

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению практических работ
по дисциплине «Инфраструктура как код и платформенная инженерия» для
студентов направления подготовки 09.04.04 Программная инженерия
Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

Содержание

ВВЕДЕНИЕ	3
Лабораторная работа №1	10
Лабораторная работа №2	22
Лабораторная работа №3	36
Лабораторная работа №4	55
Лабораторная работа №5	68
Лабораторная работа №6	83
Лабораторная работа №7	95
Лабораторная работа №8	108
Лабораторная работа №9	123
Лабораторная работа №10	141
Лабораторная работа №11	156
Лабораторная работа №12	172
Лабораторная работа №13	191
Лабораторная работа №14	212
Лабораторная работа №15	238
Лабораторная работа №16	259
Лабораторная работа №17	287
Лабораторная работа №18	337

ВВЕДЕНИЕ

Настоящий лабораторный практикум является неотъемлемой частью дисциплины «Инфраструктура как код и платформенная инженерия», реализуемой в рамках образовательной программы магистратуры по направлению 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»).

Дисциплина относится к части, формируемой участниками образовательных отношений (вариативная часть), и является одной из ключевых дисциплин профиля, формирующих профессиональные компетенции в области платформенной инженерии, инфраструктуры как кода и управления сложными распределенными системами.

Лабораторный практикум представляет собой сквозной цикл из 18 лабораторных работ, объединенных общей целью - формирование практических навыков проектирования, реализации и эксплуатации современной платформы управления инфраструктурой и парками автономных устройств.

Современная ИТ-индустрия переживает фундаментальный сдвиг в подходах к управлению инфраструктурой и разработкой программного обеспечения. Традиционные методы администрирования серверов и ручного управления конфигурациями уступают место платформенному подходу, где инфраструктура рассматривается как программируемый ресурс, управляемый через код и Git-репозитории.

Ключевые тренды, определяющие актуальность дисциплины:

Тренд	Описание
Platform Engineering	Выделение платформенных команд, создающих внутренние платформы разработчика (IDP) для ускорения доставки приложений

Тренд	Описание
Infrastructure as Code (IaC)	Управление инфраструктурой через декларативные конфигурации, хранящиеся в системах контроля версий
Policy as Code	Автоматизация проверки соответствия инфраструктуры требованиям безопасности, стоимости и compliance
GitOps	Использование Git как единого источника истины для декларативного управления инфраструктурой и приложениями
FinOps	Управление облачными затратами на уровне кода, проактивный контроль бюджета
Observability	Наблюдаемость не только приложений, но и самой платформы
Управление устройствами	Обеспечение безопасности и надежности парков автономных устройств (БПЛА, роботов, IoT)

Специфика профиля «Разработка, мониторинг и управление жизненным циклом инженерных систем» добавляет дополнительное измерение - необходимость управления не только облачной инфраструктурой, но и парками автономных устройств, что требует интеграции подходов из DevOps, SRE, безопасности и embedded-разработки.

Лабораторный практикум построен на основе знаний, полученных при изучении следующих дисциплин первого семестра:

Дисциплина	Предоставленные знания	Применение в ЛР
Проектирование и оптимизация конвейеров CI/CD	Построение пайплайнов, GitLab CI/GitHub Actions, артефакты	Шаблоны Auto-DevOps, Preview environments, ОТА-пайплайны
Системная интеграция и управление конфигурациями	Terraform, Ansible, Vault, API Gateway	Модульный IaC, Terragrunt, управление секретами
Оркестрация контейнеров и Kubernetes	Развертывание кластеров, Helm, операторы	OPA Gatekeeper, Karpenter, GitOps
Надежность и устойчивость систем	SLO/SLI, Error Budget, Chaos Engineering	SLO платформы, алерты на burn rate
Принципы и практики Observability	Prometheus, Grafana, Loki, Tempo, OpenTelemetry	Сбор телеметрии с устройств, дашборды

Таким образом, лабораторный практикум не дублирует, а надстраивается над пройденным материалом, применяя его на новом, более высоком уровне абстракции - уровне платформы как продукта.

Лабораторные работы объединены общей архитектурой - создание платформы управления парком автономных устройств (БПЛА,

роботов). Эта платформа последовательно строится от базовых IaC-операций до полноценного интеграционного решения.

Связь лабораторных работ между собой

Лабораторная работа	Базис	Обеспечивает
ЛР1-3	Базовый IaC	Модули и состояния для всей платформы
ЛР4-6	Policy as Code	Валидацию конфигураций и автоматическую блокировку
ЛР7-8	Cost & FinOps	Экономическую эффективность платформы
ЛР9-10	Security	Безопасность цепочки поставки и секретов
ЛР11-12	CI/CD как услуга	Разработку новых функций через preview-окружения
ЛР13-14	Observability & SLO	Надежность и наблюдаемость платформы
ЛР15-18	Управление устройствами	Интеграцию с реальными устройствами

Цель практикума: Формирование у обучающихся практических навыков проектирования, реализации и эксплуатации современной платформы управления инфраструктурой с использованием методологий Infrastructure as Code, Policy as Code, GitOps и Platform Engineering, а также применение этих навыков для управления парками автономных устройств.

Задачи практикума:

1. Освоить инструменты IaC на продвинутом уровне:
2. Разработка модульных Terraform конфигураций
3. Управление удаленным состоянием с блокировками
4. Применение Terragrunt для DRY-конфигураций
5. Внедрить Policy as Code в CI/CD и Kubernetes:
6. Написание политик на Rego
7. Автоматическая проверка Terraform планов
8. Admission control через OPA Gatekeeper
9. Обеспечить экономическую эффективность платформы:
10. Оценка стоимости инфраструктуры до деплоя (Infracost)
11. Автоматическая оптимизация затрат (Karpenter)
12. Реализовать Security as Code:
13. Централизованное управление секретами (HashiCorp Vault)
14. Подпись и верификация артефактов (Sigstore/Cosign)
15. Создать CI/CD как услугу:
16. Разработка reusable шаблонов пайплайнов
17. Preview-окружения по Pull Request
18. Обеспечить наблюдаемость платформы:
19. Сбор телеметрии с платформенных сервисов
20. Управление SLO и Error Budget
21. Применить платформу для управления устройствами:
22. GitOps для конфигураций устройств
23. OTA-обновления с canary и rollback
24. Сбор телеметрии с edge-устройств
25. Интеграция с внешними API

В ходе выполнения лабораторных работ используется следующее ПО:

Категория	Инструменты

Категория	Инструменты
IaC	Terraform, Terragrunt, OpenTofu
Контейнеризация	Docker, containerd
Оркестрация	Kubernetes (Minikube/Kind/K3s), Helm
CI/CD	GitLab CI / GitHub Actions
Политики	OPA, Conftest, Gatekeeper
Безопасность	Cosign (Sigstore), Trivy, Vault
Мониторинг	Prometheus, Grafana, Loki, Tempo
Наблюдаемость	OpenTelemetry Collector, Jaeger
GitOps	FluxCD, ArgoCD
Сети	Kong, Envoy, Nginx

Требования к выполнению и защите

Общие требования

Каждая лабораторная работа выполняется индивидуально в соответствии с вариантом

1. Код, конфигурации и скрипты должны храниться в Git-репозитории
2. Отчет оформляется в формате Markdown или PDF
3. Все команды и их выводы должны быть зафиксированы в отчете
4. Демонстрация работы должна включать скриншоты (или screencast)
5. 8.2. Порядок защиты
6. Представление отчета преподавателю

7. Демонстрация работающей системы
8. Ответы на контрольные вопросы
9. Обсуждение выбранных решений и альтернатив

Лабораторная работа №1

Организация модульного Terraform с частным регистром

1. Цель работы: Применить модульную организацию кода Infrastructure as Code (IaC) для повторного использования, разработав и опубликовав пользовательские Terraform-модули в частном Git-регистре, и использовать эти модули для создания двух изолированных окружений (dev/staging) с различными параметрами.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Разработать модуль Terraform для управления сетевой инфраструктурой (VPC/сеть, подсети, маршрутизация)
2. Разработать модуль Terraform для создания вычислительных ресурсов (виртуальные машины / compute instances)
3. Настроить частный Git-регистр для хранения и версионирования модулей
4. Создать конфигурации двух окружений (dev и staging), использующих разработанные модули
5. Параметризовать модули через переменные для обеспечения гибкости окружений
6. Обеспечить изоляцию состояний (state) для разных окружений

3. Краткое теоретическое обоснование

3.1. Основные компоненты Terraform

При работе с Terraform взаимодействуют несколько ключевых компонентов :

- Terraform CLI - инструмент для выполнения команд, «ядро» системы
- Конфигурационные файлы (.tf) - описывают желаемое состояние инфраструктуры

- State (состояние) - отражение реальной инфраструктуры; Terraform сравнивает желаемое состояние с действительным и предпринимает действия для устранения различий
- Провайдер - плагин, обеспечивающий взаимодействие с внешними сервисами и API

3.2. Модульный подход в Terraform

Terraform модули - это контейнеры для группы ресурсов, используемых вместе. Модули позволяют:

- Инкапсулировать сложную инфраструктурную логику
- Повторно использовать код в разных окружениях и проектах
- Стандартизировать конфигурации согласно best practices
- Упростить поддержку за счет единой точки изменений

3.3. Структура модуля

modules/

```

├── network/
│   ├── main.tf      # основные ресурсы
│   ├── variables.tf  # входные переменные
│   ├── outputs.tf    # выходные значения
│   ├── README.md     # документация
│   └── versions.tf    # требования к провайдерам
└── compute/
    ├── main.tf
    ├── variables.tf
    ├── outputs.tf
    └── README.md
  
```

3.4. Источники модулей

Terraform поддерживает различные источники (sources) для модулей:

Источник	Формат	Пример

Источник	Формат	Пример
Локальный путь	./modules/network	Для разработки
Git (HTTPS)	git::https://.../modules/network?ref=v1.0.0	Частный регистр
Git (SSH)	git::git@github.com:org/modules.git//network	Частный регистр

3.5. Версионирование модулей

Рекомендуется использовать Git-теги для версионирования:

```
git tag -a "v1.0.0" -m "First release of network module"
git push --tags
```

3.6. Изоляция окружений

Для разделения состояний разных окружений используется различная конфигурация backend:

```
text
environments/
├── dev/
│   ├── main.tf
│   ├── terraform.tfvars
│   └── backend.tf  # указывает на dev-состояние
└── staging/
    ├── main.tf
    ├── terraform.tfvars
    └── backend.tf  # указывает на staging-состояние
```

3.7. Основные команды Terraform CLI

Команда	Назначение

Команда	Назначение
terraform init	Подготовка рабочей директории, загрузка провайдеров и модулей
terraform validate	Проверка корректности конфигурации
terraform plan	Показ изменений, требуемых текущей конфигурацией
terraform apply	Создание или обновление инфраструктуры
terraform fmt	Форматирование конфигурации в стандартном стиле

4. Контрольные вопросы

1. Что такое модуль в Terraform и какие проблемы он решает?
2. Какова рекомендуемая структура директорий для модуля?
3. Какие существуют способы указания источника (source) для модуля?
4. Для чего используется версионирование модулей и как оно реализуется через Git?
5. Каким образом можно изолировать состояния разных окружений (dev/staging/prod)?
6. Что такое locals в Terraform и для каких целей он используется?
7. Какие мета-аргументы Terraform вы знаете? Приведите примеры.
8. В чем разница между variable и local значениями?
9. Как проверить модуль локально перед публикацией в Git-регистр?
10. Какие best practices модульного дизайна вы можете назвать?
11. Что произойдет при изменении входных переменных модуля?

12. Как разрешить конфликт версий провайдеров в разных модулях?
13. Какие типы переменных поддерживает Terraform?
14. Для чего нужен файл `versions.tf` в модуле?
15. Как организовать тестирование изменений модуля без публикации новой версии?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Разработать два модуля (`network` и `compute`) в соответствии со спецификацией варианта

Опубликовать модули в частном Git-регистре (можно использовать GitLab CE, Gitea или имитировать через локальный Git-репозиторий с тегами)

Создать конфигурации для двух окружений: `dev` и `staging`

2. Обеспечить, чтобы окружения использовали один и тот же источник модулей (через Git)

Настроить разные backend для хранения состояния (например, - `backend-config="key=dev/network.tfstate"`)

Продемонстрировать успешное выполнение `terraform init`, `plan`, `apply` для обоих окружений

3. Предоставить отчет о выполненной работе

5.2. Базовые параметры окружений

Для каждого варианта задаются параметры, которые должны различаться между `dev` и `staging`:

Параметр	Окружение dev	Окружение staging
Размер экземпляра	Минимальный (e2.micro / t2.micro)	Стандартный

Параметр	Окружение dev	Окружение staging
	/ standard-v1)	(e2.medium/ t2.medium/ standard-v2)
Количество экземпляров	1	2
Публичный IP	Да	Только у одного
Теги окружения	env = "dev"	env = "staging"
Backup enabled	false	true
Monitoring	basic	full

5.3. Варианты заданий

Вариант 1. Веб-приложение с балансировкой

- Модуль network: VPC, 2 приватные подсети, 1 публичная подсеть, Internet Gateway, NAT Gateway
- Модуль compute: Auto Scaling Group, Launch Template, Application Load Balancer, Target Group
- Особенности: Настроить health checks для ALB; staging должен иметь дополнительный экземпляр в reserve

Вариант 2. База данных с репликацией

- Модуль network: VPC, подсети для БД (изолированные), Security Groups с жесткими правилами
- Модуль compute: Master-Slave конфигурация (2 экземпляра), Floating IP для мастера

- Особенности: В dev - только master; в staging - master + async replica

Вариант 3. Кластер обработки данных

- Модуль network: VPC, приватные подсети для данных, публичный bastion host
- Модуль compute: 3 экземпляра + bastion, внутренний DNS
- Особенности: dev - 1 экземпляр без bastion; staging - полный кластер

Вариант 4. Микросервисная платформа

- Модуль network: VPC, подсети для сервисов, Ingress-контроллер, API Gateway
- Модуль compute: Кластер Kubernetes, node pools
- Особенности: dev - 1 control plane + 1 worker; staging - 3 control plane + 3 worker

Вариант 5. CI/CD инфраструктура

- Модуль network: VPC, публичная подсеть для CI-сервера, приватная для раннеров
- Модуль compute: CI-сервер (GitLab/Jenkins), раннеры
- Особенности: dev - все в одном экземпляре; staging - отдельные компоненты

Вариант 6. Мониторинг и observability

- Модуль network: VPC, подсети для мониторинга, выделенная storage network
- Модуль compute: Prometheus + Grafana + Loki
- Особенности: dev - single-node мониторинг; staging - HA-конфигурация

Вариант 7. Edge-вычисления (для парка устройств)

- Модуль network: Hub-Spoke топология, VPN для подключения устройств

- Модуль compute: Edge-ноды + центральный оркестратор, MQTT брокер
- Особенности: dev - 1 edge нода; staging - 3 edge ноды + кластер оркестратора

Вариант 8. Система логирования

- Модуль network: VPC с выделенной подсетью для Elasticsearch кластера
- Модуль compute: Elasticsearch (hot-warm архитектура), Kibana, Logstash
- Особенности: dev - single node ES; staging - 3 master + 5 data nodes

Вариант 9. Очереди сообщений

- Модуль network: VPC, подсети с низкой задержкой
- Модуль compute: RabbitMQ кластер или Kafka
- Особенности: dev - single broker; staging - 3-node cluster с репликацией

Вариант 10. Система кэширования

- Модуль network: VPC, подсети для Redis Sentinel кластера
- Модуль compute: Redis master + replicas + sentinels
- Особенности: dev - master + 1 replica; staging - master + 3 replicas + 3 sentinels

Вариант 11. Бэкап и восстановление

- Модуль network: VPC, изолированная подсеть для бэкапов, S3-совместимое хранилище
- Модуль compute: Application + Backup agent + Object Storage
- Особенности: dev - бэкап раз в день; staging - бэкап каждые 4 часа + репликация

Вариант 12. ML-инференс платформа

- Модуль network: VPC с GPU-подсетями, высокоскоростная сеть
- Модуль compute: GPU-экземпляры + модель сервис
- Особенности: dev - 1 GPU; staging - 4 GPU + load balancer

Вариант 13. Блокчейн сеть

- Модуль network: VPC, p2p-подсеть, bootnode
- Модуль compute: validator ноды + RPC ноды
- Особенности: dev - 2 validator ноды; staging - 7 validator нод + 3

RPC

Вариант 14. Рендеринг и GPU-графика

- Модуль network: VPC с высокопропускной сетью
- Модуль compute: GPU-кластер + очередь рендеринга
- Особенности: dev - 1 GPU-экземпляр; staging - 10 GPU-экземпляров

Вариант 15. Гибридная инфраструктура

- Модуль network: VPN между on-prem (симулируется) и cloud VPC
- Модуль compute: Экземпляры в cloud + симуляция on-prem серверов
- Особенности: dev - только on-prem симуляция; staging - hybrid с репликацией

5.4. Требования к модулям

Модуль network должен:

1. Иметь входные переменные:
environment, vpc_cidr, subnet_cidrs, enable_nat
2. Иметь выходные переменные:
vpc_id, subnet_ids, security_group_ids

Модуль compute должен:

1. Иметь входные переменные: environment, subnet_ids (из модуля network), instance_count, instance_type, assign_public_ip
2. Иметь выходные переменные: instance_ids, instance_ips
3. Код должен:
4. Использовать locals для внутренних вычислений
5. Содержать комментарии на русском или английском
6. Быть отформатирован через terraform fmt
7. Проходить terraform validate

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист (название работы, ФИО, вариант, группа)
2. Цель и задачи работы (из п. 1–2 настоящего документа)
3. Ответы на контрольные вопросы (п. 4)
4. Теоретическая часть (краткое изложение основных понятий своими словами)
5. Схема инфраструктуры (диаграмма в PlantUML, Mermaid или Draw.io)
6. Структура репозитория (дерево каталогов с пояснениями)
7. Код модуля network (основные фрагменты с комментариями)
8. Код модуля compute (основные фрагменты с комментариями)
9. Конфигурация окружения
dev (main.tf, variables.tfvars, backend.tf)
10. Конфигурация окружения
staging (main.tf, variables.tfvars, backend.tf)
11. Результаты выполнения (скриншоты terraform plan, apply, state list)
12. Заключение (достигнуты ли цель и задачи, какие сложности возникли)
13. Список использованных источников

6.2. Технические требования

- Отчет оформляется в формате Markdown (.md) или PDF
- Код должен быть представлен в виде текстовых блоков с указанием языка (````hcl````)
- Скриншоты должны быть четкими, с видимыми командами и выводами
- Объем отчета: 5–15 страниц в печатном виде

- Репозиторий с кодом должен быть доступен преподавателю
(ссылка в отчете)

6.3. Пример оформления фрагмента кода

markdown

```
```hcl
modules/network/main.tf

resource "yandex_vpc_network" "this" {
 name = var.environment
}
```
```

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов	Вес
Разработка модуля network	15	15%
Разработка модуля compute	15	15%
Публикация в Git-регистр с версионированием	10	10%
Конфигурация окружения dev	15	15%
Конфигурация окружения staging	15	15%
Ответы на контрольные вопросы	10	10%
Качество отчета	10	10%
Защита (устные ответы)	10	10%

Критерий	Макс. баллов	Вес
Итого	100	100%

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Все модули работают, код соответствует best practices, использованы locals, outputs, variables, форматирование выполнено; Git-регистр с тегами настроен; dev и staging различаются по всем заданным параметрам; контрольные вопросы проработаны; отчет полный; защита уверенная
Хорошо	75-89	Модули работают, но есть незначительные нарушения best practices (например, отсутствует README); Git-регистр настроен, но версионирование неполное; различия окружений выполнены частично; ответы на вопросы неполные; отчет содержит все разделы
Удовлетворительно	60-74	Модули работают, но код не отформатирован, документация отсутствует; Git-регистр не настроен

Оценка	Баллы	Требования
		(используются локальные пути); различия окружений минимальны; ответы на вопросы поверхностны; отчет неполный
Неудовлетворительно	<60	Модули не работают или не предоставлены; задание не выполнено; отчет отсутствует

7.3. Штрафные баллы

Нарушение	Штраф
Отсутствие форматирования (terraform fmt)	-5
Hardcoded значения вместо переменных	-5
Отсутствие outputs у модулей	-5
Разные окружения используют одно состояние	-10
Отсутствие ссылки на репозиторий в отчете	-10

Лабораторная работа №2

Управление удаленным состоянием и блокировками

1. Цель работы

Организовать командную работу с Infrastructure as Code (IaC) без конфликтов путем настройки централизованного удаленного хранилища состояния Terraform с поддержкой блокировок, а также отработать сценарии разрешения конфликтных ситуаций при одновременном изменении инфраструктуры несколькими инженерами.

Формируемые компетенции: ПК-2

2. Задачи работы

1. Развернуть S3-совместимое хранилище (MinIO) для хранения state-файлов Terraform
2. Настроить бэкенд Terraform с использованием S3-совместимого хранилища и поддержкой блокировок (через DynamoDB или аналоги)
3. Создать базовую конфигурацию инфраструктуры для тестирования
4. Сымитировать параллельную работу двух инженеров, изменяющих одну и ту же инфраструктуру
5. Проанализировать поведение Terraform при возникновении блокировки
6. Отработать сценарий принудительного снятия блокировки (force-unlock)
7. Реализовать резервное копирование и восстановление состояния

3. Краткое теоретическое обоснование

3.1. Проблема управления состоянием в команде

Terraform State - это файл, содержащий отображение между объявленной конфигурацией и реальной инфраструктурой. При работе в команде возникает несколько проблем:

Проблема	Описание	Решение
Конкурентные изменения	Два инженера одновременно применяют изменения	Блокировки состояния
Потеря состояния	Локальный файл на компьютере коллеги	Централизованное хранилище

Проблема	Описание	Решение
Неактуальное состояние	Состояние не синхронизировано с реальной инфраструктурой	Remote state + refresh
Утечка секретов	Конфиденциальные данные в state-файле	Шифрование state

3.2. Типы хранения состояния

Тип	Преимущества	Недостатки
Локальный	Простота, не требует настройки	Не подходит для команд, нет блокировок
Удаленный без блокировок	Централизован, доступен всем	Возможны конфликты
Удаленный с блокировками	Безопасная командная работа	Требует дополнительной инфраструктуры

3.3. Backend'ы Terraform

Backend - это конфигурация, определяющая, где и как хранится состояние. Terraform поддерживает множество типов backend'ов:

Backend	Блокировки	Шифрование	Применение
local	Нет	Нет	Локальная разработка
s3	Да	Да	AWS, совместимые

Backend	Блокировки	Шифрование	Применение
	DynamoDB		хранилища
azurerm	Да	Да	Azure
gcs	Да	Да	Google Cloud
etcd	Да	Нет	Кластерные среды
pg (PostgreSQL)	Да	Опционально	Реляционные БД
kubernetes	Да (lease)	Нет	K8s среды
http	Нет	Зависит	REST API

3.4. Блокировки состояния

Блокировки состояния предотвращают одновременные операции записи:

- Зачем: Предотвращает повреждение state при параллельных apply
- Как работает: Backend создает Lock (в DynamoDB, таблице БД или lease)
- Кто создает: Terraform автоматически при старте apply или plan - write

Принцип работы:

1. Инженер А запускает terraform apply
2. Terraform создает запись блокировки в DynamoDB
3. Инженер Б пытается запустить terraform apply
4. Terraform обнаруживает существующую блокировку и ожидает или выдает ошибку
5. После завершения инженера А блокировка автоматически снимается

3.5. S3-совместимые хранилища для работы в России

Для использования в российских условиях доступны следующие S3-совместимые решения:

Решение	Тип	Особенность
MinIO	Open Source	Локальное развертывание в контейнерах
Yandex Object Storage	Облачный сервис	S3-совместимый, доступен из РФ
VK Cloud S3	Облачный сервис	S3-совместимый, российский провайдер
Selectel	Облачный сервис	S3-совместимый, российский провайдер

3.6. Структура бэкенда для S3 с блокировками

```
# backend.tf
terraform {
  backend "s3" {
    bucket      = "terraform-state-bucket"
    key         = "project/dev/network.tfstate"
    region      = "ru-central1"
    endpoint     = "https://storage.yandexcloud.net"
    access_key   = var.access_key
    secret_key   = var.secret_key
    skip_region_validation = true

    # Блокировки через DynamoDB-совместимый сервис
    dynamodb_table = "terraform-locks"
    dynamodb_endpoint = "https://docapi.serverless.yandexcloud.net/ru-central1/..."
  }
}
```

3.7. Команды для работы с состоянием

Команда	Назначение

Команда	Назначение
terraform state list	Список всех ресурсов в состоянии
terraform state show <resource>	Просмотр деталей ресурса
terraform state mv	Перемещение ресурса в состоянии
terraform state rm	Удаление ресурса из состояния
terraform state pull	Получение текущего состояния
terraform state push	Обновление состояния (опасно!)
terraform force-unlock <ID>	Принудительное снятие блокировки

4. Контрольные вопросы

1. Что такое Terraform State и почему он критически важен для работы?
2. Какие проблемы возникают при хранении state-файла локально в командной работе?
3. Какие типы backend'ов поддерживает Terraform? Приведите примеры.
4. Как работает механизм блокировок состояния? Какой компонент отвечает за блокировки в S3 backend?
5. Что произойдет, если два инженера одновременно запустят terraform apply с настроенным удаленным состоянием и блокировками?
6. Как принудительно снять блокировку, если инженер прервал работу (например, закрыл ноутбук)?
7. Какие риски связаны с командой terraform force-unlock?

8. Для чего используется команда terraform refresh?
9. Как зашифровать state-файл в удаленном хранилище?
10. Чем отличается terraform state pull от terraform state show?
11. В каких случаях может потребоваться ручное редактирование state-файла?
12. Как организовать изолированные state для различных окружений (dev/staging/prod) в одном бэкенде?
13. Какие best practices управления состоянием вы знаете?
14. Какие S3-совместимые решения доступны для развертывания в российской инфраструктуре?
15. Как организовать бэкапы state-файлов и восстановление из них?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Развернуть S3-совместимое хранилище (MinIO в Docker или облачный объектный storage)
2. Настроить DynamoDB-совместимый сервис для блокировок (MinIO не поддерживает блокировки, поэтому для блокировок использовать либо PostgreSQL backend, либо DynamoDB от облачного провайдера, либо реализовать consul/etcd)
3. Создать Terraform конфигурацию для базовой инфраструктуры (согласно варианту)
4. Настроить backend с поддержкой блокировок
5. Сымитировать ситуацию конфликта (параллельный apply)
6. Продемонстрировать разрешение конфликта
7. Настроить версионирование и бэкапы state-файлов
8. Предоставить отчет о выполненной работе

5.2. Хранилище для лабораторной работы

Для выполнения работы рекомендуется использовать локальный MinIO или PostgreSQL backend как наиболее доступные варианты для учебных целей. Для блокировок:

- Вариант А (рекомендуемый): Backend pg (PostgreSQL) - поддерживает блокировки из коробки
- Вариант В: MinIO + DynamoDB (через Yandex Cloud / VK Cloud / LocalStack)
- Вариант С: Backend consul или etcd для блокировок
-

5.3. Варианты заданий

Общая структура для всех вариантов: создается простая инфраструктура, указанная в варианте, с использованием модулей из ЛР №1.

Вариант 1. Веб-сервер с нагрузкой

- Инфраструктура: 2 веб-сервера + балансировщик
- Параметр конфликта: оба инженера пытаются изменить количество реплик
- Дополнительно: реализовать state для каждого окружения

Вариант 2. Кластер баз данных

- Инфраструктура: PostgreSQL master-replica (2 экземпляра)
- Параметр конфликта: одновременное изменение параметров репликации
- Дополнительно: настроить разные state для master и replica

Вариант 3. Сетевая инфраструктура

- Инфраструктура: VPC, 2 подсети, маршрутизация
- Параметр конфликта: изменение CIDR подсетей
- Дополнительно: реализовать тегирование ресурсов через state

Вариант 4. Система мониторинга

- Инфраструктура: Prometheus + Grafana (1 сервер)
- Параметр конфликта: изменение параметров retention
- Дополнительно: настроить бэкапы состояния

Вариант 5. CI/CD компоненты

- Инфраструктура: GitLab CI runner (1 шт)
- Параметр конфликта: изменение executor settings
- Дополнительно: state с чувствительными данными

Вариант 6. Хранилище объектов

- Инфраструктура: MinIO сервер (1 экземпляр)
- Параметр конфликта: изменение bucket policy
- Дополнительно: state для конфигурации MinIO

Вариант 7. Очереди сообщений

- Инфраструктура: RabbitMQ 1 нода
- Параметр конфликта: создание очереди двумя инженерами
- Дополнительно: подключение к state другого проекта

Вариант 8. Кэш-сервер

- Инфраструктура: Redis (1 мастер)
- Параметр конфликта: изменение maxmemory политики
- Дополнительно: state с несколькими ключами

Вариант 9. DNS инфраструктура

- Инфраструктура: DNS записи (A, CNAME)
- Параметр конфликта: изменение одной зоны
- Дополнительно: импорт существующих записей в state

Вариант 10. Система логирования

- Инфраструктура: Loki + Promtail
- Параметр конфликта: изменение правил агрегации логов
- Дополнительно: миграция между backend'ами

Вариант 11. Безопасность

- Инфраструктура: Security groups, Network ACLs
- Параметр конфликта: изменение правил доступа

Дополнительно: state с несколькими источниками

Вариант 12. Auto Scaling группа

- Инфраструктура: ASG с 1-3 инстансами

- Параметр конфликта: изменение порогов масштабирования

Дополнительно: state с динамическими ресурсами

Вариант 13. Service Discovery

- Инфраструктура: Consul кластер (3 ноды)
- Параметр конфликта: регистрация сервиса
- Дополнительно: state для регистрации сервисов

Вариант 14. Система резервного копирования

- Инфраструктура: Backup сервер + хранилище
- Параметр конфликта: изменение расписания бэкапа
- Дополнительно: state для репликации

Вариант 15. API Gateway

- Инфраструктура: Kong/Nginx + маршруты
- Параметр конфликта: изменение одного маршрута
- Дополнительно: хранение state в нескольких регионах

5.4. Технические требования

1. Backend конфигурация:

- Должна поддерживать блокировки
- Должна быть вынесена в отдельный файл backend.tf
- Должна хранить state для разных окружений в разных ключах

2. Демонстрация конфликта:

- Использовать два терминала (два "инженера")
- Показать сообщение об ошибке блокировки
- Дождаться завершения первого apply
- Показать успешный apply второго

3. Force-unlock:

- Сымитировать прерванный apply (Ctrl+C)
- Показать сообщение о блокировке
- Выполнить terraform force-unlock
- Подтвердить возможность продолжения работы

5.5. Пример конфигурации backend (PostgreSQL вариант)

```
hcl
# backend.tf (PostgreSQL backend)
terraform {
  backend "pg" {
    conn_str = "postgres://user:password@localhost:5432/terraform_state?sslmode=disable"
    schema_name = "terraform_${var.environment}"
  }
}
```

5.6. Пример конфигурации backend (MinIO + DynamoDB вариант)

```
hcl
# backend.tf
terraform {
  backend "s3" {
    bucket      = "tf-state-bucket"
    key         = "lab2/network/terraform.tfstate"
    region      = "us-east-1"
    endpoint    = "http://localhost:9000"
    access_key   = "minioadmin"
    secret_key   = "minioadmin"
    skip_credentials_validation = true
    skip_region_validation     = true
    skip_metadata_api_check    = true
    force_path_style           = true

    # Для блокировок используем DynamoDB в LocalStack
    dynamodb_table = "terraform_locks"
    dynamodb_endpoint = "http://localhost:4566"
```

```
}  
}
```

5.7. Скрипты для симуляции конфликта

```
bash
```

```
# Терминал 1 - Инженер А
```

```
cd environments/dev
```

```
terraform apply --auto-approve
```

```
# (одновременно) Терминал 2 - Инженер Б
```

```
cd environments/dev
```

```
terraform apply --auto-approve
```

```
# -> Ожидаем ошибку блокировки
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист (название работы, ФИО, вариант, группа)
2. Цель и задачи работы (из п. 1–2 настоящего документа)
3. Ответы на контрольные вопросы (п. 4)
4. Теоретическая часть с описанием:
5. Проблем управления состоянием в команде
6. Архитектуры выбранного backend с блокировками
7. Схемы взаимодействия компонентов
8. Схема инфраструктуры (диаграмма с указанием компонентов state management)
9. Конфигурация backend (backend.tf с комментариями)
10. Terraform конфигурация тестовой инфраструктуры (основные ресурсы)
11. Результаты выполнения:
12. Инициализация: terraform init (скриншоты)
13. Применение: terraform apply (скриншоты)
 - Демонстрация блокировки (сообщение об ошибке)

- Force-unlock (скриншоты)
- Состояние в хранилище (список объектов)
- 14. Восстановление из бэкапа (демонстрация)
- 15. Заключение (достигнуты ли цель и задачи, выводы)
- 16. Список использованных источников

6.2. Технические требования

- Отчет оформляется в формате **Markdown** (.md) или **PDF**
- Код должен быть представлен в виде текстовых блоков с указанием языка
- Скриншоты должны содержать временные метки для демонстрации последовательности
- Объем отчета: 6–15 страниц

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Развертывание хранилища состояния (MinIO/PostgreSQL)	10
Настройка backend с поддержкой блокировок	15
Создание базовой инфраструктуры по варианту	15
Демонстрация конфликта (параллельный apply)	15
Демонстрация force-unlock	10
Ответы на контрольные вопросы	10
Качество отчета и документации	10

Критерий	Макс. баллов
Резервное копирование/восстановление состояния	5
Защита (устные ответы)	10
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Все компоненты работают, блокировки корректно предотвращают параллельные изменения, force-unlock выполнен безопасно, настроены бэкапы, контрольные вопросы проработаны, отчет полный
Хорошо	75-89	Backend настроен с блокировками, конфликт воспроизведен, но есть недочеты в демонстрации force-unlock или бэкапов, ответы на вопросы неполные
Удовлетворительно	60-74	Backend настроен без блокировок; либо блокировки есть, но не продемонстрирован конфликт; отчет неполный
Неудовлетворительно	<60	Backend не настроен; задание не выполнено; отчет отсутствует

7.3. Штрафные баллы

Нарушение	Штраф
State не централизован (хранится локально)	-15
Блокировки не реализованы	-15
Отсутствует демонстрация конфликта	-10
Использован -lock=false (обход блокировок)	-10

Лабораторная работа №3

Внедрение Terragrunt для DRY-конфигураций

1. Цель работы: Применить Terragrunt для устранения дублирования кода в Terraform-конфигурациях, организовав иерархическую структуру управления инфраструктурой с наследованием параметров, централизованным управлением backend'ами и зависимостями между модулями для обеспечения DRY-принципа (Don't Repeat Yourself).

Формируемые компетенции: ПК-2 (ИД-1, ИД-3, ИД-4, ИД-7)

2. Задачи работы

1. Изучить принципы работы Terragrunt и его отличия от нативного Terraform
2. Реорганизовать существующие Terraform-конфигурации из ЛР №1-2 в структуру Terragrunt
3. Вынести общие параметры (регион, теги, backend настройки) в корневой terragrunt.hcl
4. Настроить наследование параметров для различных окружений (dev/staging/prod)
5. Реализовать зависимости между модулями через блок dependency
6. Организовать централизованное управление версиями модулей

7. Продемонстрировать добавление нового окружения с минимальными изменениями конфигурации

3. Краткое теоретическое обоснование

3.1. Проблема дублирования в нативном Terraform

При управлении несколькими окружениями (dev, staging, prod) нативный Terraform приводит к дублированию кода:

```
environments/
├── dev/
│   ├── main.tf      # полностью повторяет staging
│   ├── variables.tf # отличается только значениями
│   ├── outputs.tf
│   └── backend.tf
├── staging/
│   ├── main.tf      # почти такой же как dev
│   ├── variables.tf
│   ├── outputs.tf
│   └── backend.tf
└── prod/
    ├── main.tf      # и снова
    └── ...
```

Последствия дублирования:

- Изменение логики требует правки 3+ файлов
- Высокий риск ошибок при ручном копировании
- Сложность поддержки и ревью
- Нарушение принципа DRY

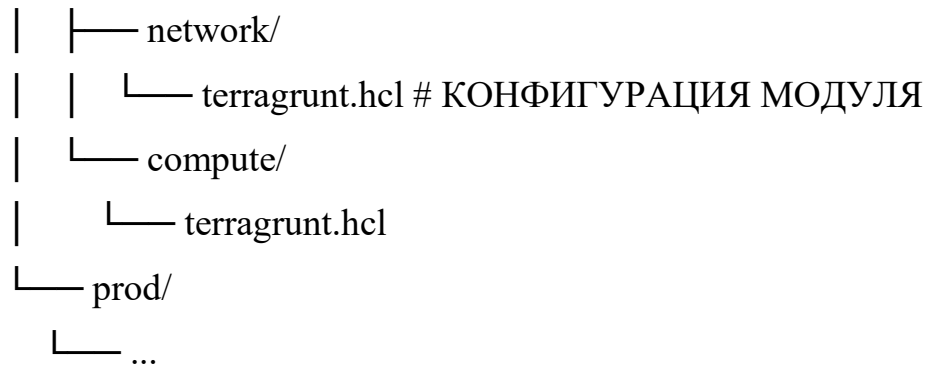
3.2. Что такое Terragrunt?

Terragrunt - это тонкая оболочка над Terraform, предоставляющая дополнительные возможности:

Возможность	Описание
DRY конфигурации	Вынос общих параметров в родительские файлы
Управление backend'ами	Централизованная настройка хранения состояния
Управление версиями модулей	Автоматическая подстановка версий из одного места
Зависимости между модулями	Автоматический порядок apply с передачей outputs
Выполнение команд на нескольких модулях	terragrunt run-all для batch-операций
Before/after хуки	Выполнение скриптов до/после Terraform команд

3.3. Архитектура Terragrunt

```
└─ infrastructure-live/      # РЕПОЗИТОРИЙ КОНФИГУРАЦИЙ
  └─ terragrunt.hcl          # КОРНЕВОЙ ФАЙЛ (глобальные
                             параметры)
    └─ account.hcl           # Файл уровня аккаунта (опционально)
      └─ region/
        └─ terragrunt.hcl    # Файл уровня региона
          └─ environments/
            └─ dev/
              └─ terragrunt.hcl # Файл окружения
```



3.4. Основные блоки Terragrunt

Блок	Назначение	Пример
terraform { source = "..." }	Указывает источник модуля	git::https://.../modules/network?ref=v1.0.0
include	Наследование параметров из родительского файла	include "root" { path = find_in_parent_folders() }
dependency	Объявление зависимости от другого модуля	dependency "network" { config_path = "./network" }
inputs	Передача переменных в модуль	inputs = { vpc_cidr = "10.0.0.0/16" }
generate	Генерация файлов (например, <u>provider.tf</u>)	generate "provider" { path = "provider.tf" ... }
remote_state	Настройка backend хранения состояния	remote_state { backend = "s3" ... }

3.5. Наследование через include

```

# infrastructure-live/terragrunt.hcl (корневой)
locals {

```

```

common_tags = {
  Organization = "NCFU"
  ManagedBy    = "Terragrunt"
}
}

# infrastructure-live/environments/dev/terragrunt.hcl
include "root" {
  path = find_in_parent_folders()
}

locals {
  environment = "dev"
}

inputs = {
  environment = local.environment
  tags        = merge(local.common_tags, { Environment = local.environment })
}

```

3.6. Зависимости между модулями

```

# environments/dev/compute/terragrunt.hcl
include "root" {
  path = find_in_parent_folders()
}

dependency "network" {
  config_path = "../network"

  # Опционально: переопределение конфигурации mock для plan
  mock_outputs = {
    vpc_id      = "vpc-mock"
    subnet_id   = "subnet-mock"
  }
  mock_outputs_allowed_terraform_commands = ["plan", "validate"]
}

inputs = {
  vpc_id      = dependency.network.outputs.vpc_id
  subnet_id   = dependency.network.outputs.subnet_id
}

```

3.7. Управление версиями модулей

```

# environments/dev/network/terragrunt.hcl
terraform {
  source = "git::https://gitlab.ncfu.ru/terraform-modules/network.git//.?ref=v1.2.0"
}

```

```
# Версию можно вынести в переменную
locals {
  module_version =
read_terragrunt_config("${get_parent_terragrunt_dir()}/versions.hcl")
}

terraform {
  source = "git::https://gitlab.ncfu.ru/terraform-
modules/network.git//.?ref=${local.module_version.locals.network_version}"
}
```

3.8. Основные команды Terragrunt

Команда	Назначение	Аналог в Terraform
terragrunt init	Инициализация модуля	terraform init
terragrunt plan	План изменений	terraform plan
terragrunt apply	Применение изменений	terraform apply
terragrunt destroy	Уничтожение ресурсов	terraform destroy
terragrunt run-all apply	Применение во всех модулях	—
terragrunt output	Получение outputs	terraform output
terragrunt hclfmt	Форматирование .hcl файлов	—

3.9. DRY-принцип в действии

До Terragrunt (на каждого окружения):

- main.tf - 100+ строк
- variables.tf - 30+ строк
- outputs.tf - 20+ строк
- backend.tf - 15 строк

Итого на окружение: ~165 строк × 3 окружения = ~500 строк кода

После Terragrunt (на каждое окружение):

- terragrunt.hcl - 15-20 строк (только отличия)

Экономия: ~90% дублирования кода

4. Контрольные вопросы

1. Что такое Terragrunt и какую проблему он решает в Terraform?
2. В чем принципиальное отличие Terragrunt от Terraform?
3. Что означает DRY-принцип и как Terragrunt его реализует?
4. Как работает наследование конфигурации через include и find_in_parent_folders()?
5. Для чего используется блок dependency? Как он помогает при работе с модулями?
6. Что такое mock_outputs и в каких ситуациях они полезны?
7. Как централизованно управлять версиями модулей в Terragrunt?
8. В чем разница между terragrunt apply и terragrunt run-all apply?
9. Как Terragrunt управляет backend'ами? Где конфигурируется хранение state?
10. Какие хуки (before/after) поддерживает Terragrunt и для чего они нужны?
11. Как организовать структуру репозитория для Terragrunt на несколько окружений?
12. Можно ли использовать Terragrunt без изменения существующих Terraform модулей?
13. Как Terragrunt обрабатывает ошибки при параллельном выполнении run-all?
14. Как передать outputs из одного модуля в другой через Terragrunt?

15. Какие best practices организации Terragrunt проектов вы знаете?
16. Как протестировать изменения модуля в одном окружении, не затрагивая другие?
17. Что делает команда `terragrunt hclfmt`?
18. Как настроить разные backend для разных окружений через Terragrunt?
19. Можно ли в Terragrunt использовать локальные пути к модулям во время разработки?
20. Как работает кэширование модулей в Terragrunt?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Взять Terraform конфигурации из ЛР №1-2 (модули `network` и `compute`)
2. Реорганизовать их в структуру Terragrunt согласно спецификации варианта
3. Настроить корневой `terragrunt.hcl` с общими параметрами
4. Создать минимум два окружения: `dev` и `staging` (для некоторых вариантов добавить `prod`)
5. Реализовать зависимости между модулями через `dependency`
6. Настроить централизованное управление backend'ами
7. Продемонстрировать, что добавление нового окружения требует минимальных изменений
8. Предоставить отчет о выполненной работе

5.2. Структура репозитория (общая для всех вариантов)

`infrastructure-live/`

<code>└─ terragrunt.hcl</code>	# Корневой с глобальными параметрами
<code>└─ versions.hcl</code>	# Версии модулей

