Deckhouse

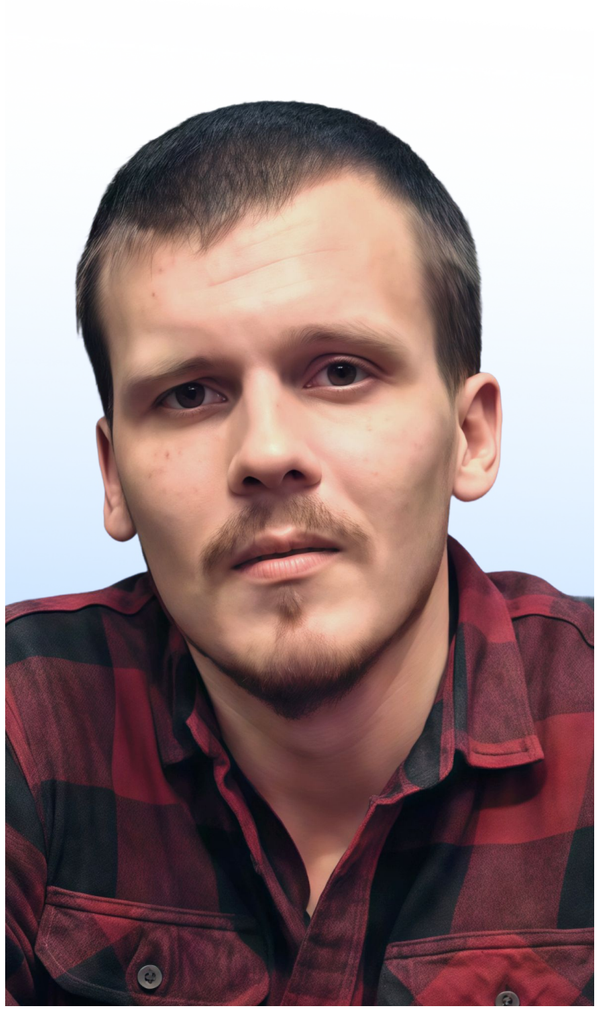
Единый контур безопасной разработки с Deckhouse: Code + Stronghold

Deckhouse

Спикеры

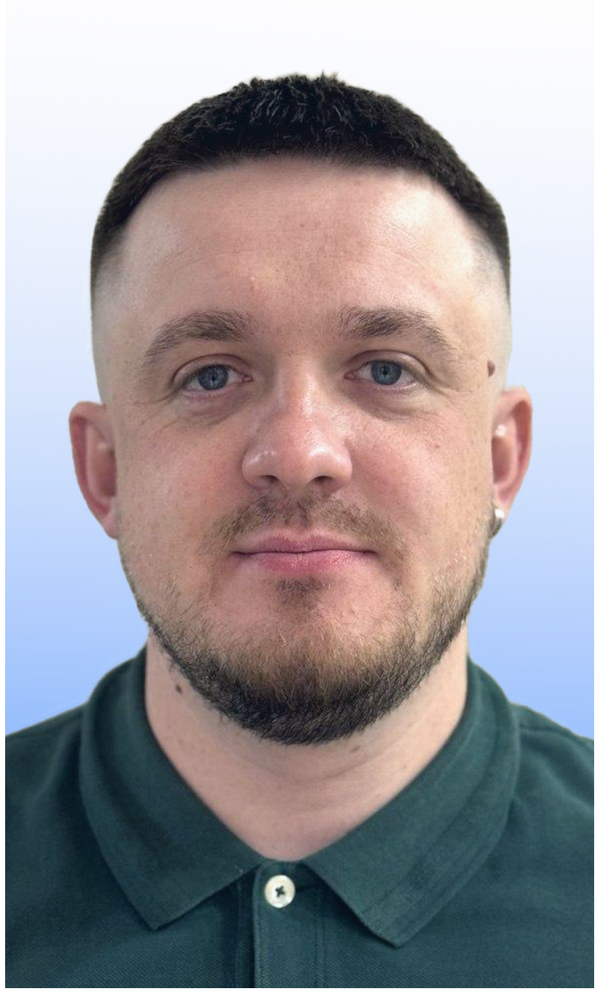
Евгений Целиков

Технический менеджер продукта Deckhouse Code



Анатолий Чуриков

Технический менеджер продукта Deckhouse Stronghold



О чём поговорим

01 О «Фланте» и экосистеме Deckhouse

02 Enterprise CI/CD — какой он?

03 Какие способы хранения секретов используют в CI/CD сегодня

04 Почему секреты нужно хранить отдельно в Vault-е
и как работает интеграция «Code + Stronghold»

05 Как безопасно управлять конфигурациями Stronghold с помощью gitops-plugin

Deckhouse

ФЛАНТ

Синергия опыта вендора, интегратора, сервисной и консалтинговой компании

Deckhouse

Deckhouse — продуктовое подразделение, разработчик продуктов для построения надёжной enterprise-инфраструктуры

DaaS

DaaS — комплексное DevOps-сопровождение инфраструктуры в режиме 24/7 силами выделенной DevOps-команды

Express 42

«Экспресс 42» — DevOps-консалтинг. От анализа узких мест в ИТ-процессах до создания роадмапа изменения ИТ для реализации цифровой трансформации

О вендоре Deckhouse

Deckhouse

17+
лет опыта
в Open Source

С 2017
года используем
Kubernetes в production

№1
контрибьютор в проекты
CNCF из России

550+
сотрудников

>260
компаний-пользователей

В топе
вендоров ИТ-решений для банков*
и промышленности**

Реестр
российского ПО

Лицензии и сертификаты
ФСТЭК России

АРПП «Отечественный
софт»

* Рейтинг «Крупнейшие ИТ-вендоры в банках», TAdviser, 2024
** Рейтинг «Крупнейшие ИТ-вендоры в промышленности», TAdviser, 2024

Три слоя для цифрового конвейера

Инфраструктура

Вертикально интегрированный набор инструментов для **полного цикла** производства цифровых продуктов.

Deckhouse Virtualization Platform (DVP)

Частное облако с простым и понятным интерфейсом для декларативного создания виртуальных машин и работы с ними и их ресурсами.

Deckhouse Kubernetes Platform (DKP)

Первая российская платформа для создания идентичных кластеров Kubernetes и управления ими в любой инфраструктуре.

Три слоя для цифрового конвейера

Трансформация

Вертикально интегрированный набор инструментов для **полного цикла** производства цифровых продуктов.

