запроса:

```
{
  "created_at": "2025-07-02T15:44:02Z",
  "updated_at": "2025-07-02T15:44:02Z",
  "event_name": "subgroup_create",
  "name": "finances",
  "path": "finances",
  "full_path": "flant-development/finances",
  "group_id": 1659,
  "parent_group_id": 1123,
  "parent_name": "Flant development",
  "parent_path": "flant-development",
  "parent_full_path": "flant-development"
}
```

Удаление вебхуков подгрупп

Заголовок запроса:

```
X-Gitlab-Event: Subgroup Hook
```

Пример запроса:

```
{
  "created_at": "2025-07-02T15:44:02Z",
  "updated_at": "2025-07-02T15:44:02Z",
  "event_name": "subgroup_destroy",
  "name": "finances",
  "path": "finances",
  "full_path": "flant-development/finances",
  "group_id": 1659,
  "parent_group_id": 1123,
  "parent_name": "Flant development",
  "parent_path": "flant-development",
  "parent_full_path": "flant-development"
}
```

Расширенный поиск

Поиск в Deckhouse Code помогает быстро находить нужную информацию по проектам, группам или всему инстансу. Результаты сортируются по релевантности и позволяют сразу перейти к исходному объекту.

Расширенный поиск на базе OpenSearch позволяет:

- находить фрагменты кода во всех доступных проектах;
- отслеживать использование устаревших функций и библиотек;
- искать комментарии в задачах и запросах на слияние;
- находить коммиты по сообщению или SHA;
- искать содержимое wiki-страниц.

Использование расширенного поиска

- Администратор должен включить расширенный поиск (OpenSearch).

Чтобы выполнить поиск:

1. В верхней панели выберите «Поиск».
2. Введите поисковый запрос.
3. Нажмите «Enter».

Расширенный поиск также доступен в контексте проекта или группы.

При включённом OpenSearch Deckhouse Code использует его как бэкенд для расширенных областей поиска (задачи, запросы на слияние, код и другие). Параметры REST API, фильтры и формат ответа описаны в разделе «API поиска».

Области поиска

Области описывают тип данных, по которым выполняется поиск.

Базовый поиск

Следующие области доступны в базовом поиске (без OpenSearch):

Область	Глобальный	Группа	Проект
Код	✗	✗	✓
Комментарии	✗	✗	✓
Коммиты	✗	✗	✓
Задачи	✓	✓	✓
Запросы на слияние	✓	✓	✓
Этапы (milestones)	✓	✓	✓
Проекты	✓	✓	✗
Пользователи	✓	✓	✓
Wiki	✗	✗	✓

Расширенный поиск

При включённом OpenSearch доступны следующие области:

Область	Глобальный	Группа	Проект
Код	✓	✓	✓
Комментарии	✓	✓	✓
Коммиты	✓	✓	✓
Задачи	✓	✓	✓
Запросы на слияние	✓	✓	✓
Этапы (milestones)	✓	✓	✓
Проекты	✓	✓	✗
Пользователи	✓	✓	✓
Wiki	✓	✓	✓

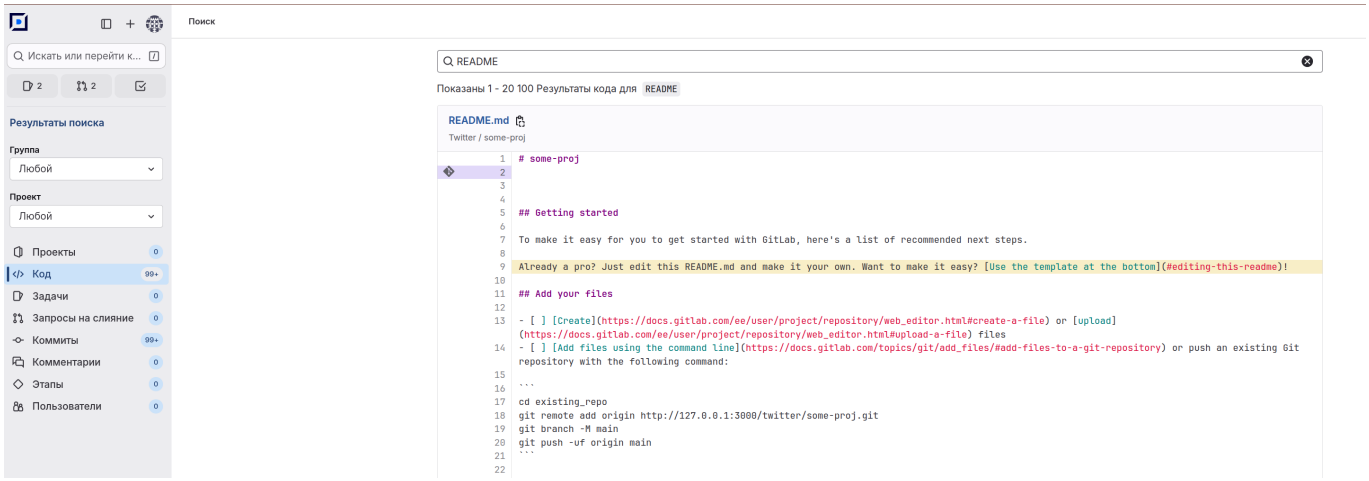
Поиск по коду, коммитам, wiki и комментариям при включённом OpenSearch выполняется через OpenSearch и учитывает **матрицу доступа**. Пользователи видят только те объекты, к которым у них есть права на чтение. Поиск по задачам, запросам на слияние и другим сущностям выполняется через базу данных.