```

└── _envcommon/           # Общие конфигурации для окружений
    ├── network.hcl       # Базовый конфиг для network
    └── compute.hcl       # Базовый конфиг для compute
└── environments/
    ├── dev/
    │   ├── terragrunt.hcl # Параметры окружения dev
    │   ├── network/
    │   │   └── terragrunt.hcl # Вызов модуля network в dev
    │   └── compute/
    │       └── terragrunt.hcl # Вызов модуля compute в dev (зависит
от network)
    ├── staging/
    │   ├── terragrunt.hcl
    │   ├── network/
    │   │   └── terragrunt.hcl
    │   └── compute/
    │       └── terragrunt.hcl
    └── prod/              # Только для вариантов 11-15
        └── ...

```

5.3. Варианты заданий

Вариант 1. Веб-приложение (stack: network + compute)

- **Компоненты:** VPC, подсети, веб-серверы (2 шт), балансировщик
 - **Параметры окружений:**
 - dev: instance_type = "small", count = 1
 - staging: instance_type = "medium", count = 2
 - prod: instance_type = "large", count = 3, enable_autoscaling = true
- (дополнительно)
- **Особенность:** добавить модуль load-balancer, зависящий от compute

Вариант 2. База данных (stack: network + database)

- **Компоненты:** VPC, подсети, PostgreSQL кластер (master + replica)

- **Параметры окружений:**

- dev: master_only, storage = 10GB
- staging: master + 1 replica, storage = 50GB, backup_enabled = true
- prod: master + 2 replicas, storage = 100GB, backup_enabled = true, backup_retention = 30

- **Особенность:** параметры репликации через dependency

Вариант 3. Кластер Kubernetes (stack: network + k8s)

- **Компоненты:** VPC, подсети, managed Kubernetes cluster, node groups

- **Параметры окружений:**

- dev: 1 node (small), version = "1.28"
- staging: 3 nodes (medium), version = "1.29"
- prod: 5-10 nodes auto-scaling, version = "1.30", private cluster

- **Особенность:** две node группы (system + workloads)

Вариант 4. Мониторинг (stack: network + monitoring)

- **Компоненты:** VPC, подсети, Prometheus, Grafana, Alertmanager

- **Параметры окружений:**

- dev: single-node, retention = 7d, basic dashboards
- staging: HA (2 nodes), retention = 30d, custom dashboards
- prod: HA + Thanos, retention = 90d, custom alerts

- **Особенность:** модуль мониторинга зависит от модуля compute (прошлой ЛР)

Вариант 5. CI/CD инфраструктура (stack: network + cicd)

- **Компоненты:** VPC, подсети, GitLab/Jenkins, runners

- **Параметры окружений:**

- dev: GitLab + 1 runner (docker executor)
- staging: GitLab HA + 3 runners (k8s executor)
- prod: GitLab Geo + 10 runners distributed

- **Особенность:** разные executor'ы для разных окружений
- Вариант 6. Обработка данных (stack: network + datalake)**

- **Компоненты:** VPC, подсети, Object Storage, Spark cluster
- **Параметры окружений:**
 - dev: 1 master + 1 worker, storage = 100GB
 - staging: 1 master + 3 workers, storage = 1TB
 - prod: HA master + 10 workers, storage = 10TB

Особенность: storage классы

Вариант 7. Очереди сообщений (stack: network + messaging)

- **Компоненты:** VPC, подсети, RabbitMQ или Kafka кластер
- **Параметры окружений:**
 - dev: 1 node (standalone)
 - staging: 3 node cluster, replication = 2
 - prod: 5 node cluster, replication = 3, tiered storage

- **Особенность:** разные версии брокера для окружений

Вариант 8. CDN и кэширование (stack: network + cache)

- **Компоненты:** VPC, подсети, Redis, CDN distribution
- **Параметры окружений:**
 - dev: Redis single, CDN disabled
 - staging: Redis Sentinel (1 master + 2 replicas), CDN enabled
 - prod: Redis Cluster (3+3), CDN + WAF enabled
- **Особенность:** зависимость от security modules

Вариант 9. Security и сеть (stack: network + security)

- **Компоненты:** VPC, подсети, Security Groups, WAF, DDoS protection
- **Параметры окружений:**
 - dev: basic security groups
 - staging: WAF enabled, rate limiting = 100rps
 - prod: WAF + DDoS, rate limiting = 1000rps, audit logging
- **Особенность:** security политики в отдельном модуле

Вариант 10. Machine Learning (stack: network + ml)

- **Компоненты:** VPC, подсети, GPU instances, Model serving

- **Параметры окружений:**

- dev: 1 GPU (development), cpu = 4
- staging: 2 GPUs (testing), cpu = 8, auto-scaling disabled
- prod: 4+ GPUs (production), cpu = 16, auto-scaling enabled

- **Особенность:** затраты (FinOps) — расчет стоимости GPU часов

Вариант 11. Microservices (stack: network + compute + service-mesh)

- **Компоненты:** VPC, подсети, K8s + Istio, API Gateway, 3 сервиса

- **Параметры окружений:**

- dev: single cluster, Istio disabled
- staging: single cluster, Istio mTLS (strict)
- prod: multi-cluster, Istio + mTLS + authorization policies

- **Особенность:** добавляется модуль service-mesh с зависимостью от k8s

Вариант 12. Serverless платформа (stack: network + serverless)

- **Компоненты:** VPC, подсети, Function as a Service, API Gateway, Queue

- **Параметры окружений:**

- dev: function timeout = 10s, memory = 128MB
- staging: timeout = 30s, memory = 256MB, provisioned concurrency = 1
- prod: timeout = 60s, memory = 512MB, provisioned concurrency = 10

- **Особенность:** зависимости между функциями

- **Вариант 13. Data Streaming (stack: network + streaming)**

- **Компоненты:** VPC, подсети, Apache Kafka, Flink, Schema Registry

- **Параметры окружений:**

- dev: Kafka 1 broker, Flink session mode
- staging: Kafka 3 brokers, Flink application mode

- prod: Kafka 5+ brokers, Flink HA, exactly-once semantics
- **Особенность:** retention policies per environment

Вариант 14. Документооборот (stack: network + docs + search)

- **Компоненты:** VPC, подсети, Document DB (MongoDB), Elasticsearch, API

- **Параметры окружений:**
 - dev: single node MongoDB + single node ES
 - staging: replica set MongoDB + 3 node ES cluster
 - prod: sharded MongoDB + 10 node ES cluster + Kibana
- **Особенность:** search и db отдельные модули

Вариант 15. Disaster Recovery (stack: network + dr)

- **Компоненты:** VPC (primary + secondary region), network, cross-region replication

- **Параметры окружений:**
 - dev: single region
 - staging: primary + backup region, RPO = 24h
 - prod: primary + active DR region, RPO = 15min, RTO = 1hour
- **Особенность:** два региона в одном Terragrunt проекте

(использование include)

5.4. Технические требования к реализации

Корневой terragrunt.hcl:

```
hcl
# infrastructure-live/terragrunt.hcl
locals {
  common_tags = {
    Project    = "ncfu-lab"
    ManagedBy  = "terragrunt"
    Environment = local.environment
  }
}
```

```

inputs = {
  common_tags = local.common_tags
}

remote_state {
  backend = "s3"
  config = {
    bucket      = "tf-state-${get_env("TF_VAR_org_id", "ncfu")}"
    key         = "${path_relative_to_include()}/terraform.tfstate"
    region      = "ru-central1"
    endpoint     = "https://storage.yandexcloud.net"
    skip_region_validation = true
  }
  generate = {
    path      = "backend.tf"
    if_exists = "overwrite_terragrunt"
  }
}

generate "provider" {
  path      = "provider.tf"
  if_exists = "overwrite_terragrunt"
  contents = <<EOF
terraform {
  required_version = ">= 1.5.0"
  required_providers {
    yandex = {
      source = "yandex-cloud/yandex"
      version = "~> 0.110"
    }
  }
}
EOF

```

```
    }  
  }  
}  
provider "yandex" {  
  zone = "ru-central1-a"  
}  
EOF  
}
```

Файл окружения environments/dev/terragrunt.hcl:

```
hcl  
include "root" {  
  path = find_in_parent_folders()  
}
```

```
locals {  
  environment = "dev"  
  region      = "ru-central1"  
}
```

```
inputs = {  
  environment = local.environment  
  region      = local.region  
}
```

Файл модуля environments/dev/network/terragrunt.hcl:

```
hcl  
include "root" {  
  path = find_in_parent_folders()  
}
```

```
include "envcommon" {
```

```
path = "${dirname(find_in_parent_folders())}/_envcommon/network.hcl"
}
```

```
inputs = {
  vpc_cidr = "10.0.0.0/16"
  subnet_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]
  enable_nat = false # dev не требует NAT
}
```

Общий конфиг _envcommon/network.hcl:

```
hcl
terraform {
  source          =          "git::https://gitlab.ncfu.ru/terraform-
modules/network.git//.?ref=${local.module_version}"
}
```

```
inputs = {
  environment = include.root.locals.environment
  tags       = include.root.inputs.common_tags
}
```

5.5. Демонстрация добавления нового окружения

После выполнения работы добавьте **окружение newenv (или test)** с минимальными изменениями и покажите:

1. Какие файлы были созданы/изменены (2-3 файла)
2. Сколько строк кода добавилось (~20-30 строк)
3. Что файл _envcommon/network.hcl остался без изменений
4. Сравнение с нативным Terraform (~200 строк дублирования)

5.6. Команды для проверки

```
bash
# Перейти в окружение
cd environments/dev/network
```

Инициализация (автоматически создаст backend.tf и provider.tf)

terragrunt init

План

terragrunt plan

Применить (если есть ресурсы)

terragrunt apply --auto-approve

Перейти в другой модуль этого же окружения

cd ../compute

terragrunt plan # автоматически использует outputs из network

Для всех модулей окружения

cd ../../ # environments/dev

terragrunt run-all plan

6. Требования к оформлению отчета

6.1. Структура отчета

1. **Титульный лист** (название работы, ФИО, вариант, группа)
2. **Цель и задачи работы** (из п. 1–2 настоящего документа)
3. **Ответы на контрольные вопросы** (п. 4)
4. **Теоретическая часть** с описанием:
 - Проблем дублирования в нативном Terraform
 - Архитектуры Terragrunt
 - DRY-принципа и его реализации
5. **Схема структуры репозитория** (дерево каталогов)
6. **Исходная конфигурация Terraform** (для сравнения — до Terragrunt)
7. **Terragrunt конфигурации:**

- Корневой terragrunt.hcl
- _envcommon/network.hcl и _envcommon/compute.hcl
- Файлы окружения dev и staging
- Файлы модулей
- 8. **Сравнительный анализ** (таблица строк кода до/после)
- 9. **Результаты выполнения команд:**
 - terragrunt init (скриншоты)
 - terragrunt plan (скриншоты)
 - terragrunt run-all plan
- 10. **Демонстрация добавления нового окружения** (какие файлы созданы)
- 11. **Заключение** (достигнуты ли цель и задачи, выводы о DRY)
- 12. **Список использованных источников**

6.2. Технические требования

- Отчет оформляется в формате **Markdown (.md)** или **PDF**
- Код должен быть представлен в виде текстовых блоков с указанием языка (**hcl**, **bash**)
 - Обязательно указать статистику сокращения кода (было X строк, стало Y строк)
 - Скриншоты команд и их выводов

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Организация структуры Terragrunt	15
Настройка корневого terragrunt.hcl (генерация provider/backend)	10

Критерий	Макс. баллов
Настройка <code>_envcommon</code> общих конфигураций	15
Реализация двух окружений (dev/staging)	15
Реализация зависимостей между модулями (dependency)	15
Ответы на контрольные вопросы	10
Демонстрация добавления нового окружения	10
Качество отчета и документации	10
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Полная структура Terragrunt, 2+ окружения, зависимости корректны, <code>_envcommon</code> используется для всех общих параметров, добавление нового окружения требует < 30 строк, контрольные вопросы проработаны
Хорошо	75-89	Terragrunt настроен, окружения созданы, но есть недочеты (частичное использование <code>_envcommon</code> ,

Оценка	Баллы	Требования
		запутанная структура), либо отсутствует демонстрация нового окружения
Удовлетворительно	60-74	Terragrunt работает, но используется только для одного окружения, либо отсутствуют зависимости, либо код дублируется
Неудовлетворительно	<60	Terragrunt не используется, либо структура не соответствует заданию, либо работа не выполнена

7.3. Штрафные баллы

Нарушение	Штраф
Отсутствует корневой terragrunt.hcl	-15
Нет генерации provider и backend	-10
Не используются dependency где необходимо	-10
Код дублируется (нарушен DRY)	-10
Отсутствует сравнение "до/после"	-5

Лабораторная работа №4

Написание политик на Rego для OPA

1. Цель работы: Освоить язык Rego для описания политик infrastructure as code с использованием Open Policy Agent (OPA), научиться

писать, тестировать и отлаживать политики безопасности и соответствия требованиям для инфраструктурных ресурсов.

Формируемые компетенции:

2. Задачи работы

1. Изучить синтаксис и основные конструкции языка Rego
2. Разработать политики для:
 - Запрета на использование публичных IP-адресов
 - Обязательного набора тегов для всех ресурсов
 - Ограничения размера дисков (не более заданного значения)
3. Написать unit-тесты для разработанных политик
4. Выполнить тестирование политик с использованием opa test
5. Проанализировать результаты тестирования и отладить политики
6. Подготовить отчет о выполненной работе

3. Краткое теоретическое обоснование

3.1. Что такое Open Policy Agent (OPA)?

Open Policy Agent (OPA) - это универсальный движок политик с открытым исходным кодом, который позволяет унифицировать применение политик во всем стеке технологий. OPA декомпозирует процесс принятия решений от логики приложения, позволяя определять политики декларативно и выполнять их в различных контекстах.

Ключевые особенности OPA:

- **Policy-as-Code** - политики хранятся в репозитории, версионируются, проходят code review
- **Декларативный язык Rego** - политики описывают *что* должно быть проверено, а не *как*
- **Универсальность** - работает с любыми JSON-данными (Kubernetes, Terraform, API-запросы и т.д.)
- **Высокая производительность** - частичная оценка, кэширование результатов

3.2. Язык Rego: основы синтаксиса

Rego - это декларативный язык, специально разработанный для выражения политик. Он основан на Datalog и поддерживает три вида равенства:

Оператор	Назначение	Пример
<code>:=</code>	Присваивание (создание новой переменной)	<code>x := 7</code>
<code>==</code>	Сравнение (проверка равенства)	<code>x == 7</code>
<code>=</code>	Унификация (поиск значений)	<code>[x, 2] = [3, y]</code>

Рекомендация: Используйте `:=` и `==` для более читаемых и безопасных политик.

3.3. Структура политики

Rego

```
package example_policy

# Значение по умолчанию (если ни одно правило не сработало)
default allow := false

# Правило, определяющее условие разрешения
allow if {
    input.user.role == "admin"
    input.resource.region != "restricted"
}

# Правило, возвращающее сообщение о нарушении
deny[msg] if {
    not allow
    msg := "Access denied: insufficient permissions"
}
```

3.4. Ключевые конструкции Rego

Правила с `if` (обязательно в OPA 1.0):

Начиная с OPA 1.0, использование `if` для всех определений правил является обязательным:

rego

```
# OPA 1.0 синтаксис
allow if {
    input.user == "admin"
}

# Старый синтаксис (более не поддерживается)
allow { input.user == "admin" }
```

Итерация по массивам и объектам:

```
# Итерация по массиву
allow if {
    some resource in input.resources
    resource.type == "compute"
}

# Доступ по индексу
deny[msg] if {
    instance := input.resources[_]           # _ означает "любой индекс"
    instance.type == "aws_instance"
    not contains(allowed_amis, instance.ami)
    msg := sprintf("AMI %s is not allowed", [instance.ami])
}
```

Вспомогательные функции:

rego

```
# Функция с входными параметрами
contains(arr, elem) true if {
    arr[_] == elem
}

# Функция с возвращаемым значением
get_instance_type(instance) := type if {
    type := instance.instance_type
}
```

3.5. Типовые паттерны политик

Паттерн 1: Список разрешенных значений

rego

```

allowed_instance_types = ["t2.micro", "t2.small", "t3.micro"]

deny[msg] if {
    resource := input.resources[_]
    resource.type == "aws_instance"
    not contains(allowed_instance_types, resource.instance_type)
    msg := sprintf("Instance type %s is not allowed", [resource.instance_type])
}

```

Паттерн 2: Проверка обязательных тегов

rego

```

required_tags = ["Name", "Environment", "Owner"]

deny[msg] if {
    resource := input.resources[_]
    resource.type == "aws_instance"
    required_tag := required_tags[_]
    not resource.tags[required_tag]
    msg := sprintf("Missing required tag '%s' on resource %s", [required_tag, resource.name])
}

```

Паттерн 3: Ограничения на значения

```

max_root_disk_size = 100 # GB

deny[msg] if {
    resource := input.resources[_]
    resource.type == "aws_instance"
    disk_size := resource.root_block_device.volume_size
    disk_size > max_root_disk_size
    msg := sprintf("Root disk size %d GB exceeds limit of %d GB", [disk_size, max_root_disk_size])
}

```

Паттерн 4: Запрет публичных IP

```

deny[msg] if {
    resource := input.resources[_]
    resource.type == "aws_instance"
    resource.associate_public_ip_address == true
    msg := sprintf("Resource %s has public IP assigned", [resource.name])
}

```

3.6. Тестирование политик

ОРА предоставляет встроенную поддержку unit-тестирования. Тесты определяются как специальные правила:

```
# test_policy_test.rego
```

```

package example_policy_test

import data.example_policy

test_allow_admin if {
    # Подготовка входных данных
    input := {"user": {"role": "admin"}, "resource": {}}

    # Выполнение проверки
    example_policy.allow with input as input
}

test_deny_public_ip if {
    resource := {"type": "aws_instance", "associate_public_ip_address": true,
"name": "test"}
    input := {"resources": [resource]}

    violations := {msg | example_policy.deny[msg] with input as input}
    count(violations) == 1 # Ожидаем одно нарушение
}

```

Команды для тестирования

```

# Запуск всех тестов в текущей директории
opa test .

# Запуск с подробным выводом
opa test . -v

# Запуск конкретного файла с тестами
opa test policies.rego policies_test.rego

# Запуск с покрытием
opa test . --coverage

```

3.7. Формат входных данных

Политики OPA работают с JSON-входными данными. Для Terraform-политик типовой формат:

```

{
  "resources": [

```



```

{
  "type": "aws_instance",
  "name": "web-server",
  "instance_type": "t2.micro",
  "ami": "ami-0aa7d40eeae50c9a9",
  "associate_public_ip_address": true,
  "tags": {
    "Name": "web-server",
    "Environment": "dev"
  },
  "root_block_device": {
    "volume_size": 30
  }
}
]
}

```

3.8. Инструменты для разработки политик

Инструмент	Назначение
opa eval	Выполнение одиночных запросов к политике
opa test	Запуск unit-тестов
opa fmt	Автоматическое форматирование Rego-кода
opa check	Проверка синтаксиса и безопасности
opa build	Сборка политик в bundle

4. Контрольные вопросы

1. Что такое Open Policy Agent (OPA) и для каких задач он применяется?
2. Какие преимущества дает подход Policy-as-Code?

3. Назовите три типа операторов равенства в Rego и поясните разницу между ними.
4. Для чего используется ключевое слово `default` в Rego?
5. Как выполняется итерация по элементам массива/объекта в Rego?
6. Что означают конструкции `input.resources[_]` и `[_]` в целом?
7. Как написать `unit`-тест для политики ОРА?
8. Какая команда используется для запуска тестов? Что означают флаги `-v` и `--coverage`?
9. Как в Rego реализовать проверку наличия элемента в списке?
10. Что такое `sprintf` и для каких целей он используется в политиках?
11. Какие изменения привнесла версия ОРА 1.0 в синтаксис Rego?
12. Каким образом ОРА интегрируется с Terraform?
13. Как передать внешние данные в политику при тестировании (конструкция `with`)?
14. В чем разница между правилами, возвращающими значение, и правилами, генерирующими набор (`set`) результатов?
15. Как организовать модульность политик с использованием `import`?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Разработать **3 политики** согласно спецификации варианта:
 - Политика А: запрет на публичные IP (варианты параметров см. ниже)
 - Политика В: обязательные теги (минимум 3 тега)
 - Политика С: ограничение размера дисков
2. Для каждой политики написать **как минимум 2 unit-теста**:
 - Тест, проверяющий успешное выполнение (положительный кейс)
 - Тест, проверяющий обнаружение нарушения (отрицательный кейс)

3. Все политики должны соответствовать синтаксису ОРА 1.0 (с использованием if)

4. Предоставить отчет с кодом политик, тестов и результатами выполнения

5.2. Варианты заданий

В каждом варианте задается тип ресурса (или несколько) и специфические параметры для политик:

Вар .	Тип ресурса	Политика А (публичные IP)	Политика В (теги)	Политика С (размер диска)
1	aws_instance	Запретить associate_public_ip_address = true	Name, Environment, Owner	Макс. 50 GB
2	aws_instance	Запретить public_ip в outputs	Project, CostCenter, ManagedBy	Макс. 100 GB
3	google_compute_instance	Запретить access_config	name, environment, team	Макс. 80 GB
4	yandex_compute_instance	Запретить nat = true	env, created-by, service	Макс. 60 GB
5	azurerm_linux_virtual_machine	Запретить public_ip_address = true	Environment, Application, Contact	Макс. 256 GB
6	openstack_compute_instance_v2	Запретить floating_ip	env, project, owner	Макс. 40 GB
7	aws_lb (Load Balancer)	Запретить scheme = "internet-facing"	Name, Environment, Terraform	N/A (логирование обязательно)
8	aws_db_instance (RDS)	Запретить publicly_accessible = true	Name, Environment, Backup	Макс. 200 GB storage

Вар .	Тип ресурса	Политика А (публичные IP)	Политика В (теги)	Политика С (размер диска)
9	google_sql_database_instance	Запретить ip_configuration.ipv4_enabled = true	environment, cost-center, owner	Макс. 100 GB
10	yandex_kubernetes_cluster	Запретить публичный endpoint	env, project, owner	(Политика В)
11	aws_redshift_cluster	Запретить publicly_accessible = true	Environment, Team, DataClassification	Макс. 1000 GB
12	azurerm_kubernetes_cluster	Запретить default_node_pool.enable_auto_scaling=false	Environment, Application, CostCenter	(Политика В)
13	aws_eks_cluster	Запретить endpoint_public_access = true	Environment, Owner, ClusterType	(Политика В)
14	google_container_cluster	Запретить private_cluster = false	environment, team, terraform	(Политика В)
15	yandex_mdb_postgresql_cluster	Запретить публичный доступ	env, project, backup-schedule	Макс. 500 GB

5.3. Дополнительные требования (усложнение)

Для оценки «отлично» необходимо выполнить одно из следующих усложнений:

Усложнение	Описание
А. Конфигурируемые политики	Параметры политик (размер диска, список тегов, допустимые типы инстансов) должны передаваться через __rule_settings__

Усложнение	Описание
В. Комбинированные правила	Политика должна учитывать комбинацию условий (например: запрет публичного IP и обязательные теги или ограничение размера)
С. Множественные ресурсы	Политика должна корректно обрабатывать несколько ресурсов во входных данных и возвращать список всех нарушений

5.4. Шаблон политики (рекомендуемая структура)

```
# policy_example.rego
package policy

import rego.v1

# =====
# Политика А: Запрет публичных IP
# =====

default allow_public_ip := true

deny_public_ip[msg] if {
    resource := input.resources[_]
    resource.type == "aws_instance"
    resource.associate_public_ip_address == true
    msg := sprintf("Resource %s has public IP assigned", [resource.name])
}

# =====
# Политика В: Обязательные теги
# =====

required_tags = ["Name", "Environment", "Owner"]

deny_missing_tags[msg] if {
    resource := input.resources[_]
    required_tag := required_tags[_]
    not resource.tags[required_tag]
    msg := sprintf("Resource %s missing required tag '%s'", [resource.name,
required_tag])
}
```

```
# =====
# Политика C: Ограничение размера диска
# =====

max_disk_size = 100 # GB

deny_disk_size[msg] if {
  resource := input.resources[_]
  resource.type == "aws_instance"
  size := resource.root_block_device.volume_size
  size > max_disk_size
  msg := sprintf("Resource %s disk size %d GB exceeds limit of %d GB",
[resource.name, size, max_disk_size])
}
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть с описанием:
 - Назначения ОРА и языка Rego
 - Особенности синтаксиса (3 типа равенства, **if**, **some**, **in**)
 - Принципов тестирования политик
5. Код разработанных политик (с комментариями)
6. Код unit-тестов (с комментариями)
7. Результаты выполнения тестов (скриншоты вывода opa test)
8. Анализ результатов (какие тесты прошли/упали, причины)
9. Заключение
10. Список использованных источников

6.2. Технические требования

11. Отчет оформляется в формате Markdown (.md) или PDF
12. Код должен быть представлен в виде текстовых блоков с указанием языка (rego)

13. Скриншоты должны содержать команды и их выводы
14. Все политики должны проходить opa check --strict

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Разработка политики А (публичные IP)	15
Разработка политики В (теги)	15
Разработка политики С (размер диска)	15
Написание unit-тестов (минимум 2 на политику)	20
Соответствие стандартам ОРА 1.0	10
Ответы на контрольные вопросы	10
Качество отчета и документации	10
Выполнение усложнения (для оценки "отлично")	+5
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Все 3 политики работают корректно, тесты покрывают положительные и отрицательные

Оценка	Баллы	Требования
		кейсы, код соответствует ОРА 1.0, выполнено усложнение, контрольные вопросы проработаны
Хорошо	75-89	Политики работают, есть тесты, но есть недочеты (неполное покрытие, отсутствует усложнение, не все ответы на вопросы)
Удовлетворительно	60-74	Политики работают, но тесты отсутствуют или неполные, есть синтаксические ошибки
Неудовлетворительно	<60	Политики не работают или не предоставлены

7.3. Штрафные баллы

Нарушение	Штраф
Использование старого синтаксиса (без if)	-10
Отсутствие тестов	-15
Политика не проходит ora check	-10
Отсутствие комментариев в коде	-5

Лабораторная работа №5

Интеграция ОРА с Terraform (Terraform ОРА)

1. Цель работы: Настроить автоматическую проверку инфраструктурных изменений с использованием Open Policy Agent (OPA) в CI/CD пайплайне, настроить Quality Gate для автоматической блокировки несоответствующих политикам изменений на этапе plan.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить принципы интеграции OPA с Terraform через JSON-представление plan'a
2. Установить и настроить инструменты: **Conftest** (рекомендуемый для CI) или raw OPA
3. Адаптировать политики из ЛР №4 для работы с Terraform plan JSON
4. Настроить CI/CD пайплайн (GitLab CI) с этапом проверки политик
5. Реализовать автоматическую остановку пайплайна при violation
6. Продемонстрировать успешное прохождение пайплайна для compliant-изменений и срыв для non-compliant

3. Краткое теоретическое обоснование

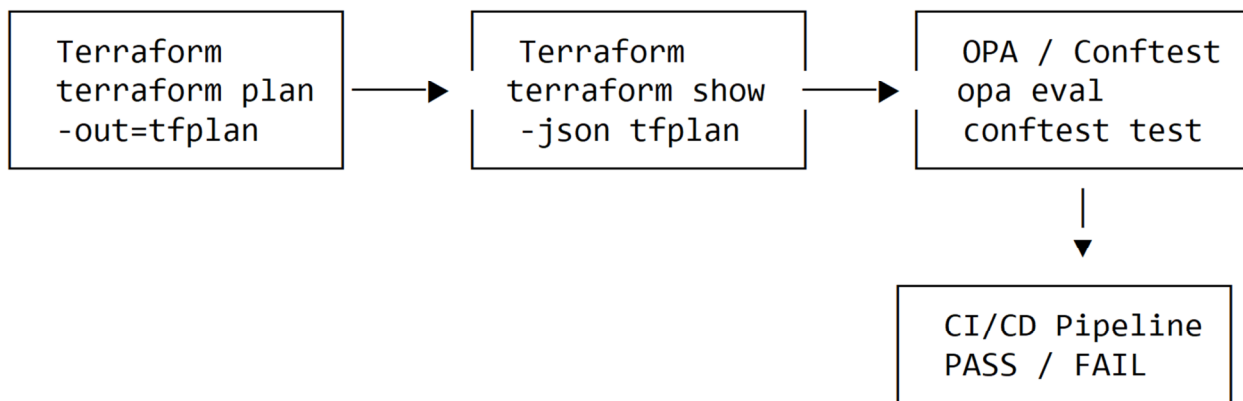
3.1. Зачем интегрировать OPA с Terraform?

Без автоматической проверки политик governance-правила живут в документации, код-ревью и институциональной памяти. Старший инженер ловит публичный S3-бакет на ревью, младший - пропускает. Compliance-проблемы всплывают на аудитах, а не до деплоя .

OPA решает эту проблему через Policy as Code:

- Инфраструктурные изменения проверяются автоматически
- Политики едины для всех команд и окружений
- Нарушения блокируются до применения изменений

3.2. Архитектура интеграции



Три шага интеграции :

```

# Шаг 1: Создать бинарный plan
terraform plan -out=tfplan

# Шаг 2: Экспортировать plan в JSON
terraform show -json tfplan > tfplan.json

# Шаг 3: Оценить политики
conftest test tfplan.json --policy policies/
  
```

3.3. Conftest vs raw OPA для CI/CD

Характеристика	Conftest	raw OPA
Установка	brew install conftest	brew install opa
Синтаксис	conftest test plan.json	opa eval --input plan.json 'data.terraform.deny'
Выходной код	Non-zero при violation	Требует --fail и правил deny

Характеристика	Conftest	raw OPA
Форматы вывода	JSON, TAP, JUnit, стандартный	--format (pretty, json, raw)
Рекомендации	Для CI/CD	Для сложных eval-запросов

Conftest - стандартный инструмент для запуска OPA политик против Terraform plan'ов в CI/CD. Он предоставляет CI-friendly вывод, множественные форматы и более простой синтаксис.

3.4. Структура Terraform plan JSON

```
{
  "format_version": "1.2",
  "terraform_version": "1.10.0",
  "planned_values": {...},
  "resource_changes": [
    {
      "address": "aws_s3_bucket.logs",
      "mode": "managed",
      "type": "aws_s3_bucket",
      "name": "logs",
      "change": {
        "actions": ["create"],
        "before": null,
        "after": {
          "bucket": "my-logs",
          "acl": "private",
          "server_side_encryption_configuration": [...]
        }
      }
    }
  ]
}
```

3.5. Ключевые поля для политик

Поле	Описание	Использование в политиках
resource_changes	Массив изменяемых ресурсов	Итерация по ресурсам
resource_changes[<u> </u>].type	Тип ресурса	resource.type == "aws_instance"
resource_changes[<u> </u>].change.actions	Действия (create, update, delete)	"create" in resource.change.actions
resource_changes[<u> </u>].change.before	Состояние до изменения	Проверка обновлений
resource_changes[<u> </u>].change.after	Состояние после изменения	Основная проверка

3.6. Адаптация политик из ЛР №4

Политики для opa eval (raw OPA) требуют специфической структуры:

```
# policy/terraform.rego
package terraform

import rego.v1

# Используйте default для правил, которые могут быть undefined
default deny := false
default allow := false

# Правило deny возвращает сообщения о нарушениях
deny contains msg if {
    resource := input.resource_changes[  
]
    resource.type == "aws_instance"
```

```

    resource.change.associate_public_ip_address == true
    msg := sprintf("Instance %s has public IP",
[resource.address
])
}

```

Для Conftest используется соглашение об именах: Conftest автоматически находит правила с именами deny, warn, violation.

```

# policy/security.rego для Conftest
package main

import future.keywords.in

deny contains msg if {
    resource := input.resource_changes[_
]
    resource.type == "aws_instance"
    resource.change.associate_public_ip_address == true
    msg := sprintf("Instance '%s' has public IP",
[resource.address
])
}

```

3.7. Quality Gate в CI/CD

Quality Gate - автоматическая проверка, останавливающая пайплайн при violation :

```

# GitLab CI этап проверки политик
opa-policy-check:
  stage: validate
  script:
    - terraform plan -out=tfplan
    - terraform show -json tfplan > tfplan.json
    - conftest test tfplan.json --policy policies/
  only:
    - merge_requests

```

При failure:

- Conftest возвращает exit code 1
- GitLab CI отмечает job как failed

- MR блокируется для merge до исправления

3.8. Conftest configuration

Для упрощения вызовов создайте .conftest.toml :

```
# .conftest.toml
[runner]
  policy = "policies"
  input = "tfplan.json"
  # Игнорировать правила warn при определении exit code
  # Полезно для постепенного внедрения
  fail_on_warn = false
```

4. Контрольные вопросы

1. Почему недостаточно полагаться только на код-ревью для проверки политик?
2. Какой формат данных OPA ожидает на входе для Terraform?
3. Какие команды Terraform необходимы для генерации plan в JSON?
4. В чем разница между terraform plan -out и terraform show -json?
5. Что такое Conftest и какая его роль в интеграции OPA с CI/CD?
6. Какие правила именования использует Conftest для обнаружения политик?
7. Как Conftest определяет exit code? При каком условии пайплайн будет остановлен?
8. Что делает опция fail_on_warn = true в .conftest.toml?
9. Какие действия terraform plan resource_changes могут содержать?
10. Почему terraform init выполняется с флагом -backend=false в CI?
11. Какие риски связаны с отсутствием backend при использовании data sources?
12. Как структурировать политики для работы с разными типами ресурсов?

13. Как в Rego проверить, что действие над ресурсом — это create, а не update?

14. Какие форматы вывода поддерживает Conftest для CI/CD?

15. Как протестировать политики локально перед запуском в CI/CD?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Адаптировать 3 политики из ЛР №4 для работы с Terraform plan JSON
2. Установить Conftest (рекомендуется) или настроить raw OPA
3. Настроить CI/CD пайплайн (GitLab CI или GitHub Actions) с этапом проверки
4. Продемонстрировать два сценария:
 - Успех: Compliance-изменения → пайплайн зеленый
 - Failure: Non-compliant изменения → пайплайн красный с сообщением
5. Предоставить отчет с кодом, конфигурациями и скриншотами

5.2. Варианты заданий

Вар	Платформа CI	Типы политик	Доп. требования
1	GitLab CI	Security (public IP, encryption)	Комбинированное правило (AND/OR)
2	GitLab CI	Cost (instance type limits)	Параметризация порогов через --data

Вар	Платформа CI	Типы политик	Доп. требования
3	GitLab CI	Tagging (required tags)	Проверка на create и update раздельно
4	GitHub Actions	Security Tagging +	Артефакты с tfplan.json
5	GitHub Actions	Cost (disk size limits)	Отдельные политики для dev/prod
6	GitHub Actions	Naming conventions	Custom warn правила
7	GitLab CI	Cross-service (S3 + IAM)	Модульные политики с import
8	GitLab CI	Region restrictions	opa eval вместо Conftest
9	GitLab CI	Backup requirements	Динамическое правило для всех ресурсов
10	GitHub Actions	Encryption requirements	JUnit отчет о нарушениях
11	GitLab CI	Security Group rules	Политика с под-правилами (helper functions)
12	GitHub Actions	Cost (multi-resource)	every keyword в Rego

Вар	Платформа CI	Типы политик	Доп. требования
13	GitLab CI	Compliance (PCI-DSS mock)	mock_outputs для тестов
14	GitHub Actions	Custom (ваш сценарий)	Интеграция с Slack уведомлениями
15	GitLab CI	Full stack (ваш кейс)	Конфигурация через внешний data.json

5.3. Шаблон политики для Terraform plan (Conftest)

```
# policies/security.rego
package main

import future.keywords.in
import future.keywords.if

# Политика А: Запрет публичных IP
deny contains msg if {
    resource := input.resource_changes[_]
    resource.type == "aws_instance""create" in resource.change.actions
    resource.change.after.associate_public_ip_address == true
    msg := sprintf("Instance '%s' has public IP",
    [resource.address
    ])
}

# Политика В: Обязательные теги
required_tags = {
    "Environment",
    "Owner",
    "Project"
}

deny contains msg if {
    resource := input.resource_changes[_]
    resource.type == "aws_instance""create" in resource.change.actions
    tags := resource.change.after.tags
```

```

        missing_tags := {tag | required_tags[tag
]; not tags[tag
]
    }
    count(missing_tags) > 0
    msg := sprintf("Instance '%s' missing tags: %v",
[resource.address, missing_tags
])
}

# Политика C: Ограничение размера диска
max_root_disk = 100

deny contains msg if {
    resource := input.resource_changes[_
    ]
    resource.type == "aws_instance""create" in resource.change.actions
    disk_size := resource.change.after.root_block_device[
    0
    ].volume_size
    disk_size > max_root_disk
    msg := sprintf("Instance '%s' root disk %dGB > %dGB limit",
[resource.address, disk_size, max_root_disk
])
}

```

5.4. Шаблон CI/CD конфигурации

GitLab CI (.gitlab-ci.yml)

```

image: hashicorp/terraform: 1.10

stages:
  - validate
  - policy-check
  - plan
  - apply

variables:
  TF_ROOT: ${CI_PROJECT_DIR
}

cache:
  paths:
    - ${TF_ROOT
} /.terraform

before_script:
  - cd ${TF_ROOT
}
  - terraform init -backend=false

```

```

terraform_validate:
  stage: validate
  script:
    - terraform fmt -check -recursive
    - terraform validate

policy-check:
  stage: policy-check
  before_script:
    - apk add --no-cache curl
    - curl -L https://github.com/open-policy-
agent/conftest/releases/download/v0.67.1/conftest_0.67.1_Linux_x86_64.tar.gz |
tar xz
    - mv conftest /usr/local/bin/
  script:
    - terraform plan -out=tfplan
    - terraform show -json tfplan > tfplan.json
    - conftest test tfplan.json --policy policies/ --output json
  artifacts:
    paths:
      - tfplan.json
    expire_in: 7 days
  rules:
    - if: $CI_MERGE_REQUEST_ID

terraform_plan:
  stage: plan
  script:
    - terraform plan -out=tfplan
  artifacts:
    paths:
      - tfplan
    expire_in: 7 days
  dependencies:
    - policy-check
  rules:
    - if: $CI_MERGE_REQUEST_ID

terraform_apply:
  stage: apply
  script:
    - terraform apply tfplan
  dependencies:
    - terraform_plan
  rules:
    - if: $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "main"

```

5.5. Конфигурация Conftest (.conftest.toml)

```
# .conftest.toml
[runner]
  policy = "policies"
  input = "tfplan.json"

  # fail_on_warn = false  # Не останавливать пайплайн на warnings
  # output = "json"      # Формат вывода: standard, json, tap, junit
```

5.6. Пример тестирования локально

```
# 1. Создать compliant конфигурацию
cat > main.tf << 'EOF'
resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafa1f0"
  instance_type = "t2.micro"
  associate_public_ip_address = false

  tags = {
    Environment = "dev"
    Owner       = "student"
    Project     = "lab5"
  }

  root_block_device {
    volume_size = 30
  }
}
EOF

# 2. Сгенерировать plan JSON
terraform init -backend=false
terraform plan -out=tfplan
terraform show -json tfplan > tfplan.json

# 3. Запустить проверку
conftest test tfplan.json --policy policies/
# Ожидаемый вывод: PASS

# 4. Создать non-compliant конфигурацию (с public IP)
# Изменить associate_public_ip_address = true

# 5. Повторить шаги 2-3
# Ожидаемый вывод: FAIL с сообщением
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть с описанием:
 - Архитектуры интеграции OPA + Terraform
 - Роли Conftest в CI/CD
 - Структуры Terraform plan JSON
5. Код адаптированных политик (3 штуки, с комментариями)
6. CI/CD конфигурация (полный файл)
7. Результаты выполнения:
 - Скриншот успешного пайплайна (compliant)
 - Скриншот упавшего пайплайна (non-compliant)
 - Лог вывода Conftest с violation
8. Сравнительный анализ (таблица поведения пайплайна)
9. Заключение
10. Список использованных источников

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Адаптация 3 политик для Terraform plan	20
Настройка Conftest / OPA	10
CI/CD конфигурация (GitLab CI/GitHub Actions)	20
Демонстрация успешного пайплайна	15

Критерий	Макс. баллов
Демонстрация срыва пайплайна	15
Ответы на контрольные вопросы	10
Качество отчета	10
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Все 3 политики работают корректно с plan JSON, CI/CD настроен, оба сценария продемонстрированы, сообщения об ошибках информативны
Хорошо	75-89	Политики работают, но есть недочеты (например, не разделены create/update), либо CI/CD без артефактов
Удовлетворительно	60-74	Confest запускается, но политики не полностью адаптированы, либо отсутствует один сценарий

Оценка	Баллы	Требования
Неудовлетворительно	<60	Интеграция не работает, политики не адаптированы

7.3. Штрафные баллы

Нарушение	Штраф
Использован raw OPA без Conftest в CI	-5
Нет проверки actions (create/update)	-10
Политики из ЛР №4 не адаптированы	-15
CI/CD не автоматизирован (ручной запуск)	-10

Лабораторная работа №6

Policy as Code в Kubernetes: OPA Gatekeeper

1. Цель работы: Научиться применять политики Policy as Code в рантайме Kubernetes с использованием OPA Gatekeeper, настроить автоматическую проверку и блокировку несоответствующих политикам workload'ов на уровне admission control.

Формируемые компетенции: ПК-2

2. Задачи работы:

- Установить OPA Gatekeeper в кластер Kubernetes с использованием Helm
- Изучить концепции ConstraintTemplate и Constraint в Gatekeeper
- Разработать ConstraintTemplate для запрета привилегированных контейнеров

- Создать Constraint, применяющий политику к Pod'ам
- Протестировать работу политики, создав Pod с нарушением
- Настроить различные режимы enforcement (deny, dryrun, warn)
- Предоставить отчет о выполненной работе

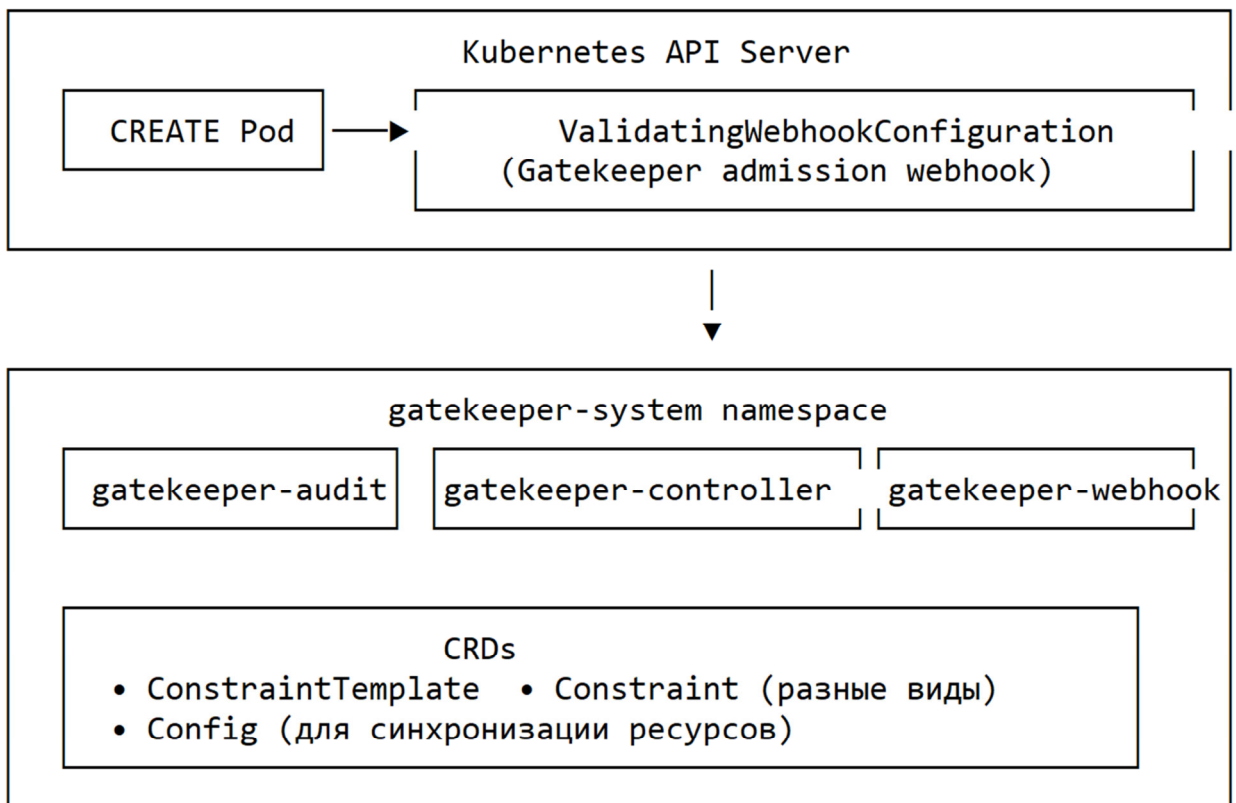
3. Краткое теоретическое обоснование

3.1. Что такое OPA Gatekeeper?

Gatekeeper - это admission controller для Kubernetes, использующий Open Policy Agent (OPA) для валидации запросов на создание и обновление Pod'ов и других ресурсов. Он расширяет OPA следующими возможностями:

1. Extensible, parameterized policy library - расширяемая параметризуемая библиотека политик
2. Native Kubernetes CRDs - для инстанцирования политик (Constraints)
3. Native Kubernetes CRDs - для расширения библиотеки политик (ConstraintTemplates)
4. Audit functionality - аудит существующих ресурсов на соответствие политикам
5. External data support - поддержка внешних данных для принятия решений

3.2. Архитектура Gatekeeper



3.3. Ключевые концепции

Концепция	Описание	Аналогия
ConstraintTemplate	Определение логики проверки на Rego и схему параметров	"Класс" или "функция"
Constraint	Инстанцирование шаблона с конкретными параметрами и областью действия	"Объект" или "вызов функции"
enforcementAction	Действие при нарушении: deny, dryrun, warn	Режим применения

3.4. Структура ConstraintTemplate

ConstraintTemplate описывает как Rego-логику, так и схему параметров :

```

apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
  
```

```

metadata:
  name: k8spspprivilegedcontainer
spec:
  crd:
    spec:
      names:
        kind: K8sPSPPrivilegedContainer # Имя CRD для Constraints
      validation:
        openAPIV3Schema: # Схема параметров
          type: object
          properties:
            exemptNamespaces:
              type: array
              items:
                type: string
      targets:
        - target: admission.k8s.gatekeeper.sh
          rego: |
            package k8spspprivileged

            violation[{"msg": msg, "details": {}}] {
              c := input_containers[_]
              c.securityContext.privileged == true
              msg := sprintf("Privileged container is not allowed: %v,
securityContext: %v", [c.name, c.securityContext])
            }

            input_containers[c] {
              c := input.review.object.spec.containers[_]
            }

            input_containers[c] {
              c := input.review.object.spec.initContainers[_]
            }

```

3.5. Режимы enforcementAction

Gatekeeper поддерживает различные действия при нарушении политики :

Режим	Описание	Применение
deny	Полная блокировка запроса (стандарт)	Production
dryrun	Проверка без блокировки,	Тестирование

Режим	Описание	Применение
	только логирование	политик
warn	Предупреждение без блокировки	Постепенное внедрение
deny (по умолчанию)	Блокировка запроса	Окончательное применение

3.6. Структура Constraint

Constraint инстанцирует ConstraintTemplate с конкретными параметрами и областью действия:

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivilegedContainer # Соответствует names.kind из
ConstraintTemplate
metadata:
  name: psp-privileged-container
spec:
  enforcementAction: deny # Опционально: deny, dryrun, warn
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces: ["default"] # Опционально: ограничить namespaces
    excludedNamespaces: ["kube-system"] # Исключения
  parameters: # Параметры (по схеме из ConstraintTemplate)
    exemptNamespaces:
      - "gatekeeper-system"
```

3.7. Входные данные для Rego

В Rego-правилах Gatekeeper входные данные доступны через переменную `input`:

Поле	Описание
input.review.object	Проверяемый Kubernetes-объект

Поле	Описание
input.review.kind.kind	Тип ресурса (Pod, Deployment и т.д.)
input.review.operation	Операция (CREATE, UPDATE, DELETE)
input.review.userInfo	Информация о пользователе, выполнившем запрос
input.parameters	Параметры из Constraint

3.8. Gatekeeper vs OPA для Kubernetes

Характеристика	OPA (pure)	Gatekeeper
Интеграция с K8s	Через sidecar (kube-mgmt)	Native CRDs
Управление политиками	ConfigMap	ConstraintTemplate + Constraint
Параметризация	Сложная	Встроенная через CRD schema
Аудит	Ограниченный	Встроенный
Простота использования	Требует ручной настройки webhook	Автоматическая интеграция

4. Контрольные вопросы

1. Что такое OPA Gatekeeper и для каких задач он применяется в Kubernetes?
2. В чем отличие Gatekeeper от "чистого" OPA при работе с Kubernetes?