Deckhouse Delivery Kit

Безопасная сборка, доставка контейнеризованного ПО и управление его жизненным циклом.

Deckhouse Stronghold

Безопасное управление жизненным циклом секретов.

Deckhouse Commander

Централизованное управление жизненным циклом парка кластеров DKP из одного интерфейса.

Deckhouse Observability Platform

Платформа централизованного мониторинга и журналирования инфраструктуры и приложений.

Три слоя для цифрового конвейера

Доставка

Вертикально интегрированный набор инструментов для **полного цикла** производства цифровых продуктов.

Deckhouse Code

Централизованное управление исходным кодом.

Deckhouse Development Platform

Комплексное решение для управления всеми этапами разработки через единое окно.

Deckhouse Marketplace

Каталог проверенного ПО от российских производителей с гарантированной совместимостью.

Экосистема продуктов Deckhouse

- Все продукты экосистемы — **в реестре российского ПО**
- Единый канал **технической поддержки** по всем продуктам
- Высокая **доступность и отказоустойчивость** всех компонентов «из коробки»
- Глубокая **интеграция** продуктов экосистемы между собой
- **Централизация** управления, мониторинга и контроля над всей экосистемой через единый UI
- Сквозное применение лучших практик **DevSecOps** на всех этапах процесса непрерывной поставки ценности

* Продукт находится в активной фазе разработки, готовится выход релизной версии

Deckhouse Professional Services — консалтинг и внедрение

Эксперты Deckhouse
берут на себя полный цикл работ —
от проектирования архитектуры
до построения готовой платформы
под ваши задачи

- Быстрый и предсказуемый старт
- Целостный подход
- Передача знаний
- Адаптация работ под клиента

На 60 % быстрее
достигается time-to-value

deckhouse.ru/services

Оставьте заявку и получите расчёт
стоимости проекта внедрения

Что такое Deckhouse Code

Deckhouse

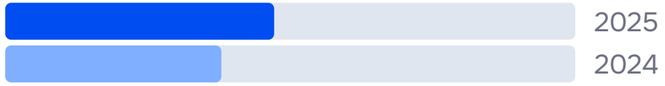
Бóльшая часть рынка использует GitLab

- Больше 70 % рынка использует GitLab для хранения кода и CI/CD
- Тренд продолжается — доля растёт год к году
- Команды знают инструмент, процессы настроены, CI/CD работает

77,3 %

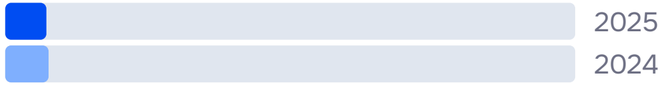
Используют GitLab в том или ином виде

47,2 % 38,0 %



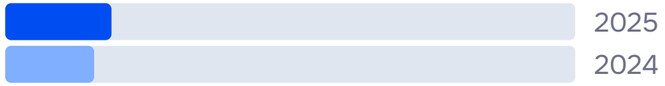
GitLab CE на своих серверах

7,1 % 7,4 %



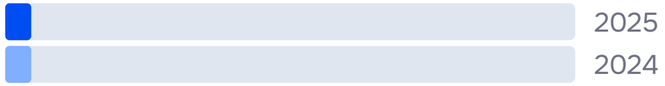
GitLab бесплатный SaaS

18,6 % 15,6 %



GitLab EE на своих серверах

4,4 % 4,4 %



GitLab платный SaaS

По данным отчёта State of DevOps

Deckhouse Code

Самостоятельный продукт на базе GitLab CE

Мы взяли проверенную основу, на которой работают миллионы разработчиков, и построили поверх неё enterprise-платформу CI/CD.

GitLab CE — фундамент, Deckhouse Code — готовый продукт с поддержкой, автоматическими обновлениями, высокой доступностью и enterprise-функционалом

Доступные редакции

	Enterprise Edition	Standard Edition
Экспертиза вендора по обслуживанию продукта	✓	✓
Глубокая интеграция всех компонентов платформы между собой	✓	✓
Максимальная автоматизация задач эксплуатации и администрирования	✓	✓
Функционал организации процесса разработки: правила ревью MR, CODEOWNERS, гибкие правила отправки изменений	✓	✓
Расширенный функционал по организации доступа и контролю за изменениями в системе (события аудита, синхронизация пользователей и групп из LDAP)	✓	✓
Групповые вебхуки и wiki	✓	✓
Функционал зеркалирования репозиториев в режимах pull и push	✓	✓
Запуск в отказоустойчивом режиме	✓	✗
Встроенные инструменты безопасности и интеграция с российскими сканерами уязвимостей	✓	✗

Дорожная карта

01 **Merge trains**

Безопасная очередь изменений, которая проверяет совместимость запросов на слияние перед последовательным слиянием в основную ветку

02 **Гибкая модель доступа**

Расширенные возможности настройки ролей и прав для разных сценариев использования

03 **Встроенные инструменты безопасности**

Встроенные сканеры безопасности, а также механизмы хранения, обработки и визуализации результатов (Semgrep, Trivy, Gitleaks, vulnerability dashboard)

04 **AI для ускорения рутинных операций**

Помощь в ревью запросов на слияние, оптимизация пайплайнов и разбор ошибок сборки

05 **Автоматизация CI/CD-инфраструктуры**

Поставка собственного оператора для Code Runner, упрощающая управление сборочными агентами в Kubernetes

06 **Гибкость поставки**

Поставка all-in-one без внешних зависимостей

07 **Интеграция с российскими сканерами уязвимостей**

Интеграция с внешними отечественными сканерами безопасности, а также ASOC/ASPM-решениями (Codescoring, Positive Technologies Application Inspector, DefectDojo)

Enterprise CI/CD — какой он?

Deckhouse

Что должно быть в enterprise CI/CD

01 Контроль изменений

02 Автоматические проверки как обязательное условие

03 Безопасность на каждом шаге

04 Единый источник правды о доступах

05 Прозрачность

Этап 1: push в ветку

Что попадает в репозиторий?