Таблицы выше описывают области поиска в веб-интерфейсе. Набор областей в REST API отличается: поиск по коду, коммитам, комментариям и wiki доступен там только на уровне проекта. Подробнее — в разделе «Области поиска в API».

Использование поиска

Общий порядок работы с поиском в Deckhouse Code:

1. Нажмите «Поиск» в верхней панели.
2. Введите поисковый запрос.
3. Нажмите «Enter» — результаты появятся на странице поиска.
4. Используйте фильтры для уточнения результатов по группе, проекту или типу объекта.



Глобальный поиск

Позволяет искать по всем проектам и группам инстанса.

1. В левом меню выберите «Поиск».
2. Введите запрос и нажмите «Enter».

Поиск в проекте

1. Перейдите в нужный проект.
2. В левом меню выберите «Поиск».
3. Введите запрос и нажмите «Enter».

Поиск по группе

1. Перейдите в нужную группу.
2. В левом меню выберите «Поиск».
3. Введите запрос и нажмите «Enter».

Дополнительные возможности

- Поиск поддерживает автодополнение по проектам, группам и пользователям.
- При включённом расширенном поиске автодополнение также работает по сообщениям коммитов, именам файлов, коду, задачам и запросам на слияние.
- При поиске можно быстро перейти к нужному коммиту по его SHA.

Синтаксис

Расширенный поиск поддерживает расширенный синтаксис запросов: точные и нечёткие совпадения, логические операторы и фильтры.

Синтаксис	Описание	Пример
`"`	Точный поиск	`"gem sidekiq"`
`~`	Нечёткий поиск	`J~ Doe`
`|`	Или	`display | banner`
`+`	И	`display +banner`
`-`	Исключение	`display -banner`
`*`	Частичное совпадение	`bug error 50*`
`\`	Экранирование	`\*md`
`#`	ID задачи (в комментариях)	`#23456`
`!`	ID запроса на слияние (в комментариях)	`!23456`

Поиск по коду

Синтаксис	Описание	Пример
`filename:`	Имя файла	`filename:*spec.rb`
`path:`	Путь в репозитории (полное или частичное совпадение)	`path:spec/workers/`
`extension:`	Расширение файла без точки	`extension:js`
`blob:`	Git object ID	`blob:998707*`

В интерфейсе поиска по коду также доступен фильтр по языку программирования.

Примеры

Запрос	Описание
`rails -filename:gemfile.lock`	Находит `rails` во всех файлах, кроме `gemfile.lock`
`RSpec.describe Resolvers -*builder`	Находит `RSpec.describe Resolvers`, исключая совпадения, начинающиеся с `builder`
`bug | (display +banner)`	Находит `bug` или одновременно `display` и `banner`
`helper -extension:yml -extension:js`	Находит `helper` во всех файлах, кроме `.yml` и `.js`
`helper path:lib/git`	Находит `helper` в файлах с путём `lib/git*` (например, `spec/lib/gitlab`)

Настройки индексации

Настройки проекта

Maintainer проекта может перейти в «Настройки» → «Поиск».

Regex для веток

Если на уровне инстанса включён режим **Разрешить регулярное выражение для веток на уровне проекта**, maintainer может указать regex для дополнительных веток. Ветка по умолчанию индексируется всегда.

Пример regex: `(feature|hotfix)/.*`

> ⚠ Изменение regex запускает полную переиндексацию проекта.

Переиндексация code и wiki

- **Переиндексировать код** — полная переиндексация кода репозитория.
- **Переиндексировать вики** — полная переиндексация wiki (если wiki-репозиторий существует).

Бейдж «Индекс актуален» показывает, завершена ли индексация для текущего состояния репозитория.

Настройки группы

Owner группы может перейти в «Настройки» → «Поиск».

Доступна переиндексация wiki группы: статус индекса и кнопка «Переиндексировать вики».

API поиска

Search REST API позволяет выполнять поиск по инстансу Deckhouse Code, отдельной группе или проекту.

Эндпоинты

Для поиска доступны следующие эндпоинты:

- `GET /api/v4/search` — поиск по инстансу Deckhouse Code;
- `GET /api/v4/groups/:id/search` (или `/api/v4/groups/:id/-/search`) — поиск по группе;
- `GET /api/v4/projects/:id/search` (или `/api/v4/projects/:id/-/search`) — поиск по проекту.