3. Что такое ConstraintTemplate и из каких компонентов он состоит?
4. Что такое Constraint и как он связан с ConstraintTemplate?
5. Какие enforcementAction поддерживаются в Gatekeeper? Чем они отличаются?
6. Как проверить работу политики в режиме dryrun перед включением блокировки?
7. Каким образом Gatekeeper получает доступ к другим ресурсам кластера (например, Pod'ам)?
8. Что такое data.inventory в Rego-правилах Gatekeeper?
9. Какие операции перехватывает admission webhook Gatekeeper?
10. Как исключить определенные namespace из действия политики?
11. Можно ли применять одну политику к разным типам ресурсов (Pod, Deployment, Service)?
12. Как протестировать Rego-логику локально перед применением в кластер?
13. Что произойдет, если Gatekeeper недоступен (как настроен failurePolicy)?
14. Как посмотреть аудит-логи Gatekeeper?
15. Какие best practices вы знаете для написания политик Gatekeeper?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Установить OPA Gatekeeper в кластер Kubernetes (Minikube/Kind) с помощью Helm
2. Разработать ConstraintTemplate для заданной политики (согласно варианту)
3. Разработать Constraint, применяющий политику к Pod'ам (или другим ресурсам)

4. Протестировать работу политики, создав compliant и non-compliant ресурсы
5. Продемонстрировать логи Gatekeeper при нарушении
6. Предоставить отчет с кодом, командами и скриншотами

5.2. Варианты заданий

Вар.	Политика	Цель	Усложнение
1	Запрет привилегированных контейнеров	Безопасность	Добавить исключения для определенных namespace
2	Запрет hostNetwork / hostPID	Безопасность	Разные enforcement для разных namespace
3	Обязательные labels	Governance	Параметризуемый список labels
4	Ограничение image registry	Supply chain	Список разрешенных registries
5	Запрет latest tag	Best practice	Предупреждение вместо блокировки (warn)
6	Ограничение ресурсов (limits/requests)	Cost/Stability	Минимальные и максимальные значения
7	Запрет root user	Безопасность	Исключения для специфичных образов
8	Запрет hostPort	Безопасность	Разрешить только для

Вар.	Политика	Цель	Усложнение
			ingress controllers
9	Принудительные tolerations	Cluster management	Проверка наличия конкретных tolerations
10	Запрет NodePort services	Безопасность	Для Services, не только Pods
11	Ограничение volume types	Безопасность	Запрет hostPath
12	Принудительные nodeSelector	Cluster management	Конкретные значения nodeSelector
13	Запрет automountServiceAccountToken	Безопасность	С проверкой для всех resources
14	Запрет insecure capabilities	Безопасность	Блокировка CAP_SYS_ADMIN и др.
15	Комбинированная политика	Security+Governance	2+ условия (privileged OR root)

5.3. Требования к реализации политик

Для всех вариантов:

1. Политика должна иметь параметризуемые исключения (например, exemptNamespaces)
2. Должны быть как минимум 2 теста (compliant и non-compliant)

3. Сообщение об ошибке должно быть информативным (указывать, что именно нарушено)

Пример: Политика запрета привилегированных контейнеров

ConstraintTemplate (template-privileged.yaml):

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspprivilegedcontainer
spec:
  crd:
    spec:
      names:
        kind: K8sPSPPrivilegedContainer
      validation:
        openAPIV3Schema:
          type: object
          properties:
            exemptNamespaces:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8spspprivileged

        violation[{"msg": msg}] {
          # Проверяем, не находится ли namespace в списке исключений
          exempt := input.parameters.exemptNamespaces[_]
          exempt == input.review.object.metadata.namespace
        } else = false {
          # Проверяем все контейнеры (обычные и init)
          container := input.review.object.spec.containers[_]
          container.securityContext.privileged == true
          msg := sprintf("Privileged container '%s' is not allowed in namespace '%s'", [container.name, input.review.object.metadata.namespace])
        }

        # Правило для init контейнеров
        violation[{"msg": msg}] {
          container := input.review.object.spec.initContainers[_]
          container.securityContext.privileged == true
          msg := sprintf("Privileged init container '%s' is not allowed", [container.name])
        }
      }
```


6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть с описанием:
5. Архитектуры OPA Gatekeeper
6. Концепций ConstraintTemplate и Constraint
7. Механизма admission webhook
8. Код ConstraintTemplate (с комментариями)
9. Код Constraint (с комментариями)
10. Код тестовых манифестов (compliant и non-compliant)
11. Результаты выполнения:
12. Скриншоты установки Gatekeeper
13. Скриншот успешного создания compliant Pod'a
14. Скриншот ошибки при создании non-compliant Pod'a
15. Скриншот логов Gatekeeper
16. Демонстрация dryrun режима (если реализовано)
17. Заключение
18. Список использованных источников

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Установка Gatekeeper	5
Разработка ConstraintTemplate	25

Критерий	Макс. баллов
Разработка Constraint	15
Демонстрация работы политики	20
Параметризация (exemptions и др.)	10
Ответы на контрольные вопросы	10
Качество отчета	10
Усложнение (для оценки "отлично")	+5
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Gatekeeper установлен, политика полностью работает, есть параметризация, продемонстрированы оба сценария (compliant/non-compliant), выполнено усложнение
Хорошо	75-89	Политика работает, но есть недочеты (нет параметризации, сообщения об ошибках)

Оценка	Баллы	Требования
		неинформативны)
Удовлетворительно	60-74	Политика работает в базовом варианте, но нет тестов или отчет неполный
Неудовлетворительно	<60	Gatekeeper не установлен, политики не работают

7.3. Штрафные баллы

Нарушение	Штраф
Нет параметров в ConstraintTemplate	-10
Нет проверки compliance	-10
Нет скриншотов логов	-5

Лабораторная работа №7

Cost as Code с Infracost

1. Цель работы: Научиться оценивать стоимость инфраструктурных изменений до их применения с использованием Infracost, интегрировать автоматический анализ затрат в CI/CD пайплайн и настроить пороговые значения для контроля бюджета на уровне Pull Request.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Установить и настроить Infracost CLI для анализа Terraform-конфигураций
2. Получить API-ключ Infracost Cloud (бесплатный уровень)

3. Настроить CI/CD пайплайн (GitLab CI или GitHub Actions) для автоматического запуска Infracost
4. Реализовать генерацию отчета о разнице стоимости между ветками
5. Настроить автоматический комментарий в Pull Request с результатами оценки
6. Создать пороговое правило (Guardrail) для блокировки изменений, превышающих бюджет
7. Продемонстрировать работу на двух сценариях (успех/отказ)

3. Краткое теоретическое обоснование

3.1. Проблема управления облачными затратами

Традиционно облачные затраты управляются постфактум - после того, как деньги уже потрачены. Инженеры могут не осознавать финансовых последствий своих инфраструктурных изменений: изменение типа инстанса, добавление реплик или включение функции могут добавить тысячи долларов к ежемесячному счету.

Основные вызовы FinOps:

1. Реактивный подход вместо проактивного
2. Отсутствие видимости затрат у разработчиков
3. Бюджеты ломаются до того, как команды узнают о проблеме
4. Упущенная экономия не возвращается

3.2. Что такое Infracost?

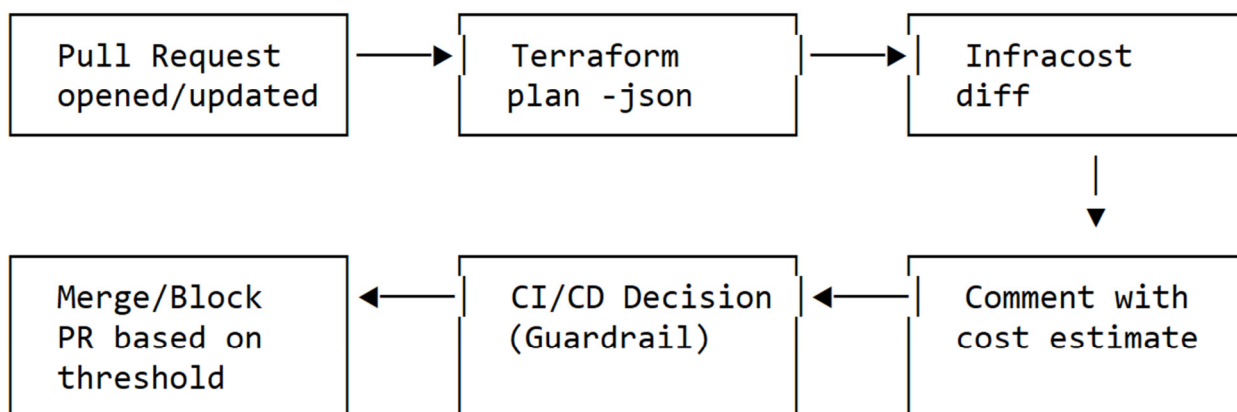
Infracost - это инструмент с открытым исходным кодом, который оценивает стоимость инфраструктуры как код (Terraform, Terragrunt, Pulumi) до ее развертывания. Он анализирует Terraform plan и показывает прогнозируемые ежемесячные затраты для каждого ресурса.

Ключевые возможности Infracost :

1. Поддержка AWS, Azure, Google Cloud, и других провайдеров

2. Интеграция с GitHub Actions, GitLab CI, Azure DevOps
3. Автоматические комментарии в Pull Request с разницей затрат
4. Guardrails (пороговые правила) для контроля бюджета

3.3. Принцип работы



Процесс включает:

- Baseline - оценка затрат на целевой ветке (main/master)
- Diff - оценка затрат на PR-ветке и сравнение с baseline
- Comment - публикация результата в PR
- Guardrail - проверка пороговых значений и блокировка при превышении

3.4. Guardrails и пороговые значения

Infracost Guardrails позволяют автоматически проверять затраты на соответствие бюджетным ограничениям .

Типы порогов :

Тип	Описание	Пример использования
Cost change	Абсолютное изменение затрат	Блокировать при добавлении > \$200/мес

Тип	Описание	Пример использования
Cost change percentage	Процентное изменение	Уведомлять при росте > 25%
Monthly cost threshold	Общий бюджет	Блокировать при превышении \$10,000/мес

Действия при нарушении :

1. notify - отправка уведомлений (email, Slack, Teams)
2. comment - добавление информации в PR
3. block - блокировка слияния до подтверждения

3.5. Формат вывода Infracost

Пример комментария в PR :

```
## 💰 Infracost estimate

**Monthly cost change:** $34.38 → $136.29 ▲ 296%

| Resource | Cost change |
|:---|:---|
| aws_instance.web[0] | $17.19 → $68.14 ▲ 296% |
| aws_instance.web[1] | $17.19 → $68.14 ▲ 296% |

**Summary:** Cost increase of $294/month (~$3,500/year)
```

3.6. Структура Infracost конфигурации

Файл infracost.yml определяет проекты для анализа:

```
version: 0.1

projects:
  - name: dev-environment
    path: environments/dev
    terraform_var_files:
      - terraform.tfvars
    terraform_workspace: dev
```

```
- name: prod-environment
  path: environments/prod
  terraform_var_files:
    - prod.tfvars
```

4. Контрольные вопросы

1. Какую проблему решает Infracost в контексте FinOps?
2. В чем разница между infracost breakdown и infracost diff?
3. Как Infracost получает данные о ценах на облачные ресурсы?
4. Что такое Guardrails и какие типы порогов поддерживаются?
5. Какие действия можно настроить при нарушении порогового значения?
6. Какова роль Infracost Cloud в процессе оценки затрат?
7. Для чего нужен API-ключ Infracost?
8. Как Infracost интегрируется с GitLab CI? Приведите пример конфигурации.
9. Что такое --behavior в команде infracost comment? Какие значения поддерживаются?
10. Как можно настроить блокировку PR при превышении бюджета?
11. Какие ограничения имеет Infracost при оценке затрат?
12. Почему важно запускать Infracost после terraform init?
13. Какие форматы вывода поддерживает Infracost для CI/CD?
14. Как можно протестировать Guardrails локально, не отправляя PR?
15. В чем преимущество подхода Shift-left в управлении затратами?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Зарегистрироваться в Infracost Cloud и получить API-ключ
3. Установить Infracost CLI локально для тестирования
4. Настроить CI/CD пайплайн (GitLab CI или GitHub Actions) для автоматического запуска Infracost
5. Создать Guardrail с пороговым значением согласно варианту

6. Продемонстрировать два сценария:
7. Успех: Изменение в рамках бюджета → PR проходит
8. Failure: Изменение превышает бюджет → PR блокируется с комментарием
9. Предоставить отчет с кодом, конфигурациями и скриншотами

5.2. Варианты заданий

Вар.	CI/CD платформа	Тип порога	Значение порога	Cloud провайдер
1	GitHub Actions	Абсолютное увеличение	> \$50	AWS
2	GitHub Actions	Процентное увеличение	> 30%	AWS
3	GitHub Actions	Бюджет (общий)	> \$1000	AWS
4	GitHub Actions	Абсолютное увеличение	> \$100	GCP
5	GitHub Actions	Процентное увеличение	> 50%	GCP
6	GitHub Actions	Бюджет (общий)	> \$500	GCP
7	GitLab CI	Абсолютное увеличение	> \$30	Azure
8	GitLab CI	Процентное	> 20%	Azure

Вар.	CI/CD платформа	Тип порога	Значение порога	Cloud провайдер
		увеличение		
9	GitLab CI	Бюджет (общий)	> \$2000	Azure
10	GitLab CI	Комбинированный	Абс > \$50 ИЛИ % > 25%	AWS
11	GitHub Actions	Комбинированный	Бюджет > \$800 И % > 40%	GCP
12	GitLab CI	Несколько guardrails	Разные для dev/prod	AWS
13	GitHub Actions	Кастомный скрипт	Проверка конкретного ресурса	AWS
14	GitLab CI	Абсолютное уменьшение	< -\$20 (экономия)	AWS
15	GitHub Actions	Многоуровневый	Warn > 50,Block>50,Block> 200	AWS

5.3. Требования к реализации

Для всех вариантов:

CI/CD пайплайн должен запускаться автоматически при создании/обновлении PR

1. Пайплайн должен выполнять terraform init, plan, infracost breakdown и infracost diff
2. В PR должен публиковаться комментарий с разницей затрат
3. При превышении порога CI/CD пайплайн должен падать (exit code $\neq 0$)
4. Необходимо продемонстрировать оба сценария в отчете

5.4. Шаблон конфигурации (GitLab CI)

Файл .gitlab-ci.yml:

```
image: hashicorp/terraform:1.10

stages:
  - plan
  - cost-estimate

variables:
  TF_ROOT: ${CI_PROJECT_DIR}
  INFRACOST_API_KEY: $INFRACOST_API_KEY

cache:
  paths:
    - ${TF_ROOT}/.terraform

before_script:
  - cd ${TF_ROOT}
  - terraform init -backend=false

terraform_plan:
  stage: plan
  script:
    - terraform plan -out=tfplan
    - terraform show -json tfplan > tfplan.json
  artifacts:
    paths:
      - tfplan.json
    expire_in: 7 days
  rules:
    - if: $CI_MERGE_REQUEST_ID

infracost:
  stage: cost-estimate
  image: infracost/infracost:ci-latest
  dependencies:
    - terraform_plan
  script:
    # Получение baseline из целевой ветки
```

```

- apk add --no-cache git
- git fetch origin $CI_MERGE_REQUEST_TARGET_BRANCH_NAME
- git checkout $CI_MERGE_REQUEST_TARGET_BRANCH_NAME -- ${TF_ROOT}
- |
  if [ -f "infracost.yml" ]; then
    infracost breakdown --config-file infracost.yml \
      --format json --out-file /tmp/infracost-base.json
  else
    infracost breakdown --path=${TF_ROOT} \
      --format json --out-file /tmp/infracost-base.json
  fi

# Генерация diff из текущей ветки
- git checkout $CI_COMMIT_SHA -- ${TF_ROOT}
- |
  if [ -f "infracost.yml" ]; then
    infracost diff --config-file infracost.yml \
      --format json --compare-to /tmp/infracost-base.json \
      --out-file /tmp/infracost.json
  else
    infracost diff --path=${TF_ROOT} \
      --format json --compare-to /tmp/infracost-base.json \
      --out-file /tmp/infracost.json
  fi

# Публикация комментария в Merge Request
- infracost comment gitlab --path=/tmp/infracost.json \
  --repo=$CI_PROJECT_PATH \
  --merge-request=$CI_MERGE_REQUEST_IID \
  --gitlab-token=$GITLAB_TOKEN \
  --behavior=update
rules:
- if: $CI_MERGE_REQUEST_ID

```

5.5. Шаблон конфигурации (GitHub Actions)

Файл `.github/workflows/infracost.yml`:

```

name: Infracost Cost Estimation

on:
  pull_request:
    paths:
      - '**.tf'
      - '**.tfvars'

jobs:
  infracost:
    name: Infracost
    runs-on: ubuntu-latest
    permissions:

```

```

  contents: read
  pull-requests: write

env:
  TF_ROOT: environments/production

steps:
  - name: Checkout base branch
    uses: actions/checkout@v4
    with:
      ref: '${{ github.event.pull_request.base.ref }}'

  - name: Setup Infracost
    uses: infracost/actions/setup@v3
    with:
      api-key: '${{ secrets.INFRACOST_API_KEY }}'

  - name: Generate baseline
    run: |
      infracost breakdown --path=${TF_ROOT} \
        --format=json \
        --out-file=/tmp/infracost-base.json

  - name: Checkout PR branch
    uses: actions/checkout@v4

  - name: Generate diff
    run: |
      infracost diff --path=${TF_ROOT} \
        --format=json \
        --compare-to=/tmp/infracost-base.json \
        --out-file=/tmp/infracost.json

  - name: Post PR comment
    uses: infracost/actions/comment@v1
    with:
      path: /tmp/infracost.json
      behavior: update

```

5.6. Настройка Guardrail (Infracost Cloud)

После интеграции Infracost в CI/CD, необходимо настроить Guardrail:

1. Войти в Infracost Cloud dashboard
2. Перейти в Governance > Cost Guardrails
3. Создать новый Guardrail с параметрами:

4. `name`: "Budget Threshold Guardrail"
5. `description`: "Block PR if cost increase exceeds threshold"

```

6.
7. thresholds:
8.   - type: cost_change
9.     value: 50                                # $50 monthly
10.    action: block
11.
12.   - type: percent_change
13.     value: 25                                # 25% increase
14.     action: notify                            # Notify Slack, Teams
15.
16. notifications:
17.   - type: email
18.     recipients: ["team-lead@example.com"]
19.   - type: slack
20.     webhook_url: "https://hooks.slack.com/services/..."

```

5.7. Тестовые сценарии

Сценарий 1: Успех (изменение в рамках бюджета)

```

# Изменение: t2.micro → t2.small (+~$10/month)
resource "aws_instance" "web" {
  instance_type = "t2.small" # вместо "t2.micro"
}

```

Ожидаемый результат: PR проходит, комментарий показывает изменение.

Сценарий 2: Отказ (превышение бюджета)

```

# Изменение: t2.micro → t3.xlarge (+~$150/month)
resource "aws_instance" "web" {
  instance_type = "t3.xlarge" # превышает порог $50
}

```

Ожидаемый

результат: PR блокируется, CI/CD пайплайн падает, комментарий содержит предупреждение.

5.8. Локальное тестирование

```
# Установка Infracost
curl -L https://github.com/infracost/infracost/releases/latest/download/infracost-linux-amd64.tar.gz
| tar xz
sudo mv infracost /usr/local/bin/

# Настройка API ключа
export INFRACOST_API_KEY=your-api-key

# Запуск оценки для директории
infracost breakdown --path=environments/dev

# Сравнение двух конфигураций
infracost diff --path=environments/dev --compare-to environments/prod
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть с описанием:
5. Проблем управления затратами в облаке
6. Принципов работы Infracost
7. Концепции Guardrails
8. Установка и настройка Infracost (скриншоты)
9. CI/CD конфигурация (полный файл)
10. Guardrail конфигурация (скриншоты из Infracost Cloud)
11. Результаты выполнения:
12. Скриншот успешного PR с комментарием
13. Скриншот заблокированного PR
14. Лог CI/CD пайплайна
15. Анализ результатов (таблица затрат)
16. Заключение
17. Список использованных источников

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Установка и настройка Infracost	10
Получение API-ключа и настройка секретов	5
CI/CD конфигурация	25
Настройка Guardrail в Infracost Cloud	15
Демонстрация успешного сценария	15
Демонстрация сценария отказа (блокировка)	15
Ответы на контрольные вопросы	10
Качество отчета	5
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Полная интеграция Infracost с CI/CD, оба сценария работают корректно, Guardrail настроен и блокирует PR, комментарии информативны
Хорошо	75-89	Интеграция работает, но есть недочеты (нет блокировки,

Оценка	Баллы	Требования
		только уведомление) либо отсутствует один сценарий
Удовлетворительно	60-74	Infracost запускается, но комментарий не публикуется в PR, либо нет Guardrail
Неудовлетворительно	<60	Infracost не настроен, задание не выполнено

7.3. Штрафные баллы

Нарушение	Штраф
API-ключ закоммичен в репозиторий	-10
Нет автоматического комментария в PR	-15
Guardrail не настроен	-15
Отсутствует скриншот CI/CD	-5

Лабораторная работа №8

FinOps: автоматическая оптимизация затрат в Kubernetes с Karpenter

1. Цель работы: Научиться настраивать автоматическое масштабирование кластера Kubernetes на основе фактической загрузки с

использованием Karpenter, реализовать экономию затрат через динамическое управление узлами и настройку бюджетных алертов для контроля расходов.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить принципы работы Karpenter как современного инструмента автоматического провижининга узлов
2. Установить и настроить Karpenter в кластере Kubernetes (Minikube/Kind с симуляцией или реальный EKS/Yandex Cloud)
3. Настроить NodePool с поддержкой Spot-инстансов и политиками консолидации
4. Реализовать автоматическое масштабирование на основе кастомных метрик (Prometheus + HPA + Karpenter)
5. Настроить механизм "засыпания" кластера (scale to zero) для нерабочего времени
6. Добавить бюджетный алерт (через Prometheus Alertmanager или внешние инструменты)
7. Продемонстрировать экономию затрат на практических сценариях

3. Краткое теоретическое обоснование

3.1. Проблема: плата за простаивающие ресурсы

Типичная среда разработки и тестирования активна лишь около 40 часов в неделю, однако предприятия платят за полные 168 часов. Кластеры продолжают работать ночью, в выходные и праздники, генерируя счета за неиспользуемые ресурсы.

Ключевые вызовы FinOps в Kubernetes :

1. Request vs. Reality gap - контейнеры запрашивают больше ресурсов, чем реально используют

2. Jagged headroom - небольшие неиспользуемые промежутки памяти, накапливающиеся в кластере
3. Legacy thinking - чрезмерно жесткие ограничения на типы инстансов
4. Hidden EBS tax - неоптимизированные корневые диски узлов

3.2. Karpenter: современный подход к автомасштабированию

Karpenter - это инструмент провижининга узлов, который вместо предопределенных Auto Scaling Groups напрямую создает EC2-инстансы (или аналоги) оптимального типа и размера для ожидающих подов .

Ключевые возможности Karpenter:

1. Мгновенная реакция на неподходящие поды (секунды вместо минут)
2. Выбор оптимального семейства инстансов (c, m, r, t - compute, general, memory, burstable)
3. Поддержка Spot-инстансов со стратегиями разнообразия
4. Автоматическая консолидация - уплотнение и замена узлов на более дешевые

3.3. Консолидация: ключ к экономии

Karpenter автоматически обнаруживает узлы, которые можно оптимизировать, и заменяет их :

Тип консолидации	Когда происходит	Экономия
WhenEmptyOrUnderutilized	Узел частично недогружен	Замена на меньший/дешевый инстанс
WhenEmpty	Узел полностью пуст	Удаление узла
Replace	Другой тип инстанса	Замена на более

Тип консолидации	Когда происходит	Экономия
	дешевле/эффективнее	выгодный

Пример из практики :

1. Запущены 2 узла m5.large (8GB RAM + 2 vCPU)
2. Нагрузка снижается до 1 пода с 1GB RAM
3. Karpenter определяет, что дешевле использовать один c5.large
4. Процесс: cordon → drain → terminate → создание нового узла c5.large
5. Итог: работа на более дешевом инстансе, двойная экономия

3.4. Spot-инстансы для дополнительной экономии

Spot-инстансы (прерываемые) могут давать экономию до 70-90% по сравнению с On-Demand. Karpenter умеет управлять Spot-пулами, автоматически восстанавливая вытесненные поды.

3.5. scale-to-zero: засыпание кластера

Истинная экономия включает не только подбор правильных инстансов, но и их выключение в простое :

```
# Подход: cron job только масштабирует поды, но не останавливает ноды
kubectl scale deployment --all --replicas=0 # ❌ не хватает
```

Правильная реализация scale-to-zero требует:

1. Масштабирования подов → 0
2. Настройки Karpenter на агрессивную консолидацию (consolidateAfter: 30s)
3. Перехвата входящего трафика через Ingress для «пробуждения»

3.6. Механизм пробуждения (Wake-on-Request)

Ingress может перехватывать запросы к спящему окружению, удерживать их, восстанавливать кластер, а затем направлять трафик :

annotations:

qovery.com/wake-on-request: "true"

qovery.com/idle-timeout: "4h"

3.7. Контроль затрат и алерты

Для управления бюджетом необходима система мониторинга:

1. Kubecost / OpenCost для визуализации затрат
2. Prometheus + Alertmanager для алертов при превышении бюджета
3. OPA/Gatekeeper для принудительной маркировки ресурсов (cost-center)

3.8. Отличие подхода от традиционного

Аспект	Cluster Autoscaler	Karpenter
Подход	Работает с ASG (заранее определенные пулы)	Непосредственно выбирает лучший инстанс
Скорость	Минуты	Секунды
Консолидация	Ограниченная	Продвинутая (WhenUnderutilized + Replace)
Spot	Сложно	Встроенная поддержка
Экономия	Средняя	Высокая (20-50% дополнительно)

4. Контрольные вопросы

1. Какая основная причина переплаты за облачные ресурсы в Kubernetes (request vs utilization gap)?
2. Что такое Karpenter и чем он отличается от Cluster Autoscaler?
3. Какие типы консолидации поддерживает Karpenter? Объясните разницу.
4. Как работает механизм consolidateAfter и для чего он нужен?
5. Что происходит при установке аннотации karpenter.sh/do-not-disrupt: "true"?
6. Какие преимущества дают Spot-инстансы? Какие риски?
7. Почему обычный kubectl scale replicas=0 не останавливает оплату за узлы?
8. Как организовать механизм «пробуждения» спящего кластера через Ingress?
9. Какие метрики необходимо отслеживать для оценки эффективности FinOps?
10. Как Karpenter выбирает тип инстанса для размещения подов?
11. Что такое NodePool в Karpenter и какие параметры там настраиваются?
12. Как ограничить максимальное количество CPU/памяти в кластере через Karpenter?
13. Какие инструменты используются для визуализации затрат в Kubernetes?
14. Как защитить критический под от вытеснения при консолидации?
15. Какие метрики Karpenter нужно мониторить для отладки (консолидация, запуск узлов)?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

Развернуть Karpenter в кластере Kubernetes (можно использовать Kind + симуляцию провайдера, Minikube, или реальный облачный кластер)

Настроить NodePool с поддержкой Spot-инстансов и политикой консолидации

Развернуть демонстрационное приложение с HPA на основе CPU/кастомных метрик

Настроить бюджетный алерт (например, через Prometheus + Alertmanager или Kubecost)

Доказать экономию - показать, что кластер масштабируется до нуля/R1 в нерабочее время

Предоставить отчет с кодом, манифестами, скриншотами дашбордов

5.2. Варианты заданий

Вар.	Платформа	Тип экономии	Специфические требования
1	AWS EKS (симуляция)	Spot-инстансы + консолидация	Включить WhenEmptyOrUnderutilized, consolidateAfter=1m
2	AWS EKS (симуляция)	scale-to-zero по расписанию	CronJob для остановки в 20:00-09:00
3	AWS EKS (симуляция)	multi-arch (AMD64 + ARM64)	Graviton для экономии на compute-heavy
4	Yandex Cloud	Yandex Compute + Karpenter	Использование прерываемых ВМ (preemptible)
5	Kind + fake-provider	Демонстрация логики без облака	Эмуляция консолидации

Вар.	Платформа	Тип экономии	Специфические требования
6	AWS EKS	HPA на основе RabbitMQ queue depth	Кастомные метрики + Karpenter
7	AWS EKS	batch-защита + консолидация	Защита длительных джобов от прерывания
8	Yandex Cloud	Grafana-дашборд FinOps	Визуализация затрат, метрики Karpenter
9	AWS EKS	EBS-оптимизация	Минимальный root volume (gp3, 20GB)
10	AWS EKS	Multi-NodePool	dev (small+spot) vs prod (large+on-demand)
11	Kind	CI/CD + эфемерные среды	Поднятие сред под PR, удаление после
12	AWS EKS	Spot разнообразие	5+ семейств инстансов (c, m, r, t)
13	Yandex Cloud	Ingress-пробуждение	Спящий mode при нулевом трафике
14	AWS EKS	FinOps + Kubecost	Бюджет \$500/day, алерт при превышении
15	AWS EKS	Комбинированный	Spot + scale-to-zero + EBS оптимизация

5.3. Базовая конфигурация NodePool

`apiVersion`: karpenter.sh/v1

`kind`: NodePool

`metadata`:

`name`: finops nodepool

`spec`:

`template`:

`spec`:

`requirements`:

Поддержка spot и on-demand [citation:4]

- `key`: karpenter.sh/capacity-type

`operator`: In

`values`: ["spot", "on-demand"]

Семейства инстансов: вычислительные, general, память, burstable

- `key`: karpenter.k8s.aws/instance category

`operator`: In

`values`: ["c", "m", "r", "t"]

Минимум 4 поколение

- `key`: karpenter.k8s.aws/instance generation

`operator`: Gt

`values`: ["4"]

Архитектура: x86_64 (опционально ARM)

- `key`: kubernetes.io/arch

`operator`: In

`values`: ["amd64"]

`nodeClassRef`:

`name`: default

`limits`:

`cpu`: "100" # Общее ограничение


```
memory: 200Gi
disruption:
  consolidationPolicy: WhenEmptyOrUnderutilized # Агрессивная
консолидация [citation:5]
  consolidateAfter: 1m # Быстрая реакция [citation:2]
```

5.4. Защита критических нагрузок от консолидации

Для long-running batch-задач:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: critical_batch_job
spec:
  template:
    metadata:
      annotations:
        karpenter.sh/do-not-disrupt: "true" # [citation:3][citation:8]
    spec:
      containers:
        - name: worker
          image: myapp latest
          restartPolicy: Never
```

5.5. HPA на основе кастомной метрики

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
```

```
name: my app
minReplicas: 1
maxReplicas: 100
metrics:
- type: Resource
  resource:
    name: cpu
    target:
      type: Utilization
      averageUtilization: 60
- type: Pods
  pods:
    metric:
      name: messages_queue_depth # кастомная метрика
    target:
      type: AverageValue
      averageValue: 10
```

5.6. Бюджетный алерт (Prometheus + Alertmanager)

```
groups:
- name: finops
  rules:
- alert: DailyBudgetExceeded
  expr: |
    avg_over_time(kubecost_cluster_monthly_rate[12h]) * 30 > 500
  for: 1h
  annotations:
    summary: "Cluster budget may exceed $500/month"
    description: "Current projected cost: {{ $value }} USD"
- alert: KarpenterConsolidationNotWorking
  expr: |
```

```
karpenter_nodes Consolidated_total[10m] == 0
and karpenter_nodes count > 3
```

```
for: 30m
```

```
annotations:
```

```
summary: "No consolidation for 30 mins while nodes >3"
```

5.7. Cron-расписание для scale-to-zero (опционально)

```
apiVersion: batch/v1
```

```
kind: CronJob
```

```
metadata:
```

```
name: sleep nonprod
```

```
spec:
```

```
schedule: "0 20 * * 1-5" # 8 PM weekdays
```

```
jobTemplate:
```

```
spec:
```

```
template:
```

```
spec:
```

```
containers:
```

```
- name: scaler
```

```
image: bitnami/kubectl latest
```

```
command:
```

```
- /bin/sh
```

```
- c
```

```
- |
```

```
kubectl scale deployment -n staging --all --replicas=0
```

```
kubectl scale statefulset -n staging --all --replicas=0
```

```
restartPolicy: Never
```