01
Push Rules
Push в ветку
Проверка автора, формата и подписи коммита
- Коммиты провалидированы

02
CODEOWNERS + Approval Rules
Мерж-реквест
Обязательные апрувы и автоназначение ревьюеров
- Все апрувы получены

03
CI/CD
Пайплайн
Сборка, тесты, quality gates. Секреты из Vault
- Пайплайн успешно пройден

04
Мерж
Мерж в main
Только после всех апрувов и успешного пайплайна
- Все проверки пройдены

05
Registry + Packages
Артефакты
Образы в container registry, пакеты в package registry. Готовы к деплою
- Артефакт опубликован

06
Deploy
Деплой
Проверенный образ в целевом окружении
- Деплой прошёл успешно

На уровне всей платформы

Аудит событий
Каждое действие зафиксировано: кто, когда, что изменил

LDAP/SAML
Доступы синхронизируются из корпоративного каталога автоматически

Дашборд доступов
Все токены, SSH- и GPG-ключи — на виду в одном месте

Функции Deckhouse Code на этом этапе

Push Rules
Проверка автора, формата коммита, подписи, запрет нежелательных файлов

Правила отправки изменений (Push Rules)

Что это:

Правила, ограничивающие пуш коммитов в репозитории. Реализованы на уровне системы, группы и проектов с опциональной возможностью переопределения

Зачем:

Дисциплина и безопасность разработки, а также предотвращение ошибок в коде

Сценарии использования:

- Соблюдение внутренних стандартов разработки
- Гарантия корпоративной ответственности
- Повышение качества сборок

Смотрим 👀

Этап 2: мерж-реквест

Кто должен подтвердить изменение?

01
Push Rules
Push в ветку
Проверка автора, формата и подписи коммита
Коммиты провалидированы

02
CODEOWNERS + Approval Rules
Мерж-реквест
Обязательные апрувы и автоназначение ревьюеров
Все апрувы получены

03
CI/CD
Пайплайн
Сборка, тесты, quality gates. Секреты из Vault
Пайплайн успешно пройден

04
Мерж
Мерж в main
Только после всех апрувов и успешного пайплайна
Все проверки пройдены

05
Registry + Packages
Артефакты
Образы в container registry, пакеты в package registry. Готовы к деплою
Артефакт опубликован

06
Deploy
Деплой
Проверенный образ в целевом окружении
Деплой прошёл успешно

На уровне всей платформы

Аудит событий
Каждое действие зафиксировано: кто, когда, что изменил

LDAP/SAML
Доступы синхронизируются из корпоративного каталога автоматически

Дашборд доступов
Все токены, SSH- и GPG-ключи — на виду в одном месте

Функции Deckhouse Code на этом этапе

Approval Rules
Обязательные апрувы от нужных людей и команд

CODEOWNERS
Автоматическое назначение ревьюеров по изменённым файлам

Безопасник

— Мы хотим контролировать изменения кодовой базы!

Техлид

— Без моего ревью в мастер ни ногой!

СТО

— Ну или моего!

Внешний аудитор

— Хочу всё и сразу!

Тестировщик

— Пишите СРАЗУ тестируемый код! Не умеете — мы будем ревьюить!

Ревьюер

— А я это не могу ревьюить, этот кусок только Вася знает!

ТехПис

— Вы почему доку без меня меняете?! Как мне за неё потом отвечать??

Правила ревью запросов на слияние

Что это:

Механизм настройки правил
обязательного ревью
перед слиянием кода

Зачем:

Гарантирует соблюдение процессов
качества и безопасности

Сценарии использования:

- Двойное ревью с обязательным QA-подтверждением
- Обязательное ИБ-ревью для security-проектов
- Единые политики ревью для всех команд

Функциональность владения кодом (CODEOWNERS)

Что это:

Автоматическое назначение ревьюеров
в зависимости от изменяемых файлов
или директорий

Зачем:

Ускоряет код-ревью и гарантирует,
что за каждый модуль отвечают эксперты

Сценарии использования:

- Автоназначение ревьюеров
  по типу файлов
- Компонентное владение кодом
- Обязательный апрув
  владельца сервиса

Смотрим 👀

Этапы 3–4: пайплайн и мерж

Что проверяется перед попаданием в main?

01
Push Rules
Push в ветку
Проверка автора, формата и подписи коммита
Коммиты провалидированы

02
CODEOWNERS + Approval Rules
Мерж-реквест
Обязательные апрувы и автоназначение ревьюеров
Все апрувы получены

03
CI/CD
Пайплайн
Сборка, тесты, quality gates. Секреты из Vault
Пайплайн успешно пройден

04
Мерж
Мерж в main
Только после всех апрувов и успешного пайплайна
Все проверки пройдены

05
Registry + Packages
Артефакты
Образы в container registry, пакеты в package registry. Готовы к деплою
Артефакт опубликован

06
Deploy
Деплой
Проверенный образ в целевом окружении
Деплой прошёл успешно

На уровне всей платформы

Аудит событий
Каждое действие зафиксировано: кто, когда, что изменил

LDAP/SAML
Доступы синхронизируются из корпоративного каталога автоматически

Дашборд доступов
Все токены, SSH- и GPG-ключи — на виду в одном месте

Функции Deckhouse Code на этом этапе

CI/CD
Сборка, тесты, анализ качества, quality gates

Deckhouse Stronghold (Vault)
Секреты динамически подгружаются в пайплайн, не хранятся в коде

Блокировка мержа
Мерж в основную ветку невозможен без успешного пайплайна — код не попадает в main, пока сборка и тесты не пройдены

Какие способы хранения секретов используют в CI/CD сегодня

Принципы работы с кодом
и этапы его доставки

Deckhouse

Способы хранения секретов

01 Хранение в репозитории

- Секреты хранятся внутри репозитория вместе с кодом
- Секреты могут быть зашифрованы или обёрнуты в Base64
- Секреты подкладываются в переменные окружения или конфигурационный файл при запуске или деплое приложения

\+ Очень просто

\+ Изменения видны в коммитах

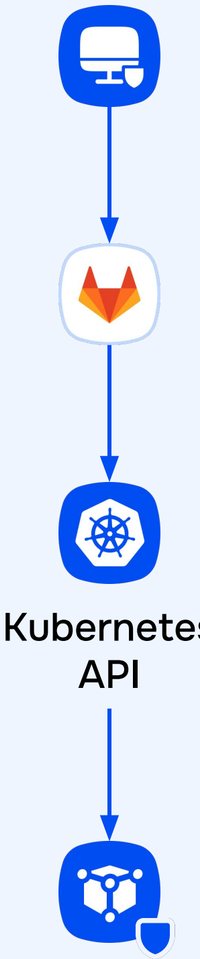
\− Небезопасно, так как секреты передаются в открытом виде