Все эндпоинты требуют аутентификации.

Области поиска в API

Область поиска задаётся обязательным параметром `scope`. Поддерживаемые значения зависят от эндпоинта:

Значение `scope`	Инстанс	Группа	Проект	Бэкенд при включённом OpenSearch
`projects`				PostgreSQL
`users`				PostgreSQL
`snippet_titles`				PostgreSQL
`issues`				OpenSearch (`advanced`)
`work_items`				OpenSearch (`advanced`)
`merge_requests`				OpenSearch (`advanced`)
`milestones`				OpenSearch (`advanced`)
`notes`				OpenSearch (`advanced`)
`wiki_blobs`				OpenSearch (`advanced`)
`commits`				OpenSearch (`advanced`)
`blobs`				OpenSearch (`advanced`)

В заголовке ответа `X-Search-Type` возвращается фактически использованный тип поиска.

Набор областей в API отличается от областей поиска в веб-интерфейсе: значения `blobs`, `commits`, `notes` и `wiki_blobs` поддерживаются только эндпоинтом проекта, тогда как в интерфейсе поиск по этим областям доступен и глобально, и по группе.

Параметры запроса

Общие параметры

Параметр	Тип	Обязательный	Эндпоинты	Примечание
`search`	string	Да	Все	Поисковый запрос
`scope`	string	Да	Все	Область поиска. Доступные значения описаны в таблице выше
`confidential`	boolean	Нет	Все	Передаётся в службу поиска
`include_archived`	boolean	Нет	Инстанс, группа	Параметр недоступен для поиска по проекту
`page` / `per_page`	integer	Нет	Все	Постраничный вывод со смещением (offset)
`ref`	string	Нет	Проект	Ветка или тег для поиска в проекте
`state`	string	Нет	Все	Состояние объекта: `all`, `opened`, `closed`, `merged`
`type`	array[string]	Нет	Все	Фильтр типа work item (фактически применяется при `scope=work_items`)

Дополнительные параметры

Поддержка дополнительных параметров зависит от выбранной области поиска. Если параметр передан с неподдерживаемым значением `scope`, API возвращает ответ `400` с сообщением `<PARAM_NAME> is supported only for <SCOPE_LIST>`.

Параметр	Тип	Применяется к `scope`	Ограничения
`author_username`	string	`merge_requests`	Фильтр по автору
`exclude_forks`	boolean	`work_items`, `issues`	Только в этих `scope`
`fields`	array[string]	`work_items`, `issues`	Поддерживается только значение `title`. Для других значений API возвращает `400`
`label_name`	array[string]	`work_items`, `issues`, `merge_requests`	Поддерживаются значения через запятую
`language`	array[string]	`blobs`	Поддерживаются значения через запятую
`not_author_username`	string	`merge_requests`	Исключение по автору
`not_source_branch`	string	`merge_requests`	Исключающий фильтр
`not_target_branch`	string	`merge_requests`	Исключающий фильтр
`num_context_lines`	integer	`blobs`	Поддерживается диапазон `0..20`
`source_branch`	string	`merge_requests`	Точный фильтр по исходной ветке
`target_branch`	string	`merge_requests`	Точный фильтр по целевой ветке

Заголовки ответа

API может возвращать следующие заголовки:

- `X-Search-Type` — фактически использованный тип поиска;
- `X-Search-Aggregations` — присутствует только когда OpenSearch включён и для выбранной области поиска доступны агрегаты.

Состав агрегатов зависит от значения `scope`:

Значение `scope`	Агрегаты
`blobs`	`language`
`work_items`, `issues`	`work_item_type_ids`, `labels`
`merge_requests`	`labels`

Тело ответа

Эндпоинт возвращает JSON-массив объектов, тип которых зависит от выбранной области поиска:

Значение `scope`	Тип объекта
`issues`	`IssueBasic`
`work_items`	`WorkItem`
`merge_requests`	`MergeRequestBasic`
`milestones`	`Milestone`
`notes`	`Note`
`commits`	`Commit`
`blobs`	`Blob`
`wiki_blobs`	`Blob`
`projects`	`BasicProjectDetails`
`users`	`UserBasic`
`snippet_titles`	`Snippet`

Примеры запросов

Поиск по инстансу: issues/work items с метками и полями

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/search?
scope=issues&search=deploy&fields=title&label_name=team%3Aplatform&exclude_forks=true"
```

Поиск по группе: запросы на слияние с фильтрами

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/groups/my-group/-/search?
scope=merge_requests&search=release&source_branch=release%2F1.2&not_author_username=bo
t"
```

Поиск по проекту: blobs с контекстными строками

```
curl --request GET \
  --header "PRIVATE-TOKEN: <ACCESS_TOKEN>" \
  --url "https://code.example.com/api/v4/projects/my-group%2Fmy-project/-/search?
scope=blobs&search=deploy&num_context_lines=5&language=Ruby"
```