5.8. Симуляция нагрузки для демонстрации

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: load generator
spec:
  replicas: 5
  template:
    spec:
      containers:
      - name: loader
        image: busybox latest
        command:
        - /bin/sh
        - -c
        - "while true; do dd if=/dev/urandom of=/dev/null bs=1M count=100;
done"

      resources:
        requests:
          cpu: "0.5"
          memory: "256Mi"
```

Демонстрация:

Создать нагрузку → Karpenter запускает новые узлы (секунды)

Удалить/уменьшить нагрузку → консолидация и удаление узлов (1-2 минуты)

Проверить логи Karpenter: `kubectl logs -n karpenter deployment/karpenter -c controller --tail=50`

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (Karpenter, консолидация, Spot-инстансы, scale-to-zero)

5. Архитектура решения (схема взаимодействия компонентов)
6. Конфигурация NodePool и EC2NodeClass
7. Манифесты HPA + нагрузки
8. Настройка алертов (Prometheus/Alertmanager/Kubecost)
9. Результаты выполнения:
- 10.Скриншоты логов Karpenter при запуске узла
- 11.Скриншоты логов при консолидации
- 12.Графики загрузки кластера (до/после)
- 13.Бюджетный алерт (скриншот)
- 14.Расчет экономии (сравнение без Karpenter / с Karpenter)
- 15.Заключение
- 16.Список использованных источников

6.2. Технические требования

Отчет оформляется в формате Markdown (.md) или PDF

Все манифесты должны быть в виде текстовых блоков с указанием языка

Расчет экономии должен быть с цифрами (в рублях/\$ или условных единицах)

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Установка и настройка Karpenter	15
Конфигурация NodePool с Spot/консолидацией	15
Демонстрация автоматического провижининга узлов	15
Демонстрация консолидации/масштабирования вниз	15

Критерий	Макс. баллов
Настройка бюджетного алерта	10
Демонстрация scale-to-zero (или его эмуляции)	10
Ответы на контрольные вопросы	10
Качество отчета и расчет экономии	10
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Karpenter полностью настроен, консолидация работает, Spot-инстансы включены, бюджетные алерты и scale-to-zero реализованы, расчёт экономии обоснован
Хорошо	75-89	Базовый Karpenter работает, консолидация есть, но нет Spot, либо неполные алерты
Удовлетворительно	60-74	Karpenter установлен, но в демо один сценарий (только рост), консолидация не показана
Неудовлетворительно	<60	Karpenter не установлен, задание не выполнено

7.3. Штрафные баллы

Нарушение	Штраф

Нарушение	Штраф
Нет демонстрации консолидации узлов	-10
Нет расчёта экономии	-10
Нет алертов	-10

Лабораторная работа №9

Security as Code: управление секретами с HashiCorp Vault

1. Цель работы: Научиться организовывать централизованное и безопасное управление секретами на платформе Kubernetes с использованием HashiCorp Vault, настроить аутентификацию приложений через Kubernetes Service Account и интегрировать Vault с приложением через sidecar injector для динамического получения секретов без их хранения в репозиториях или переменных окружения.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Развернуть HashiCorp Vault в кластере Kubernetes с помощью Helm
2. Настроить хранилище секретов (KV secrets engine) для статических секретов
3. Настроить Kubernetes authentication method для аутентификации приложений
4. Создать политики доступа (policies) и роли для Service Account
5. Развернуть Vault Agent Injector и настроить автоматическое внедрение секретов

6. Разработать демонстрационное приложение, получающее секреты из Vault
7. Продемонстрировать работу механизма динамического получения секретов

3. Краткое теоретическое обоснование

3.1. Проблема управления секретами в Kubernetes

Традиционный подход к управлению секретами в Kubernetes имеет ряд ограничений:

Проблема	Описание	Риски
Безопасность по умолчанию	Kubernetes Secrets хранятся в etcd в base64 (не зашифрованы)	Компрометация etcd → доступ ко всем секретам
Отсутствие аудита	Нет встроенного механизма логирования доступа к секретам	Невозможно отследить, кто и когда обращался к секрету
Статичность	Секреты нельзя динамически генерировать или ротировать	Утечка секрета требует передеплой приложений
Сложность управления	Секреты разбросаны по множеству namespace	Отсутствие централизованного контроля

3.2. HashiCorp Vault: решение для управления секретами

HashiCorp Vault - инструмент для централизованного управления секретами с поддержкой:

1. Шифрование секретов при хранении (encryption at rest)
2. Аудит всех операций с секретами
3. Динамические секреты (генерация по требованию с TTL)
4. Поддержка множества auth methods (Kubernetes, LDAP, JWT, токены)
5. Автоматическая ротация секретов

3.3. Способы интеграции Vault с Kubernetes

HashiCorp Vault предоставляет три основных способа интеграции с Kubernetes :

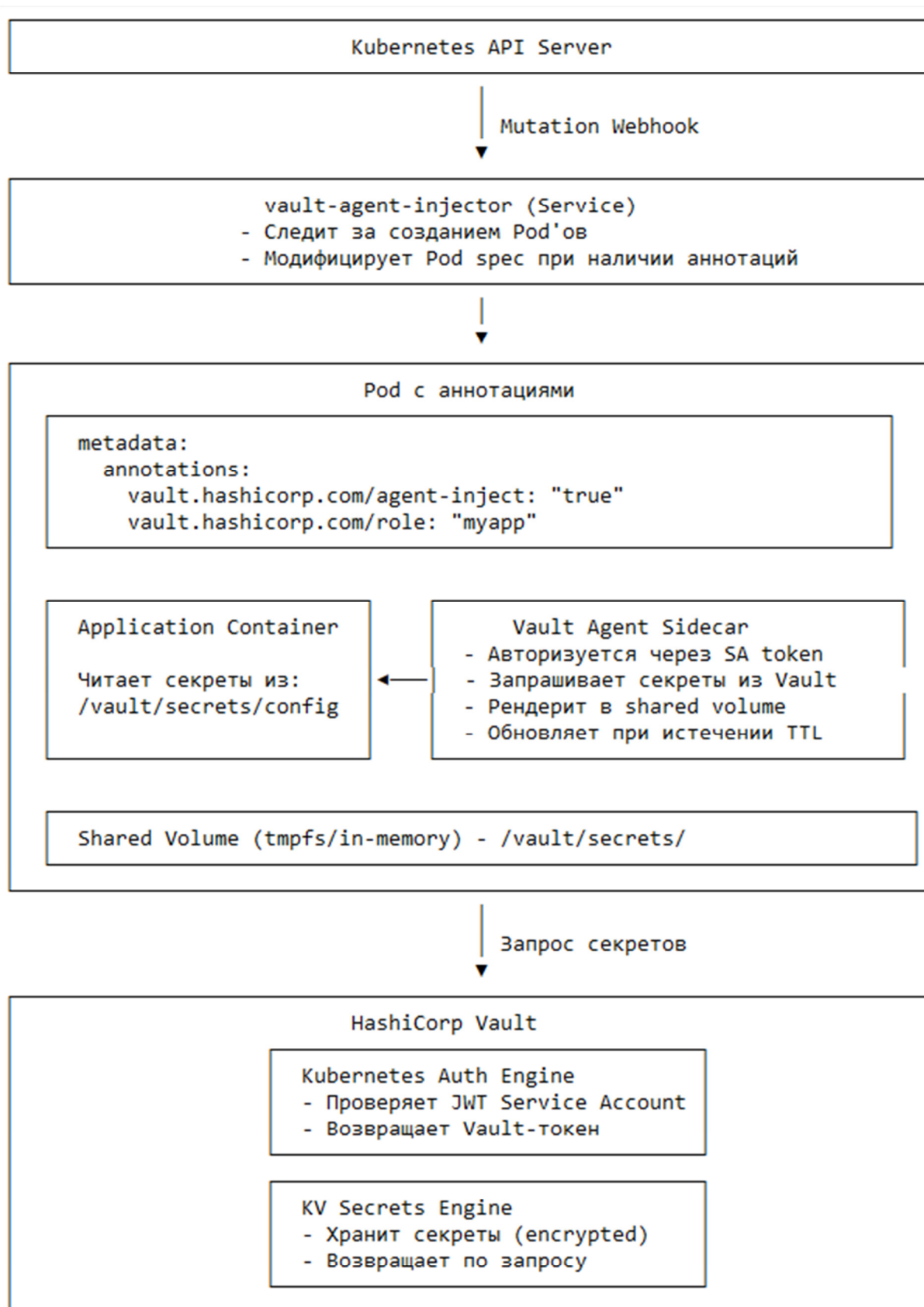
Характеристика	Vault Agent Injector (Sidecar)	Vault CSI Provider	Vault Secrets Operator	Выбор в ЛР
Структура	Sidecar контейнер в каждом Pod	DaemonSet на каждом узле	CRD + Controller в кластере	☒ Sidecar
Поддержка auth	Все методы (Kubernetes, JWT, AWS, GCP)	Только Kubernetes	Kubernetes, AppRole, AWS, GCP, JWT	—
Хранение секретов	Ephemeral volumes (tmpfs)	Ephemeral или K8s Secrets	K8s Secrets или эфемерно	—
Обновление секретов	Автоматическое (рендеринг при	Не поддерживается автоматически	Поддерживается через синхронизацию	☒ Auto-renew

Характеристика	Vault Agent Injector (Sidecar)	Vault CSI Provider	Vault Secrets Operator	Выбор в ЛР
	изменении)	ки		
Templating	Да (мощный шаблонизатор)	Нет	Да	☒ Важно
Нагрузка на Vault	Высокая (per-pod соединения)	Средняя (per-node)	Низкая (один контроллер)	—
Требуемые привилегии	Нет	Да (hostPath)	Нет	—

Почему выбран Agent Injector?:

1. Поддерживает все типы секретов (статический KV, динамические PKI, Database)
2. Не требует привилегированного доступа к узлам (в отличие от CSI)
3. Поддерживает автоматическое обновление секретов и токенов
4. Имеет мощный механизм шаблонизации для форматирования вывода секретов

3.4. Архитектура Vault Agent Injector



3.5. Аутентификация через Kubernetes Service Account

Vault использует Kubernetes TokenReview API для верификации service account токенов :

Процесс аутентификации:

1. Pod получает JWT-токен своего service account (монтируется в /var/run/secrets/kubernetes.io/serviceaccount/token)
2. Vault Agent отправляет этот токен в Vault
3. Vault обращается к Kubernetes API с запросом TokenReview
4. При успешной верификации Vault возвращает временный токен
5. Vault Agent использует токен для доступа к секретам

Конфигурация Kubernetes auth:

```
vault auth enable kubernetes
vault write auth/kubernetes/config kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"
```

3.6. Политики доступа (Policies)

Политики в Vault определяют, какие операции разрешены на каких путях :

```
# policy.hcl
path "secret/data/myapp/*" {
  capabilities = ["read"] # Только чтение
}

path "secret/data/myapp/config" {
  capabilities = ["read", "list"]
}

path "database/creds/myapp-db" {
  capabilities = ["read"]
}
```

3.7. Аннотации Vault Agent Injector

Для включения инъекции секретов в Pod необходимо добавить аннотации:

Аннотация	Описание	Пример

Аннотация	Описание	Пример
<code>vault.hashicorp.com/agent-inject</code>	Включить инъекцию	"true"
<code>vault.hashicorp.com/role</code>	Имя роли для аутентификации	"myapp"
<code>vault.hashicorp.com/agent-inject-secret-<file></code>	Путь к секрету в Vault	"secret/data/myapp/config"
<code>vault.hashicorp.com/agent-inject-template-<file></code>	Шаблон для форматирования	См. ниже
<code>vault.hashicorp.com/agent-requests-mem</code>	Лимиты памяти для sidecar	"64Mi"
<code>vault.hashicorp.com/agent-limits-cpu</code>	Лимиты CPU для sidecar	"250m"

3.8. Шаблонизация секретов

Vault Agent поддерживает мощный шаблонизатор для форматирования секретов:

```

annotations:
  vault.hashicorp.com/agent-inject-template-config: |
    {{- with secret "secret/data/myapp/config" -}}
    DATABASE_URL=postgres://{{ .Data.data.db_user }}:{{ .Data.data.db_pass }}@postgres:5432/myapp
    API_KEY={{ .Data.data.api_key }}
    JWT_SECRET={{ .Data.data.jwt_secret }}
    {{- end -}}

```

3.9. Динамические секреты

Vault может генерировать временные секреты по запросу (например, учетные данные для базы данных с TTL 1 час). При истечении TTL секрет автоматически отзывается.

3.10. Безопасность: изоляция секретов

Важно понимать: при использовании Vault Agent Injector секреты хранятся в памяти (tmpfs) и доступны только внутри Pod'a . Они не попадают в etcd, что снижает риск их компрометации.

4. Контрольные вопросы

1. Какие проблемы управления секретами решает HashiCorp Vault?
2. В чем отличие Vault от стандартных Kubernetes Secrets?
3. Какие методы интеграции Vault с Kubernetes существуют? Какой выбран в ЛР и почему?
4. Как работает механизм аутентификации через Kubernetes Service Account?
5. Для чего нужны политики (policies) в Vault?
6. Какую роль выполняет Vault Agent Injector? Как он модифицирует Pod'ы?
7. Какие аннотации необходимо добавить в Pod для включения инъекции секретов?
8. Что такое шаблонизация секретов и когда ее использовать?
9. Чем отличаются статические и динамические секреты?
10. Как происходит обновление секретов при их ротации в Vault?
11. Где физически хранятся секреты при использовании Agent Injector? Почему это безопасно?
12. Как ограничить доступ к секретам на уровне Kubernetes (RBAC)?
13. Какие типы секретов (secrets engines) поддерживает Vault?
14. Как организовать секрет-менеджмент для мультитенантного кластера?

15. Какие best practices вы знаете для управления секретами в production?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Установить Vault в кластер Kubernetes с помощью Helm
3. Настроить KV secrets engine и создать секреты согласно варианту
4. Настроить Kubernetes auth и создать политику/роль
5. Разработать демонстрационное приложение (скрипт на Python/bash/node), которое:
6. Читает секреты из файла (/vault/secrets/config)
7. Выводит их в лог (но не в stdout production!)
8. Демонстрирует, что секрет успешно получен
9. Продемонстрировать работу через kubectl logs

5.2. Варианты заданий

Ва р.	Тип секретов	Количество	Специфика	Усложнение
1	Database credentials	2	PostgreSQL	Динамические DB-секреты (Database engine)
2	API ключи сервисов	3	Google Maps, Stripe, Twilio	Шаблон JSON
3	JWT секреты и сертификаты	2 + 1 cert	PKI engine для TLS	PKI интеграция

Ва р.	Тип секретов	Количество	Специфика	Усложнение
4	Комбинация разных пути	2 пути	config и credentials	OPA для доступа
5	Ключи шифрования	2 ключа (AES-256)	Base64 encoded	Vault Transform engine
6	LDAP/AD учетные данные	3 параметра	user, domain, password	Множественны е файлы
7	S3/S3- совместимые ключи	access + secret	MinIO	AWS auth method
8	SMTP настройки	host, user, password, port	Mailtrap/Real SMTP	Сложный шаблон
9	Идентификат оры OAuth/OIDC	client_id, secret, redirect_uri	Google OAuth	AppRole auth
10	API Gateway ключи	3 ключа (rate limit tiers)	Kong	Проверка TTL
11	MongoDB replica set	3 набора creds	Разные роли (read/write/ad min)	Подключение к MongoDB

Ва р.	Тип секретов	Количество	Специфика	Усложнение
12	Redis Cache пароли	master + replica	Sentinel mode	Внедрение через env
13	Составной микросервис	5+ секретов	Multi- annotation	Разные пути
14	Kafka SASL/SSL	username, password, keystore	Schema Registry	PKI + Kafka
15	Ключи шифрования данных	2 ключа + ID	KMS Mock	Transit engine

5.3. Базовая конфигурация

Шаг 1. Установка Vault через Helm:

```
# Добавление репозитория HashiCorp
helm repo add hashicorp https://helm.releases.hashicorp.com
helm repo update

# Установка Vault (dev mode для лабораторной работы)
helm install vault hashicorp/vault \
  --namespace vault \
  --create-namespace \
  --set "server.dev.enabled=true" \
  --set "injector.enabled=true"

# Проверка установки
kubectl get pods -n vault
```

Шаг 2. Настройка секретов в Vault:

```
# Вход в pod Vault
kubectl exec -it -n vault vault-0 -- /bin/sh

# Включение KV v2 engine
vault secrets enable -path=secret kv-v2

# Создание секретов (пример для варианта 1)
vault kv put secret/myapp/database \
  db_host="postgres.prod.svc.cluster.local" \
  db_port="5432" \
  db_user="myapp_user" \
  db_pass="myapp_secure_password_2024"

# Выход
exit
```

Шаг 3. Настройка Kubernetes auth:

```
kubectl exec -it -n vault vault-0 -- /bin/sh

# Включение Kubernetes auth
vault auth enable kubernetes

# Настройка подключения к Kubernetes API
vault write auth/kubernetes/config \
  kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443"

# Создание политики (пример)
vault policy write myapp-policy - <<EOF
path "secret/data/myapp/*" {
  capabilities = ["read", "list"]
}
EOF

# Создание роли, связывающей service account с политикой
vault write auth/kubernetes/role/myapp \
  bound_service_account_names=myapp-sa \
  bound_service_account_namespaces=default \
  policies=myapp-policy \
  ttl=1h

exit
```

Шаг 4. Service Account для приложения:

```
# serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: myapp-sa
  namespace: default
```

Шаг 5. Демонстрационное приложение с аннотациями Vault:

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vault-demo-app
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vault-demo
  template:
    metadata:
      labels:
        app: vault-demo
      annotations:
        vault.hashicorp.com/agent-inject: "true"
        vault.hashicorp.com/role: "myapp"
        vault.hashicorp.com/agent-inject-secret-config:
"secret/data/myapp/database"
        vault.hashicorp.com/agent-inject-template-config: |
          {{- with secret "secret/data/myapp/database" -}}
          DATABASE_URL=postgres://{{ .Data.data.db_user }}:{{ .Data.data.db_pass
}}@{{ .Data.data.db_host }}:{{ .Data.data.db_port }}/myapp
          DB_USER={{ .Data.data.db_user }}
          DB_PASS={{ .Data.data.db_pass }}
          DB_HOST={{ .Data.data.db_host }}
          DB_PORT={{ .Data.data.db_port }}
          {{- end -}}
    spec:
      serviceAccountName: myapp-sa
      containers:
        - name: app
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args:
            - |
```

```

echo "=== Starting application ==="
echo "Secrets will be injected into /vault/secrets/"
echo "Waiting for secrets..."
sleep 5
if [ -f /vault/secrets/config ]; then
    echo "=== Secrets loaded successfully ==="
    cat /vault/secrets/config
    # Source the file to load env variables
    source /vault/secrets/config
    echo "=== Environment variables set ==="
    echo "DB_USER=$DB_USER"
    echo "DB_HOST=$DB_HOST"
    echo "DB_PORT=$DB_PORT"
    # DO NOT echo passwords in production!
else
    echo "ERROR: Secrets file not found!"
    exit 1
fi
echo "=== Simulation successful ==="
sleep infinity

```

5.4. Альтернативная реализация приложения (Python)

Для более реалистичного примера можно использовать Python:

```

# app.py
import os
import sys

def main():
    secrets_file = "/vault/secrets/config"

    if not os.path.exists(secrets_file):
        print(f"ERROR: Secrets file {secrets_file} not found")
        sys.exit(1)

    # Парсим переменные из файла
    with open(secrets_file, 'r') as f:
        for line in f:
            if '=' in line:
                key, value = line.strip().split('=', 1)
                os.environ[key] = value

    # Используем секреты из переменных окружения
    db_user = os.environ.get('DB_USER', 'unknown')
    db_host = os.environ.get('DB_HOST', 'unknown')
    db_port = os.environ.get('DB_PORT', 'unknown')

    print(f"Database connection: user={db_user}, host={db_host}, port={db_port}")
    # DO NOT print passwords in production!

```

```
return 0

if __name__ == "__main__":
    sys.exit(main())
```

Dockerfile для Python приложения:

```
FROM python:3.11-slim
COPY app.py /app.py
CMD ["python", "/app.py"]
```

5.5. Проверка работы

```
# Применить ServiceAccount и Deployment
kubectl apply -f serviceaccount.yaml
kubectl apply -f deployment.yaml

# Проверить Pod и логи
kubectl get pods
kubectl logs <pod-name> -c app

# Проверить Vault Agent sidecar логи
kubectl logs <pod-name> -c vault-agent

# Для отладки инъекции
kubectl describe pod <pod-name>
```

5.6. Ожидаемый результат

Логи приложения должны содержать:

```
=== Starting application ===
Secrets will be injected into /vault/secrets/
Waiting for secrets...
=== Secrets loaded successfully ===
DATABASE_URL=postgres://myapp_user:myapp_secure_password_2024@postgres.prod.svc.cluster.local:5432/m
yapp
DB_USER=myapp_user
DB_PASS=myapp_secure_password_2024
DB_HOST=postgres.prod.svc.cluster.local
DB_PORT=5432
=== Environment variables set ===
DB_USER=myapp_user
DB_HOST=postgres.prod.svc.cluster.local
DB_PORT=5432
=== Simulation successful ===
```

Логи sidecar контейнера (vault-agent) должны показывать успешную аутентификацию и получение секретов.

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы (п. 4)
4. Теоретическая часть с описанием:
5. Проблем управления секретами в Kubernetes
6. Архитектуры Vault и Agent Injector
7. Механизма аутентификации через Kubernetes Service Account
8. Схема взаимодействия компонентов (диаграмма)
9. Установка и настройка Vault (команды и их вывод)
10. Конфигурация секретов и политик (код)
11. Манифесты Kubernetes (ServiceAccount, Deployment)
12. Результаты выполнения:
13. Скриншоты установки Vault
14. Скриншот создания секретов
15. Скриншот логов приложения с секретами
16. Скриншот логов vault-agent
17. Анализ безопасности (почему этот подход безопаснее стандартных Secrets)

18.Заключение

19.Список использованных источников

6.2. Технические требования

1. Отчет оформляется в формате Markdown (.md) или PDF
2. Все команды и манифесты должны быть в текстовых блоках с указанием языка
3. Код приложения должен быть представлен полностью
4. Скриншоты должны быть четкими, с видимыми командами и выводами

7. Критерии оценивания

7.1. Шкала оценивания

Критерий	Макс. баллов
Установка Vault и Agent Injector	10
Настройка секретов в Vault	15
Настройка Kubernetes auth и политик	15
Создание Service Account	5
Разработка демонстрационного приложения	20
Настройка аннотаций и Deployment	15
Демонстрация работы (логи)	10
Ответы на контрольные вопросы	10

Критерий	Макс. баллов
Качество отчета	5
Итого	100

7.2. Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Vault установлен, секреты и политики настроены, приложение работает, секреты корректно инжектятся, логи демонстрируют успешное получение, выполнено усложнение (вариант с динамическими секретами)
Хорошо	75-89	Vault работает, приложение получает секреты, но есть недочеты (например, нет проверки ошибок, неоптимальный шаблон)
Удовлетворительно	60-74	Vault установлен, но приложение не работает или секреты инжектятся некорректно
Неудовлетворительно	<60	Vault не установлен, задание не выполнено

7.3. Штрафные баллы

Нарушение	Штраф
Секреты закоммичены в репозиторий	-15
Отсутствует ServiceAccount	-10
Нет демонстрации логов	-10

Лабораторная работа №10

SLSA-совместимая цепочка поставки: Sigstore и верификация образов

1. Цель работы: Обеспечить целостность и происхождение артефактов в конвейере поставки программного обеспечения с использованием Sigstore (Cosign) для подписи контейнерных образов, генерации SBOM и настройки admission controller для верификации подписей при деплое в Kubernetes.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить концепции SLSA (Supply-chain Levels for Software Artifacts) и Sigstore
2. Установить и настроить Cosign для подписи контейнерных образов
3. Сгенерировать ключевую пару (или настроить keyless signing)
4. Подписать контейнерный образ и сохранить подпись в OCI registry
5. Сгенерировать SBOM (Software Bill of Materials) для образа
6. Настроить Kubernetes admission controller для верификации подписей

7. Продемонстрировать успешный деплой подписанного образа и блокировку неподписанного

3. Краткое теоретическое обоснование

3.1. Проблема безопасности цепочки поставки

Современные атаки на программное обеспечение все чаще нацелены не на уязвимости в коде, а на цепочку поставки (supply chain). Злоумышленники могут:

1. Внедрить вредоносный код в open-source зависимости
2. Скомпрометировать CI/CD пайплайн и подменить образы
3. Атаковать registry и заменить легитимный образ
4. Ключевые вопросы безопасности:
5. Откуда взялся этот образ? Кто его собрал?
6. Был ли он изменен после сборки?
7. Какие компоненты в него входят?

3.2. SLSA (Supply-chain Levels for Software Artifacts)

SLSA - это framework для обеспечения безопасности цепочки поставки, определяющий уровни зрелости от 0 до 4 .

Уровень	Требования	Что дает
SL SA 1	Provenance (происхождение) в build-системе	Доказательство, что образ собран в CI
SL SA 2	Подпись provenance, host build service	Доказательство, что сборка выполнена в изолированной среде
SL	Неподделываемая provenance,	Сильные гарантии

Уровень	Требования	Что дает
SA 3	герметичная сборка	воспроизводимости
SL SA 4	Два независимых reviewer'a, воспроизводимая сборка	Максимальный уровень доверия

3.3. Sigstore и Cosign

Sigstore - это открытая корпорация, предоставляющая инструменты для подписи и верификации артефактов с использованием "keyless" подхода .

Cosign - основной инструмент Sigstore:

```
# Подпись образа
cosign sign --key cosign.key myregistry/myapp:latest

# Верификация
cosign verify --key cosign.pub myregistry/myapp:latest
```

Keyless signing (без управления ключами) использует временные сертификаты от Fulcio CA, привязанные к OIDC identity (GitHub Actions, Google Cloud, etc.).

3.4. SBOM (Software Bill of Materials)

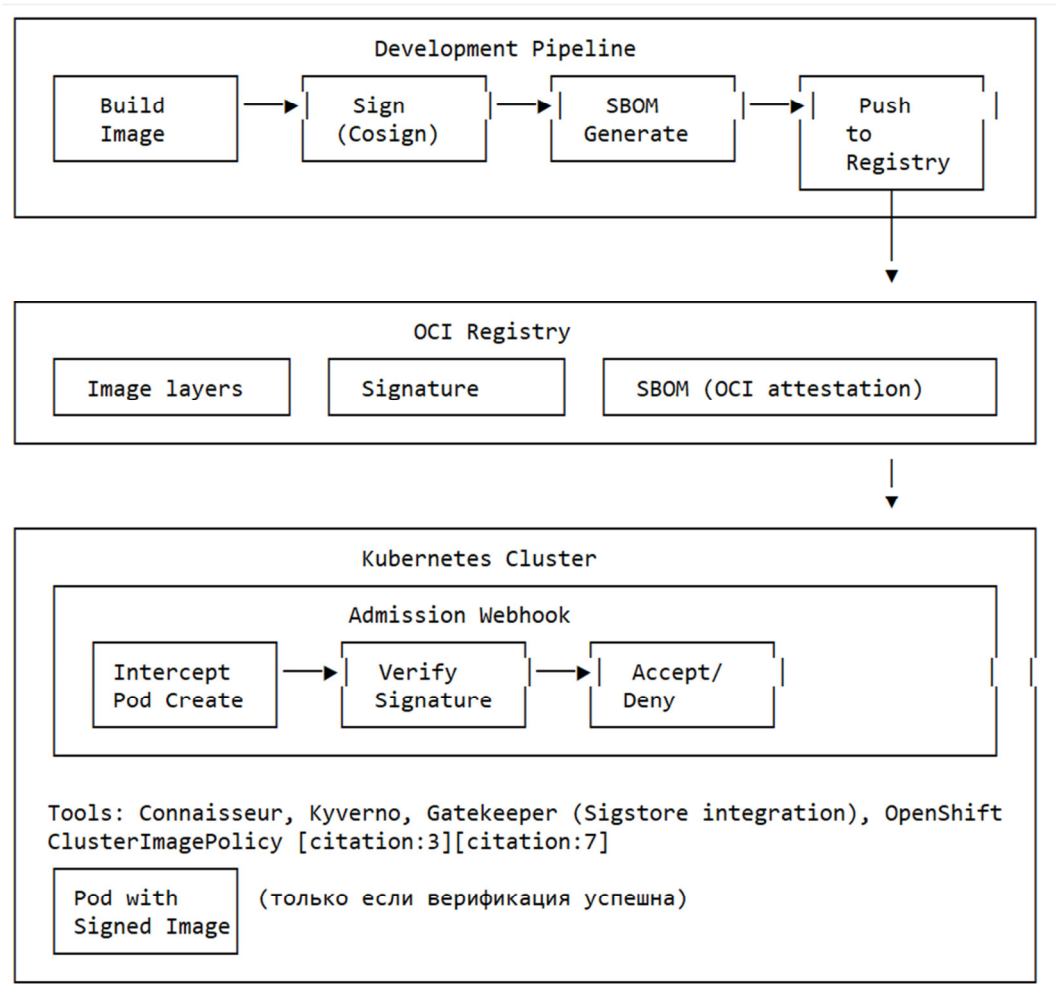
SBOM - это формальный список всех компонентов, входящих в программный продукт . Поддерживаемые форматы:

CycloneDX (JSON/XML) - популярный для приложений, поддерживает complex dependency graphs

SPDX - стандарт Linux Foundation, более юридически ориентированный

```
# Генерация SBOM с помощью Syft или Trivy
syft myregistry/myapp:latest -o cyclonedx-json > sbom.json
trivy image --format cyclonedx --output sbom.json myregistry/myapp:latest
```

3.5. Архитектура верификации



3.6. Admission Controller варианты

Решение	Механизм	Pros	Cons	Выбор
Connaisseur	Validating Webhook	Простая настройка через Helm, pre-	Меньше гибкости	☒ Для ЛР

Решение	Механизм	Pros	Cons	Выбор
		configured правила		
Kyverno	Native CRDs + Policies	Очень гибкий, ClusterPolicy ресурсы	Сложнее настройк а	—
Gatekeeper + Sigstore	OPA + ConstraintTempl ate	Мощные policy возможност и	Требует Rego	—
OpenShift ClusterImagePoli cy	Native API	Интегриров ан в ОСР	Только OpenShif t	—

Для лабораторной работы рекомендуется Connaisseur как самый простой для быстрого старта и понимания концепции.

4. Контрольные вопросы

1. Что такое SLSA и какие уровни зрелости он определяет?
2. Какие проблемы безопасности цепочки поставки решает подпись образов?
3. Что такое Sigstore и из каких компонентов он состоит (Fulcio, Rekor, Cosign)?
4. В чем разница между key-based и keyless signing?

5. Что такое SBOM и какие форматы поддерживаются (CycloneDX, SPDX)?
6. Как Cosign хранит подписи в OCI registry?
7. Как работает admission webhook для верификации подписей?
8. Какую роль играет Connaisseur? Как он интегрируется с Kubernetes?
9. Что такое certificate-identity-regex и certificate-oidc-issuer при keyless верификации?
10. Как проверить подпись образа локально с помощью cosign verify?
11. Какие метаданные содержит SLSA provenance attestation?
12. Как интегрировать подпись образов в CI/CD пайплайн?
13. Как настроить исключения (exemptions) для определенных namespace в admission controller?
14. Какие best practices вы знаете для безопасной цепочки поставки?
15. Почему важен мониторинг Rekor transparency log?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Сгенерировать ключевую пару для подписи (или настроить keyless)
2. Создать тестовый образ (можно использовать простой nginx + кастомную страницу)
3. Подписать образ с помощью Cosign и сохранить подпись в registry
4. Сгенерировать SBOM в формате CycloneDX или SPDX
5. Настроить Connaisseur (или Kyverno) в Kubernetes для верификации
6. Продемонстрировать:
7. Успешный деплой подписанного образа

8. Ошибку при деплое неподписанного образа

9. Предоставить отчет со всеми командами, манифестами и скриншотами

5.2. Варианты заданий

Вар.	Тип подписи	SBOM метод	Admission tool	Специфика
1	Key-based (Cosign)	Syft	Connaisseur	Базовый сценарий
2	Key-based (Cosign)	Trivy	Connaisseur	SBOM от Trivy
3	Key-based (Cosign)	Snyk	Connaisseur	SBOM от Snyk CLI
4	Keyless (GitHub OIDC)	Syft	Kyverno	CI/CD интеграция
5	Keyless (GitLab OIDC)	Syft	Kyverno	GitLab CI
6	Key-based (HashiCorp Vault)	Trivy	Connaisseur	Vault для хранения ключей
7	Key-based (PKI, create CA)	Syft	Connaisseur	BYO-PKI
8	Keyless (Google OIDC)	Syft	Kyverno	GCP Workload Identity

Вар.	Тип подписи	SBOM метод	Admission tool	Специфика
9	Key-based (AWS KMS)	Trivy	Connaisseur	KMS для ключей
10	Keyless (Azure OIDC)	Syft	Kyverno	Azure AD
11	Key-based (multi-signature)	Syft	Connaisseur	Два signer'a
12	Key-based attestations +	Syft + provenance	Kyverno	SLSA provenance
13	Keyless	cdxgen	Connaisseur	SBOM от cdxgen
14	Key-based (cosign)	Syft + SPDX	Connaisseur	SPDX формат
15	Key-based Vault +	Syft + gype	Kyverno	Vulnerability scanning

5.3. Инструкция по выполнению (Базовый вариант 1)

Шаг 1. Установка инструментов


```
# Установка Cosign (Linux)
curl -L -o cosign https://github.com/sigstore/cosign/releases/latest/download/cosign-linux-amd64
chmod +x cosign
sudo mv cosign /usr/local/bin/

# Установка Syft (для SBOM)
curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s -- -b /usr/local/bin

# Проверка
cosign version
syft version
```

Шаг 2. Генерация ключевой пары

bash

Генерация ключей (будет запрошен пароль)

cosign generate-key-pair

Результат: cosign.key (приватный), cosign.pub (публичный)

Шаг 3. Создание и публикация тестового образа

```
# Dockerfile
cat > Dockerfile << 'EOF'
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
EOF

# Кастомная страница
echo "Build: $(date) | Signed with Cosign" > index.html

# Сборка образа (замените myregistry на ваш registry, например, ghcr.io/username, cr.yandex/...)
IMAGE=myregistry/demo-app:v1.0.0
docker build -t $IMAGE .
docker push $IMAGE
```

Шаг 4. Подпись образа

```
# Подпись образа приватным ключом
cosign sign --key cosign.key $IMAGE

# Проверка подписи
cosign verify --key cosign.pub $IMAGE
```

Ожидаемый вывод:

Verification for myregistry/demo-app:v1.0.0 --

The following checks were performed on each of these signatures:

- The cosign claims were validated
- The signatures were verified against the specified public key
- Any certificates were verified against the Fulcio roots.

Шаг 5. Генерация SBOM

bash

```
# Генерация SBOM в формате CycloneDX JSON
syft $IMAGE -o cyclonedx-json > sbom.json

# (Опционально) Просмотр SBOM
cat sbom.json | jq '.components[] | {name, version}'
```

Шаг 6. Установка Connaisseur

```
# values-connaisseur.yaml
image:
  repository: securesystemsengineering/connaisseur
  tag: latest

policy:
  - name: "signed-images-policy"
    pattern: "myregistry/*"
    verification:
      level: "all"
      trust:
        - type: "publicKey"
          publicKey: |
            -----BEGIN PUBLIC KEY-----
            (скопируйте содержимое cosign.pub)
            -----END PUBLIC KEY-----
    notify: true
```

```
# Установка Connaissance
helm repo add connaissance https://charts.connaissance.de
helm upgrade --install connaissance connaissance/connaissance \
  --namespace connaissance --create-namespace \
  --values values-connaissance.yaml
```

Шаг 7. Тестирование: подписанный образ

```
# deployment-signed.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-signed
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
        - name: app
          image: myregistry/demo-app:v1.0.0 # ПОДПИСАННЫЙ
```

```
kubectl apply -f deployment-signed.yaml
# Ожидаемый результат: Deployment создан успешно
kubectl get pods -w
```

Шаг 8. Тестирование: неподписанный образ

```
# Создание неподписанной версии
docker tag myregistry/demo-app:v1.0.0 myregistry/demo-app:unsigned
docker push myregistry/demo-app:unsigned

# Попытка деплоя
cat > deployment-unsigned.yaml << EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-unsigned
spec:
  replicas: 1
```

```

selector:
  matchLabels:
    app: demo-unsigned
template:
  metadata:
    labels:
      app: demo-unsigned
  spec:
    containers:
      - name: app
        image: myregistry/demo-app:unsigned
EOF
kubectrl apply -f deployment-unsigned.yaml

```

Ожидаемый вывод (ошибка):

Error from server: error when creating "deployment-unsigned.yaml":
 admission webhook "connaisseur.securesystems.de" denied the request:
 No valid signature found for image(s): myregistry/demo-app:unsigned

5.4. Альтернатива: Kyverno вместо Connaisseur

Для Kyverno создается ClusterPolicy:

```

apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: check-image-signature
spec:
  validationFailureAction: Enforce
  rules:
    - name: require-cosign-signature
      match:
        any:
          - resources:
              kinds:
                - Pod
      verifyImages:
        - image: "myregistry/*"
          key: |-
            -----BEGIN PUBLIC KEY-----
            ...
            -----END PUBLIC KEY-----

```

5.5. Keyless signing (варианты 4-5, 8, 10)

Keyless signing не требует управления ключами - сертификат выпускается Fulcio на основе OIDC identity :

```
# B GitHub Actions workflow
- name: Sign the image with Cosign (keyless)
  run: |
    cosign sign ${ env.REGISTRY }/${ env.IMAGE_NAME }:${ env.github.sha } \
      --yes
```

Верификация с проверкой OIDC identity:

```
cosign verify $IMAGE \
  --certificate-identity-regexp "https://github.com/your-org/.*" \
  --certificate-oidc-issuer "https://token.actions.githubusercontent.com"
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть:
5. SLSA framework и уровни
6. Sigstore экосистема (Fulcio, Rekor, Cosign)
7. SBOM и форматы
8. Схема архитектуры (CI/CD → Sign → Registry → Admission → K8s)
9. Код и команды:
10. Генерация ключей
11. Сборка и подпись образа
12. Генерация SBOM
13. Connaisseur/Kyverno конфигурация
14. Результаты выполнения:

- 15.Скриншот верификации подписи (cosign verify)
- 16.Скриншот деплоя подписанного образа
- 17.Скриншот ошибки для неподписанного образа
- 18.Логи Connaisseur webhook
- 19.SBOM анализ (структура, ключевые компоненты)
- 20.Заключение
- 21.Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Генерация ключей и настройка Cosign	10
Подпись образа и верификация	15
Генерация SBOM	10
Установка и настройка admission controller	15
Демонстрация успешного деплоя	15
Демонстрация блокировки неподписанного образа	15
Ответы на контрольные вопросы	10
Качество отчета	10
Итого	100

Детальные критерии

Оценка	Баллы	Требования

Оценка	Баллы	Требования
Отлично	90-100	Все 3 этапа подписи, SBOM и верификации работают; оба сценария (успех/блокировка) продемонстрированы; SBOM проанализирован
Хорошо	75-89	Подпись и верификация работают, вебхук настроен, но есть недочеты (например, нет SBOM)
Удовлетворительно	60-74	Cosign работает, но admission controller не настроен или работает не полностью
Неудовлетворительно	<60	Подпись не выполнена, верификация не настроена

Штрафные баллы

Нарушение	Штраф

Нарушение	Штраф
Приватный ключ закоммичен	-15
Нет демонстрации ошибки для неподписанного образа	-15
Отсутствует SBOM	-10

Лабораторная работа №11

Auto-DevOps шаблон как услуга

1. Цель работы: Создать готовый CI/CD шаблон (Auto-DevOps) для предоставления разработчикам возможности подключения полноценного пайплайна в новый проект одной строкой кода, с использованием эфемерных раннеров в Kubernetes и интеграцией всех необходимых этапов (линтинг, тестирование, сборка, SAST, деплой).

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить механизмы повторного использования CI/CD конфигураций в GitLab CI
2. Разработать reusable template (шаблон) для CI/CD пайплайна, включающий:
3. Линтинг кода (например, ESLint для JS/TS или flake8 для Python)
4. Unit-тестирование с генерацией отчетов
5. SAST (статический анализ безопасности)
6. Сборка Docker-образа
7. Публикация образа в registry
8. Деплой в тестовое окружение (Kubernetes)
9. Настроить эфемерные раннеры (ephemeral runners) в Kubernetes

10. Создать тестовый репозиторий приложения, подключающий шаблон одной строкой
11. Продемонстрировать работу пайплайна

3. Краткое теоретическое обоснование

3.1. Проблема: повторяющиеся пайплайны в GitLab

При разработке множества микросервисов или проектов возникает проблема дублирования кода CI/CD пайплайнов. Каждый новый проект требует написания одного и того же набора этапов: тесты, сборка, SAST, деплой.

3.2. Решение: GitLab CI Templates

GitLab поддерживает включение внешних конфигураций через ключевое слово `include`. Это позволяет хранить общие части пайплайна в централизованном репозитории и переиспользовать их.

Способы включения:

1. `include:local` - локальный файл
2. `include:remote` - URL
3. `include:template` - встроенный шаблон GitLab
4. `include:project` - файл из другого проекта (с указанием ветки/тега)

3.3. Структура Auto-DevOps шаблона

`ci-templates/`

—	<code>.gitlab-ci.yml</code>	# Основной файл шаблона
—	<code>templates/</code>	
—	<code>stages.yml</code>	# Объявление этапов
—	<code>lint.yml</code>	# Линтинг
—	<code>test.yml</code>	# Тестирование
—	<code>sast.yml</code>	# Безопасность
—	<code>build.yml</code>	# Сборка образа

```
| └─ deploy.yml      # Деплой
└─ scripts/
    └─ common.sh      # Общие скрипты
```

3.4. Эфемерные раннеры в Kubernetes

Эфемерный (ephemeral) раннер - раннер, который создается под каждый job и уничтожается после его завершения. Преимущества:

Изоляция между jobs

Отсутствие устаревших артефактов на раннере

Масштабирование под нагрузку

```
# Установка GitLab Runner с executor = kubernetes
helm install gitlab-runner gitlab/gitlab-runner \
  --set rbac.create=true \
  --set runners.privileged=true \
  --set runners.image=alpine \
  --set runners.kubernetes.pod_annotations."cluster-autoscaler\.kubernetes\.io/safe-to-evict"=true
```

3.5. Шаблон .gitlab-ci.yml

Пример базового шаблона:

```
# templates/stages.yml
stages:
  - lint
  - test
  - sast
  - build
  - deploy

# templates/lint.yml
lint:js:
  stage: lint
```

```

image: node:18
script:
  - npm ci
  - npm run lint
rules:
  - if: $CI_MERGE_REQUEST_ID
  - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

# templates/build.yml
build:docker:
  stage: build
  image: docker:24
  services:
    - docker:24-dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA .
    - docker push $CI_REGISTRY_IMAGE $CI_COMMIT_SHORT_SHA
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

```

3.6. Параметризация шаблонов

Шаблоны могут принимать переменные:

yaml

```

# templates/deploy.yml
.deploy_template:
  stage: deploy
  image: bitnami/kubectl:latest
  script:
    - kubectl set image deployment/$CI_PROJECT_NAME
      $CI_PROJECT_NAME=$CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
    - kubectl rollout status deployment/$CI_PROJECT_NAME
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH && $DEPLOY_ENABLED ==
"true"

```

3.7. Подключение шаблона в проекте

```
# .gitlab-ci.yml в проекте разработчика
include:
  - project: 'platform/ci-templates'
    ref: 'v1.0.0'
    file: '.gitlab-ci.yml'

variables:
  DEPLOY_ENABLED: "true"
  APP_NAME: "my-service"
  NAMESPACE: "development"
```

3.8. Переопределение этапов в проекте

```
# Переопределение переменных или добавление кастомного job
include:
  - project: 'platform/ci-templates'
    file: '.gitlab-ci.yml'

# Дополнительный кастомный job
custom-job:
  stage: test
  script:
    - echo "Running custom tests"
  rules:
    - if: $CI_MERGE_REQUEST_ID

# Переопределение существующего job
lint:js:
  before_script:
    - npm install -g eslint plugin-custom
```

3.9. GitLab CI Variables в шаблонах

Переменная	Описание
CI_PROJECT_NAME	Имя проекта

Переменная	Описание
CI_PROJECT_NAMESPACE	Namespace проекта
CI_REGISTRY_IMAGE	URL образа в registry
CI_COMMIT_SHORT_SHA	Короткий хэш коммита
CI_MERGE_REQUEST_IID	ID Merge Request
CI_DEFAULT_BRANCH	Основная ветка (обычно main/master)

4. Контрольные вопросы

1. Какие способы включения внешних конфигураций поддерживает GitLab CI?
2. Что такое эфемерный раннер и какие преимущества он дает?
3. Как параметризовать шаблон для разных проектов?
4. Каким образом проект может переопределить отдельные этапы шаблона?
5. Какие правила (rules) следует использовать для разных этапов (MR, основная ветка)?
6. Как защитить секреты (токены, пароли) в шаблоне при использовании в проектах?
7. Как организовать версионирование шаблонов (semver, Git tags)?
8. Как протестировать изменения шаблона без публикации новой версии?
9. Какие метрики эффективности пайплайна можно собирать (DORA metrics)?
10. Как реализовать conditional деплой (только при определенном теге)?
11. В чем преимущество include:project перед include:remote?

12. Как настроить GitLab Runner для работы с недоверенными проектами?
13. Как добавить кастомные SAST анализаторы в шаблон?
14. Как организовать сквозное (e2e) тестирование после деплоя?
15. Какие best practices вы знаете для создания reusable шаблонов?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Создать репозиторий с шаблонами CI/CD
2. Разработать шаблон согласно спецификации варианта, включающий минимум 4 этапа
3. Настроить эфемерные раннеры в Kubernetes (или использовать shared runners)
4. Создать тестовый проект, который подключает шаблон одной строкой include
5. Запустить пайплайн и продемонстрировать успешное выполнение
6. Предоставить отчет со всеми конфигурациями и скриншотами

5.2. Варианты заданий

Вар .	Язык/платформа	Тестирование	SAST	Деплой в	Особенность
1	Node.js	Jest	Semgrep	K8s (namespace)	npm caching
2	Python	pytest	Bandit	K8s	pip caching

Вар .	Язык/платформа	Тестирование	SAST	Деплой в	Особенность
3	Go	go test	gosec	K8s	go mod caching
4	Java	Maven	SpotBugs	K8s	Maven caching
5	.NET	dotnet test	SecurityCode Scan	K8s	NuGet caching
6	Ruby	RSpec	brakeman	K8s	Bundler caching
7	Rust	cargo test	cargo-audit	K8s	cargo caching
8	PHP	PHPUnit	Psalm	K8s	Composer caching
9	Node.js (TS)	Jest + ESLint	Semgrep	Helm chart	Vite build
10	Python (FastAPI)	pytest + flake8	Bandit	K8s + Helm	OpenAPI validation
11	Go + gRPC	go test + buf lint	gosec	K8s	protocol generation
12	Scala	sbt test	Scalafix	K8s	sbt

Вар .	Язык/платформа	Тестирование	SAST	Деплой в	Особенность
			x		caching
13	Kotlin	gradle test	detekt	K8s	Gradle caching
14	Multi-lang (Node+Python)	separate jobs	Trivy	K8s	Conditional jobs
15	Bash/Shell script	shellcheck	Trivy	K8s	Docker multi-stage

5.3. Общая структура шаблона (.gitlab-ci.yml)

```
# ci-templates/.gitlab-ci.yml
---
stages:
  - lint
  - test
  - sast
  - build
  - deploy

variables:
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  SAST_EXCLUDED_ANALYZERS: "eslint"

# === Линтинг ===
include: "templates/lint.yml"

# === Тестирование ===
include: "templates/test.yml"
```



```
# === Безопасность (SAST) ===  
include: "templates/sast.yml"  
  