\− Доступ к секретам есть у всех, кто имеет доступ в репозиторий

- Секреты хранятся внутри репозитория с кодом в виде CI/CD-переменных
- Секреты хранятся в зашифрованном виде
- Секреты могут маскироваться. Их видимость может фильтроваться по веткам и тегам

\+ Очень просто

\− Доступ есть у всех, кто имеет доступ уровня maintainer в репозиторий, и у администраторов хранилища кода

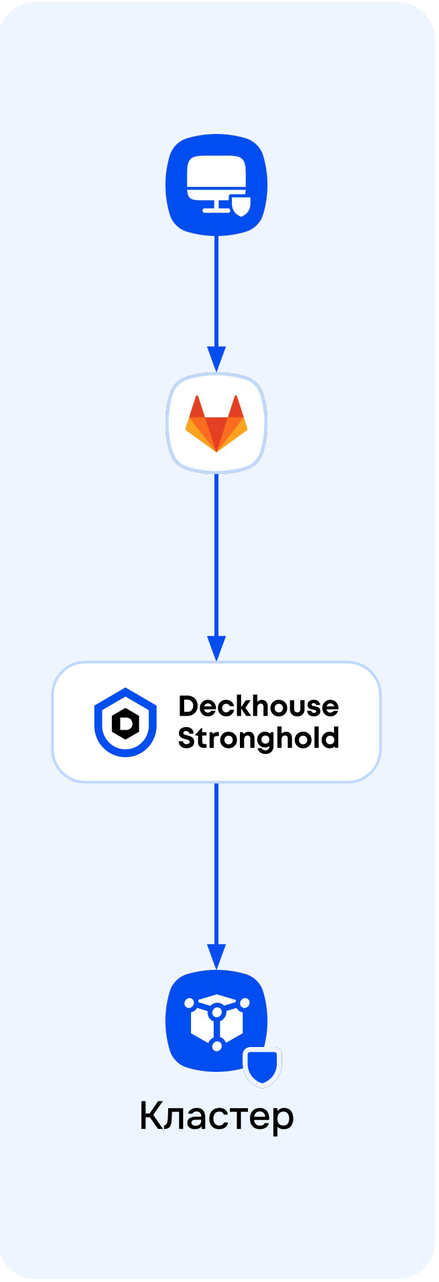


Способы хранения секретов

02 Использование хранилища секретов

- Секреты хранятся внутри хранилища секретов
- Хранятся в зашифрованном виде
- Возможна доставка несколькими безопасными способами

+ Все секреты хранятся в одном месте
+ Управление жизненным циклом секретов и доступом к ним на единых принципах
+ Доставка секретов безопасна на всех этапах и не требует разработки «особых» механизмов



Как хранят секреты в GitLab CI/CD

GitLab предоставляет три уровня хранения и несколько атрибутов для каждой переменной

Уровень инстанса

Доступны во всех проектах. Управляет только администратор. Используется для глобальных токенов и общих сервисов

Уровень группы

Доступны во всех проектах группы и подгрупп. Наследуются автоматически — удобно, но создаёт неявную связь

Уровень проекта

Доступны только в конкретном проекте. Управляется maintainer'ом

Проблема ролевой модели GitLab

GitLab CI/CD Variables не разделяют
«кто может запускать пайплайн»
и «кто должен знать секрет»

Что может maintainer

Добавить Job с printenv → все секреты в логах

Другие векторы риска

- Секреты доступны через GitLab API при наличии токена
- Нет ротации: прописали однажды — живут годами
- Нет audit log: непонятно, кто и когда менял/читал
- Owner группы видит переменные всех дочерних проектов

Проблема ротации

Один секрет используется в 30 пайплайнах →
при ротации нужно обновить его в каждом проекте
вручную. Про часть забудут

Проблемы интеграции CE GitLab + Vault

01 Нет нативного secrets: keyword — только в GitLab Premium/Ultimate. В CE весь механизм — самописный код

02 Нет стандарта внутри компании: команда А пишет Bash, команда Б — Python, команда В тащит Vault Agent

03 В ряде случаев приходится настраивать политики хранения секретов для «распечатывания» секретов

04 Каждый новый проект начинает с нуля — нет переиспользуемого стандарта и документации «из коробки»

05 В случае обновления GitLab или Vault велик риск того, что все самописные интеграции сломаются

Интеграция с хранилищами секретов

Что это:

Поддержка подключения к любым Vault-совместимым системам управления секретами, включая Deckhouse Stronghold

Зачем:

Секреты динамически подгружаются в пайплайны для безопасного хранения и централизованного управления ключами, паролями и токенами. Это исключает хранение секретов в коде и повышает соответствие требованиям безопасности и аудита

Сценарии использования:

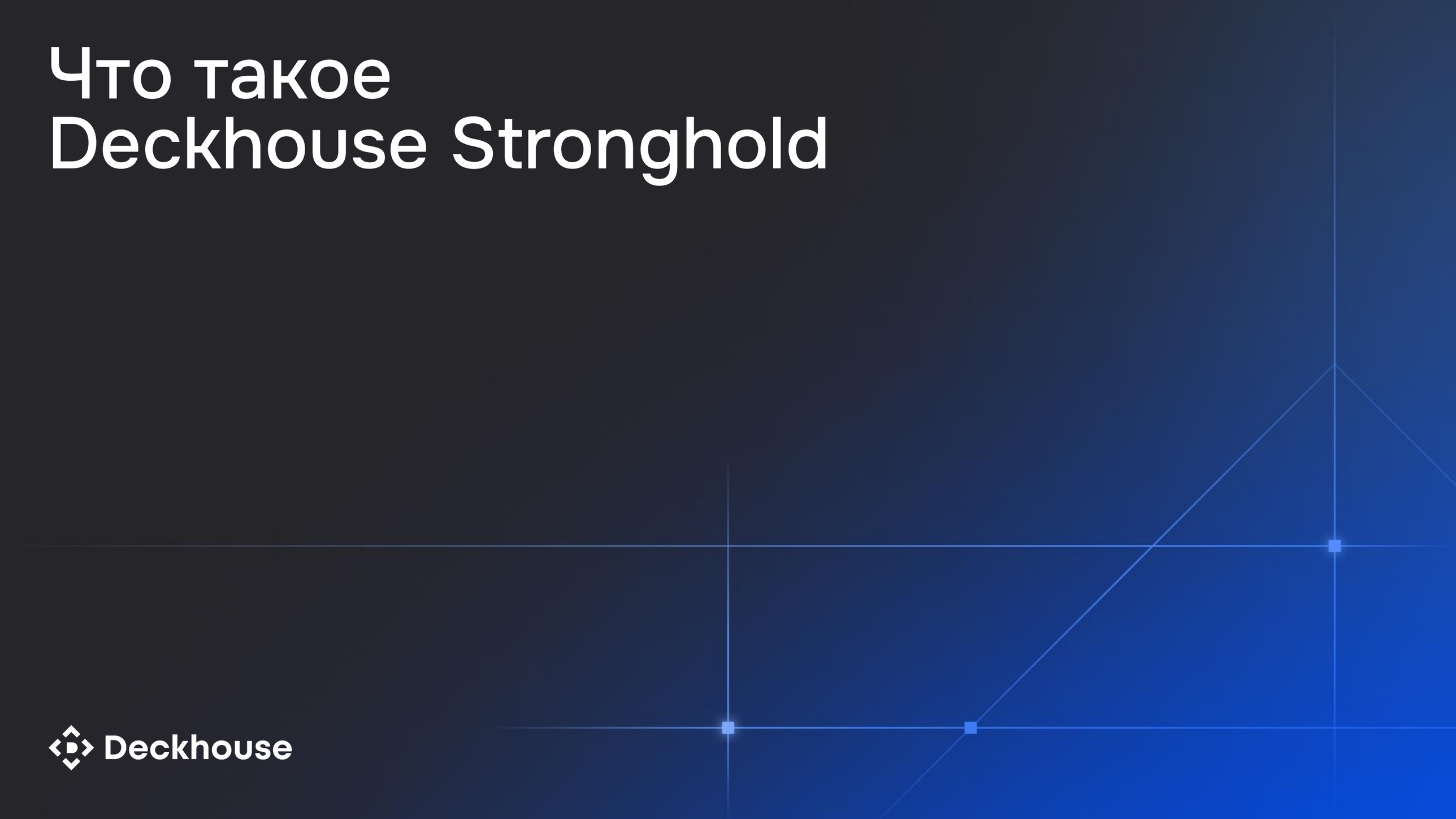
- **Безопасный деплой в продакшен** — пайплайн автоматически получает временные токены доступа и удаляет их по завершении
- **Единый контроль для всех команд** — секреты управляются централизованно, что исключает дублирование и «забытые» ключи
- **Независимость от конкретного вендора** — можно использовать Stronghold или любое другое Vault-совместимое решение, сохраняя независимость от конкретного вендора

Сравнение подходов

	CI/CD variables	CE + Bash/Vault	GitLab EE + Vault	Code + Stronghold
Нативный secrets: keyword	✗	✗	✓	✓
Short-lived-токены	✗	✓	✓	✓
Audit log	✗	✓	✓	✓
Единая точка ротации секрета	✗	✓	✓	✓
Недоступность prod-секретов для разработчика	✗	✓	✓	✓
Отсутствие лицензионных рисков	✗	✗	✗	✓
Один вендор / поддержка	✗	✗	✗	✓
CODEOWNERS: защита .gitlab-ci.yml	✗	✗	✓	✓
Экосистема платформы	✗	✗	✓	✓
Реестр отечественного ПО	✗	✗	✗	✓

Смотрим 👀

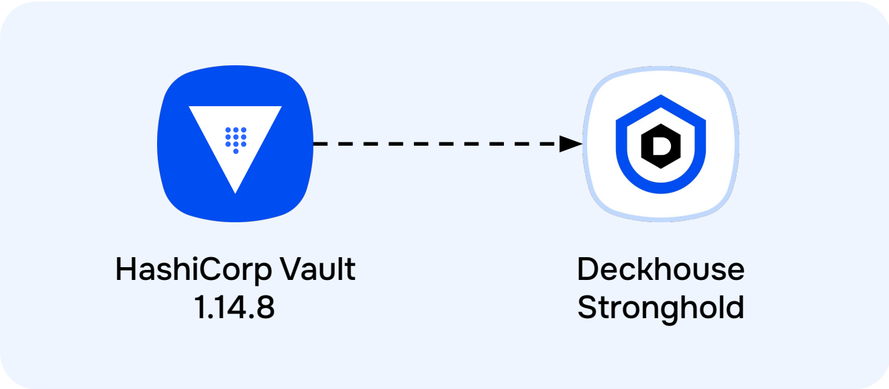
Что такое Deckhouse Stronghold



Deckhouse

Deckhouse Stronghold

Решение для централизованного управления жизненным циклом секретов. Защищает пароли, ключи API, сертификаты, SSH-ключи, токены и другие конфиденциальные данные от утечек, а также обеспечивает безопасную доставку секретов в приложения.



Мы **не копируем**
HashiCorp Vault,
мы **переосмысливаем**
хранилище секретов
и **создаём стандарт**
для российского рынка

Основные возможности продукта

- Хранение секретов как key-value
- Доступ к хранилищу через API и UI
- Хранение данных в зашифрованном виде
- Разграничение доступа с помощью гибкого набора политик
- Отказоустойчивость «из коробки».
- Совместимость с API HashiCorp Vault
- Полная наблюдаемость жизненного цикла секретов
- Интеграция с внутренними и внешними сервисами

Возможные сценарии использования

Для кого полезно

- Команды DevOps и DevSecOps
- Сотрудники подразделений ИБ
- Разработчики

01 Управление жизненным циклом секретов

02 Аутентификация и авторизация

03 Управление инфраструктурой открытых ключей (PKI)