# === Сборка образа ===  
include: "templates/build.yml"  
  
# === Деплой ===  
include: "templates/deploy.yml"
```

5.4. Шаблон lint (пример для Python)

```
# templates/lint.yml  
lint:flake8:  
  stage: lint  
  image: python:3.11  
  before_script:  
    - pip install flake8  
  script:  
    - flake8 . --max-line-length=120 --count --statistics  
  artifacts:  
    paths:  
      - flake8-report.txt  
    when: always  
  rules:  
    - if: $CI_MERGE_REQUEST_ID  
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

5.5. Шаблон test (пример для Node.js)

yaml

```
# templates/test.yml  
test:jest:  
  stage: test  
  image: node:18  
  before_script:  
    - npm ci --cache .npm --prefer-offline
```

```

script:
  - npm run test -- --coverage
  - npm run test integration --if-present
coverage: '/All files[^]*\\[^]*s+([\\d\\.]+)/'
artifacts:
  reports:
    junit: junit.xml
  paths:
    - coverage/
  expire_in: 7 days
rules:
  - if: $CI_MERGE_REQUEST_ID
  - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
cache:
  key: $CI_COMMIT_REF_SLUG
  paths:
    - .npm
    - node_modules

```

5.6. Шаблон SAST (GitLab native)

```

# templates/sast.yml
sast:
  stage: sast
  variables:
    SEARCH_MAX_DEPTH: 5
  rules:
    - if: $CI_MERGE_REQUEST_ID
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

secret-detection:
  stage: sast

```

```

rules:
  - if: $CI_MERGE_REQUEST_ID

dependency-scanning:
  stage: sast
  rules:
    - if: $CI_MERGE_REQUEST_ID

```

5.7. Шаблон build

```

# templates/build.yml
build:docker:
  stage: build
  image: docker:24
  services:
    - docker:24 dind
  variables:
    DOCKER_IMAGE: $CI_REGISTRY_IMAGE
    DOCKER_TAG: $CI_COMMIT_SHORT_SHA
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
      $CI_REGISTRY
    - docker build -t $DOCKER_IMAGE:$DOCKER_TAG .
    - docker push $DOCKER_IMAGE:$DOCKER_TAG
    - |
      if [ $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH ]; then
        docker tag $DOCKER_IMAGE:$DOCKER_TAG $DOCKER_IMAGE:latest
        docker push $DOCKER_IMAGE:latest
      fi
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
    - if: $CI_MERGE_REQUEST_ID

```

5.8. Шаблон deploy в Kubernetes

```

# templates/deploy.yml

```

deploy:k8s:

stage: deploy

image: bitnami/kubectl:latest

script:

```
- kubectl config use context $KUBE_CONTEXT
- |
  if [ $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH ]; then
    NAMESPACE="${KUBE_NAMESPACE:-production}"
    REPLICAS="${PROD_REPLICAS:-2}"
  else
    NAMESPACE="${KUBE_NAMESPACE:-preview}"
    REPLICAS=1
  fi
- |
  cat <<EOF | kubectl apply -n $NAMESPACE -f -
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: $CI_PROJECT_NAME
  spec:
    replicas: $REPLICAS
    selector:
      matchLabels:
        app: $CI_PROJECT_NAME
    template:
      metadata:
        labels:
          app: $CI_PROJECT_NAME
      spec:
        containers:
          - name: app
            image: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
            ports:
              - containerPort: 80
  EOF
- kubectl rollout status -n $NAMESPACE deployment/$CI_PROJECT_NAME
```

rules:

```
- if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH && $DEPLOY_ENABLED ==  
"true"  
- if: $CI_MERGE_REQUEST_ID && $DEPLOY_PREVIEW == "true"
```

5.9. Подключение шаблона в тестовом проекте

```
# тестовый проект: .gitlab-ci.yml  
include:  
- project: 'platform/ci-templates'  
  ref: 'v1.0.0'  
  file: '.gitlab-ci.yml'  
  
variables:  
  DEPLOY_ENABLED: "true"  
  KUBE_NAMESPACE: "dev"  
  DEPLOY_PREVIEW: "true"
```

5.10. Настройка эфемерных раннеров

```
# Установка GitLab Runner в Kubernetes  
helm repo add gitlab https://charts.gitlab.io  
helm upgrade --install gitlab-runner gitlab/gitlab-runner \  
  --namespace gitlab-runner \  
  --create-namespace \  
  --set gitlabUrl=https://gitlab.example.com \  
  --set runnerRegistrationToken=$REGISTRATION_TOKEN \  
  --set rbac.create=true \  
  --set runners.privileged=true \  
  --set runners.kubernetes.helper_image="alpine:latest" \  
  --set runners.kubernetes.pod_annotations."cluster-autoscaler\.kubernetes\.io/safe-to-evict"="true"
```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист

2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (include механизм, эфемерные раннеры, структура шаблона)
5. Схема архитектуры (взаимодействие шаблон → раннеры → K8s)
6. Код шаблона (все файлы .gitlab-ci.yml и вспомогательные шаблоны)
7. Код тестового проекта (.gitlab-ci.yml, демонстрирующий include)
8. Настройка раннеров (Helm values или конфигурация)
9. Результаты выполнения: (скриншоты пайплайна в GitLab, успешное выполнение всех этапов)
10. Заключение
11. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Разработка шаблона (мин. 4 этапа)	25
Корректное использование include	10
Настройка эфемерных раннеров	15
Создание тестового проекта с подключением шаблона	10
Демонстрация успешного пайплайна	20
Ответы на контрольные вопросы	10
Качество отчета	10

Критерий	Макс. баллов
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Шаблон включает 5+ этапов, корректно параметризован, раннеры в K8s работают, тестовый проект успешно выполняет пайплайн
Хорошо	75-89	Шаблон работает, но есть недочеты (не все этапы, нет кэширования)
Удовлетворительно	60-74	Шаблон создан, но не все этапы проходят, либо раннеры не эфемерные
Неудовлетворительно	<60	Шаблон не создан, тестовый проект не работает

Штрафные баллы

Нарушение	Штраф
Hardcoded значения вместо переменных	-10
Нет кэширования зависимостей	-10
Нет правила rules для MR/main ветки	-10
Отсутствует SAST этап	-10

Лабораторная работа №12

Preview-окружения по Pull Request

1. **Цель работы:** Настроить автоматическое развертывание изолированных preview-окружений в Kubernetes для каждого Merge Request/Pull Request, обеспечивая разработчикам возможность тестирования изменений в живых средах перед слиянием в основную ветку.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить концепцию preview-окружений и их роль в CI/CD процессе
2. Настроить автоматический деплой приложения в изолированный namespace Kubernetes при создании MR/PR
3. Использовать переменную CI_COMMIT_REF_SLUG (GitLab CI) или HEAD_REF_SLUG (GitHub Actions) для уникальной идентификации окружения
4. Настроить публикацию комментария в MR/PR со ссылкой на развернутое окружение

5. Реализовать автоматическое удаление окружения при закрытии MR/PR
6. Продемонстрировать работу preview-окружений на практическом примере

3. Краткое теоретическое обоснование

3.1. Что такое Preview Environment?

Preview Environment (также known как Review App, Dynamic Environment) - это временная, изолированная копия приложения, развертываемая для каждого Merge Request. Она позволяет:

1. Тестировать изменения в реальной инфраструктуре до слияния
2. Демонстрировать результаты работы стейкхолдерам
3. Выполнять интеграционное тестирование с зависимыми сервисами
4. Обнаруживать проблемы на ранних стадиях

3.2. GitLab CI: Review Apps

GitLab имеет встроенную поддержку Review Apps через механизм dynamic environments. Ключевые возможности:

1. Автоматическое создание environment при job
2. Публикация ссылки на MR
3. Автоматическая остановка при закрытии MR
4. Экспозиция метрик и мониторинг

3.3. GitHub Actions: Preview Environments

В GitHub Actions для реализации preview-окружений используется комбинация:

1. pull_request trigger для перехвата событий PR
2. github.event.pull_request.head.ref для уникального идентификатора
3. workflow_run или pull_request (closed) для очистки

3.4. Уникальные идентификаторы окружений

Платформа	Переменная	Пример значения
GitLab CI	CI_COMMIT_REF_SLUG	feature-new-login
GitLab CI	CI_ENVIRONMENT_SLUG	feature-new-login-3f2a1b
GitHub Actions	github.head_ref	feature/new-login
GitHub Actions	github.event.pull_request.number	42

Трансформация в безопасное имя namespace:

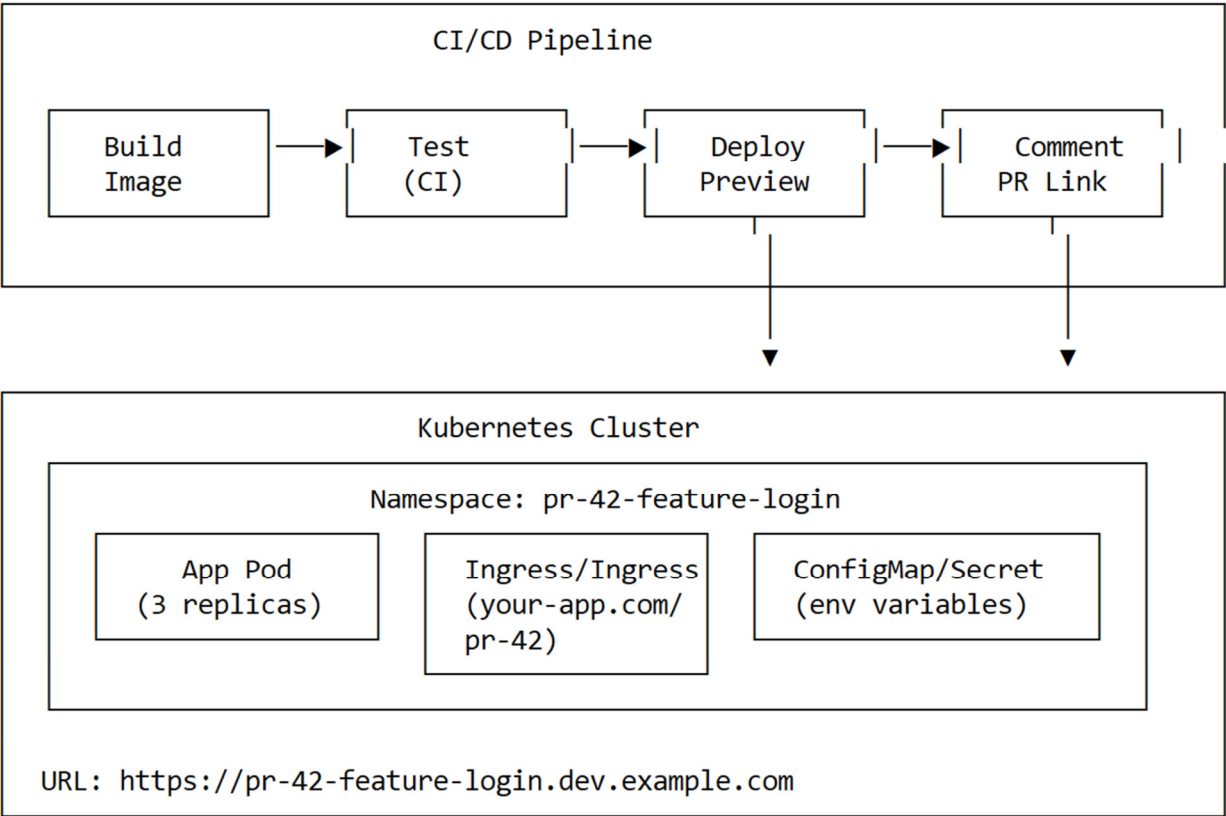
```
bash
```

```
# Допустимые символы в Kubernetes namespace: a-z, 0-9, -
```

```
# Ограничение длины: 63 символа
```

```
NAMESPACE="pr-${CI_MERGE_REQUEST_IID}-${CI_COMMIT_REF_SLUG:0:40}"
```

3.5. Архитектура



3.6. Жизненный цикл preview-окружения

Событие	Действие	Результат
Создан MR	Deploy job запускается	Создан namespace с приложением
Обновлен MR (push)	Deploy job перезапускается	Обновлено приложение
Закрыт MR	Stop job запускается	Namespace удален
Слит MR	Stop job запускается	Окружение удалено

3.7. GitHub Actions: Ветка vs PR номер

Важное различие: В GitHub Actions события pull_request и push используют разные refs.

```
# ☒ Правильно: используем номер PR для уникальности
- name: Get PR number
  run: echo "PR_NUMBER=${{ github.event.pull_request.number }}" >>
$GITHUB_ENV
```

```
# ☐ Неправильно: github.head_ref может совпадать между разными PR
- name: Deploy
  run: |
    NAMESPACE="pr-${{ github.head_ref }}" # может конфликтовать
```

3.8. Именованние URL для preview-окружений

Рекомендуемые паттерны:

```
# Паттерн 1: Subdomain (требует wildcard DNS)
URL: "https://${NAMESPACE}.preview.example.com"
```

```
# Паттерн 2: Path-based (Ingress с правилами)
URL: "https://example.com/preview/${NAMESPACE}"
```

```
# Паттерн 3: Port-based (NodePort, не рекомендуется для production)
URL: "https://example.com:${NODE_PORT}"
```

3.9. Очистка preview-окружений

Для предотвращения накопления ресурсов:

Автоматическая очистка при закрытии MR (обязательно)

TTL-контроллер для старых окружений (опционально)

CronJob для регулярной очистки (backup)

4. Контрольные вопросы

1. Что такое preview environment и какие проблемы она решает?
2. Как GitLab CI реализует механизм Review Apps?
3. Какая переменная в GitLab CI используется для создания уникального идентификатора окружения?
4. Как передать URL preview-окружения в комментарий MR?

5. Как настроить автоматическое удаление окружения при закрытии MR?
6. В чем разница между CI_ENVIRONMENT_SLUG и CI_COMMIT_REF_SLUG?
7. Какие ограничения Kubernetes накладываются на имена namespace?
8. Как обеспечить уникальность URL для разных MR?
9. Как защитить preview-окружения от несанкционированного доступа?
10. Какие ресурсы должны быть изолированы между preview-окружениями?
11. Как обрабатывать конфликты при одновременном деплое нескольких MR на один кластер?
12. Какие метрики можно собирать с preview-окружений?
13. Какова стратегия очистки старых preview-окружений (при зависании)?
14. Как интегрировать preview-окружения с системой мониторинга?
15. Какие best practices вы знаете для управления preview-окружениями?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

Независимо от варианта, необходимо:

1. Настроить CI/CD пайплайн для автоматического деплоя preview-окружений
2. Реализовать генерацию уникального namespace на основе идентификатора MR/PR
3. Развернуть Ingress для маршрутизации трафика на preview-окружения
4. Настроить публикацию комментария со ссылкой на окружение

5. Реализовать автоудаление окружения при закрытии MR/PR

6. Продемонстрировать создание, обновление и удаление окружений

5.2. Варианты заданий

Ва.	CI/CD платформа	Приложение	БД/зависимости	Особенность
1	GitLab CI	Python/Flask	PostgreSQL	Поддомены (wildcard)
2	GitLab CI	Node.js/Express	MongoDB	Path-based Ingress
3	GitLab CI	Go/HTTP	Redis	TLS сертификаты
4	GitLab CI	Python/Django	PostgreSQL	Миграции БД
5	GitLab CI	Ruby on Rails	PostgreSQL + Redis	Sidekiq workers
6	GitHub Actions	Node.js/Express	MySQL	Path-based Cleanup +
7	GitHub Actions	Python/FastAPI	PostgreSQL	Subdomain
8	GitHub Actions	Java/Spring Boot	H2 (in-memory)	Prometheus метрики
9	GitHub Actions	PHP/Laravel	MySQL	Seed data

В.а.	CI/CD платформа	Приложение	БД/зависимости	Особенность
0	1 GitLab CI	Static Site (React)	No DB	S3-like storage
1	1 GitLab CI	.NET Core	SQL Server Express	Windows containers
2	1 GitHub Actions	Vue.js	API mock	Nginx config
3	1 GitLab CI	gRPC сервис	etcd	gRPC reflection
4	1 GitHub Actions	WebSocket (Node)	Redis	Sticky sessions
5	1 GitLab CI	Full Stack (React+Go)	PostgreSQL	Multi-container

5.3. Базовый шаблон для GitLab CI

```
# .gitlab-ci.yml
---
stages:
  - build
  - deploy preview
  - stop preview

variables:
  REGISTRY_IMAGE: $CI_REGISTRY_IMAGE
  KUBE_NAMESPACE_BASE: "pr"
```

```

build:docker:
  stage: build
  image: docker:24
  services:
    - docker:24:dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
      $CI_REGISTRY
    - docker build -t $REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA .
    - docker push $REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
  rules:
    - if: $CI_MERGE_REQUEST_ID

deploy-preview:
  stage: deploy preview
  image: bitnami/kubectl:latest
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    url: https://$CI_ENVIRONMENT_SLUG.preview.example.com
    on_stop: stop preview
  script:
    # Генерация уникального namespace
    - NAMESPACE="$KUBE_NAMESPACE_BASE-${CI_MERGE_REQUEST_ID}"
    - echo "Namespace: $NAMESPACE"

    # Создание namespace если не существует
    - kubectl create namespace $NAMESPACE --dry-run=client -o yaml |
      kubectl apply -f -

    # Создание ConfigMap с переменными окружения
    - |
      cat <<EOF | kubectl apply -n $NAMESPACE -f -
      apiVersion: v1
      kind: ConfigMap
      metadata:
        name: app-config

```



```
data:
  APP_NAME: "Preview-${CI_MERGE_REQUEST_ID}"
  APP_ENV: "preview"
  DATABASE_URL:
"postgresql://postgres:password@postgres.default.svc.cluster.local:5432/app_${CI_MERGE_REQUEST_ID}"
EOF
```

```
# Деплой приложения
```

```
- |
cat <<EOF | kubectl apply -n $NAMESPACE -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: preview-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: preview-app
      mr-id: "${CI_MERGE_REQUEST_ID}"
  template:
    metadata:
      labels:
        app: preview-app
        mr-id: "${CI_MERGE_REQUEST_ID}"
    spec:
      containers:
      - name: app
        image: ${REGISTRY_IMAGE}:${CI_COMMIT_SHORT_SHA}
        ports:
        - containerPort: 8080
        envFrom:
        - configMapRef:
            name: app-config
EOF
```

```
- |
```

```
cat <<EOF | kubectl apply -n $NAMESPACE -f -
apiVersion: v1
kind: Service
metadata:
  name: preview-service
spec:
  selector:
    app: preview-app
  ports:
    - port: 8080
      targetPort: 8080
EOF
```

```
- |
cat <<EOF | kubectl apply -n $NAMESPACE -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: preview-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: "${CI_ENVIRONMENT_SLUG}.preview.example.com"
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: preview-service
                port:
                  number: 8080
EOF
```

```
# Ожидание готовности
```

```
- kubectl rollout status -n $NAMESPACE deployment/preview app
rules:
```

```

    - if: $CI_MERGE_REQUEST_ID

stop-preview:
  stage: stop preview
  image: bitnami/kubectl latest
  environment:
    name: review/${CI_COMMIT_REF_SLUG}
    action: stop
  script:
    - NAMESPACE="${KUBE_NAMESPACE_BASE}-${CI_MERGE_REQUEST_ID}"
    - echo "Deleting namespace: $NAMESPACE"
    - kubectl delete namespace $NAMESPACE --ignore-not-found=true
  rules:
    - if: $CI_MERGE_REQUEST_ID
    - when: manual
  allow_failure: true

```

5.4. GitHub Actions шаблон

```

# .github/workflows/preview.yml
name: Preview Environment

on:
  pull_request:
    types: [opened, synchronize, closed]
  workflow_dispatch:

jobs:
  deploy-preview:
    if: github.event_name != 'pull_request' || github.event.action !=
'closed'
    runs-on: ubuntu latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

```

```

- name: Log in to Container Registry
  uses: docker/login action@v3
  with:
    registry: ${{ secrets.REGISTRY_URL }}
    username: ${{ secrets.REGISTRY_USERNAME }}
    password: ${{ secrets.REGISTRY_PASSWORD }}

- name: Build and push Docker image
  uses: docker/build-push action@v5
  with:
    context: .
    push: true
    tags: ${{ secrets.REGISTRY_URL }}/preview:${{
github.event.pull_request.number }}

- name: Set up kubectl
  uses: azure/setup-kubectl@v4
  with:
    version: 'latest'

- name: Configure kubectl
  run: |
    echo "${{ secrets.KUBE_CONFIG }}" | base64 -d > kubeconfig
    export KUBECONFIG=$PWD/kubeconfig

- name: Deploy to preview namespace
  run: |
    export KUBECONFIG=$PWD/kubeconfig
    NAMESPACE="pr-${{ github.event.pull_request.number }}"

    kubectl create namespace $NAMESPACE --dry-run=client -o yaml
    | kubectl apply -f -

    envsubst < deploy/preview deployment.yaml | kubectl apply -n
$NAMESPACE -f -

    envsubst < deploy/preview service.yaml | kubectl apply -n
$NAMESPACE -f -

```

```
envsubst < deploy/preview ingress.yaml | kubectl apply -n
$NAMESPACE -f -
```

```
kubectl rollout status -n $NAMESPACE deployment/preview app -
-timout=5m
```

```
env:
  PR_NUMBER: ${GITHUB_EVENT_PULL_REQUEST_NUMBER}
  IMAGE_TAG:    ${SECRETS_REGISTRY_URL}/preview:${GITHUB_EVENT_PULL_REQUEST_NUMBER}
```

- name: Find Comment
 uses: peter-evans/find-comment@v3
 id: fc
 with:
 issue-number: \${GITHUB_EVENT_PULL_REQUEST_NUMBER}
 comment-author: 'github-actions[bot]'
 body-includes: Preview environment deployed
- name: Create or update comment
 uses: peter-evans/create-or-update-comment@v4
 with:
 issue-number: \${GITHUB_EVENT_PULL_REQUEST_NUMBER}
 comment-id: \${STEPS.FC.outputs.comment-id}
 edit-mode: replace
 body: |
 ## 🚀 Preview environment deployed!

Resource	URL
Application	https://pr-\${GITHUB_EVENT_PULL_REQUEST_NUMBER}.preview.example.com
Namespace	`pr-\${GITHUB_EVENT_PULL_REQUEST_NUMBER}`

This environment will be automatically deleted when the PR is closed.

```

destroy-preview:
  if: github.event_name == 'pull_request' && github.event.action ==
'closed'

  runs-on: ubuntu latest

  steps:
    - name: Set up kubectl
      uses: azure/setup kubectl@v4

    - name: Configure kubectl
      run: |
        echo "${{ secrets.KUBE_CONFIG }}" | base64 -d > kubeconfig
        export KUBECONFIG=$PWD/kubeconfig

    - name: Delete preview namespace
      run: |
        export KUBECONFIG=$PWD/kubeconfig
        NAMESPACE="pr-${{ github.event.pull_request.number }}"
        kubectl delete namespace $NAMESPACE --ignore-not-found=true

```

5.5. Ingress для Wildcard Subdomain

yaml

Ingress для обработки всех preview поддоменов

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: preview wildcard ingress

namespace: default

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

nginx.ingress.kubernetes.io/use-regex: "true"

spec:

ingressClassName: nginx

tls:

- hosts:

- "*.preview.example.com"

```
secretName: preview-tls
rules:
- host: "*.preview.example.com"
  http:
    paths:
    - path: /
      pathType: Prefix
    backend:
      service:
        name: preview-service
        port:
          number: 8080
```

5.6. Демонстрационная конфигурация

```
# deploy/preview-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: preview-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: preview-app
  template:
    metadata:
      labels:
        app: preview-app
    spec:
      containers:
      - name: app
        image: ${IMAGE_TAG}
        ports:
```

```
- containerPort: 8080
env:
- name: APP_VERSION
  value: "PR-${PR_NUMBER}"
- name: NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
```

5.7. Комментарий в MR/PR

🚀 Preview environment deployed!

Resource	URL
Application	https://pr-42-feature-login.preview.example.com
API Health	https://pr-42-feature-login.preview.example.com/health
Metrics	https://pr-42-feature-login.preview.example.com/metrics

Deployment details:

- Commit: `abc1234` (2 minutes ago)
- Status: ☒ Running
- Replicas: 1/1
- Namespace: `pr-42`

This environment will be **automatically deleted** when this PR is closed.

✨ **Need to test something? Comment `/deploy` to force redeploy.**

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (preview environments, dynamic environments, lifecycle)
5. Схема архитектуры (взаимодействие CI/CD → K8s → Ingress)
6. CI/CD конфигурация (полный файл)
7. Kubernetes манифесты (Deployment, Service, Ingress)
8. Результаты выполнения:
9. Скриншот создания MR и запуска пайплайна
10. Скриншот комментария со ссылкой
11. Скриншот работающего preview-окружения
12. Скриншот удаления при закрытии MR
13. Заключение
14. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Настройка CI/CD пайплайна для preview-деплойа	20
Генерация уникального namespace/окружения	15
Публикация комментария со ссылкой	10
Настройка Ingress с уникальным URL	15
Автоудаление при закрытии MR/PR	15
Демонстрация всех сценариев (создание/обновление/удаление)	15

Критерий	Макс. баллов
Ответы на контрольные вопросы	10
Качество отчета	5
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Preview-окружения работают полностью автоматически, URL генерируется уникально, удаление корректное, комментарий информативный
Хорошо	75-89	Preview работает, но есть недочеты (например, нет автоматического комментария, или URL не уникальный)
Удовлетворительно	60-74	Preview создается, но удаление не работает или URL конфликтует
Неудовлетворительно	<60	Preview не создается

Штрафные баллы

Нарушение	Штраф
Окружения не очищаются	-15
Нет комментария со ссылкой	-10
Жестко закодированный namespace	-10
Конфликт при одновременных PR	-10

Лабораторная работа №13

Наблюдаемость платформы: сбор метрик платформенных сервисов

1. Цель работы: Научиться применять OpenTelemetry для сбора телеметрии (метрик, логов и трасс) с платформенных компонентов инфраструктуры как кода, настроить инструментирование платформенного API и создать дашборды для мониторинга производительности платформы.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить принципы наблюдаемости платформы (observability of the platform)
2. Развернуть OpenTelemetry Collector в кластере Kubernetes
3. Инструментировать платформенный API (например, Terragrunt wrapper или CI/CD оркестратор) для отправки трасс и метрик
4. Настроить экспорт телеметрии в Prometheus и Jaeger/Grafana Tempo
5. Создать дашборд в Grafana для мониторинга платформенных сервисов
6. Проанализировать собранные метрики и выявить узкие места

3. Краткое теоретическое обоснование

3.1. Наблюдаемость платформы vs наблюдаемость приложения

В предыдущих дисциплинах изучалась наблюдаемость приложений, работающих на платформе. Данная работа фокусируется на наблюдаемости самой платформы - компонентов IaC, CI/CD, оркестрации, которые обеспечивают работу приложений.

Аспект	Наблюдаемость приложения	Наблюдаемость платформы
Объект мониторинга	Бизнес-приложение	CI/CD, Terraform, Karpenter, Vault
Ключевые метрики	RPS, latency, error rate	Время выполнения пайплайна, частота деплоев, успешность provisioning
Пользователи	Конечные пользователи	Разработчики, SRE-инженеры
SLO	Доступность API	Время создания preview-окружения

3.2. Три столпа наблюдаемости платформы

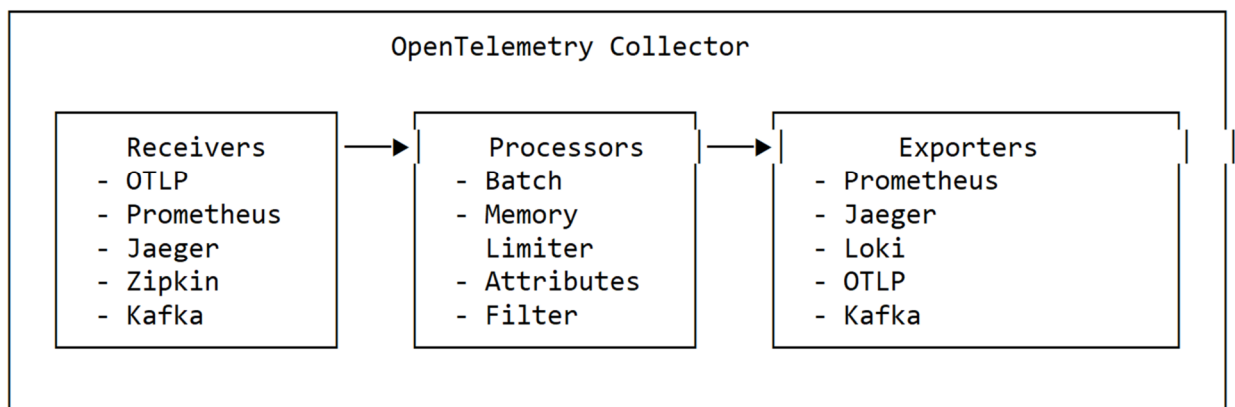
Столп	Что измеряем	Инструменты
Метрики	Количество вызовов API, длительность операций, частота ошибок	Prometheus, VictoriaMetrics

Столп	Что измеряем	Инструменты
Логи	События выполнения операций, ошибки, предупреждения	Loki, ELK Stack
Трассы	Полный путь запроса через платформенные сервисы	Jaeger, Grafana Tempo

3.3. OpenTelemetry Collector

OpenTelemetry Collector - это прокси-сервер для приёма, обработки и экспорта телеметрии.

Архитектура Collector:



3.4. Установка OpenTelemetry Collector в Kubernetes

Установка через Helm

```

helm repo add open-telemetry https://open-
telemetry.github.io/opentelemetry-helm-charts
helm upgrade --install opentelemetry-collector open-
telemetry/opentelemetry-collector \
  --namespace observability --create-namespace \
  --values collector-values.yaml

```

3.5. Инструментирование платформенного API

Пример инструментирования Terragrunt обёртки на Python:

```
from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import
OTLPSpanExporter
from opentelemetry.instrumentation.requests import RequestsInstrumentor
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

# Настройка трассировки
provider = TracerProvider()
processor = BatchSpanProcessor(OTLPSpanExporter(endpoint="http://otel-
collector:4317"))
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)

tracer = trace.get_tracer(__name__)

@tracer.start_as_current_span("terragrunt.plan")
def run_terragrunt_plan(module_path):
    with tracer.start_as_current_span("terraform.init"):
        # Логика init
        pass

    with tracer.start_as_current_span("terraform.plan"):
        # Логика plan
        pass

    return plan_output
```

3.6. Инструментирование через OpenTelemetry SDK (Go)

```
package main
```

```

import (
    "context"
    "log"
    "time"

    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
    semconv "go.opentelemetry.io/otel/semconv/v1.21.0"
)

func initTracer() func() {
    exporter, err := otlptracegrpc.New(context.Background(),
        otlptracegrpc.WithEndpoint("otel-collector:4317"),
        otlptracegrpc.WithInsecure(),
    )
    if err != nil {
        log.Fatal(err)
    }

    tp := sdktrace.NewTracerProvider(
        sdktrace.WithBatcher(exporter),
        sdktrace.WithResource(resource.NewWithAttributes(
            semconv.SchemaURL,
            semconv.ServiceNameKey.String("platform-api"),
            semconv.ServiceVersionKey.String("1.0.0"),
        )),
    )
    otel.SetTracerProvider(tp)

    return func() {
        tp.Shutdown(context.Background())
    }
}

```

3.7. Метрики платформы (RED метод)

RED метод для каждого платформенного сервиса:

Метрика	Описание	Пример использования
Rate	Количество запросов в секунду	Частота вызовов Terragrunt
Errors	Частота ошибок	Процент неудачных terraform apply
Duration	Длительность операций	Время выполнения пайплайна

3.8. Сбор метрик из CI/CD пайплайна

```
# .gitlab-ci.yml этап отправки метрик
metrics:
  stage: post-deploy
  script:
    - |
      curl -X POST http://otel-collector:4318/v1/metrics \
        -H "Content-Type: application/json" \
        -d '{
          "resource_metrics": [{
            "resource": {
              "attributes": [{"key": "service.name", "value":
{"string_value": "gitlab-ci"}}]
            },
            "scope_metrics": [{
              "metrics": [{
                "name": "pipeline.duration_seconds",
                "gauge": {
                  "data_points": [{
```



```

        "time_unix_nano": '${date +%s%N}',
        "as_double": '${PIPELINE_DURATION}'
    }}
}
}}
}}
}}
}
rules:
  - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

```

3.9. Метрики для отслеживания

Категория	Метрика	Целевое значение
Preview-окружения	preview_env_create_duration_seconds	< 120 секунд
Terraform операции	terraform_plan_duration_seconds	< 30 секунд
Terraform операции	terraform_apply_success_total	> 99%
Деплой	deploy_duration_seconds	< 300 секунд
Ошибки	platform_api_error_rate	< 1%
Очередь	pipeline_queue_duration_seconds	< 60 секунд
Загрузка	runner_active_count	> 1

3.10. SLO платформы

```
# slo-config.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: platform slos
data:
  slos.yaml: |
    - name: preview_env_creation
      sli: latency
      threshold: 120s
      slo: 0.95
      window: 7d

    - name: terraform_plan_success
      sli: success_rate
      threshold: 0.99
      window: 1d

    - name: pipeline_availability
      sli: availability
      threshold: 0.999
      window: 30d
```

4. Контрольные вопросы

1. В чем разница между наблюдаемостью приложения и наблюдаемостью платформы?
2. Какие три столпа наблюдаемости выделяют в OpenTelemetry?
3. Какую роль выполняет OpenTelemetry Collector?
4. Какие типы receiver'ов поддерживает OpenTelemetry Collector?
5. Как инструментировать приложение на Python для отправки трасс?
6. Что такое RED метод и как он применяется к платформенным сервисам?
7. Какие метрики критичны для платформы управления инфраструктурой?

8. Как настроить экспорт метрик из GitLab CI в OpenTelemetry Collector?
9. Для чего нужен процессор batch в OpenTelemetry Collector?
10. Как организовать распределённую трассировку через несколько платформенных сервисов?
11. Что такое контекстная корреляция (context propagation) и зачем она нужна?
12. Как выявить узкое место в платформе с помощью трассировки?
13. Какие best practices сбора метрик платформы вы знаете?
14. Как настроить алерты на основе платформенных метрик?
15. Что такое Service Level Objective (SLO) для платформы?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Развернуть OpenTelemetry Collector в Kubernetes с настроенными receivers и exporters
3. Инструментировать платформенный сервис согласно варианту
4. Настроить сбор метрик в Prometheus и трасс в Jaeger/Tempo
5. Создать дашборд в Grafana для визуализации ключевых метрик
6. Проанализировать собранные данные и выявить узкие места
7. Предоставить отчет с конфигурациями, кодом и скриншотами

5.2. Варианты заданий

Вар.	Платформенный сервис	Язык	Тип телеметрии	Особенность
1	Terragrunt wrapper	Python	Traces + Metrics	Трассировка plan/apply/destroy

Вар.	Платформенный сервис	Язык	Тип телеметрии	Особенность
2	GitLab CI custom executor	Go	Metrics + Logs	Сбор метрик очереди пайплайнов
3	Preview environment operator	Python	Traces + Metrics	Время создания окружения
4	Kubernetes admission webhook	Go	Traces	Проверка политик Gatekeeper
5	Argo CD operator	Python	Metrics + Logs	Синхронизация состояния
6	Vault secret injector	Go	Traces	Время получения секретов
7	Karpenter controller	Go	Metrics	Количество запущенных узлов
8	Crossplane provider	Python	Traces + Metrics	Провижининг ресурсов
9	Custom Terraform provider	Go	Metrics	Частота вызовов provider

Вар.	Платформенный сервис	Язык	Тип телеметрии	Особенность
10	Cloud cost optimizer	Python	Metrics + Traces	FinOps операции
11	Policy as Code engine (OPA)	Rego wrapper	Traces	Время оценки политик
12	Service catalog API	Go	Metrics	Частота запросов к каталогу
13	Backup operator	Python	Traces + Logs	Время бэкапа
14	Blue-green deploy controller	Go	Metrics + Traces	Время переключения трафика
15	Platform API gateway	Node.js	Metrics + Traces + Logs	Полная наблюдаемость

5.3. Конфигурация OpenTelemetry Collector

```
# collector-values.yaml
mode: deployment

config:
  receivers:
    otlp:
      protocols:
        grpc:
```

```
    endpoint: 0.0.0.0:4317
  http:
    endpoint: 0.0.0.0:4318

prometheus:
  config:
    scrape_configs:
      - job_name: 'platform-metrics'
        scrape_interval: 30s
        static_configs:
          - targets:
              - 'platform-api:8080'
              - 'terragrunt-wrapper:9090'

processors:
  batch:
    timeout: 1s
    send_batch_size: 1024

  memory_limiter:
    check_interval: 1s
    limit_mib: 512

exporters:
  prometheus:
    endpoint: "0.0.0.0:8889"
    namespace: platform

  otlp:
    endpoint: "jaeger-collector:4317"
    tls:
      insecure: true

  loki:
    endpoint: "http://loki:3100/loki/api/v1/push"

service:
  pipelines:
```

```

traces:
  receivers: [otlp]
  processors: [memory_limiter, batch]
  exporters: [otlp]

metrics:
  receivers: [otlp, prometheus]
  processors: [memory_limiter, batch]
  exporters: [prometheus]

logs:
  receivers: [otlp]
  processors: [batch]
  exporters: [loki]

```

5.4. Инструментирование платформенного API (Python)

```

# platform_api.py
import time
from opentelemetry import trace
from opentelemetry exporter otlp.proto.grpc.metric_exporter import
OTLPMetricExporter
from opentelemetry exporter otlp.proto.grpc.trace_exporter import
OTLPSpanExporter
from opentelemetry metrics import get_meter_provider,
set_meter_provider
from opentelemetry sdk metrics import MeterProvider
from opentelemetry sdk metrics.export import
PeriodicExportingMetricReader
from opentelemetry sdk resources import SERVICE_NAME, Resource
from opentelemetry sdk trace import TracerProvider
from opentelemetry sdk trace export import BatchSpanProcessor

# Настройка песочца
resource = Resource(attributes={
    SERVICE_NAME: "platform-api",
    "service.version": "1.0.0",

```

```

        "deployment.environment": "production"
    })

# Настройка трассировки
trace_provider = TracerProvider(resource=resource)
trace_processor = BatchSpanProcessor(
    OTLPSpanExporter(endpoint="http://otel-collector:4317",
insecure=True)
)
trace_provider.add_span_processor(trace_processor)
trace.set_tracer_provider(trace_provider)

# Настройка метрик
metric_reader = PeriodicExportingMetricReader(
    OTLPMetricExporter(endpoint="http://otel-collector:4317",
insecure=True)
)
meter_provider = MeterProvider(resource=resource,
metric_readers=[metric_reader])
set_meter_provider(meter_provider)

tracer = trace.get_tracer(__name__)
meter = get_meter_provider().get_meter("platform.api")

# Создание метрик
operation_counter = meter.create_counter(
    "platform.api.operations_total",
    description="Total number of platform API operations"
)

operation_duration = meter.create_histogram(
    "platform.api.operation_duration_seconds",
    description="Duration of platform API operations"
)

active_operations = meter.create_up_down_counter(
    "platform.api.active_operations",
    description="Number of active operations"

```



```

)

@app.route("/api/terragrunt/plan", methods=["POST"])
def terragrunt_plan():
    with tracer.start_as_current_span("terragrunt.plan") as span:
        start_time = time.time()
        active_operations.add(1)
        operation_counter.add(1, {"operation": "plan", "status":
"started"})

        try:
            # Имитация выполнения плана
            result = run_terragrunt_plan(request.json)
            duration = time.time() - start_time
            operation_duration.record(duration, {"operation": "plan",
"status": "success"})
            operation_counter.add(1, {"operation": "plan", "status":
"success"})

            span.set_attribute("plan.duration_ms", duration * 1000)
            span.set_status(trace.StatusCode.OK)
            return {"status": "success", "duration_ms": duration *
1000}

        except Exception as e:
            duration = time.time() - start_time
            operation_duration.record(duration, {"operation": "plan",
"status": "error"})
            operation_counter.add(1, {"operation": "plan", "status":
"error"})

            span.set_status(trace.StatusCode.ERROR, str(e))
            return {"status": "error", "error": str(e)}, 500

        finally:
            active_operations.add(-1)

```

5.5. Дашборд Grafana

```

{
  "dashboard": {

```

```

    "title": "Platform Observability",
    "panels": [
      {
        "title": "API Request Rate",
        "targets": [
          {
            "expr": "rate(platform_api_operations_total[5m])",
            "legendFormat": "{{operation}}"
          }
        ]
      },
      {
        "title": "Operation Duration (P95)",
        "targets": [
          {
            "expr": "histogram_quantile(0.95,
sum(rate(platform_api_operation_duration_seconds_bucket[5m])) by (le,
operation))",
            "legendFormat": "{{operation}}"
          }
        ]
      },
      {
        "title": "Active Operations",
        "targets": [
          {
            "expr": "platform_api_active_operations",
            "legendFormat": "active"
          }
        ]
      },
      {
        "title": "Error Rate",
        "targets": [
          {
            "expr":
"sum(rate(platform_api_operations_total{status=\"error\"}[5m])) /
sum(rate(platform_api_operations_total[5m]))",

```

```

        "legendFormat": "error_rate"
    }
]
},
{
    "title": "Preview Environment Creation Time",
    "targets": [
        {
            "expr": "histogram_quantile(0.95,
sum(rate(preview_env_create_duration_seconds_bucket[5m])) by (le))",
            "legendFormat": "P95"
        },
        {
            "expr": "histogram_quantile(0.99,
sum(rate(preview_env_create_duration_seconds_bucket[5m])) by (le))",
            "legendFormat": "P99"
        }
    ],
    "alert": {
        "conditions": [
            {
                "evaluator": {"type": "gt", "params": [120]},
                "unitary": {"type": "gt", "params": [120]}
            }
        ],
        "name": "Preview environment creation too slow"
    }
}
]
}
}

```

5.6. Алертинг в Prometheus

```

# prometheus-alerts.yaml
groups:
  - name: platform_alerts

```

```

rules:
  - alert: HighPlatformAPILatency
    expr: |
      histogram_quantile(0.99,
sum(rate(platform_api_operation_duration_seconds_bucket[5m])) by (le,
operation)
      ) > 60
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High latency on {{ $labels.operation }}"
      description: "P99 latency is {{ $value }}s, exceeding 60s
threshold"

  - alert: HighPlatformErrorRate
    expr: |
      sum(rate(platform_api_operations_total{status="error"}[5m]))
/
      sum(rate(platform_api_operations_total[5m])) > 0.05
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High error rate on platform API"
      description: "Error rate is {{ $value | humanizePercentage
}}}"

  - alert: PreviewEnvironmentCreationSlow
    expr: |
      histogram_quantile(0.95,
      sum(rate(preview_env_create_duration_seconds_bucket[10m]))
by (le)
      ) > 180
    for: 10m
    labels:
      severity: warning

```

```
annotations:  
  summary: "Preview environment creation is slow"  
  description: "P95 creation time is {{ $value }}s, expected <  
180s"
```

```
- alert: NoActiveRunners  
  expr: runner_active_count == 0  
  for: 5m  
  labels:  
    severity: critical  
  annotations:  
    summary: "No active GitLab runners"  
    description: "No runners available for pipeline execution"
```