04 Секреты для CI/CD

05 Динамические секреты для баз данных

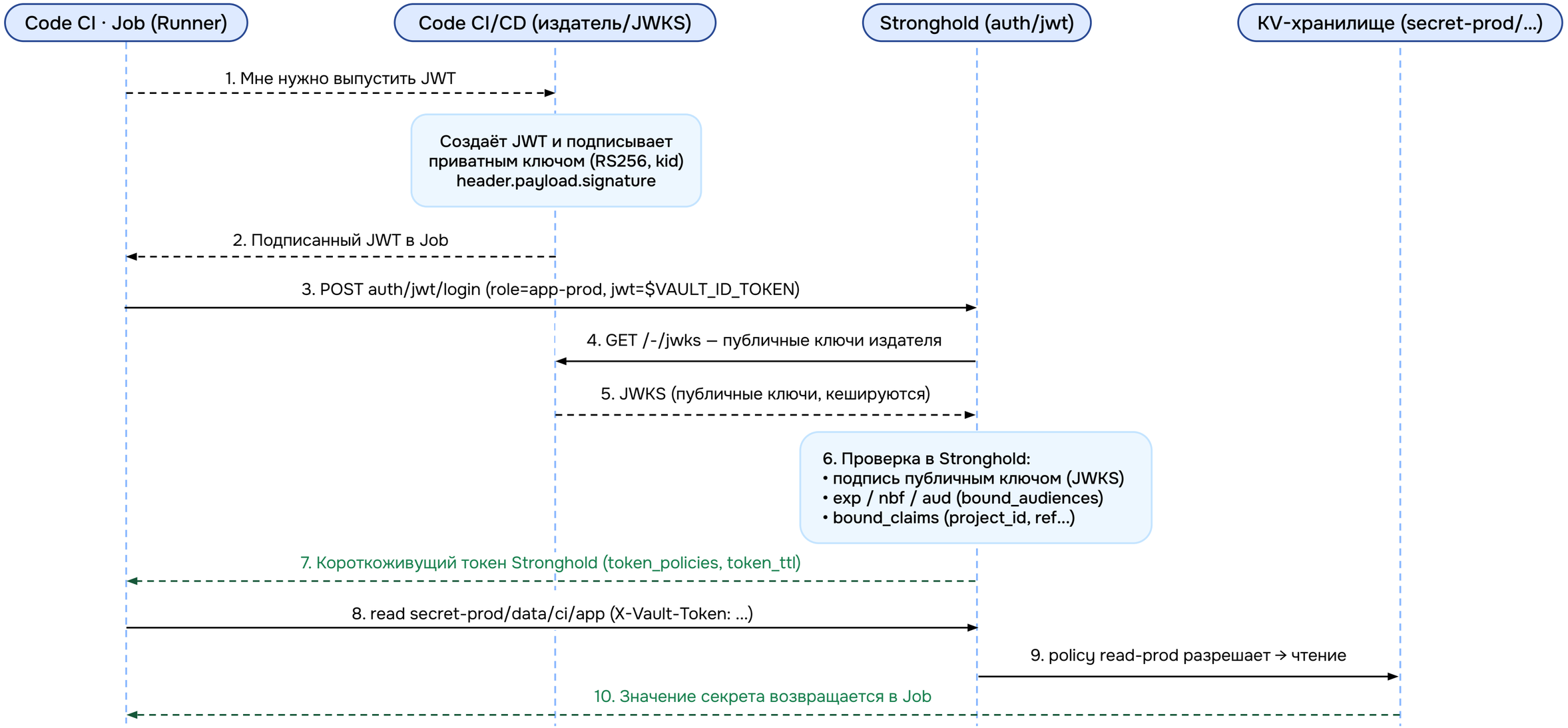
06 Подпись SSH-ключей

Почему секреты нужно хранить отдельно в Vault-е и как работает интеграция «Code + Stronghold»

Как работает интеграция
и как её правильно готовить

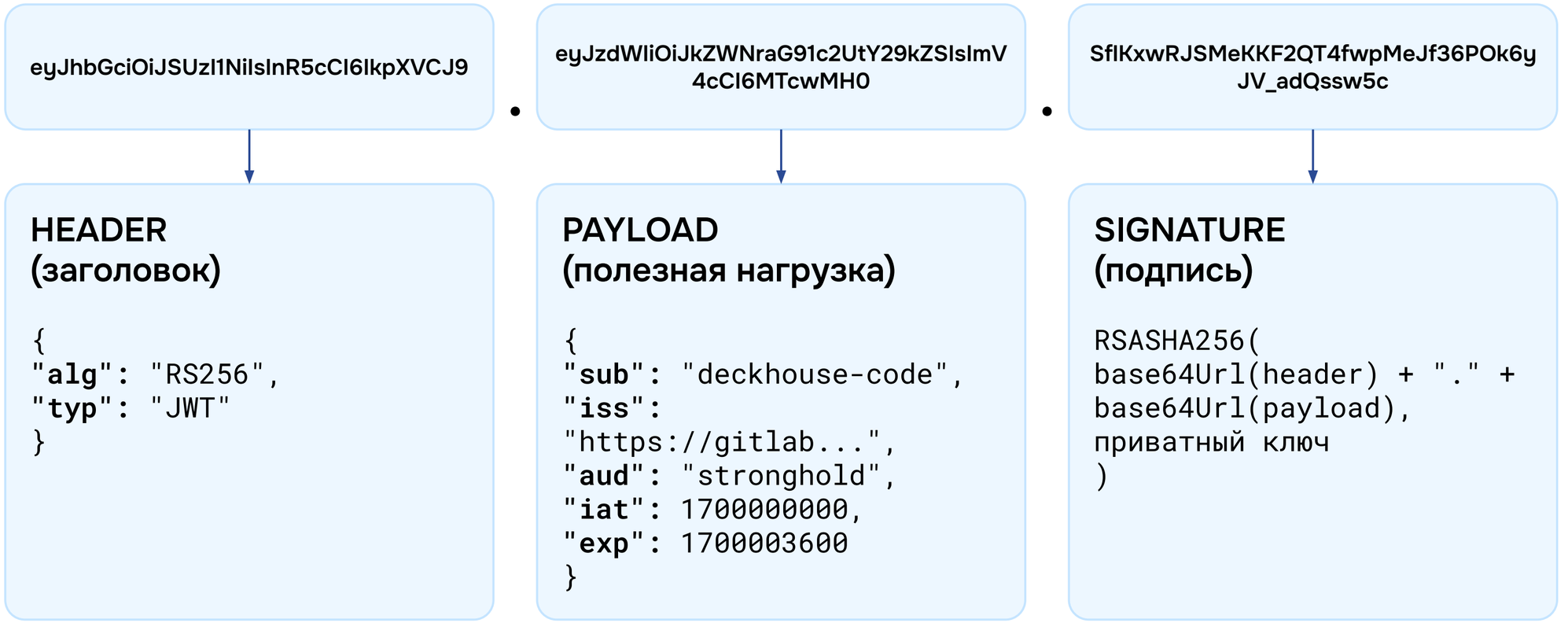
Deckhouse

Как работает аутентификация через JWT



Препарируем JWT

Закодированный токен (Base64URL):