5.7. Тестовый сценарий

```
# Симуляция нагрузки на платформенный API  
for i in {1..100}; do  
  curl -X POST http://platform-api/api/terraform/plan \  
    -H "Content-Type: application/json" \  
    -d '{"module_path": "environments/dev/network", "variables":  
{"env": "dev"}}'  
  sleep 0.5  
done
```

5.8. Ожидаемые метрики

Метрика	Значение (ожидаемое)
platform_api_operations_total	100 (при симуляции)
platform_api_operation_duration_seconds	~30-60 секунд
platform_api_active_operations	0-2

Метрика	Значение (ожидаемое)
preview_env_create_duration_seconds	~120 секунд
pipeline_queue_duration_seconds	~5-10 секунд

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (наблюдаемость платформы, OpenTelemetry Collector, RED метод)
5. Схема архитектуры (взаимодействие компонентов)
6. Конфигурация OpenTelemetry Collector
7. Код инструментирования платформенного сервиса
8. Конфигурация Prometheus и Grafana
9. Результаты выполнения:
10. Скриншоты логов Collector
11. Скриншоты дашборда Grafana
12. Скриншоты трасс в Jaeger/Tempo
13. Скриншоты алертов
14. Анализ собранных данных (таблицы, графики)
15. Заключение
16. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов

Критерий	Макс. баллов
Развертывание OpenTelemetry Collector	15
Инструментирование платформенного сервиса	25
Настройка экспорта метрик и трасс	15
Создание дашборда в Grafana	15
Настройка алертов	10
Демонстрация сбора телеметрии	10
Ответы на контрольные вопросы	10
Качество отчета	5
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Collector развернут, все 3 типа телеметрии собираются, дашборд отражает метрики, алерты настроены, анализ проведён
Хорошо	75-89	Метрики и трассы

Оценка	Баллы	Требования
		собираются, но нет логов или алертов
Удовлетворительно	60-74	Collector работает, но инструментирование неполное
Неудовлетворительно	<60	Collector не развернут, телеметрия не собирается

Штрафные баллы

Нарушение	Штраф
Нет трассировки (только метрики)	-10
Нет дашборда	-10
Collector без exporters	-10
Отсутствует анализ собранных метрик	-5

Лабораторная работа №14

SLO и Error Budget платформы

1. Цель работы: Научиться управлять надежностью платформы как продукта с использованием SLO (Service Level Objectives) и Error Budget, определить ключевые SLI для платформенных сервисов, настроить мониторинг соблюдения SLO и алерты на сжигание бюджета ошибок.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить концепции SLI, SLO, SLA и Error Budget применительно к платформе
2. Определить ключевые SLI для платформенных сервисов (на основе ЛР №13)
3. Установить целевые SLO для каждого SLI
4. Рассчитать Error Budget и настроить его мониторинг
5. Настроить алерты на скорость сжигания бюджета ошибок (burn rate)
6. Создать SLO-дашборд в Grafana для визуализации соблюдения SLO
7. Продемонстрировать сценарии исчерпания Error Budget

3. Краткое теоретическое обоснование

3.1. SLI, SLO, SLA для платформы

Понятие	Определение	Пример для платформы
SLI (Service Level Indicator)	Количественная метрика качества сервиса	Время создания preview-окружения
SLO (Service Level Objective)	Целевое значение SLI	95% окружений создаются за < 120 секунд
SLA (Service Level Agreement)	Юридическое обязательство перед пользователем	Штраф при недоступности > 0.1% в месяц

Важно: SLO должны быть строже SLA (обычно $SLO = SLA \times 0.9$)

3.2. SLI для платформенных сервисов

Категория	SLI	Метрика
Доступность	Успешность запросов к API	$\frac{\text{success\_requests}}{\text{total\_requests}}$
Латентность	Время выполнения операции	P95, P99 latency
Производительность	Пропускная способность	Количество операций в секунду
Качество	Успешность деплоев	$\frac{\text{successful\_deploys}}{\text{total\_deploys}}$

3.3. Error Budget

Error Budget - это допустимое количество ошибок за период времени, рассчитываемое как:

$$\text{Error Budget} = 1 - \text{SLO}$$

Пример:

$$\text{SLO} = 99.9\% \text{ (три девятки)}$$

$$\text{Error Budget} = 0.1\% \text{ (или 43 минуты недоступности в месяц)}$$

Использование Error Budget:

Сценарий	Действие
Бюджет > 50%	Разработка новых функций разрешена
Бюджет между 50% и 10%	Требуется подтверждение на рискованные изменения
Бюджет < 10%	Фокус на надежность, новые функции

Сценарий	Действие
	замораживаются
Бюджет = 0%	Только исправления критических проблем

3.4. Burn Rate (скорость сжигания бюджета)

Burn Rate показывает, насколько быстро расходуется Error Budget:

Burn Rate	Интерпретация
1	Бюджет будет исчерпан ровно в конце периода
< 1	Бюджет расходуется медленнее
> 1	Бюджет расходуется быстрее

Алерты на основе Burn Rate:

Burn Rate	Длительность	Действие
> 1	1 час	Предупреждение (warning)
> 3	30 минут	Критический (critical)
> 10	5 минут	Срочный (page)

3.5. Расчет Error Budget

```
def calculate_error_budget(slo_percent, total_requests,
error_requests):
    """
```

Расчет оставшегося бюджета ошибок

Args:

slo_percent: SLO в процентах (например, 99.9)

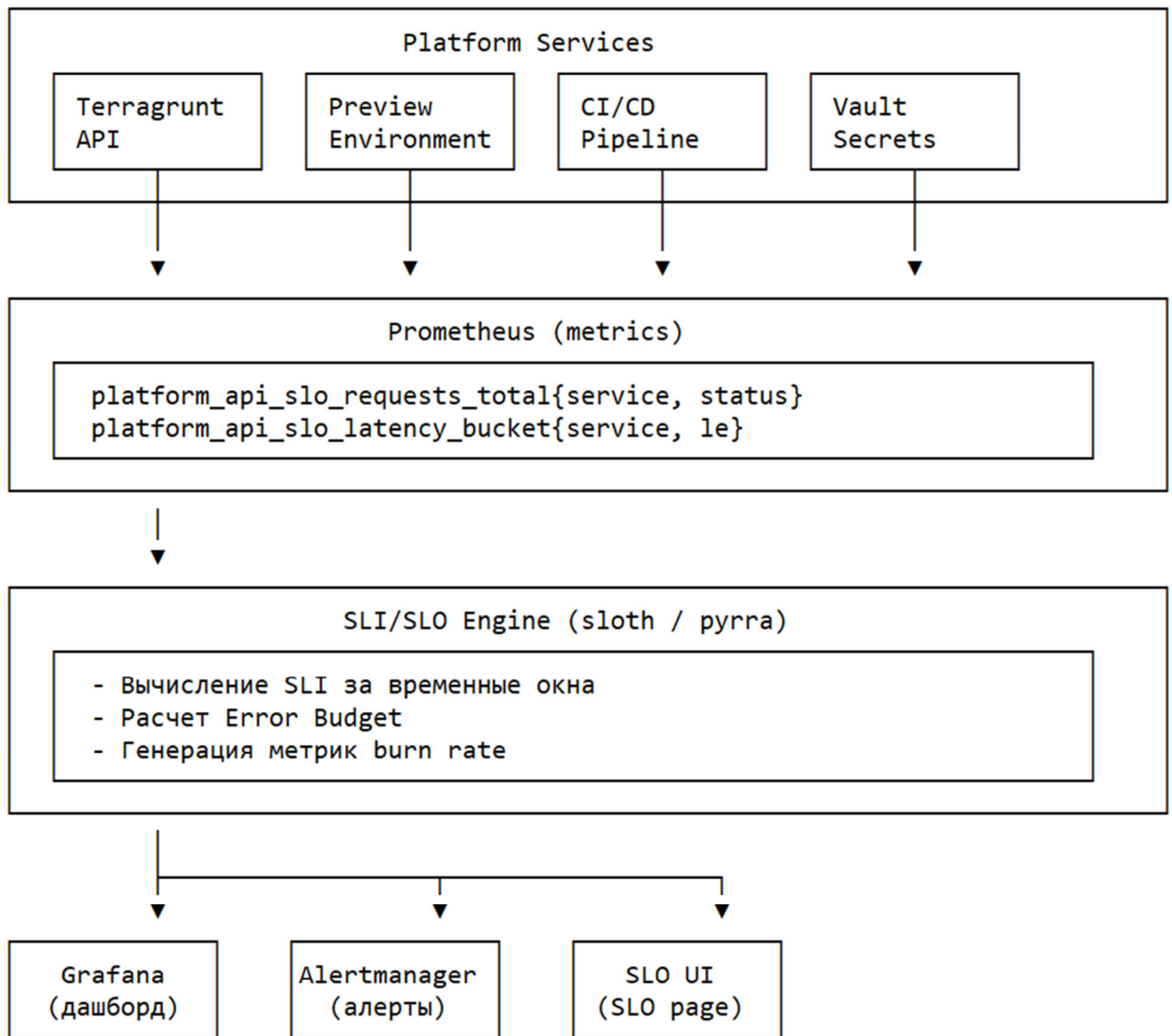
```

        total_requests: Общее количество запросов
        error_requests: Количество ошибочных запросов
    """
    allowed_errors = total_requests * (1 - slo_percent / 100)
    remaining_errors = allowed_errors - error_requests
    remaining_budget_percent = (remaining_errors / allowed_errors) *
100

    return {
        "allowed_errors": allowed_errors,
        "error_requests": error_requests,
        "remaining_errors": remaining_errors,
        "remaining_budget_percent": remaining_budget_percent,
        "burn_rate": error_requests / (allowed_errors / 30) # за 30
дней
    }

```

3.6. Архитектура SLO мониторинга



3.7. Sloth для генерации SLO метрик

Sloth - инструмент для генерации Prometheus правил SLO на основе конфигурации:

```
# slo-config.yaml
version: "1.0"
service: "platform-api"
labels:
  team: "platform"
  env: "production"

slos:
  - name: "api-availability"
```

```

objective: 99.9
description: "API успешно отвечает на запросы"

sli:
  events:
    error_query: |

sum(rate(platform_api_requests_total{status="error"}[{{.window}}]))
  total_query: |
    sum(rate(platform_api_requests_total[{{.window}}]))

alerting:
  page_alert:
    labels:
      severity: "page"
    annotations:
      summary: "API availability SLO violation"
  ticket_alert:
    labels:
      severity: "ticket"

- name: "preview-latency"
  objective: 95
  description: "Preview окружение создается за < 120 секунд"

sli:
  latency:
    query: |
      histogram_quantile(0.95,

sum(rate(preview_env_create_duration_seconds_bucket[{{.window}}])) by (le)
  )
  threshold: 120

```

3.8. Pyrra для SLO управления

Pyrra - альтернатива Sloth с веб-интерфейсом для управления SLO:

```

# pyrra-slo.yaml
apiVersion: pyrra.dev/v1alpha1
kind: ServiceLevelObjective
metadata:
  name: platform api availability
spec:
  service: platform api
  description: "API успешно обрабатывает запросы"
  indicator:
    ratio:
      errors:
        metric: platform_api_requests_total{status="error"}
      total:
        metric: platform_api_requests_total
  target: "99.9"
  window: 30d

```

3.9. Дашборд SLO

Ключевые элементы SLO дашборда:

Компонент	Описание
SLO status	Текущее состояние (достигается/нарушается)
Error Budget	Остаток бюджета в процентах и днях
Burn rate graph	График скорости сжигания за последние часы
SLI timeline	Исторические значения SLI
Alert burn rate	Пороги алертов

3.10. Управление инцидентами на основе SLO

При срабатывании алерта на высокий burn rate:

Инцидент: High SLO burn rate

Время: 2026-05-14 14:30 UTC

SLI: api-availability (SLO: 99.9%)

Current burn rate: 12.5 (за последние 5 минут)

Действия:

1. [] Остановить все non-critical deployments
2. [] Проверить последние изменения в платформе
3. [] Провести blameless postmortem
4. [] Обновить статус SLO в документации

Эскалация:

- @oncall-sre (немедленно)
- @platform-team (после 15 минут)
- @engineering-manager (после 30 минут)

3.11. Обновление SLO на основе данных

SLO не статичны - их следует пересматривать:

Частота	Действие
Еженедельно	Проверка burn rate, корректировка алертов
Ежемесячно	Анализ соблюдения SLO, пересмотр целей
Ежеквартально	Обновление SLO на основе бизнес-требований

4. Контрольные вопросы

1. Что такое SLI, SLO и SLA? В чем разница между ними?
2. Как рассчитывается Error Budget? Приведите формулу.
3. Какие SLI критичны для платформы управления инфраструктурой?
4. Что такое burn rate и как его использовать для алертинга?

5. Какие действия предпринимаются при исчерпании Error Budget?
6. Как настроить SLO для сервиса с несколькими типами запросов?
7. В чем разница между SLO доступности и SLO латентности?
8. Как выбрать окно для SLO (7 дней, 30 дней)? Какие компромиссы?
9. Какие инструменты существуют для управления SLO (Sloth, Pyrra)?
10. Как связаны SLO платформы и SLO приложений, работающих на платформе?
11. Как часто следует пересматривать SLO?
12. Что происходит, когда SLO не достигается несколько месяцев подряд?
13. Как визуализировать Error Budget на дашборде?
14. Какие типы алертов следует настроить для SLO?
15. Как интегрировать SLO в процесс принятия решений о релизах?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Определить 3 SLI для заданного платформенного сервиса
3. Установить SLO для каждого SLI
4. Настроить сбор метрик в Prometheus (на основе ЛР №13)
5. Настроить Sloth или Pyrra для генерации SLO метрик
6. Создать SLO дашборд в Grafana
7. Настроить алерты на burn rate
8. Продемонстрировать работу SLO и обработку инцидентов
9. Предоставить отчет с конфигурациями, скриншотами и анализом

5.2. Варианты заданий

Вар.	Платформенный сервис	SLI 1	SLI 2	SLI 3	SLO цель
1	Terragrunt API	Доступность	Латентность plan	Латентность apply	99.9% P95<60s
2	Preview Environment	Доступность	Время создания	Время удаления	95% <180s <60s
3	CI/CD Pipeline	Успешность	Длительность	Очередь	98% <10min <60s
4	GitLab Runner	Доступность	Время ответа	Загрузка	99.5% P99<5s
5	Vault Secrets	Доступность	Время получения	Частота ошибок	99.99% <100ms
6	Karpenter	Успешность провижинга	Время запуска узла	Частота вытеснений	99% <120s <5%
7	Argo CD	Доступность API	Время синхронизации	Успешность синхр.	99.9% <60s

Вар.	Платформенный сервис	SLI 1	SLI 2	SLI 3	SLO цель
8	Crossplane	Доступность	Время провижинга	Успешность	99.5% / <300s
9	OPA Gatekeeper	Доступность webhook	Латентность проверки	Успешность	99.99% / <50ms
10	Service Catalog	Доступность	Время ответа	Точность данных	99.5% / P95<200ms
11	Backup Operator	Успешность бэкапа	Время бэкапа	Время восстановления	99.9% / <1h / <4h
12	Cost Optimizer	Доступность	Точность прогнозов	Время расчета	99% / 95% / <5min
13	Platform API Gateway	Доступность	Латентность	Error rate	99.95% / P99<100ms
14	Blue-green Controller	Доступность	Время переключения	Успешность	99.9% / <30s

Вар.	Платформенный сервис	SLI 1	SLI 2	SLI 3	SLO цель
			ючения	ь	99.9%
15	Multi-platform SLO	Комбини рованны й	(выбрат ь 3)	(выбр ать 3)	99.9%

5.3. Конфигурация Sloth

```
# slo-config.yaml
version: "1.0"
service: "platform-preview"
labels:
  team: "platform"
  service_type: "preview"

slos:
- name: "preview-availability"
  objective: 95
  description: "Preview окружения успешно создаются"

sli:
  events:
    error_query: |
      sum by (namespace) (
        rate(preview_env_create_total{status="error"}[{{.window}}])
      )
  total_query: |
    sum by (namespace) (
      rate(preview_env_create_total[{{.window}}])
    )

alerting:
```

```

page_alert:
  labels:
    severity: "page"
    slo: "preview-availability"
  annotations:
    summary: "Preview environment availability SLO violation"
    description: "SLO {{ .value }}% for {{ .labels.service }}"
  threshold: 95
ticket_alert:
  labels:
    severity: "ticket"
  threshold: 97

- name: "preview-latency"
  objective: 95
  description: "Preview окружение создается за < 180 секунд"

sli:
  latency:
    query: |
      histogram_quantile(0.95,
        sum by (namespace) (
          rate(preview_env_create_duration_seconds_bucket[{{.window}}])
            ) by (le, namespace)
        )
    threshold: 180

alerting:
  page_alert:
    labels:
      severity: "page"
    annotations:
      summary: "Preview environment creation too slow"
    threshold: 180

- name: "preview-cleanup"
  objective: 99

```

```

description: "Preview окружения удаляются после закрытия PR"

sli:
  events:
    error_query: |
      sum(rate(preview_env_delete_orphaned_total[{{.window}}]))
    total_query: |
      sum(rate(preview_env_delete_total[{{.window}}]))

alerting:
  ticket_alert:
    labels:
      severity: "ticket"
    threshold: 99

```

5.4. Запуск Sloth и генерация правил

```

# Установка Sloth
curl -L https://github.com/slok/sloth/releases/latest/download/sloth-
linux-amd64 -o sloth
chmod +x sloth
sudo mv sloth /usr/local/bin/

# Генерация Prometheus правил
sloth generate -c slo-config.yaml -o prometheus-rules.yaml

# Проверка правил
sloth validate -c slo-config.yaml

```

5.5. Prometheus правила (сгенерированные Sloth)

```

# prometheus-rules.yaml
groups:
- name: "platform-preview.slos"
  interval: 1m
  rules:

```

```

# SLI метрики за разные окна
- record: "slo:preview_availability:error_budget:30d"
  expr: |
    max_overall = 1 - 0.95
    (1 - (
      sum_overall_errors / sum_overall_total
    )) / max_overall

# Burn rate за разные периоды
- record: "slo:preview_availability:error_burn_rate:5m"
  expr: |
    (1 - (
      sum_overall_errors[5m] / sum_overall_total[5m]
    )) / (1 - 0.95)

- record: "slo:preview_availability:error_burn_rate:1h"
  expr: |
    (1 - (
      sum_overall_errors[1h] / sum_overall_total[1h]
    )) / (1 - 0.95)

- record: "slo:preview_availability:error_burn_rate:1d"
  expr: |
    (1 - (
      sum_overall_errors[1d] / sum_overall_total[1d]
    )) / (1 - 0.95)

```

5.6. Алерты на Burn Rate

```

# prometheus-alerts-slo.yaml
groups:
- name: slo_alerts
  interval: 1m
  rules:

```

- `alert: PreviewSLOBurnRateHigh`
`expr: |`
`slo:preview_availability:error_burn_rate:5m > 3`
`and`
`slo:preview_availability:error_burn_rate:1h > 2`
`for: 2m`
`labels:`
`severity: critical`
`slo: preview availability`
`annotations:`
`summary: "High SLO burn rate for preview environments"`
`description: |`
`Burn rate is {{ $value }} (threshold 3) for last 5 minutes.`
`Remaining error budget: {{ printf "%.1f" $value }} days.`
`Action: Stop non-critical changes.`

- `alert: PreviewSLOBurnRateWarning`
`expr: slo preview_availability:error_burn_rate:1d > 1.5`
`for: 10m`
`labels:`
`severity: warning`
`annotations:`
`summary: "SLO burn rate above normal"`

- `alert: PreviewSLOViolation`
`expr: slo preview_availability:error_budget:30d <= 0`
`for: 5m`
`labels:`
`severity: critical`
`annotations:`
`summary: "Preview environment SLO violated"`
`description: "Error budget exhausted for 30d window"`

- `alert: PreviewSLONearViolation`
`expr: slo preview_availability:error_budget:30d < 0.1`
`for: 15m`
`labels:`
`severity: warning`


```
    annotations:
      summary: "Preview environment SLO nearly violated"
      description: "Only {{ $value | humanizePercentage }} of error
budget remaining"
```

5.7. Дашборд SLO в Grafana

```
{
  "dashboard": {
    "title": "Platform SLO Dashboard",
    "panels": [
      {
        "title": "SLO Status Overview",
        "type": "stat",
        "targets": [
          {
            "expr": "slo:preview_availability:error_budget:30d",
            "legendFormat": "Preview Availability"
          },
          {
            "expr": "slo:preview_latency:error_budget:30d",
            "legendFormat": "Preview Latency"
          },
          {
            "expr": "slo:preview_cleanup:error_budget:30d",
            "legendFormat": "Preview Cleanup"
          }
        ],
        "fieldConfig": {
          "defaults": {
            "unit": "percent",
            "thresholds": {
              "steps": [
                {"color": "green", "value": 0},
                {"color": "yellow", "value": 10},
                {"color": "red", "value": 5}
              ]
            }
          }
        }
      }
    ]
  }
}
```

```

    }
  }
},
{
  "title": "Burn Rate (5m vs 1h)",
  "type": "timeseries",
  "targets": [
    {
      "expr": "slo:preview_availability:error_burn_rate:5m",
      "legendFormat": "5m burn rate"
    },
    {
      "expr": "slo:preview_availability:error_burn_rate:1h",
      "legendFormat": "1h burn rate"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "thresholds": {
        "steps": [
          {"color": "green", "value": 0},
          {"color": "yellow", "value": 1},
          {"color": "orange", "value": 3},
          {"color": "red", "value": 5}
        ]
      }
    }
  }
},
{
  "title": "Remaining Error Budget (days)",
  "type": "gauge",
  "targets": [
    {
      "expr":
"slo:preview_availability:error_budget_remaining_days",
      "legendFormat": "days remaining"
    }
  ]
}

```

```

    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "d",
      "thresholds": {
        "steps": [
          {"color": "red", "value": 0},
          {"color": "yellow", "value": 3},
          {"color": "green", "value": 7}
        ]
      }
    }
  }
},
{
  "title": "SLI Timeline (30d window)",
  "type": "timeseries",
  "targets": [
    {
      "expr": "slo:preview_availability:sli:30d",
      "legendFormat": "Actual SLI"
    },
    {
      "expr": "0.95",
      "legendFormat": "SLO Target (95%)"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
      "min": 90,
      "max": 100
    }
  }
},
{
  "title": "Error Budget Consumption",

```

```

    "type": "timeseries",
    "targets": [
      {
        "expr": "1 - slo:preview_availability:error_budget:30d",
        "legendFormat": "Budget consumed"
      }
    ],
    "fieldConfig": {
      "defaults": {
        "unit": "percent",
        "color": {"mode": "thresholds"},
        "thresholds": {
          "steps": [
            {"color": "green", "value": 0},
            {"color": "yellow", "value": 80},
            {"color": "orange", "value": 90},
            {"color": "red", "value": 95}
          ]
        }
      }
    }
  ],
  "refresh": "1m"
}

```

5.8. Демонстрация сценариев

Сценарий 1: Нормальная работа (SLO достигается)

```

# Симуляция успешных операций
for i in {1..1000}; do
  curl -X POST platform-api/preview/create -d '{"pr_id": 42}'
  sleep 0.1
done
# Ожидаемый результат: Error Budget > 50%

```

Сценарий 2: Нарушение SLO (высокий burn rate)

```
# Симуляция ошибок (90% неудачных операций)
for i in {1..100}; do
    curl -X POST platform-api/preview/create -d '{"pr_id": 42}' &
done
# Ожидаемый результат: Burn rate > 5, алерт срабатывает
```

Сценарий 3: Истощение Error Budget

```
# Длительная симуляция ошибок
for day in {1..30}; do
    for i in {1..1000}; do
        curl -X POST platform-api/preview/create -d '{"pr_id": 42}' &
    done
    sleep 86400 # 24 часа (только для демонстрации, используйте
ускоренный режим)
done
# Ожидаемый результат: Error Budget = 0%, новые фичи блокируются
```

5.9. Интеграция с CI/CD (Quality Gate)

```
# .gitlab-ci.yml quality gate
slo-check:
  stage: pre deploy
  script:
    - |
      # Проверка Error Budget перед деплоем
      ERROR_BUDGET=$(curl -s
http://prometheus/api/v1/query?query=slo:preview_availability:error_budget:30
d | jq '.data.result[0].value[1]')
      if [ $(echo "$ERROR_BUDGET < 0.1" | bc) -eq 1 ]; then
        echo "Error budget below 10% - blocking deployment"
        exit 1
      else
        echo "Error budget is ${ERROR_BUDGET}% - deployment allowed"
```

```
fi
rules:
- if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

5.10. SLO статус страница

html

```
<!-- status.html -->
<!DOCTYPE html>
<html>
<head>
<title>Platform SLO Status</title>
<meta http-equiv="refresh" content="60">
</head>
<body>
<h1>Platform Service Level Objectives</h1>
<table>
<tr>
<th>Service Level</th>
<th>Target</th>
<th>Actual (30d)</th>
<th>Status</th>
<th>Remaining Budget</th>
</tr>
<tr>
<td>Preview Availability</td>
<td>95%</td>
<td id="availability-value">--%</td>
<td id="availability-status"></td>
<td id="availability-budget"></td>
</tr>
<tr>
```

```

<td>Preview Latency (P95)</td>
<td>< 180s</td>
<td id="latency-value">--s</td>
<td id="latency-status"></td>
<td id="latency-budget"></td>
</tr>
</table>
<script>
  setInterval(async () => {
    const response = await fetch('/api/slo-status');
    const data = await response.json();

    // Обновление значений
    document.getElementById('availability-value').innerText =
data.availability.actual + '%';
    document.getElementById('availability-status').innerText =
data.availability.meets_slo ? '✓' : '✗';
    document.getElementById('availability-budget').innerText =
data.availability.error_budget_remaining + '%';
  }, 5000);
</script>
</body>
</html>

```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы

4. Теоретическая часть (SLI/SLO/Error Budget, Burn Rate, Sloth/Pyrra)
5. Определение SLI для выбранного сервиса (таблица с обоснованием)
6. SLO конфигурация (Sloth YAML)
7. Prometheus правила и алерты
8. Grafana дашборд (скриншот и JSON)
9. Результаты выполнения:
- 10.Скриншоты дашборда с SLO статусом
- 11.Скриншоты алертов в Alertmanager
- 12.Демонстрация сценариев (нормальный, нарушение, исчерпание)
- 13.Анализ эффективности SLO (таблица метрик за период)
- 14.Заключение
- 15.Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Определение SLI и установка SLO	15
Настройка Sloth/Pyrra	15
Конфигурация Prometheus правил	15
Настройка алертов на burn rate	15
Создание SLO дашборда в Grafana	15
Демонстрация сценариев	10
Ответы на контрольные вопросы	10

Критерий	Макс. баллов
Качество отчета	5
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Все 3 SLO настроены, метрики собираются, алерты срабатывают на high burn rate, дашборд полный, оба сценария продемонстрированы
Хорошо	75-89	SLO настроены, но алерты не работают или нет интеграции с CI/CD
Удовлетворительно	60-74	SLO настроены только для одного SLI, дашборд неполный
Неудовлетворительно	<60	SLO не настроены, метрики не собираются

Штрафные баллы

Нарушение	Штраф
Нет обоснования выбора SLO	-10
Отсутствует дашборд	-10

Нарушение	Штраф
Нет алертов на burn rate	-15
Sloth конфигурация не проходит валидацию	-5

Лабораторная работа №15

Управление конфигурациями устройств через Git + OPA

1. Цель работы: Научиться управлять конфигурациями парка автономных устройств (БПЛА, роботов, IoT-устройств) через GitOps подход с использованием Open Policy Agent (OPA) для автоматической валидации изменений конфигураций перед применением.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Создать Git-репозиторий для хранения конфигураций устройств
2. Разработать структуру конфигурационных файлов для парка устройств
3. Написать OPA-политики для валидации конфигураций (безопасность, ограничения, обязательные поля)
4. Настроить CI/CD пайплайн для автоматической проверки конфигураций при Pull Request
5. Реализовать автоматическое применение конфигураций через GitOps оператор (Flux/ArgoCD)
6. Продемонстрировать процесс изменения конфигурации устройства с валидацией и применением

3. Краткое теоретическое обоснование

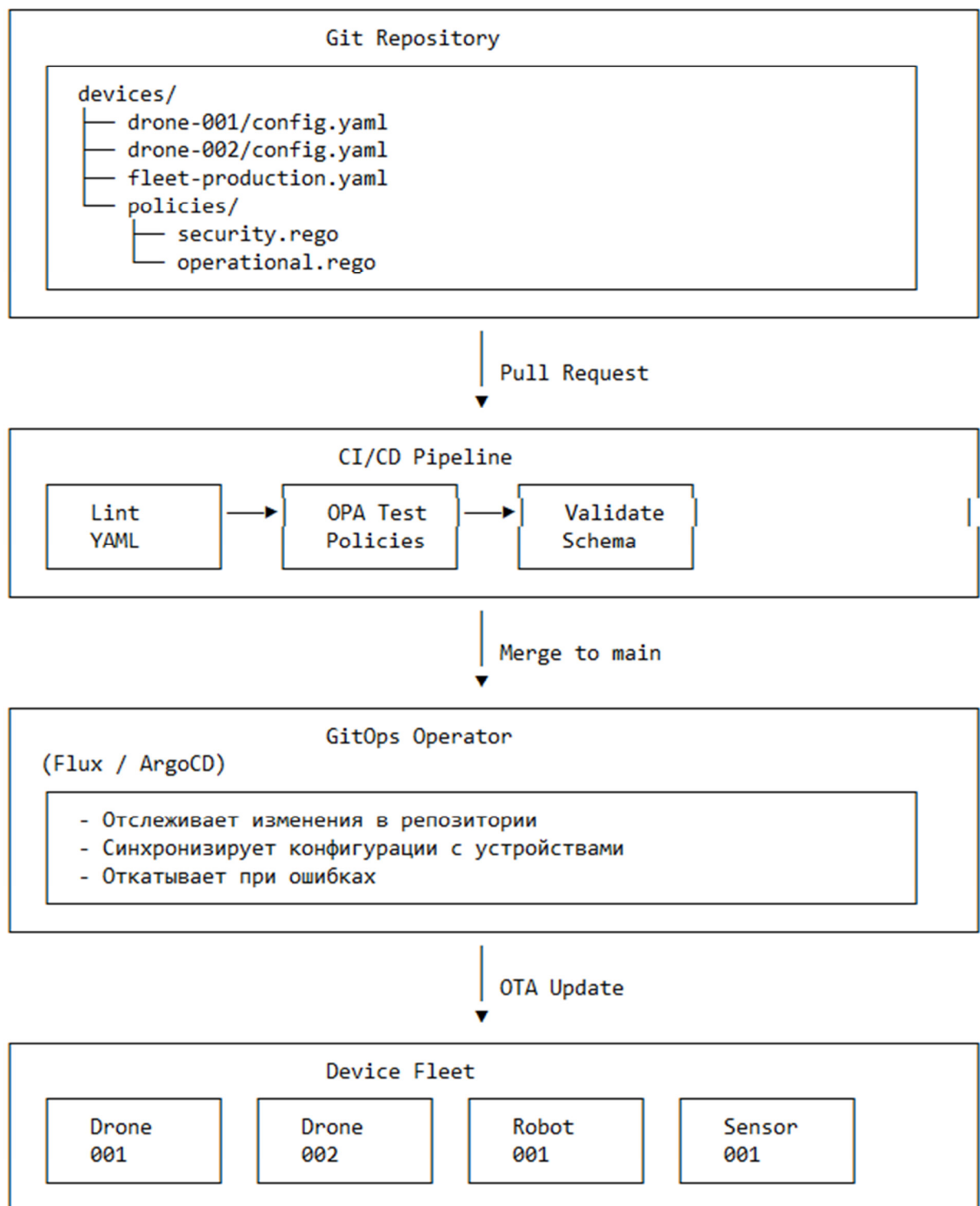
3.1. Проблема управления конфигурациями парка устройств

Управление тысячами распределенных устройств (БПЛА, роботов, IoT-датчиков) создает уникальные вызовы:

Проблема	Описание	Риски
Масштаб	Ручное изменение конфигурации каждого устройства невозможно	Неоднородность конфигураций, ошибки
Безопасность	Необходимо предотвращать опасные конфигурации	Превышение высоты, запрещенные зоны
Версионирование	Нет истории изменений, сложный откат	Трудности при расследовании инцидентов
Автоматизация	Отсутствие CI/CD для конфигураций	Долгий цикл изменений

3.2. GitOps для конфигураций устройств

GitOps подход для управления конфигурациями устройств:



3.3. Структура конфигурации устройства

```
# devices/drone-001/config.yaml
apiVersion: fleet.dev/v1
kind: DeviceConfiguration
metadata:
```

```
name: drone 001
namespace: production
labels:
  type: "quadcopter"
  region: "north"
  fleet: "delivery"
spec:
  device:
    id: "DR-001-ABCD"
    type: "DJI Matrice 300"
    firmware_version: "1.2.3"

  flight:
    max_altitude: 120          # метров
    max_speed: 15              # м/с
    max_distance: 5000        # метров
    no_fly_zones:
      - name: "airport"
        type: "circle"
        center: [55.751244, 37.618423]
        radius: 5000
      - name: "military-base"
        type: "polygon"
        points: [[55.75, 37.60], [55.76, 37.61], [55.75, 37.62]]

  camera:
    resolution: "4K"
    frame_rate: 30
    stream_url: "rtsp://camera.local/stream"

  telemetry:
    report_interval: 5          # секунд
    data_points: ["gps", "battery", "altitude", "speed"]

  safety:
    emergency_landing_enabled: true
    return_to_home_on_signal_loss: true
    max_signal_loss_time: 30    # секунд
```

```
battery_min_percent: 20

mission:
  allowed_modes: ["autonomous", "waypoint", "loiter"]
  default_mode: "waypoint"
```

3.4. ОРА политики для валидации конфигураций

```
# policies/device-config.rego
package device_config

import rego v1

# Проверка обязательных полей
required_fields = {
  "spec.device.id",
  "spec.device.type",
  "spec.flight.max_altitude",
  "spec.safety.battery_min_percent"
}

violation[msg] if {
  field := required_fields[_]
  not has_field(input, field)
  msg := sprintf("Missing required field: %s", [field])
}

has_field(obj, field_path) if {
  path := split(field_path, ".")
  value := walk(obj, path)
  value != null
}

# Ограничение высоты
max_allowed_altitude = 120

violation[msg] if {
```

```

        altitude := input spec.flight.max_altitude
        altitude > max_allowed_altitude
        msg := sprintf("Max altitude %d m exceeds limit of %d m",
[altitude, max_allowed_altitude])
    }

# Запрет опасных зон (пример)
dangerous_zones = {
    "airport", "military-base", "nuclear-plant"
}

violation|msg| if {
    zone := input spec.flight.no_fly_zones[_]
    zone name in dangerous_zones
    msg := sprintf("No-fly zone '%s' is not allowed", [zone.name])
}

# Проверка минимального заряда батареи
min_battery = 15

violation|msg| if {
    min_battery_percent := input.spec.safety.battery_min_percent
    min_battery_percent < min_battery
    msg := sprintf("Min battery %d%% is below required %d%%",
[min_battery_percent, min_battery])
}

# Разрешенные режимы полета
allowed_modes = {"autonomous", "waypoint", "loiter", "manual"}

violation|msg| if {
    mode := input spec.mission.allowed_modes[_]
    not mode in allowed_modes
    msg := sprintf("Mission mode '%s' is not allowed", [mode])
}

# Версия прошивки должна быть актуальной
required_firmware = "1.2.3"

```

```

violation[msg] if {
    firmware := input.spec.device.firmware_version
    firmware != required_firmware
    msg := sprintf("Firmware version %s does not match required %s",
[firmware, required_firmware])
}

# Комбинированное правило: высота и скорость не могут быть слишком
большими
violation[msg] if {
    altitude := input.spec.flight.max_altitude
    speed := input.spec.flight.max_speed
    altitude > 100
    speed > 20
    msg := sprintf("High altitude (%d m) and high speed (%d m/s)
combination is unsafe", [altitude, speed])
}

```

3.5. Пакетные конфигурации (флот)

```

# devices/fleet-production.yaml
apiVersion: fleet.dev/v1
kind: FleetConfiguration
metadata:
  name: production fleet
  namespace: production
spec:
  selectors:
    - type: "quadcopter"
    - region: "north"

  defaults:
    flight:
      max_altitude: 100
      max_speed: 12
    safety:
      battery_min_percent: 25

```



```

    emergency_landing_enabled: true

overrides:
  - selector:
      drone_id: "DR-001-ABCD"
    config:
      flight.max_altitude: 150
      camera.frame_rate: 60

```

3.6. GitOps с FluxCD

Установка FluxCD для синхронизации конфигураций:

```

# Установка Flux
flux bootstrap github \
  --owner=<org> \
  --repository=fleet-configs \
  --branch=main \
  --path=./devices \
  --personal

```

Kustomization для конфигураций:

```

yaml
# flux/kustomization.yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: device configs
  namespace: flux system
spec:
  interval: 1m
  path: ./devices
  prune: true
  validation: client
  healthChecks:
    - apiVersion: fleet.dev/v1
      kind: DeviceConfiguration
      name: drone-001

```

3.7. ОРА интеграция с Flux/ArgoCD

```
# Артефакт валидации перед применением
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: fleet configs
spec:
  interval: 30s
  url: https://gitlab.com/fleet/configs
  ref:
    branch: main
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: fleet configs
spec:
  interval: 1m
  path: ./devices
  prune: true
  validation: client
  postBuild:
    substitute:
      ENVIRONMENT: "production"
    substituteFrom:
      - kind: ConfigMap
        name: fleet policies
```

3.8. Мониторинг применения конфигураций

```
# prometheus/metrics.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: device agent
```

```
spec:
  selector:
    matchLabels:
      app: device agent
  endpoints:
    - port: metrics
      path: /metrics
      interval: 30s
# Метрики для отслеживания
# device_config_applied_total{device, status}
# device_config_last_applied_timestamp{device}
# device_config_validation_errors_total{device, rule}
```

4. Контрольные вопросы

1. Какие проблемы управления конфигурациями решает GitOps подход?
2. Как ОРА используется для валидации конфигураций устройств?
3. Какие типы политик безопасности необходимо применять к конфигурациям БПЛА?
4. Как организовать иерархию конфигураций (устройство → группа → флот)?
5. Как реализовать сапагу-обновления конфигураций?
6. Что происходит при нарушении политики ОРА в CI/CD пайплайне?
7. Как обеспечить откат конфигурации при проблемах?
8. Какие метрики важны для мониторинга состояния конфигураций устройств?
9. Как интегрировать ОРА с GitOps оператором (Flux/ArgoCD)?
10. Как обрабатывать конфликты при параллельных обновлениях конфигураций?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Создать Git-репозиторий для конфигураций устройств
3. Разработать структуру конфигураций для 5+ устройств
4. Написать 5+ OPA политик для валидации
5. Настроить CI/CD пайплайн (GitLab CI/GitHub Actions) с проверкой политик
6. Развернуть GitOps оператор (FluxCD или ArgoCD) для синхронизации
7. Продемонстрировать полный цикл: PR → валидация → merge → применение
8. Предоставить отчет с конфигурациями, политиками и демонстрацией

5.2. Варианты заданий

Вар.	Тип устройства	Ключевые параметры	Особые политики
	Квадрокоптер-доставщик	Высота, скорость, зоны доставки	Запрет полетов над жилыми зонами
2	Сельскохозяйственный дрон	Маршруты опрыскивания, химикаты	Ограничение ночных полетов
3	Промышленный робот	Зоны безопасности, скорости	Одновременное движение (deadlock)
4	Автономный	Маршруты,	Зоны с низкой

Вар.	Тип устройства	Ключевые параметры	Особые политики
	автомобиль	ограничения скорости	связью
3	Морской дрон	Глубина, зоны навигации	Запрет входа в порты
6	Охранный дрон	Время работы, зоны патрулирования	Запрет съемки частных зон
7	Складской робот	Пути, стеллажи, подъемы	Взаимные блокировки
8	Метеозонд	Высота, время полета	Запрет пересечения авиатрасс
9	Спасательный дрон	Радиус поиска, время автономии	Приоритет миссий
10	Строительный робот	Точность, зоны работ	Безопасность персонала
11	Трубопроводный дрон	Давление, температура, утечки	Частота инспекций
12	Подводный аппарат	Глубина, время погружения	Зоны с холодными течениями
	Энергоаудит дрон	Высоковольтные	Безопасные

Вар.	Тип устройства	Ключевые параметры	Особые политики
3		линии	дистанции
4	Реагирование на ЧС	Время отклика, загрузка датчиков	Приоритет экстренных вызовов
5	Комбинированный флот	3 типа устройств	Гибридные политики

5.3. Структура репозитория

fleet-configs/

├── devices/

| ├── drone-001/

| | └── config.yaml

| ├── drone-002/

| | └── config.yaml

| ├── robot-001/

| | └── config.yaml

| ├── fleet-production.yaml

| └── fleet-staging.yaml

├── policies/

| ├── security.rego

| ├── operational.rego

| ├── safety.rego

| └── test/

| | ├── security_test.rego

| | └── operational_test.rego

```
|— flux/
|   |— kustomization.yaml
|— .gitlab-ci.yml
|— Makefile
|— README.md
```

5.4. CI/CD конфигурация (GitLab CI)

```
# .gitlab-ci.yml
stages:
  - validate
  - test
  - deploy

variables:
  OPA_VERSION: "0.67.1"

.validate_base:
  stage: validate
  before_script:
    - curl -L -o opa https://github.com/open-policy-agent/opa/releases/download/v${OPA_VERSION}/opa_linux_amd64
    - chmod +x opa
    - mv opa /usr/local/bin/opa

validate-syntax:
  extends: .validate_base
  script:
    - for file in $(find devices -name "*.yaml"); do
      jq eval 'true' $file > /dev/null || exit 1;
    done
  rules:
    - if: $CI_MERGE_REQUEST_ID
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

validate-policies:
```

```

    extends: .validate_base
    script:
      - opa test policies/
      - opa check --strict policies/
    rules:
      - if: $CI_MERGE_REQUEST_ID
      - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

validate-device-configs:
    extends: .validate_base
    script:
      - |
        for config in $(find devices -name "config.yaml"); do
          echo "Validating $config"
          opa eval --data policies/ --input $config --format pretty \
            'data.device_config.violation' | grep -q "\[" && exit 1 ||
true
        done
    artifacts:
      reports:
        junit: junit.xml
    rules:
      - if: $CI_MERGE_REQUEST_ID
      - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

validate-fleet:
    extends: .validate_base
    script:
      - |
        for fleet in $(find devices -name "fleet-*.yaml"); do
          echo "Validating $fleet"
          opa eval --data policies/ --input $fleet --format pretty \
            'data.fleet_config.violation' | grep -q "\[" && exit 1 ||
true
        done
    rules:
      - if: $CI_MERGE_REQUEST_ID
      - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

```



```

deploy-flux:
  stage: deploy
  image: bitnami/kubectl:latest
  script:
    - kubectl apply -k ./flux
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
  environment:
    name: production

```

5.5. Тесты ОРА политик