Тип токена и алгоритм подписи.
alg — как подписан токен (RS256, HS256...),
typ — всегда JWT

Claims — данные о субъекте и токене.
sub — кто,
iss — кто выдал,
aud — для кого,
iat — время выпуска,
exp — срок действия,

Гарантирует целостность и подлинность.
Подписывается приватным ключом издателя, проверяется публичным ключом (JWKS).
Изменили payload → подпись не сойдётся

JWT Claims — что можно использовать в bound_claims

`project_id`	ID проекта — самый точный способ ограничить доступ
`project_path`	Путь до проекта, например root/my-project
`ref`	Git-ветка или тег, инициировавший запуск пайплайна
`ref_protected`	Только защищённые ветки могут получить секрет
`environment`	Название окружения (production, staging)
`namespace_path`	Путь группы/пространства, например groups/dev

Почему JWT безопаснее ?

Без «секретов для секретов»

Пайплайн не хранит пароль от хранилища — токен выдаётся под конкретную задачу

Доверие по подписи

JWT подписан издателем (GitLab) и проверяется по JWKS — без общего пароля

Контекст внутри запроса

Bound_claims используется для авторизации доступа

Короткий срок жизни

Токен и доступ живут минуты — утёкший быстро устаревает

Автоматическая ротация

Новый токен на каждый запуск, без ручного обновления секретов

Аудит и отзыв

Журнал входов + отзыв сменой роли/политики, без правки пайплайнов

Конфигурация CI — переменные окружения

`VAULT_SERVER_URL`	Обязательно	Stronghold-серверы, например https://stronghold.example.com
`VAULT_AUTH_ROLE`		Название роли в Stronghold. По умолчанию — роль из конфигурации JWT auth
`VAULT_AUTH_PATH`		Путь до метода аутентификации. По умолчанию — jwt
`VAULT_NAMESPACE`		Неймспейс Stronghold при многоуровневой иерархии

Настраиваем Stronghold

01

Включить JWT-аутентификацию

Vault принимает ID-токены через метод JWT. Указываются OIDC Discovery URL и issuer — это URL вашего Deckhouse Code

```
stronghold auth enable jwt
stronghold write auth/jwt/config \
oidc_discovery_url="https://code.
example.com" \
bound_issuer="https://code.
example.com"
```

02

Создать роль с bound_claims

bound_claims ограничивает доступ: только токены с указанными значениями (например, project_id, project_path) смогут аутентифицироваться. Без этого любой CI-токен получит доступ

```
stronghold write
auth/jwt/role/gitlab-role - <<EOF
{
   "role_type": "jwt",
   "user_claim": "sub",
    "bound_audiences": ["vault"],
    "bound_claims": {
       "project_id": "23"
     },
    "policies": ["gitlab-policy"],
     "ttl": "1h"
  }
EOF
```

03

Написать ACL-политику

Политика задаёт конкретные пути в Vault, к которым разрешено обращаться. Принцип минимальных прав: только read к нужным секретам

```
stronghold policy write
gitlab-policy - <<EOF
path "kv/data/code/vault-demo" {
  capabilities = ["read"]
}
EOF
```

Использование секретов в .gitlab-ci.yml

```yaml
stages:
   - test

stronghold-demo:
  stage: test
  image: alpine

  # 1. Запрос JWT-токена с audience = vault
  id_tokens:
    VAULT_ID_TOKEN:
      aud: https://stronghold.example.com

  # 2. Объявление секрета: путь@движок
  secrets:
    DATABASE_PASSWORD:
      vault: code/stronghold-demo/DATABASE_PASSWORD@kv
      token: $VAULT_ID_TOKEN
      file: false
  script:
    - echo $DATABASE_PASSWORD
```

Анатомия пути к секрету

`code/stronghold-demo/DATABASE_PASSWORD@kv`

01

Путь до секрета в Stronghold

`code/vault-demo`

02

Имя поля внутри секрета

`DATABASE_PASSWORD`

03

Точка монтирования Secret Engine (kv-v2 по умолчанию)

`kv`

Почему нужно использовать bound_claims?

Deckhouse

Смотрим 👀

Как управлять конфигурацией Stronghold?

Deckhouse

Stronghold-GitOps-плагин

Что это:

- Плагин для Stronghold с GitOps-подходом
- Следит за Git-репозиторием и применяет изменения конфигурации
- Состояние хранится в Stronghold

Зачем:

- Управлять политиками, ролями, движками декларативно через Git
- Вместо ручных stronghold write — конфигурация как код (YAML/HCL)
- Изменения попадают в хранилище только через кворум подписей

Ценность:

- Безопасность: изменения только по доверенной PGP-подписи
- Прозрачность и аудит: вся история изменений — в Git
- Автоматизация: плагин применяет конфигурацию из репозитория по pull-модели без ручных операций

Что нужно для начала работы?

01 Включить плагин: `SHA=$(sha256sum $PWD/gitops | awk '{print $1;}')`

```
stronghold plugin register -command gitops -sha256 $SHA -version=v0.0.1
secret gitops
stronghold secrets enable gitops
```

02 Подключить репозиторий:

```
stronghold write gitops/configure/git_repository \
git_repo_url="https://gitlab.com/user/vault-gitops-configuration.git" \
      required_number_of_verified_signatures_on_commit=1 \
      git_poll_period=1m
```

Что нужно для начала работы?

03 Создаём приватные ключи:

```
gpg --quick-generate-key "key1 <key1@example.com>" rsa4096
gpg --quick-generate-key "key2 <key2@example.com>" rsa4096
```

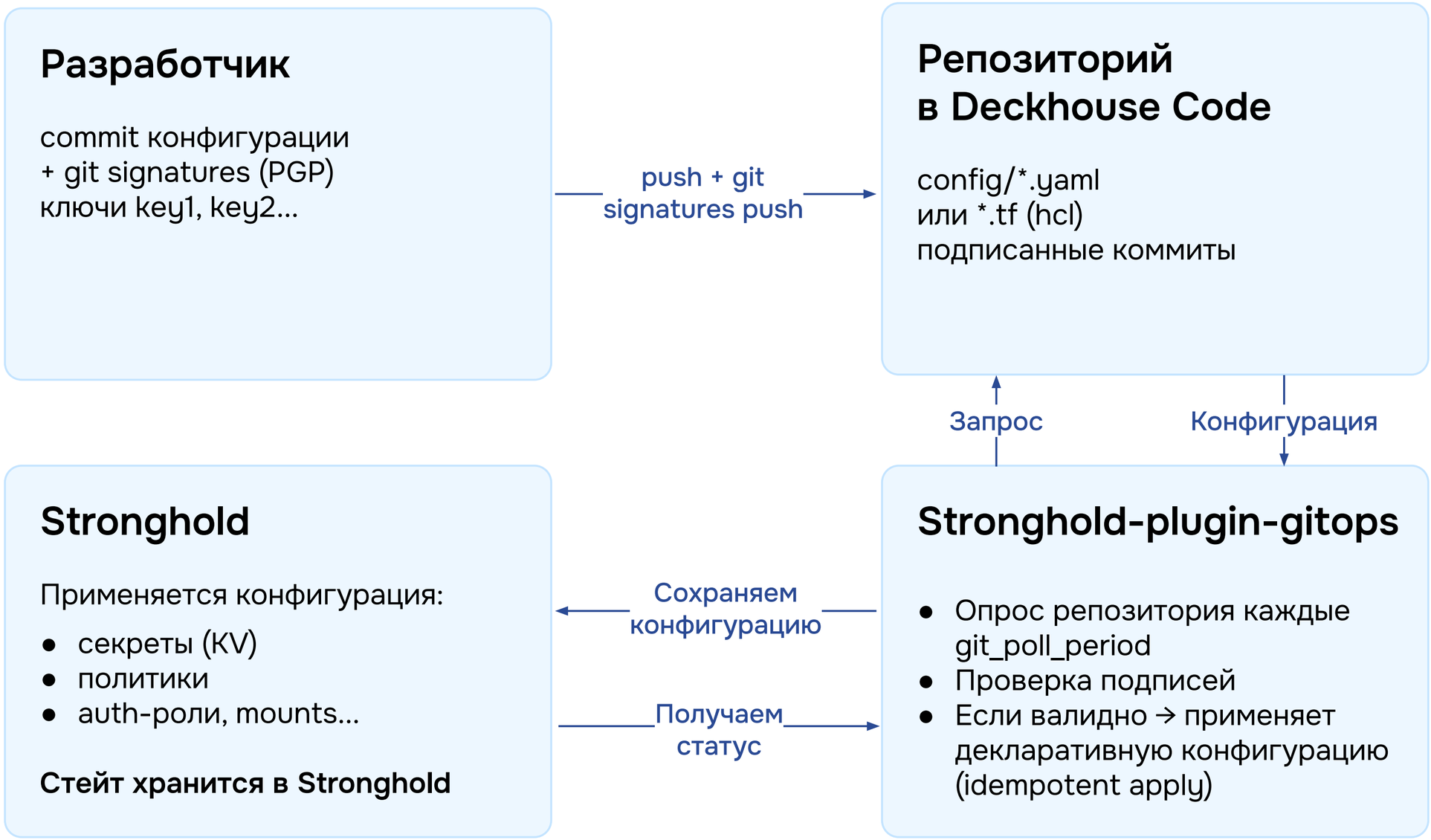
04 Загружаем публичные части ключей:

```
stronghold write gitops/configure/trusted_pgp_public_key/key1
public_key=@key1.pgp
stronghold write gitops/configure/trusted_pgp_public_key/key2
public_key=@key2.pgp
```

05 Даём доступ плагину к API:

```
TOKEN=$(vault token create -orphan -period=7d -policy=root
-display-name="gitops-plugin" -wrap-ttl 1m -field=wrapping_token)
stronghold write gitops/configure/vault
vault_addr=http://127.0.0.1:8200 wrapping_token=$TOKEN
```

Как работает плагин



Смотрим 👀

Наши планы

01

Соответствие требованиям российского законодательства

- Релиз Deckhouse Stronghold CSE 1.19: новые функции безопасности и соответствие требованиям Приказа ФСТЭК России № 117
- Расширение списка поддерживаемых HSM (ViPNet)
- Оценка влияния СКЗИ в соответствии с требованиями ПКЗ-2005

02

Отказоустойчивость и масштабирование

- Поддержка российских систем (RuBackup, Кибербэкап, Береста) для резервного копирования данных

03

Централизованное управление и контроль

- Поддержка российских решений для управления доступом (например, РЕД АДМ)
- Поддержка аутентификации через Kerberos

Наши планы

04 **Безопасность CI/CD-конвейера и приложений**

- Интеграция с российскими CI/CD-инструментами
- Управление секретами в ML-пайплайнах (например, Airflow)

05 **Удобство использования для пользователей**

- Миграция веб-интерфейса с Ember.js на Vue
- Генерация отчетов об использовании (Utilization Report)

06 **Работа в любой инфраструктуре**

- Поддержка российских платформ контейнеризации (Боцман, Штурвал)

Почему Code + Stronghold работает нативно

01 Экосистема DKP — Code и Stronghold интегрированы с мониторингом, логированием, сетевыми политиками платформы

02 Российский вендор — «Флант» — и Deckhouse в реестре отечественного ПО. Поддержка на русском языке, контракты по российскому праву

03 CODEOWNERS + protected branches — защита .gitlab-ci.yml от несанкционированных изменений. Stronghold гарантирует, КТО получит секрет, CODEOWNERS гарантирует, что в пайплайне не появится код для его компрометации

Промокод со скидкой 15 % на курс от Deckhouse Академии «Инструменты безопасности в Deckhouse Kubernetes Platform»

Напишите промокод

CODE15

На почту education@flant.ru

До 28 сентября включительно

Подробнее о курсе

Онлайн-анонс большого продуктового обновления Deckhouse

В сентябре Deckhouse выходит
на новую траекторию развития.

Мы расскажем, как обновили продукты,
чтобы ваша инфраструктура стала
более гибкой, управляемой и готовой
к любым задачам и нагрузкам.

Регистрируйтесь
и приглашайте коллег!

Зарегистрироваться

прямой эфир

17.09 12:00

Deckhouse

Узнайте больше

о предлагаемом нами новом отраслевом стандарте безопасной разработки и о том, как комплексное решение Deckhouse Code + Stronghold обеспечит быстрый и прозрачный для всех команд пайплайн, готовый к РБПО

Подробнее о Deckhouse Code

Подробнее о Deckhouse Stronghold

deckhouse.ru