```

# policies/test/security_test.rego
package device_config_test

import data device_config
import rego v1

test_valid_config if {
  config := {
    "spec": {
      "device": {"id": "DR-001", "type": "quadcopter",
"firmware_version": "1.2.3"},
      "flight": {"max_altitude": 100, "max_speed": 10,
"no_fly_zones": []},
      "safety": {"battery_min_percent": 25},
      "mission": {"allowed_modes": ["autonomous", "waypoint"]}
    }
  }
  count(device_config violation with input as config) == 0
}

test_invalid_altitude if {
  config := {
    "spec": {

```

```

        "device": { "id": "DR-001", "type": "quadcopter",
"firmware_version": "1.2.3"},
        "flight": { "max_altitude": 150, "max_speed": 10,
"no_fly_zones": []},
        "safety": { "battery_min_percent": 25},
        "mission": { "allowed_modes": ["autonomous"]}
    }
}

count(device_config violation with input as config) > 0
}

test_dangerous_zone if {
    config := {
        "spec": {
            "device": { "id": "DR-001", "type": "quadcopter",
"firmware_version": "1.2.3"},
            "flight": {
                "max_altitude": 100,
                "max_speed": 10,
                "no_fly_zones": [{"name": "airport"}]
            },
            "safety": { "battery_min_percent": 25},
            "mission": { "allowed_modes": ["autonomous"]}
        }
    }

    violations := {msg | device_config violation[msg] with input as
config}

    some msg, contains(msg, "airport")
}

```

5.6. Makefile для локальной разработки

```

# Makefile
.PHONY: help validate test deploy

help
    @echo "Available targets:"
    @echo "  validate - Validate YAML syntax"

```

```

@echo "  test      - Run OPA tests"
@echo "  deploy    - Apply configurations"

validate:
  @echo "Validating YAML files..."
  @for file in $(find devices -name "*.yaml"); do \
    echo "Checking $$file"; \
    yq eval 'true' $$file > /dev/null || exit 1; \
  done

test:
  @echo "Running OPA tests..."
  @opa test policies/ -v

deploy: validate test
  @echo "Applying configurations..."
  @kubectl apply -f flux/kustomization.yaml

dev-validate:
  @for config in $(find devices -name "config.yaml"); do \
    echo "Validating $$config"; \
    opa eval --data policies/ --input $$config \
      'data.device_config.violation' --format pretty; \
  done

```

5.7. Мониторинг конфигураций

```

# prometheus/device-exporter.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: device exporter config
data:
  config.yaml: |
    devices:
      - id: drone-001
        endpoint: "http://drone-001.local:9090/metrics"

```

```
- id: drone-002
  endpoint: "http://drone-002.local:9090/metrics"
```

```
scrape_interval: 30s
metrics:
- name: device_config_applied
  type: counter
  labels: ["device", "version"]
- name: device_config_last_applied
  type: gauge
  labels: ["device"]
```

5.8. Демонстрационный сценарий

1. Клонирование репозитория

```
git clone https://gitlab.com/fleet/configs
cd configs
```

2. Создание feature-ветки

```
git checkout -b feature/update-drone-001-altitude
```

3. Изменение конфигурации

```
sed -i 's/max_altitude: 100/max_altitude: 130/' devices/drone-
001/config.yaml
```

4. Локальная проверка

```
make test
```

Ожидаемый вывод: violation: Max altitude 130 m exceeds limit of 120 m

5. Исправление (устанавливаем допустимое значение)

```
sed -i 's/max_altitude: 130/max_altitude: 110/' devices/drone-
001/config.yaml
```

6. Повторная проверка

```
make test
```

Ожидаемый вывод: PASS

7. Push и создание MR

```
git commit -am "Increase drone-001 altitude to 110m"
```

```
git push origin feature/update-drone-001-altitude
```

8. CI/CD автоматически запустит валидацию

9. После merge Flux применит конфигурацию к устройству

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (GitOps для конфигураций, ОРА валидация)
5. Схема архитектуры
6. Структура репозитория
7. Примеры конфигураций устройств (минимум 3)
8. ОРА политики (полный код с комментариями)
9. CI/CD конфигурация
10. Результаты выполнения:
11. Скриншот успешной валидации
12. Скриншот нарушения политики
13. Скриншот CI/CD пайплайна
14. Скриншот применения конфигурации
15. Заключение
16. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Структура репозитория и конфигураций	15

Критерий	Макс. баллов
Разработка ОРА политик (минимум 5)	25
Написание тестов для политик	15
CI/CD конфигурация	15
Демонстрация валидации и применения	15
Ответы на контрольные вопросы	10
Качество отчета	5
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Полная структура, 5+ политик, тесты проходят, CI/CD работает, демонстрация полного цикла
Хорошо	75-89	Политики работают, но нет тестов или неполная CI/CD
Удовлетворительно	60-74	Базовые политики есть, но не все проверяются
Неудовлетворительно	<60	Политики не работают

Штрафные баллы

Нарушение	Штраф
Менее 5 политик	-10
Нет тестов	-10
Нет CI/CD	-10
Нет демонстрации	-10

Лабораторная работа №16

ОТА-обновления как услуга платформы

1. Цель работы: Реализовать массовое обновление прошивок устройств (Over-The-Air) как платформенный сервис с использованием стратегий канареечного и A/B-обновления, с возможностью отката при ошибках, управляемый через Git-репозиторий и интегрированный с CI/CD пайплайном.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить архитектуру ОТА-обновлений для парка автономных устройств
2. Разработать сервис управления ОТА-обновлениями
3. Реализовать стратегии канареечного и A/B-обновления
4. Настроить механизм подписи и верификации обновлений (Sigstore/Cosign)
5. Реализовать автоматический откат при обнаружении ошибок
6. Создать GitOps-пайплайн для управления версиями прошивок
7. Продемонстрировать процесс обновления парка устройств

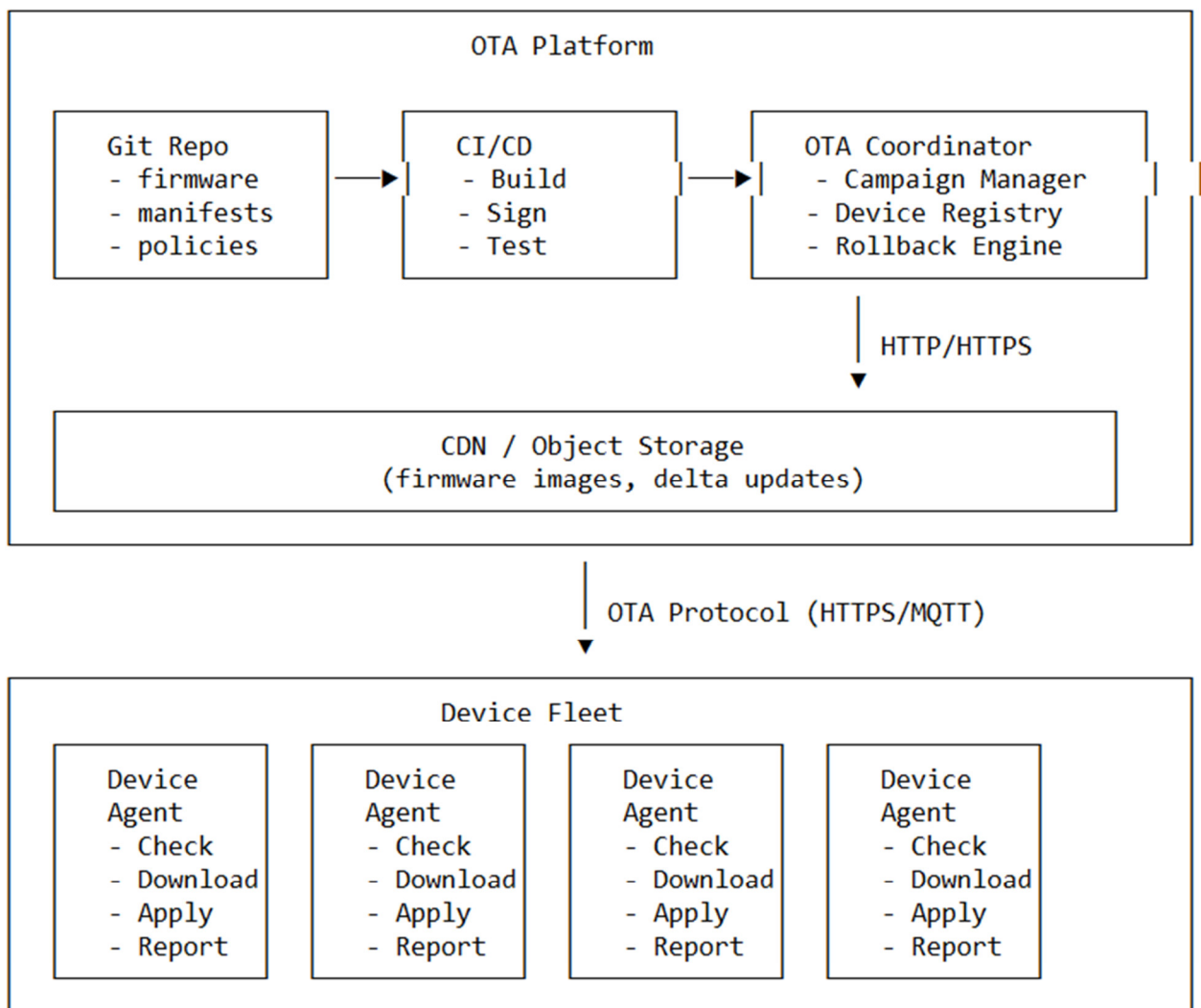
3. Краткое теоретическое обоснование

3.1. Что такое OTA-обновления?

OTA (Over-The-Air) - технология беспроводной доставки обновлений программного обеспечения на удаленные устройства. Ключевые аспекты:

Аспект	Описание
Безопасность	Обновления должны быть подписаны и зашифрованы
Надежность	Возможность отката при неудачном обновлении
Масштабируемость	Поддержка тысяч устройств одновременно
Контроль	Поэтапный rollout, A/B тестирование

3.2. Архитектура OTA-платформы



3.3. Стратегии обновления

Стратегия	Описание	Преимущества	Недостатки
Canary	Обновление малой группы (1-5%)	Быстрое обнаружение проблем	Малый охват тестирования
A/B	Разные версии на разных группах	Сравнение производительности	Сложность управления
Rolling	Постепенное обновление по	Контролируемый rollout	Длительное время

Стратегия	Описание	Преимущества	Недостатки
	группам		
Blue-Green	Полная замена парка	Быстрый откат	Двойные затраты ресурсов

3.4. Модель данных для OTA-кампании

```
# ota/campaign.yaml
```

```
apiVersion: ota.platform/v1
```

```
kind: UpdateCampaign
```

```
metadata:
```

```
  name: drone-firmware v2 0-1
```

```
  namespace: ota system
```

```
spec:
```

```
  # Целевая версия
```

```
  target_version: "2.0.1"
```

```
  target_firmware_url: "https://ota-storage/drone-firmware-2.0.1.bin"
```

```
  firmware_checksum: "sha256:abc123..."
```

```
  firmware_signature: "sig:def456..."
```

```
# Стратегия обновления
```

```
strategy:
```

```
  type: "canary"
```

```
  initial_percentage: 1
```

```
  increment: 5
```

```
  increment_interval: "10m"
```

```
  max_percentage: 100
```

```
  failure_threshold: 5    # количество ошибок для отката
```

Селектор устройств

selector:

match_labels:

device_type: "quadcopter"

fleet: "delivery"

region: "north"

Требования к устройству

requirements:

min_battery_percent: 50

min_signal_strength: -70

required_firmware_version: "1.9.x"

Health checks после обновления

health_checks:

- name: "connectivity"

timeout: 60s

expected_result: "online"

- name: "sensors"

timeout: 30s

expected_result: "all_operational"

Автоматический откат

rollback:

enabled: true

trigger: "failure_rate > 10%"

auto_rollback_delay: "5m"

3.5. OTA Agent на устройстве

```
# ota_agent.py

import hashlib
import json
import requests
from datetime import datetime

class OTAAgent:

    def __init__(self, device_id, api_endpoint):
        self.device_id = device_id
        self.api_endpoint = api_endpoint
        self.current_version = self.get_current_firmware_version()

    def check_for_updates(self):
        """Проверка наличия обновлений"""
        response = requests.get(
            f'{self.api_endpoint}/api/v1/devices/{self.device_id}/campaign'
        )
        if response.status_code == 200:
            return response.json()
        return None

    def download_firmware(self, url, checksum):
        """Загрузка прошивки с проверкой целостности"""
        response = requests.get(url, stream=True)
        firmware_data = b''

        for chunk in response.iter_content(chunk_size=8192):
            firmware_data += chunk
```

```
# Проверка контрольной суммы
actual_checksum = hashlib.sha256(firmware_data).hexdigest()
if actual_checksum != checksum:
    raise Exception(f"Checksum mismatch: expected {checksum}, got {actual_checksum}")

return firmware_data

def apply_firmware(self, firmware_data):
    """Применение прошивки на устройстве"""
    # Сохранение текущей версии для отката
    self.save_backup()

    # Запись новой прошивки
    with open("/firmware/current.bin", "wb") as f:
        f.write(firmware_data)

    # Перегрузка устройства
    self.reboot_device()

    # Ожидание и проверка здоровья
    if self.wait_for_health_check():
        self.report_status("success")
    else:
        self.rollback()
        self.report_status("failed")

def rollback(self):
    """Откат к предыдущей версии"""
    self.restore_backup()
```

```
self reboot_device()
```

```
def report_status(self status):  
    """Отправка статуса обновления на платформу"""  
    requests post(  
        f'{self api_endpoint}/api/v1/devices/{self device_id}/status',  
        json={  
            "status": status,  
            "version": self current_version,  
            "timestamp": datetime utcnow().isoformat()  
        }  
    )
```

3.6. Подпись и верификация обновлений

```
# Генерация ключей для подписи прошивок
```

```
cosign generate-key-pair
```

```
# Подпись прошивки
```

```
cosign sign-blob --key cosign.key firmware.bin --output-signature  
firmware.sig
```

```
# Верификация на устройстве
```

```
cosign verify-blob --key cosign.pub --signature firmware.sig firmware.bin
```

3.7. OTA Coordinator (Kubernetes)

```
# ota-coordinator.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
  name: ota coordinator
  namespace: ota system
spec:
  replicas: 3
  selector:
    matchLabels:
      app: ota coordinator
  template:
    metadata:
      labels:
        app: ota coordinator
    spec:
      containers:
        - name: coordinator
          image: ota-platform/coordinator v1.0.0
          ports:
            - containerPort: 8080
            - containerPort: 9090 # metrics
          env:
            - name: DATABASE_URL
              value: "postgresql://ota-db:5432/ota"
            - name: REDIS_URL
              value: "redis://ota-redis:6379"
            - name: STORAGE_ENDPOINT
              value: "https://ota-storage.example.com"
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
```

```
    initialDelaySeconds: 30
    periodSeconds: 10
  readinessProbe:
    httpGet:
      path: /ready
      port: 8080
    initialDelaySeconds: 5
    periodSeconds: 5
  resources:
    requests:
      cpu: 500m
      memory: 512Mi
    limits:
      cpu: 2000m
      memory: 2Gi
```

3.8. Мониторинг OTA-кампаний

```
# ota_metrics.prom
# Метрики OTA-обновлений
ota_updates_total{device_type, campaign, status}
ota_update_duration_seconds{device_type, campaign}
ota_devices_pending_total{campaign}
ota_devices_success_total{campaign}
ota_devices_failed_total{campaign}
ota_campaign_rollback_total{campaign}

# Burn rate алерты
alert: OTAUpdateHighFailureRate
expr: |
```


`rate(ota_updates_total{status="failed"}[5m]) /`

`rate(ota_updates_total[5m]) > 0.1`

for: 2m

annotations:

summary: "OTA update failure rate exceeded 10%"

4. Контрольные вопросы

1. Что такое OTA-обновления и почему они критичны для парка автономных устройств?
2. Какие стратегии обновления существуют? В чем их преимущества и недостатки?
3. Как обеспечить безопасность OTA-обновлений (подпись, шифрование, целостность)?
4. Как работает канареечное обновление? Как выбрать процент канареечной группы?
5. Как реализовать автоматический откат при обнаружении проблем?
6. Как OTA-платформа масштабируется на тысячи устройств?
7. Какие метрики необходимо отслеживать в процессе OTA-обновления?
8. Как интегрировать OTA-обновления с GitOps подходом?
9. Что такое delta-обновления и когда они эффективны?
10. Как обрабатывать устройства с низким зарядом батареи или плохим сигналом?
11. Как тестировать OTA-обновления в симуляционной среде перед rollout?
12. Какие протоколы передачи данных оптимальны для OTA (HTTP, MQTT, CoAP)?
13. Как обеспечить мониторинг состояния обновления на каждом устройстве?

14. Какие требования к хранилищу прошивок (репликация, CDN)?
15. Как организовать постепенный rollout с учетом географического расположения устройств?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Разработать OTA-координатор (API сервис) согласно спецификации
3. Реализовать стратегию обновления (canary или A/B)
4. Настроить подпись обновлений с помощью Cosign
5. Создать GitOps пайплайн для управления версиями
6. Настроить мониторинг OTA-кампаний в Prometheus/Grafana
7. Продемонстрировать процесс обновления парка устройств (5+ устройств)
8. Показать откат при обнаружении ошибок

5.2. Варианты заданий

Вар.	Тип устройств	Стратегия	Особенность
	Квадрокоптеры	Canary (1%, 5%, 20%, 100%)	Гезонирование
2	Сельхоз дроны	Rolling (10% каждые 5 мин)	Время полевых работ
3	Доставщики	Canary + A/B сравнение	Приоритет срочности

Вар.	Тип устройств	Стратегия	Особенность
4	Охранные дроны	Blue-Green	Смена смены
3	Промышленные роботы	Canary (цех за цехом)	Безопасность персонала
6	Морские дроны	Rolling (по регионам)	Погодные условия
7	Метеозонды	A/B тестирование	Точность измерений
8	Спасательные дроны	Canary (10%, затем 100%)	Экстренная готовность
9	Складские роботы	Rolling (зоны склада)	Пиковые часы
0	Строительные роботы	Canary (площадка)	Рабочие смены
1	Подводные аппараты	A/B (глубина)	Время погружения
2	Трубопроводные дроны	Rolling (секции)	Непрерывность инспекции
3	Энергодроны	Canary (ЛЭП)	Высоковольтные зоны
4	Реагирование на ЧС	Canary + escalation	Время отклика

Вар.	Тип устройств	Стратегия	Особенность
5	Комбинированный флот	Multi-strategy	Разные стратегии для разных типов

5.3. Структура OTA-платформы

ota-platform/

```

├── coordinator/
│   ├── api/
│   │   ├── campaigns.py
│   │   ├── devices.py
│   │   └── webhooks.py
│   ├── engine/
│   │   ├── rollout.py
│   │   ├── rollback.py
│   │   └── scheduler.py
│   ├── storage/
│   │   ├── firmware.py
│   │   └── manifests.py
│   ├── models/
│   │   ├── campaign.py
│   │   ├── device.py
│   │   └── update.py
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app.py
├── agent/
│   ├── ota_agent.py
│   ├── updater.py
│   └── health.py

```

```
|— manifests/
|   |— campaigns/
|   |   └─ drone-v2.0.1.yaml
|   └─ devices/
|       └─ drone-001.yaml
|       └─ drone-002.yaml
|— policies/
|   └─ ota.rego
|— k8s/
|   └─ coordinator.yaml
|   └─ database.yaml
|   └─ ingress.yaml
|— .gitlab-ci.yml
|— Makefile
└─ README.md
```

5.4. API OTA-координатора

```
# app.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Optional
from datetime import datetime
import json
import redis
import asyncpg

app = FastAPI(title="OTA Coordinator")

# Pydantic модели
```

```
class Campaign(BaseModel):
```

```
    name: str
```

```
    target_version: str
```

```
    target_firmware_url: str
```

```
    firmware_checksum: str
```

```
    strategy: dict
```

```
    selector: dict
```

```
    requirements: dict
```

```
class DeviceStatus(BaseModel):
```

```
    device_id: str
```

```
    status: str
```

```
    version: str
```

```
    campaign: Optional[str]
```

```
    timestamp: datetime
```

```
@app.post("/api/v1/campaigns")
```

```
async def create_campaign(campaign: Campaign):
```

```
    """Создание новой ОТА-кампании"""
```

```
    # Сохранение в базу данных
```

```
    # Инициализация rollout
```

```
    # Возврат ID кампании
```

```
    pass
```

```
@app.get("/api/v1/devices/{device_id}/campaign")
```

```
async def get_device_campaign(device_id: str):
```

```
    """Получение активной кампании для устройства"""
```

```
    # Проверка устройства
```

```
    # Определение доли обновления
```

```
    # Возврат кампании или null
```

```
pass
```

```
@app.post("/api/v1/devices/{device_id}/status")
```

```
async def report_device_status(device_id: str, status: DeviceStatus):
```

```
    """Прием статуса от устройства"""
```

```
    # Обновление метрик
```

```
    # Проверка порога ошибок
```

```
    # Автоматический откат при необходимости
```

```
pass
```

```
@app.get("/api/v1/campaigns/{campaign_name}/metrics")
```

```
async def get_campaign_metrics(campaign_name: str):
```

```
    """Получение метрик кампаний"""
```

```
    # Успешные обновления
```

```
    # Ошибки
```

```
    # Прогресс rollout
```

```
pass
```

```
@app.post("/api/v1/campaigns/{campaign_name}/rollback")
```

```
async def rollback_campaign(campaign_name: str):
```

```
    """Принудительный откат кампании"""
```

```
pass
```

```
# Rollout engine
```

```
class RolloutEngine:
```

```
    def __init__(self, redis_client):
```

```
        self.redis = redis_client
```

```
    def should_update_device(self, device_id: str, campaign: Campaign) ->
```

```
bool:
```

```
"""Определение, должно ли устройство получить обновление"""
strategy = campaign["strategy"]

if strategy["type"] == "canary":
    # Хэширование device_id для детерминированного выбора
    hash_value = hash(device_id) % 100
    return hash_value < strategy["current_percentage"]

elif strategy["type"] == "rolling":
    # Проверка группы устройства
    group = self.get_device_group(device_id)
    return group <= strategy["current_group"]

return True

def increment_rollout(self, campaign_name: str):
    """Увеличение процента/группы обновления"""
    campaign = self.get_campaign(campaign_name)
    strategy = campaign["strategy"]

    if strategy["type"] == "canary":
        new_percentage = min(
            strategy["current_percentage"] + strategy["increment"],
            strategy["max_percentage"]
        )
    elif strategy["type"] == "rolling":
        new_group = min(
            strategy["current_group"] + 1,
            strategy["total_groups"]
        )
```



```
# Обновление кампании
self.update_campaign(campaign_name,
current_percentage=new_percentage)

# Логирование
self.log_rollout_progress(campaign_name, new_percentage)
```

5.5. Симулятор устройств

```
# simulator.py
import asyncio
import aiohttp
import random
import hashlib
from datetime import datetime

class DeviceSimulator:
    def __init__(self, device_id, api_endpoint):
        self.device_id = device_id
        self.api_endpoint = api_endpoint
        self.current_version = "1.0.0"
        self.online = True

    async def run(self):
        while True:
            try:
                # Проверка обновлений
                async with aiohttp.ClientSession() as session:
                    async with session.get(
```

```

f" {self api_endpoint} /api/v1/devices/ {self device_id} /campaign"
    ) as resp:
        if resp.status == 200:
            campaign = await resp.json()
            if campaign and campaign["target_version"] !=
self.current_version:

                await self.perform_update(session, campaign)

        await asyncio.sleep(30) # Проверка каждые 30 секунд
    except Exception as e:
        print(f"Error: {e}")
        await asyncio.sleep(60)

    async def perform_update(self, session, campaign):
        print(f"[ {self device_id} ] Starting update to
{campaign['target_version']}")

        # Проверка требований
        if random.random() > 0.95: # 5% ошибок
            await self.report_status(session, "failed")
            return

        # Симуляция загрузки
        await asyncio.sleep(random.uniform(5, 15))

        # Симуляция применения обновления
        await asyncio.sleep(random.uniform(3, 10))

        # Обновление версии

```

```

self.current_version = campaign["target_version"]

# Отчет об успехе
await self.report_status(session, "success")
print(f"[{self.device_id}] Update completed")

async def report_status(self, session, status):
    await session.post(
        f"{self.api_endpoint}/api/v1/devices/{self.device_id}/status",
        json={
            "device_id": self.device_id,
            "status": status,
            "version": self.current_version,
            "timestamp": datetime.utcnow().isoformat()
        }
    )

# Запуск симуляторов
async def main():
    devices = [DeviceSimulator(f"drone-{i:03d}", "http://localhost:8080") for
i in range(1, 51)]
    await asyncio.gather(*[d.run() for d in devices])

if __name__ == "__main__":
    asyncio.run(main())

```

5.6. GitLab CI для OTA

```

# .gitlab-ci.yml
stages:

```

- build
- sign
- test
- deploy

variables:

FIRMWARE_VERSION: "v2.0.1"

build-firmware:

stage: build

image: docker:24

script:

- docker build -t firmware \${FIRMWARE_VERSION} -f

Dockerfile.firmware .

- docker save firmware \${FIRMWARE_VERSION} -o firmware.bin

artifacts:

paths:

- firmware.bin

sign-firmware:

stage: sign

image: sigstore/cosign latest

script:

- cosign sign-blob --key \${COSIGN_KEY} firmware.bin --output-

signature firmware.sig

artifacts:

paths:

- firmware.bin
- firmware.sig

needs:

- build firmware

upload-firmware:

stage: deploy

script:

- aws s3 cp firmware.bin s3 //ota

storage/firmware/\${FIRMWARE_VERSION}/firmware.bin

- aws s3 cp firmware.sig s3 //ota

storage/firmware/\${FIRMWARE_VERSION}/firmware.sig

create-campaign:

stage: deploy

script:

- |

```
cat > campaign.yaml << EOF
```

```
apiVersion: ota.platform/v1
```

```
kind: UpdateCampaign
```

```
metadata:
```

```
  name: drone-${FIRMWARE_VERSION}
```

```
spec:
```

```
  target_version: "${FIRMWARE_VERSION#v}"
```

```
  target_firmware_url: "https://ota-
```

```
storage/firmware/${FIRMWARE_VERSION}/firmware.bin"
```

```
  firmware_checksum: "$(sha256sum firmware.bin | cut -d' ' -f1)"
```

```
strategy:
```

```
  type: "canary"
```

```
  initial_percentage: 1
```

```
  increment: 5
```

```
  increment_interval: "10m"
```

```
selector:
```

```
    match_labels:
        device_type: "quadcopter"
EOF
- kubectl apply -f campaign.yaml
```

needs:

- sign firmware
- upload firmware

5.7. ОРА политики для ОТА

```
# policies/ota.rego
```

```
package ota
```

```
# Запрет обновления при низком заряде батареи
```

```
violation[msg] if {
    device := input device
    device battery_percent < input requirements min_battery_percent
    msg := sprintf("Battery too low: %d%%", [device battery_percent])
}
```

```
# Запрет обновления при плохом сигнале
```

```
violation[msg] if {
    device := input device
    device signal_strength > input requirements max_signal_strength
    msg := sprintf("Signal too weak: %d dBm", [device signal_strength])
}
```

```
# Проверка совместимости версий
```

```
violation[msg] if {
    device := input device
```

```

        not version_compatible(device current_version
input requirements required_firmware_version)

        msg := sprintf("Incompatible version: %s", [device current_version])
    }

```

```

version_compatible(current, required) if {
    # Семантическое сравнение версий
    parts_current := split(current, ".")
    parts_required := split(required, ".")
    parts_current[0] == parts_required[0] # major version must match
}

```

5.8. Мониторинг OTA (Grafana)

```

{
    "dashboard": {
        "title": "OTA Campaign Dashboard",
        "panels": [
            {
                "title": "Campaign Progress",
                "type": "gauge",
                "targets": [
                    {
                        "expr": "ota_devices_success_total / ota_devices_total"
                    }
                ]
            },
            {
                "title": "Rollout Speed",
                "type": "timeseries",

```

```

    "targets": [
      {
        "expr": "rate(ota_updates_total{status='success'}[5m])"
      }
    ]
  },
  {
    "title": "Failure Rate",
    "type": "timeseries",
    "targets": [
      {
        "expr": "rate(ota_updates_total{status='failed'}[5m]) /
rate(ota_updates_total[5m])"
      }
    ]
  },
  {
    "title": "Devices by Status",
    "type": "piechart",
    "targets": [
      {
        "expr": "sum by (status)
(ota_updates_total{status!~'started|pending'})"
      }
    ]
  },
  {
    "title": "Campaign Timeline",
    "type": "table",
    "targets": [

```



```

    {
      "expr": "ota_campaign_progress"
    }
  ]
}
]
}
}
}

```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (ОТА архитектура, стратегии)
5. Схема архитектуры ОТА-платформы
6. Код ОТА-координатора
7. Код ОТА-агента
8. Конфигурация кампаний
9. ОРА политики
10. Результаты выполнения:
11. Скриншоты создания кампании
12. Скриншоты прогресса rollout
13. Скриншоты отката при ошибках
14. Скриншоты метрик и дашбордов
15. Заключение
16. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов

Критерий	Макс. баллов
Разработка OTA-координатора	20
Реализация стратегии обновления	20
Разработка OTA-агента	15
Подпись и верификация	10
GitOps пайплайн	10
Демонстрация rollout и отката	15
Ответы на контрольные вопросы	10
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Полная OTA платформа, оба сценария (rollout/откат) работают, подпись настроена
Хорошо	75-89	OTA работает, но отсутствует подпись или откат
Удовлетворительно	60-74	Базовая реализация, только один тип стратегии
Неудовлетворительно	<60	OTA не работает

Лабораторная работа №17

Сбор телеметрии с устройств в платформу наблюдаемости

1. Цель работы: Настроить сбор телеметрии с edge-устройств (БПЛА, роботов, IoT-датчиков) в централизованную платформу наблюдаемости с использованием OpenTelemetry Collector на устройствах, обеспечить корреляцию телеметрии устройств с платформенными событиями и создать единые дашборды для мониторинга парка устройств.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить архитектуру сбора телеметрии с распределенных edge-устройств
2. Разработать легковесный OpenTelemetry Collector для устройств с ограниченными ресурсами
3. Реализовать сбор метрик (GPS, заряд батареи, статус, температура)
4. Настроить буферизацию данных при потере связи с центральным сервером
5. Реализовать отправку телеметрии в центральную платформу (Prometheus/VictoriaMetrics)
6. Создать дашборд в Grafana для мониторинга парка устройств
7. Настроить алерты на критические состояния устройств

3. Краткое теоретическое обоснование

3.1. Особенности сбора телеметрии с edge-устройств

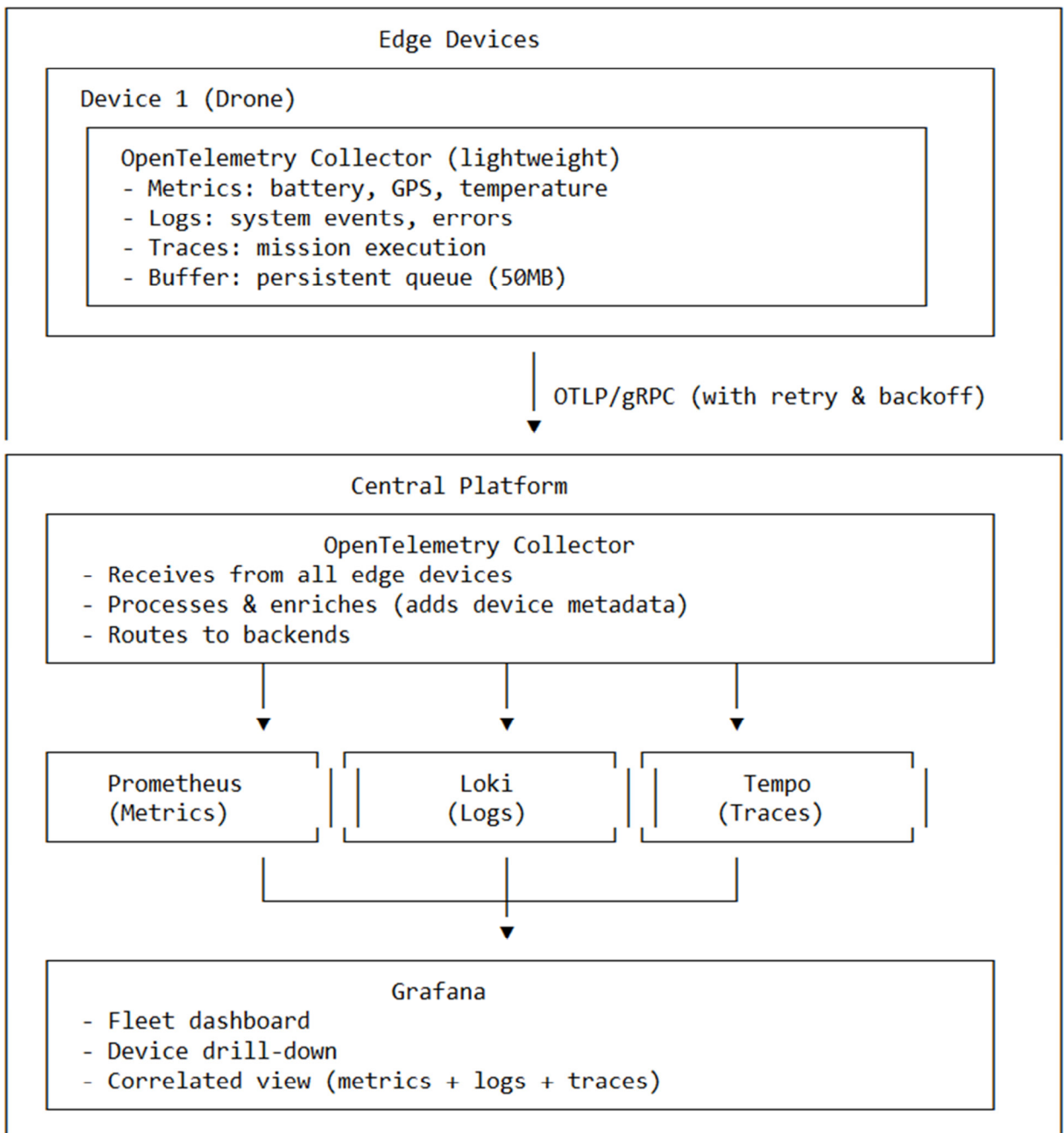
Сбор телеметрии с распределенных устройств имеет уникальные

ВЫЗОВЫ:

Вызов	Описание	Решение

Вызов	Описание	Решение
Ограниченные ресурсы	CPU, память, энергопотребление	Легковесный collector, оптимизированные форматы
Нестабильная связь	Прерывистое соединение, высокая задержка	Буферизация, retry, сжатие
Масштаб	Тысячи устройств	Federation, агрегация на edge
Безопасность	TLS, аутентификация устройств	mTLS, токены, OAuth2
Гетерогенность	Разные типы устройств, протоколы	Единый формат (OTLP), нормализация

3.2. Архитектура сбора телеметрии



3.3. Модель данных телеметрии

```
# metrics.yaml
apiVersion: otel/v1
kind: Metric
metadata:
  name: device.telemetry
labels:
```

device_id: "drone-001"

device_type: "quadcopter"

fleet: "delivery"

region: "north"

firmware: "2.0.1"

spec:

metrics:

- name: "device.battery.percent"

type: gauge

value: 78.5

unit: "%"

timestamp: "2026-05-14T10:30:00Z"

- name: "device.gps.latitude"

type: gauge

value: 55.751244

unit: "degrees"

- name: "device.gps.longitude"

type: gauge

value: 37.618423

unit: "degrees"

- name: "device.gps.altitude"

type: gauge

value: 120.5

unit: "m"

- name: "device.temperature.cpu"

type: gauge

value: 45.2

unit: "°C"

- name: "device.signal.rssi"

type: gauge

value: -67

unit: "dBm"

- name: "device.status"

type: gauge

value: 1 # 1=online, 0=offline, 2=fault

unit: "state"

- name: "device.mission.progress"

type: gauge

value: 45

unit: "%"

- name: "device.mission.distance"

type: counter

value: 12500

unit: "m"

- name: "device.errors.total"

type: counter

value: 3

unit: "errors"

3.4. Легковесный OpenTelemetry Collector для edge

```
# edge-collector-config.yaml

receivers:

  hostmetrics:
    collection_interval: 10s

  scrapers:
    cpu:
    memory:
    load:

# Кастомные метрики устройства
prometheus/device:
  config:
    scrape_configs:
      - job_name: 'device-metrics'
        static_configs:
          - targets: ['localhost:9090']

  filelog:
    include: [/var/log/device/*.log]
    start_at: end
    operators:
      - type: json_parser
        parse_to: body

processors:
  memory_limiter:
    check_interval: 1s
    limit_mib: 100 # Ограничение для edge устройств

batch:
```


timeout: 10s

send_batch_size: 1024

Добавление метаданных устройства

attributes/device:

actions:

- key: device.id

- value: "\${DEVICE_ID}"

- action: upsert

- key: device.type

- value: "\${DEVICE_TYPE}"

- action: upsert

- key: fleet

- value: "\${FLEET}"

- action: upsert

Сжатие для экономии трафика

compression/gzip:

level: best_speed

exporters:

Экспорт с буферизацией и retry

otlphttp/central:

endpoint: "https://telemetry.platform.com/v1/traces"

compression: gzip

retry_on_failure:

- enabled: true

- initial_interval: 1s

- max_interval: 30s

- max_elapsed_time: 300s

```
queue:
  enabled: true
  num_consumers: 2
  queue_size: 10000
```

```
service:
```

```
  pipelines:
```

```
    metrics:
```

```
      receivers: [hostmetrics, prometheus/device]
      processors: [memory_limiter, batch, attributes/device]
      exporters: [otlphttp/central]
```

```
    logs:
```

```
      receivers: [filelog]
      processors: [memory_limiter, batch, attributes/device]
      exporters: [otlphttp/central]
```

3.5. Центральный OpenTelemetry Collector

```
# central-collector-config.yaml
```

```
receivers:
```

```
  otlp:
```

```
    protocols:
```

```
      grpc:
```

```
        endpoint: 0.0.0.0:4317
```

```
      http:
```

```
        endpoint: 0.0.0.0:4318
```

```
processors:
```

```
  memory_limiter:
```

```
    check_interval: 1s
```

```
limit_mib: 1024

batch:
  timeout: 1s
  send_batch_size: 8192

# Обогащение данных (добавление гео-информации)
transform/geo:
  error_mode: ignore
  metric_statements:
    - context: datapoint
      statements:
        - set(attributes["country"], get_geo_country(attributes["longitude"],
attributes["latitude"]))

# Фильтрация аномалий
filter/outlier:
  metrics:
    include:
      match_type: regexp
      metric_names: ["device.*"]
    exclude:
      match_type: strict
      metric_names: ["device.battery.percent"]
  conditions:
    - metric_name == "device.temperature.cpu" && value > 85

exporters:
  prometheus:
    endpoint: "0.0.0.0:8889"
```

namespace: edge

loki:

endpoint: "http://loki:3100/loki/api/v1/push"

otlp:

endpoint: "tempo:4317"

tls:

insecure: true

service:

pipelines:

metrics:

receivers: [otlp]

processors: [memory_limiter, batch, transform/geo]

exporters: [prometheus]

logs:

receivers: [otlp]

processors: [memory_limiter, batch]

exporters: [loki]

traces:

receivers: [otlp]

processors: [memory_limiter, batch]

exporters: [otlp]

3.6. Edge Agent на Python

```
# edge_agent.py
```

```
import asyncio
```

```
import json
```

```

import random

import time

import os

import psutil

from opentelemetry import metrics, trace

from opentelemetry exporter otlp proto grpc metric_exporter import
OTLPMetricExporter

from opentelemetry exporter otlp proto grpc trace_exporter import
OTLPSpanExporter

from opentelemetry sdk metrics import MeterProvider

from opentelemetry sdk metrics export import

PeriodicExportingMetricReader

from opentelemetry sdk resources import SERVICE_NAME Resource

from opentelemetry sdk trace import TracerProvider

from opentelemetry sdk trace export import BatchSpanProcessor


class DeviceTelemetryAgent:

    def __init__(self, device_id: str, device_type: str, fleet: str):

        self.device_id = device_id

        self.device_type = device_type

        self.fleet = fleet


# Настройка ресурса
resource = Resource(attributes={
    SERVICE_NAME: "device-agent",
    "device.id": device_id,
    "device.type": device_type,
    "fleet": fleet,
    "service.version": "1.0.0"
})

```

Настройка метрик с буферизацией

```
self metric_reader = PeriodicExportingMetricReader(  
    OTLPMetricExporter(  
        endpoint="https://telemetry.platform.com:4317",  
        insecure=False,  
        timeout=30,  
        retry_max=3  
    ),  
    export_interval_millis=15000, # Каждые 15 секунд  
    export_timeout_millis=10000  
)
```

```
self meter_provider = MeterProvider(  
    resource=resource,  
    metric_readers=[self metric_reader]  
)  
metrics.set_meter_provider(self meter_provider)  
self meter = metrics.get_meter(__name__)
```

Создание метрик

```
self battery_gauge = self meter.create_gauge(  
    "device.battery.percent",  
    description="Battery charge level",  
    unit="%"  
)  
  
self gps_latitude = self meter.create_gauge(  
    "device.gps.latitude",  
    description="GPS latitude",  
    unit="degrees"
```

```
)  
self gps_longitude = self meter create_gauge(  
    "device.gps.longitude",  
    description="GPS longitude",  
    unit="degrees"  
)  
self gps_altitude = self meter create_gauge(  
    "device.gps.altitude",  
    description="GPS altitude",  
    unit="m"  
)  
self temp_gauge = self meter create_gauge(  
    "device.temperature.cpu",  
    description="CPU temperature",  
    unit="°C"  
)  
self signal_gauge = self meter create_gauge(  
    "device.signal.rssi",  
    description="Signal strength",  
    unit="dBm"  
)  
self mission_progress = self meter create_gauge(  
    "device.mission.progress",  
    description="Current mission progress",  
    unit="%"  
)  
self error_counter = self meter create_counter(  
    "device.errors.total",  
    description="Total errors count",  
    unit="errors"
```

```
)
```

```
# Настройка трассировки
```

```
tracer_provider = TracerProvider(resource=resource)
```

```
tracer_provider.add_span_processor(
```

```
    BatchSpanProcessor(OTLPSpanExporter(
```

```
        endpoint="https://telemetry.platform.com:4317",
```

```
        insecure=False
```

```
    ))
```

```
)
```

```
trace.set_tracer_provider(tracer_provider)
```

```
self.tracer = trace.get_tracer(__name__)
```

```
# Локальный буфер для офлайн-режима
```

```
self.offline_buffer = []
```

```
self.is_online = True
```

```
async def collect_metrics(self):
```

```
    """Сбор метрик устройства"""
```

```
    # Имитация сбора реальных данных
```

```
    self.battery_gauge.set(random.uniform(50, 100))
```

```
    self.gps_latitude.set(random.uniform(55.7, 55.8))
```

```
    self.gps_longitude.set(random.uniform(37.5, 37.7))
```

```
    self.gps_altitude.set(random.uniform(80, 150))
```

```
    self.temp_gauge.set(random.uniform(35, 65))
```

```
    self.signal_gauge.set(random.uniform(-85, -45))
```

```
    self.mission_progress.set(random.uniform(0, 100))
```

```
# Проверка аномалий
```

```
if random.random() < 0.05: # 5% ошибок
```



```
self error_counter add(1, {"type": "sensor_error"})
await self log_error("Sensor reading anomaly")
```

```
async def log_error(self, message: str):
    """Логирование ошибок"""
    # Использование OpenTelemetry logging
    logger = self meter_provider get_logger(__name__)
    logger emit({
        "severity": "ERROR",
        "body": message,
        "attributes": {
            "device.id": self device_id
            "error.type": "sensor"
        }
    })
```

```
async def execute_mission(self, waypoints: list):
    """Трассировка выполнения миссии"""
    with self tracer start_as_current_span("mission.execute") as span:
        span set_attribute("waypoints.count", len(waypoints))
        span set_attribute("device.id", self device_id)

    for i, wp in enumerate(waypoints):
        with self tracer start_as_current_span(f"waypoint.{i}") as
wp_span:
            wp_span set_attribute("waypoint.lat", wp["lat"])
            wp_span set_attribute("waypoint.lon", wp["lon"])
            wp_span set_attribute("waypoint.alt", wp["alt"])

    # Симуляция полета к точке
```

```

        await asyncio.sleep(1)

        # Обновление прогресса
        self.mission_progress.set((i + 1) / len(waypoints) * 100)

    async def health_check(self):
        """Health check соединения с платформой"""
        return self.is_online

    async def run(self):
        """Основной цикл агента"""
        while True:
            try:
                await self.collect_metrics()

            if not await self.health_check():
                # Офлайн режим - буферизация
                self.is_online = False
                self.offline_buffer.append({
                    "timestamp": time.time(),
                    "metrics": await self.get_current_metrics()
                })
            else:
                self.is_online = True
                # Отправка буферизованных данных
                if self.offline_buffer:
                    for data in self.offline_buffer:
                        await self.send_metrics(data)
                    self.offline_buffer.clear()

```

```

        await asyncio.sleep(10)
    except Exception as e:
        print(f'Error in telemetry collection: {e}')
        await asyncio.sleep(30)

```

Dockerfile для edge агента

```
DOCKERFILE = """
```

```
FROM python:3.11-slim
```

```
WORKDIR /app
```

```

RUN pip install opentelemetry-api opentelemetry-sdk \
    opentelemetry-exporter-otlp psutil

```

```
COPY edge_agent.py .
```

```
CMD ["python", "edge_agent.py"]
```

```
"""
```

3.7. Дашборд состояния парка устройств (Grafana)

```

{
  "dashboard": {
    "title": "Device Fleet Telemetry",
    "panels": [
      {
        "title": "Fleet Health Overview",
        "type": "stat",
        "targets": [
          {

```

```
    "expr": "count(device_status {status='online'})",
    "legendFormat": "Online"
  },
  {
    "expr": "count(device_status {status='offline'})",
    "legendFormat": "Offline"
  },
  {
    "expr": "count(device_status {status='fault'})",
    "legendFormat": "Fault"
  }
]
},
{
  "title": "Battery Distribution",
  "type": "histogram",
  "targets": [
    {
      "expr": "device_battery_percent",
      "legendFormat": "battery"
    }
  ]
},
{
  "title": "Device Map",
  "type": "geomap",
  "targets": [
    {
      "expr": "device_gps_latitude and device_gps_longitude",
      "legendFormat": "{{device_id}}",
```

```
    "map_type": "coordinates"
  }
]
},
{
  "title": "Temperature by Device",
  "type": "timeseries",
  "targets": [
    {
      "expr": "device_temperature_cpu",
      "legendFormat": "{{device_id}}"
    }
  ]
},
{
  "title": "Mission Progress",
  "type": "gauge",
  "targets": [
    {
      "expr": "device_mission_progress"
    }
  ]
},
{
  "title": "Error Rate",
  "type": "timeseries",
  "targets": [
    {
      "expr": "rate(device_errors_total[5m])",
      "legendFormat": "{{device_id}}"
    }
  ]
}
```

```

    }
  ]
},
{
  "title": "Last Telemetry Received",
  "type": "table",
  "targets": [
    {
      "expr": "timestamp(device_last_seen)",
      "format": "table"
    }
  ]
}
]
}
]
}
}

```

3.8. Алерты для парка устройств

```

# prometheus-alerts-device.yaml
groups:
- name: device_alerts
  interval: 1m
  rules:
  - alert: DeviceOffline
    expr: device_last_seen < time() - 300
    for: 5m
    labels:
      severity: warning
      type: connectivity

```

annotations:

summary: "Device {{ \$labels.device_id }} is offline"

description: "No telemetry received for 5 minutes"

- alert: LowBattery

expr: device_battery_percent < 20

for: 1m

labels:

severity: warning

type: battery

annotations:

summary: "Device {{ \$labels.device_id }} battery low"

description: "Battery at {{ \$value }}%"

- alert: CriticalBattery

expr: device_battery_percent < 10

for: 30s

labels:

severity: critical

type: battery

annotations:

summary: "Device {{ \$labels.device_id }} critical battery"

description: "Device will land soon"

- alert: HighTemperature

expr: device_temperature_cpu > 70

for: 2m

labels:

severity: warning

type: hardware

annotations:

summary: "High temperature on {{ \$labels.device_id }}"

description: "Temperature at {{ \$value }}°C"

- alert: MissionStalled

expr: device_mission_progress < 10 and device_mission_progress > 0

for: 10m

labels:

severity: warning

type: mission

annotations:

summary: "Mission stalled on {{ \$labels.device_id }}"

description: "Progress hasn't increased in 10 minutes"

- alert: HighErrorRate

expr: rate(device_errors_total[5m]) > 1

for: 5m

labels:

severity: critical

type: errors

annotations:

summary: "High error rate on {{ \$labels.device_id }}"

description: "Error rate {{ \$value }} errors/min"

- alert: SignalWeak

expr: device_signal_rssi < -80

for: 1m

labels:

severity: warning

type: connectivity

annotations:

summary: "Weak signal on {{ \$labels.device_id }}"

description: "RSSI {{ \$value }} dBm"

)

type DeviceTelemetry struct {

DeviceID string

Battery float64

Latitude float64

Longitude float64

Altitude float64

Temperature float64

SignalRSSI float64

MissionProgress float64

Errors int64

Timestamp time.Time

}

type EdgeAgent struct {

deviceID string

deviceType string

fleet string

buffer []DeviceTelemetry

metricMeter metric.Meter

metrics *DeviceMetrics

offlineMode bool

}

type DeviceMetrics struct {

```

Battery    metric Float64ObservableGauge
Latitude   metric Float64ObservableGauge
Longitude   metric Float64ObservableGauge
Altitude    metric Float64ObservableGauge
Temperature metric Float64ObservableGauge
Signal      metric Float64ObservableGauge
MissionProgress metric Float64ObservableGauge
Errors      metric Int64Counter
}

```

```

func NewEdgeAgent(deviceID string, deviceType string, fleet string) (*EdgeAgent,
error) {
    // Настройка ресурса
    res, err := resource.New(context.Background(),
        resource.WithAttributes(
            semconv.ServiceNameKey.String("device-agent"),
            attribute.String("device.id", deviceID),
            attribute.String("device.type", deviceType),
            attribute.String("fleet", fleet),
        ),
    )
    if err != nil {
        return nil, err
    }

    // Настройка OTLP экспортера
    metricExporter, err := otelmetricgrpc.New(context.Background(),
        otelmetricgrpc.WithEndpoint("telemetry.platform.com:4317"),
        otelmetricgrpc.WithInsecure(),
    )
}

```

```
if err != nil {
    return nil, err
}

// Настройка MeterProvider
reader := periodic.New(
    metricExporter,
    periodic.WithInterval(15*time.Second),
)
meterProvider := sdkmetric.NewMeterProvider(
    sdkmetric.WithResource(res),
    sdkmetric.WithReader(reader),
)
otel.SetMeterProvider(meterProvider)

// Настройка TraceProvider
traceExporter, err := oteltracegrpc.New(context.Background(),
    oteltracegrpc.WithEndpoint("telemetry.platform.com:4317"),
    oteltracegrpc.WithInsecure(),
)
if err != nil {
    return nil, err
}
traceProvider := sdktrace.NewTracerProvider(
    sdktrace.WithBatcher(traceExporter),
    sdktrace.WithResource(res),
)
otel.SetTracerProvider(traceProvider)

meter := meterProvider.Meter("device.metrics")
```

```

agent := &EdgeAgent {
    deviceID: deviceID,
    deviceType: deviceType,
    fleet: fleet
    buffer: []DeviceTelemetry {},
}

```

// Создание observable метрик

```

metrics := &DeviceMetrics {}

```

```

metrics.Battery, err = meter.Float64ObservableGauge(
    "device.battery.percent",
    metric.WithDescription("Battery charge level"),
    metric.WithUnit("%"),
)

```

```

if err != nil {
    return nil, err
}

```

// Добавление callback для обновления метрик

```

_, err = meter.RegisterCallback(
    func(ctx context.Context, o metric.Observer) error {
        data := agent.collectTelemetry()
        o.ObserveFloat64(metrics.Battery, data.Battery)
        o.ObserveFloat64(metrics.Latitude, data.Latitude)
        o.ObserveFloat64(metrics.Longitude, data.Longitude)
        o.ObserveFloat64(metrics.Altitude, data.Altitude)
        o.ObserveFloat64(metrics.Temperature, data.Temperature)
        o.ObserveFloat64(metrics.Signal, data.SignalRSSI)
    },
)

```

```

        o ObserveFloat64(metrics.MissionProgress, data.MissionProgress)
    }
    return nil
},
metrics.Battery,
metrics.Latitude,
metrics.Longitude,
metrics.Altitude,
metrics.Temperature,
metrics.Signal,
metrics.MissionProgress,
)
if err != nil {
    return nil, err
}

agent.metrics = metrics
agent.metricMeter = meter

return agent, nil
}

```

```

func (a *EdgeAgent) collectTelemetry() DeviceTelemetry {
    // Имитация сбора телеметрии с реальных датчиков
    return DeviceTelemetry{
        DeviceID:    a.deviceID,
        Battery:     50 + rand.Float64()*50,
        Latitude:    55.75 + rand.Float64()*0.1,
        Longitude:   37.60 + rand.Float64()*0.1,
        Altitude:    80 + rand.Float64()*70,
        Temperature: 35 + rand.Float64()*30,
    }
}

```

```

SignalRSSI: -85 + rand.Float64()*40,
MissionProgress rand.Float64() * 100,
Errors: 0,
Timestamp time.Now(),
}
}

```

```

func (a *EdgeAgent) Run(ctx context.Context) {
    tracer := otel.Tracer("device-agent")

    ticker := time.NewTicker(10 * time.Second)
    defer ticker.Stop()

    for {
        select {
        case <-ctx.Done():
            return
        case <-ticker.C:
            a.sendTelemetry(ctx, tracer)
        }
    }
}

```

```

func (a *EdgeAgent) sendTelemetry(ctx context.Context, tracer
trace.Tracer) {
    _, span := tracer.Start(ctx, "telemetry.collect")
    defer span.End()

    span.SetAttributes(
        attribute.String("device.id", a.deviceID),

```

```

        attribute.String("fleet", a fleet),
    )

    // Логика отправки с буферизацией
    // ...
}

func main() {
    agent, err := NewEdgeAgent("drone-001", "quadcopter", "delivery")
    if err != nil {
        log.Fatal(err)
    }

    ctx := context.Background()
    agent.Run(ctx)
}

```

3.9. Настройка сжатия и оптимизация трафика (продолжение)

```

# edge-collector-config-full.yaml
receivers:
  hostmetrics:
    collection_interval: 10s
  scrapers:
    cpu:
    memory:
    disk:
    network:

# Кастомные метрики устройства через HTTP endpoint
prometheus/device:
  config:
    scrape_configs:
      - job_name: 'device-metrics'
        scrape_interval: 5s
        static_configs:
          - targets: ['localhost:9090']

# Сбор логов устройства
filelog:
  include: [/var/log/device/*.log]

```

```

include_file_name: false
include_file_path: true
start_at: end
operators:
  - type: json_parser
    parse_to: body
  - type: add
    field: attributes.log_type
    value: device_log

# OTLP receiver для трассировки миссий
otlp:
  protocols:
    grpc:
      endpoint: 0.0.0.0:4317
    http:
      endpoint: 0.0.0.0:4318

processors:
  # Ограничение памяти для edge (критично для устройств с 512MB RAM)
  memory_limiter:
    check_interval: 1s
    limit_mib: 96          # Максимум 96MB для collector
    spike_limit_mib: 16
    check_interval: 2s

  # Пакетная обработка для снижения числа запросов
  batch:
    timeout: 10s
    send_batch_size: 512
    send_batch_max_size: 1024

  # Добавление метаданных устройства
  attributes/device:
    actions:
      - key: device.id
        value: "${DEVICE_ID}"
        action: upsert
      - key: device.type
        value: "${DEVICE_TYPE}"
        action: upsert
      - key: device.fleet
        value: "${FLEET_NAME}"
        action: upsert
      - key: device.region
        value: "${REGION}"
        action: upsert
      - key: device.firmware
        value: "${FIRMWARE_VERSION}"
        action: upsert

  # Сжатие данных для экономии трафика (до 90% экономии)
  compression/gzip:
    level: best_speed      # best_compression / best_speed / default

  # Фильтрация несущественных метрик

```



```
filter/low_priority:
  metrics:
    exclude:
      match_type: regexp
      metric_names:
        - "host.disk.*"
        - "host.network.*"
  logs:
    exclude:
      match_type: regexp
      expressions:
        - 'body matches "DEBUG.*"'

# Дедупликация (при потере связи и повторной отправке)
dedup:
  dedup_key: trace_id

exporters:
  # Основной экспортер с retry и буферизацией
  otlphttp/primary:
    endpoint: "https://telemetry.platform.com/v1/traces"
    compression: gzip
    retry_on_failure:
      enabled: true
      initial_interval: 1s
      max_interval: 30s
      max_elapsed_time: 300s
    queue:
      enabled: true
      num_consumers: 2
      queue_size: 5000      # Буфер на 5000 событий
    headers:
      Authorization: "Bearer ${API_TOKEN}"

  # Fallback экспортер (при недоступности основного)
  otlphttp/backup:
    endpoint: "https://backup-telemetry.platform.com/v1/traces"
    compression: gzip
    retry_on_failure:
      enabled: true
      max_interval: 60s

  # Локальное логирование для отладки
  logging:
    logLevel: warn
    sampling_initial: 5
    sampling_thereafter: 200

service:
  telemetry:
    metrics:
      address: 0.0.0.0:8888
      level: basic

  pipelines:
    metrics:
```

```

        receivers: [hostmetrics, prometheus/device]
        processors: [memory_limiter, batch, attributes/device,
filter/low_priority]
        exporters: [otlphttp/primary, logging]

logs:
    receivers: [filelog]
    processors: [memory_limiter, batch, attributes/device,
filter/low_priority]
    exporters: [otlphttp/primary, logging]

traces:
    receivers: [otlp]
    processors: [memory_limiter, batch, attributes/device, dedup]
    exporters: [otlphttp/primary, otlphttp/backup, logging]

```

3.10. Управление буфером при потере связи

```

# edge_agent_with_buffer.py
import sqlite3
import json
import threading
import time
from datetime import datetime
from typing import List, Dict, Any
import logging

class PersistentTelemetryBuffer:
    """
    Персистентный буфер телеметрии с использованием SQLite
    для сохранения данных при потере связи и перезагрузке устройства
    """

    def __init__(self, db_path: str = "/var/lib/telemetry/buffer.db"):
        self.db_path = db_path
        self.conn = None
        self._init_db()
        self.logger = logging.getLogger(__name__)

    def _init_db(self):
        """Инициализация SQLite базы данных для буфера"""
        self.conn = sqlite3.connect(self.db_path,
check_same_thread=False)
        cursor = self.conn.cursor()

        cursor.execute("""
        CREATE TABLE IF NOT EXISTS telemetry_buffer (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            device_id TEXT NOT NULL,
            data_type TEXT NOT NULL, -- metrics, logs, traces
            payload TEXT NOT NULL,
            timestamp REAL NOT NULL,
            retry_count INTEGER DEFAULT 0,

```

```

        created_at REAL NOT NULL
    )
    """

    cursor.execute("""
        CREATE INDEX IF NOT EXISTS idx_timestamp
        ON telemetry_buffer(timestamp)
    """)

    cursor.execute("""
        CREATE INDEX IF NOT EXISTS idx_retry
        ON telemetry_buffer(retry_count)
    """)

    self.conn.commit()
    self.logger.info(f"Telemetry buffer initialized at {db_path}")

    def add(self, device_id: str, data_type: str, payload: Dict[str,
Any]):
        """Добавление данных в буфер"""
        try:
            cursor = self.conn.cursor()
            cursor.execute("""
                INSERT INTO telemetry_buffer
                (device_id, data_type, payload, timestamp, created_at)
                VALUES (?, ?, ?, ?, ?)
            """, (
                device_id,
                data_type,
                json.dumps(payload),
                time.time(),
                time.time()
            ))
            self.conn.commit()
            self.logger.debug(f"Added {data_type} to buffer for device
{device_id}")
        except Exception as e:
            self.logger.error(f"Failed to add to buffer: {e}")

    def get_batch(self, limit: int = 100) -> List[Dict[str, Any]]:
        """Получение batch данных для отправки (FIFO)"""
        cursor = self.conn.cursor()
        cursor.execute("""
            SELECT id, device_id, data_type, payload, timestamp,
retry_count

            FROM telemetry_buffer
            ORDER BY created_at ASC
            LIMIT ?
        """, (limit,))

        rows = cursor.fetchall()
        return [
            {
                "id": row[0],
                "device_id": row[1],
                "data_type": row[2],
            }
        ]

```

```

        "payload": json.loads(row[3]),
        "timestamp": row[4],
        "retry_count": row[5]
    }
    for row in rows
]

def mark_sent(self, ids: List[int]):
    """Удаление успешно отправленных данных"""
    if not ids:
        return

    placeholders = ",".join("?" * len(ids))
    cursor = self.conn.cursor()
    cursor.execute(f"""
        DELETE FROM telemetry_buffer WHERE id IN ({placeholders})
    """, ids)
    self.conn.commit()
    self.logger.info(f"Cleared {len(ids)} items from buffer")

def mark_retry(self, ids: List[int]):
    """Увеличение счетчика retry для неудачных отправок"""
    cursor = self.conn.cursor()
    for id in ids:
        cursor.execute("""
            UPDATE telemetry_buffer
            SET retry_count = retry_count + 1
            WHERE id = ?
        """, (id,))
    self.conn.commit()

def get_stats(self) -> Dict[str, Any]:
    """Получение статистики буфера"""
    cursor = self.conn.cursor()

    cursor.execute("SELECT COUNT(*) FROM telemetry_buffer")
    total = cursor.fetchone()[0]

    cursor.execute("""
        SELECT data_type, COUNT(*)
        FROM telemetry_buffer
        GROUP BY data_type
    """)
    by_type = dict(cursor.fetchall())

    cursor.execute("SELECT AVG(retry_count) FROM telemetry_buffer")
    avg_retry = cursor.fetchone()[0] or 0

    return {
        "total_items": total,
        "by_type": by_type,
        "avg_retry_count": avg_retry,
        "estimated_size_mb": total * 2048 / (1024 * 1024) # ~2KB
    }

```

per item

```

def cleanup_old(self, max_age_hours: int = 168):
    """Очистка старых данных (по умолчанию 7 дней)"""
    cutoff_time = time.time() - (max_age_hours * 3600)
    cursor = self.conn.cursor()
    cursor.execute("""
        DELETE FROM telemetry_buffer
        WHERE created_at < ? AND retry_count > 10
    """, (cutoff_time,))
    deleted = cursor.rowcount
    self.conn.commit()

    if deleted > 0:
        self.logger.warning(f"Cleaned up {deleted} stale items from
buffer")

    return deleted

def close(self):
    """Закрытие соединения с БД"""
    if self.conn:
        self.conn.close()

class EnhancedTelemetrySender:
    """Отправка телеметрии с буферизацией и экспоненциальным backoff"""

    def __init__(self, endpoint: str, device_id: str, api_token: str):
        self.endpoint = endpoint
        self.device_id = device_id
        self.api_token = api_token
        self.buffer = PersistentTelemetryBuffer()
        self.session = None
        self.running = True
        self.logger = logging.getLogger(__name__)
        self.send_thread = threading.Thread(target=self._send_loop,
daemon=True)
        self.send_thread.start()

    async def _get_session(self):
        if self.session is None or self.session.closed:
            import aiohttp
            self.session = aiohttp.ClientSession(
                headers={
                    "Authorization": f"Bearer {self.api_token}",
                    "Content-Type": "application/json"
                },
                timeout=aiohttp.ClientTimeout(total=30)
            )
        return self.session

    async def send(self, data_type: str, payload: Dict[str, Any]):
        """Отправка данных (с буферизацией при ошибке)"""
        # Добавляем метаданные
        payload["device_id"] = self.device_id
        payload["timestamp"] = datetime.utcnow().isoformat()

```

```

try:
    session = await self._get_session()
    async with session.post(
        f"{self.endpoint}/{data_type}",
        json=payload
    ) as response:
        if response.status == 200:
            self.logger.debug(f"Sent {data_type} successfully")
            return True
        else:
            self.logger.warning(f"Failed to send {data_type}:
HTTP {response.status}")
            self._buffer_for_retry(data_type, payload)
            return False
except Exception as e:
    self.logger.error(f"Send error: {e}")
    self._buffer_for_retry(data_type, payload)
    return False

def _buffer_for_retry(self, data_type: str, payload: Dict[str,
Any]):
    """Сохранение данных в буфер для повторной отправки"""
    self.buffer.add(self.device_id, data_type, payload)

    async def flush_buffer(self):
        """Отправка всех накопленных данных из буфера"""
        batch = self.buffer.get_batch(limit=50)
        if not batch:
            return

        self.logger.info(f"Flushing {len(batch)} items from buffer")

        session = await self._get_session()
        sent_ids = []
        retry_ids = []

        for item in batch:
            try:
                async with session.post(
                    f"{self.endpoint}/{item['data_type']}",
                    json=item['payload']
                ) as response:
                    if response.status == 200:
                        sent_ids.append(item['id'])
                    else:
                        retry_ids.append(item['id'])
            except Exception:
                retry_ids.append(item['id'])

        if sent_ids:
            self.buffer.mark_sent(sent_ids)

        if retry_ids:
            self.buffer.mark_retry(retry_ids)
            self.logger.warning(f"Retry scheduled for {len(retry_ids)}
items")

```

```

async def _send_loop(self):
    """Фоновый цикл отправки буфера"""
    while self.running:
        try:
            await self.flush_buffer()
            await asyncio.sleep(30) # Каждые 30 секунд
        except Exception as e:
            self.logger.error(f"Send loop error: {e}")
            await asyncio.sleep(60)

async def get_buffer_stats(self) -> Dict[str, Any]:
    """Получение статистики буфера"""
    return self.buffer.get_stats()

async def close(self):
    """Корректное завершение работы"""
    self.running = False
    await self.flush_buffer() # Финальная отправка
    if self.session:
        await self.session.close()
    self.buffer.close()

```

3.11. Трассировка миссий БПЛА

```

# mission_tracing.py
from opentelemetry import trace
from opentelemetry.trace import Status, StatusCode
from contextlib import contextmanager
import time

class MissionTracer:
    """Трассировка выполнения полетных миссий"""

    def __init__(self):
        self.tracer = trace.get_tracer(__name__)

    @contextmanager
    def mission_span(self, mission_id: str, device_id: str,
                    waypoints_count: int):
        """Создание span для всей миссии"""
        with self.tracer.start_as_current_span("mission.execute") as span:
            span.set_attributes({
                "mission.id": mission_id,
                "device.id": device_id,
                "mission.waypoints_count": waypoints_count,
                "mission.start_time": time.time()
            })

            yield span

            span.set_attribute("mission.end_time", time.time())

```

```

        duration = span.attributes["mission.end_time"] -
span.attributes["mission.start_time"]
        span.set_attribute("mission.duration_seconds", duration)

    @contextmanager
    def waypoint_span(self, waypoint_index: int, waypoint: dict):
        """Создание span для отдельной точки маршрута"""
        with
self.tracer.start_as_current_span(f"mission.waypoint.{waypoint_index}") as
span:
            span.set_attributes({
                "waypoint.index": waypoint_index,
                "waypoint.latitude": waypoint.get("lat"),
                "waypoint.longitude": waypoint.get("lon"),
                "waypoint.altitude": waypoint.get("alt", 100),
                "waypoint.action": waypoint.get("action", "fly")
            })

            yield span

    @contextmanager
    def telemetry_span(self, device_id: str, metric_name: str):
        """Трассировка сбора и отправки телеметрии"""
        with self.tracer.start_as_current_span("telemetry.collect") as
span:
            span.set_attributes({
                "device.id": device_id,
                "telemetry.metric": metric_name
            })

            yield span

    def record_error(self, span, error: Exception, context: str):
        """Запись ошибки в span"""
        span.set_status(Status(StatusCode.ERROR, str(error)))
        span.set_attribute("error.context", context)
        span.set_attribute("error.type", error.__class__.__name__)
        span.record_exception(error)

    def record_event(self, span, event_name: str, attributes: dict =
None):
        """Запись события в span"""
        span.add_event(event_name, attributes)

# Пример использования при выполнении миссии
async def execute_mission_with_tracing(
    device_id: str,
    mission_id: str,
    waypoints: list,
    mission_tracer: MissionTracer
):
    """Выполнение миссии с полной трассировкой"""

    with mission_tracer.mission_span(mission_id, device_id,
len(waypoints)) as mission_span:

```



```

for i, waypoint in enumerate(waypoints):
    with mission_tracer.waypoint_span(i, waypoint) as wp_span:
        try:
            # Выполнение полета к точке
            await fly_to_waypoint(device_id, waypoint)

            # Запись события достижения точки
            mission_tracer.record_event(
                wp_span,
                "waypoint.reached",
                {"remaining_waypoints": len(waypoints) - i - 1}
            )

        except Exception as e:
            mission_tracer.record_error(wp_span, e,
"waypoint_execution")
            raise

mission_tracer.record_event(mission_span, "mission.completed")

```

3.12. Дашборд состояния парка устройств (Grafana) - полная версия

```

{
  "dashboard": {
    "title": "Device Fleet Telemetry Dashboard",
    "tags": ["device", "fleet", "telemetry"],
    "timezone": "browser",
    "refresh": "30s",
    "panels": [
      {
        "id": 1,
        "title": "Fleet Health Status",
        "type": "stat",
        "gridPos": {"h": 4, "w": 4, "x": 0, "y": 0},
        "targets": [
          {
            "expr": "count(device_status{status='online'})",
            "legendFormat": "Online",
            "refId": "A"
          },
          {
            "expr": "count(device_status{status='offline'})",
            "legendFormat": "Offline",
            "refId": "B"
          },
          {
            "expr": "count(device_status{status='fault'})",
            "legendFormat": "Fault",
            "refId": "C"
          }
        ]
      },
      {
        "fieldConfig": {

```

```

        "defaults": {
            "unit": "none",
            "thresholds": {
                "steps": [
                    {"color": "green", "value": null}
                ]
            }
        },
    },
    {
        "id": 2,
        "title": "Battery Distribution",
        "type": "histogram",
        "gridPos": {"h": 6, "w": 8, "x": 4, "y": 0},
        "targets": [
            {
                "expr": "device_battery_percent",
                "legendFormat": "battery",
                "refId": "A"
            }
        ],
        "options": {
            "bucketSize": 10,
            "buckets": [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
        }
    },
    {
        "id": 3,
        "title": "Device Map (Live Tracking)",
        "type": "geomap",
        "gridPos": {"h": 12, "w": 12, "x": 12, "y": 0},
        "targets": [
            {
                "expr": "device_gps_latitude and device_gps_longitude",
                "legendFormat": "{{device_id}}",
                "refId": "A"
            }
        ],
        "options": {
            "view": {"lat": 55.76, "lon": 37.62, "zoom": 10},
            "basemap": {"name": "OpenStreetMap"},
            "layers": [
                {
                    "name": "Devices",
                    "type": "markers",
                    "tooltip": "{{device_id}}: Battery {{battery}}%",
                    "color": {
                        "field": "battery",
                        "scheme": "RdYlGn",
                        "range": [0, 100]
                    }
                }
            ]
        }
    }
],
},

```

```

{
  "id": 4,
  "title": "Temperature by Device",
  "type": "timeseries",
  "gridPos": {"h": 6, "w": 8, "x": 0, "y": 6},
  "targets": [
    {
      "expr": "device_temperature_cpu",
      "legendFormat": "{{device_id}}",
      "refId": "A"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "celsius",
      "thresholds": {
        "steps": [
          {"color": "green", "value": 0},
          {"color": "orange", "value": 65},
          {"color": "red", "value": 75}
        ]
      }
    }
  }
},
{
  "id": 5,
  "title": "Mission Progress",
  "type": "gauge",
  "gridPos": {"h": 6, "w": 4, "x": 8, "y": 6},
  "targets": [
    {
      "expr": "avg(device_mission_progress)",
      "legendFormat": "avg",
      "refId": "A"
    }
  ],
  "options": {
    "showThresholdLabels": true,
    "showThresholdMarkers": true
  },
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
      "min": 0,
      "max": 100,
      "thresholds": {
        "steps": [
          {"color": "green", "value": 0},
          {"color": "yellow", "value": 80}
        ]
      }
    }
  }
},
{

```

```

    "id": 6,
    "title": "Signal Strength",
    "type": "timeseries",
    "gridPos": {"h": 6, "w": 8, "x": 0, "y": 12},
    "targets": [
      {
        "expr": "device_signal_rssi",
        "legendFormat": "{{device_id}}",
        "refId": "A"
      }
    ],
    "fieldConfig": {
      "defaults": {
        "unit": "dBm",
        "thresholds": {
          "steps": [
            {"color": "green", "value": -50},
            {"color": "yellow", "value": -70},
            {"color": "red", "value": -80}
          ]
        }
      }
    }
  },
  {
    "id": 7,
    "title": "Error Rate by Device",
    "type": "timeseries",
    "gridPos": {"h": 6, "w": 8, "x": 8, "y": 12},
    "targets": [
      {
        "expr": "rate(device_errors_total[5m])",
        "legendFormat": "{{device_id}}",
        "refId": "A"
      }
    ],
    "fieldConfig": {
      "defaults": {
        "unit": "cpm",
        "thresholds": {
          "steps": [
            {"color": "green", "value": 0},
            {"color": "yellow", "value": 0.5},
            {"color": "red", "value": 2}
          ]
        }
      }
    }
  },
  {
    "id": 8,
    "title": "Last Telemetry Received",
    "type": "table",
    "gridPos": {"h": 8, "w": 12, "x": 0, "y": 18},
    "targets": [

```

```

        "expr": "timestamp(device_last_seen)",
        "format": "table",
        "refId": "A"
    }
  ],
  "transformations": [
    {
      "id": "sortBy",
      "options": {"fields": {"Value": {"desc": false}}}
    }
  ]
},
{
  "id": 9,
  "title": "Buffer Status (Edge)",
  "type": "stat",
  "gridPos": {"h": 4, "w": 4, "x": 8, "y": 0},
  "targets": [
    {
      "expr": "device_buffer_size",
      "legendFormat": "buffer",
      "refId": "A"
    }
  ]
},
{
  "id": 10,
  "title": "Trace Analysis - Mission Duration",
  "type": "timeseries",
  "gridPos": {"h": 6, "w": 12, "x": 12, "y": 12},
  "targets": [
    {
      "expr": "histogram_quantile(0.5,
sum(rate(mission_duration_bucket[5m])) by (1e))",
      "legendFormat": "P50",
      "refId": "A"
    },
    {
      "expr": "histogram_quantile(0.95,
sum(rate(mission_duration_bucket[5m])) by (1e))",
      "legendFormat": "P95",
      "refId": "B"
    },
    {
      "expr": "histogram_quantile(0.99,
sum(rate(mission_duration_bucket[5m])) by (1e))",
      "legendFormat": "P99",
      "refId": "C"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "s"
    }
  }
}
}

```

```

    ],
    "time": {
      "from": "now-1h",
      "to": "now"
    }
  }
}

```

3.13. Алерты для парка устройств (полная версия)

```

# prometheus-alerts-device-full.yaml
groups:
- name: device_connectivity
  interval: 30s
  rules:
  - alert: DeviceOffline
    expr: device_last_seen < time() - 300
    for: 5m
    labels:
      severity: warning
      type: connectivity
      component: device
    annotations:
      summary: "Device {{ $labels.device_id }} is offline"
      description: "No telemetry received for 5 minutes. Last seen:
        {{ $value | humanizeTimestamp }}"
      runbook: "Check device connectivity, network status, and
        power supply"

  - alert: DeviceOfflineCritical
    expr: device_last_seen < time() - 900
    for: 10m
    labels:
      severity: critical
      type: connectivity
    annotations:
      summary: "Device {{ $labels.device_id }} critical offline"
      description: "No telemetry received for 15 minutes"
      page: "true"

- name: device_battery
  interval: 1m
  rules:
  - alert: LowBattery
    expr: device_battery_percent < 20
    for: 1m
    labels:
      severity: warning
      type: battery
    annotations:
      summary: "Device {{ $labels.device_id }} battery low"
      description: "Battery at {{ $value }}%. Return to base
        recommended"

```

```

- alert: CriticalBattery
  expr: device_battery_percent < 10
  for: 30s
  labels:
    severity: critical
    type: battery
  annotations:
    summary: "Device {{ $labels.device_id }} critical battery"
    description: "Battery at {{ $value }}%. Emergency landing
initiated"
    page: "true"

- alert: BatteryDrainRateHigh
  expr: deriv(device_battery_percent[5m]) < -5
  for: 5m
  labels:
    severity: warning
    type: battery
  annotations:
    summary: "High battery drain on {{ $labels.device_id }}"
    description: "Battery draining at {{ $value }}%/min"

- name: device_temperature
  interval: 30s
  rules:
    - alert: HighTemperature
      expr: device_temperature_cpu > 70
      for: 2m
      labels:
        severity: warning
        type: hardware
      annotations:
        summary: "High temperature on {{ $labels.device_id }}"
        description: "CPU temperature at {{ $value }}°C. Expected
damage above 80°C"

    - alert: CriticalTemperature
      expr: device_temperature_cpu > 80
      for: 30s
      labels:
        severity: critical
        type: hardware
      annotations:
        summary: "Critical temperature on {{ $labels.device_id }}"
        description: "Temperature {{ $value }}°C. Immediate action
required"
        page: "true"

- name: device_mission
  interval: 1m
  rules:
    - alert: MissionStalled
      expr: device_mission_progress < 10 and device_mission_progress
> 0
      for: 10m

```

```

labels:
  severity: warning
  type: mission
annotations:
  summary: "Mission stalled on {{ $labels.device_id }}"
  description: "Progress hasn't increased in 10 minutes.
Current progress: {{ $value }}%"

- alert: MissionTimeout
  expr: mission_duration_seconds > 3600
  for: 0m
  labels:
    severity: warning
    type: mission
  annotations:
    summary: "Mission timeout on {{ $labels.device_id }}"
    description: "Mission exceeded 1 hour duration"

- name: device_errors
  interval: 1m
  rules:
    - alert: HighErrorRate
      expr: rate(device_errors_total[5m]) > 1
      for: 5m
      labels:
        severity: critical
        type: errors
      annotations:
        summary: "High error rate on {{ $labels.device_id }}"
        description: "Error rate {{ $value }} errors/min"
        page: "true"

    - alert: SensorError
      expr: increase(device_errors_total{error_type="sensor"}[5m]) >

5
      for: 0m
      labels:
        severity: warning
        type: errors
      annotations:
        summary: "Sensor errors on {{ $labels.device_id }}"
        description: "Multiple sensor failures detected"

- name: device_signal
  interval: 30s
  rules:
    - alert: WeakSignal
      expr: device_signal_rssi < -80
      for: 1m
      labels:
        severity: warning
        type: connectivity
      annotations:
        summary: "Weak signal on {{ $labels.device_id }}"
        description: "RSSI {{ $value }} dBm. Risk of disconnection"

```



```

- alert: SignalFluctuation
  expr: stddev_over_time(device_signal_rssi[5m]) > 15
  for: 5m
  labels:
    severity: warning
    type: connectivity
  annotations:
    summary: "Signal fluctuation on {{ $labels.device_id }}"
    description: "Unstable connection detected"

- name: device_buffer
  interval: 1m
  rules:
    - alert: BufferOverflow
      expr: device_buffer_size > 4500
      for: 5m
      labels:
        severity: warning
        type: buffer
      annotations:
        summary: "Telemetry buffer near capacity on {{
100 $labels.device_id }}"
        description: "Buffer at {{ $value }} items. Data loss risk"

    - alert: BufferStalled
      expr: device_buffer_drain_rate == 0 and device_buffer_size >

      for: 10m
      labels:
        severity: critical
        type: buffer
      annotations:
        summary: "Buffer not draining on {{ $labels.device_id }}"
        description: "Telemetry not reaching platform"
        page: "true"

- name: device_fleet_aggregate
  interval: 1m
  rules:
    - alert: FleetHealthDegraded
      expr: (count(device_status{status="online"}) /
count(device_status)) < 0.8
      for: 10m
      labels:
        severity: critical
        type: fleet
      annotations:
        summary: "Fleet health degraded"
        description: "Less than 80% of devices online. Online: {{
$value }}%"
        page: "true"

    - alert: HighAverageTemperature
      expr: avg(device_temperature_cpu) > 60
      for: 10m
      labels:

```

```

severity: warning
type: fleet
annotations:
  summary: "High average temperature across fleet"
  description: "Average temperature {{ $value }}°C"

```

3.14. Метрики производительности edge agent

```

# edge_agent_metrics.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: edge agent metrics
data:
  metrics.txt: |
    # HELP edge_agent_buffer_size Current size of telemetry buffer
    # TYPE edge_agent_buffer_size gauge
    edge_agent_buffer_size{device_id="drone-001"} 0

    # HELP edge_agent_buffer_drain_rate Rate of buffer drainage
    (items/sec)
    # TYPE edge_agent_buffer_drain_rate gauge
    edge_agent_buffer_drain_rate{device_id="drone-001"} 3.4

    # HELP edge_agent_send_success_total Total successful telemetry
    sends
    # TYPE edge_agent_send_success_total counter
    edge_agent_send_success_total{device_id="drone-
    001",data_type="metrics"} 12345

    # HELP edge_agent_send_errors_total Total failed telemetry sends
    # TYPE edge_agent_send_errors_total counter
    edge_agent_send_errors_total{device_id="drone-
    001",error_type="timeout"} 12

    # HELP edge_agent_memory_usage_bytes Memory usage of the agent
    # TYPE edge_agent_memory_usage_bytes gauge
    edge_agent_memory_usage_bytes{device_id="drone-001"} 45800000

    # HELP edge_agent_cpu_usage_percent CPU usage of the agent
    # TYPE edge_agent_cpu_usage_percent gauge
    edge_agent_cpu_usage_percent{device_id="drone-001"} 2.5

    # HELP edge_agent_uptime_seconds Agent uptime
    # TYPE edge_agent_uptime_seconds counter
    edge_agent_uptime_seconds{device_id="drone-001"} 86400

```

3.15. Демонстрационный сценарий

```

# demo_telemetry.sh - Полный сценарий демонстрации

#!/bin/bash

```

```

echo "=== Device Telemetry Platform Demo ==="
echo

# 1. Запуск симуляторов устройств
echo "1. Starting device simulators..."
for i in {1..5}; do
    docker run -d \
        --name device-simulator-$i \
        -e DEVICE_ID="drone-00$i" \
        -e DEVICE_TYPE="quadcopter" \
        -e FLEET="demo-fleet" \
        -e OTLP_ENDPOINT="http://central-collector:4317" \
        device-simulator:latest
done
echo "    Started 5 device simulators"
sleep 5

# 2. Проверка получения телеметрии
echo
echo "2. Checking telemetry reception..."
curl -s
"http://prometheus:9090/api/v1/query?query=device_battery_percent" | jq
'.data.result[].metric.device_id'
echo

# 3. Симуляция ошибки (отключение сети)
echo
echo "3. Simulating network outage on drone-003..."
docker exec device-simulator-3 ./simulate_outage.sh
sleep 10

# 4. Проверка буферизации
echo "    Checking buffer size for drone-003..."
curl -s "http://device-simulator-3:8888/metrics" | grep
"edge_agent_buffer_size"

# 5. Восстановление сети
echo
echo "4. Restoring connectivity..."
docker exec device-simulator-3 ./restore_network.sh
sleep 30

# 6. Проверка отправки буфера
echo "    Buffer should be drained..."
curl -s "http://device-simulator-3:8888/metrics" | grep
"edge_agent_buffer_size"

# 7. Открытие дашборда в браузере
echo
echo "5. Opening Grafana dashboard..."
echo "    URL: http://localhost:3000/d/device-telemetry"
echo "    Credentials: admin/admin"
echo
echo "=== Demo complete ==="

```

6. Требования к оформлению отчета (дополнение к ЛР17)

6.1. Структура отчета (дополнительные разделы)

1. Анализ эффективности сбора телеметрии:
2. Объем данных до/после сжатия
3. Процент успешных доставок
4. Статистика буферизации при потере связи
5. Latency сбора и доставки телеметрии
6. Оптимизации для edge устройств:
7. Memory footprint analysis
8. CPU usage profile
9. Battery consumption impact
10. Network bandwidth savings

6.2. Дополнительные критерии оценивания

Критерий	Макс. баллов
Эффективность буферизации при потере связи	10
Сжатие данных и экономия трафика	5
Трассировка миссий (span correlation)	5
Производительность edge agent	5

Итого (ЛР17): $100 + 25 = 125$ баллов (с бонусом за полную реализацию)

6.3. Checklist выполнения

1. Edge agent развернут и собирает метрики
2. Данные отправляются в центральный collector
3. Буферизация работает при потере связи
4. После восстановления связи буфер отправлен

5. В Grafana отображаются все устройства
6. Геолокационная карта работает
7. Алерты срабатывают при нарушениях
8. Трассировка миссий видна в Jaeger/Tempo

Лабораторная работа №18

Интеграция платформы с внешними API (карты, аналитика)

1. Цель работы: Обеспечить безопасное и надежное взаимодействие платформы управления парком устройств с внешними API (картографические сервисы, системы аналитики, облачные хранилища) через единый платформенный шлюз с реализацией rate limiting, кэширования, mTLS и политик доступа через OPA.

Формируемые компетенции: ПК-2

2. Задачи работы:

1. Изучить паттерны интеграции платформы с внешними API
2. Развернуть API Gateway (Kong/Nginx/Envoy) для маршрутизации запросов к внешним сервисам
3. Реализовать rate limiting для защиты от превышения квот внешних API
4. Настроить кэширование ответов внешних API для снижения нагрузки и задержек
5. Реализовать mTLS для безопасного взаимодействия с внешними сервисами
6. Интегрировать OPA для централизованного управления политиками доступа

7. Создать сервис-адаптер для конкретного внешнего API (карты или аналитика)

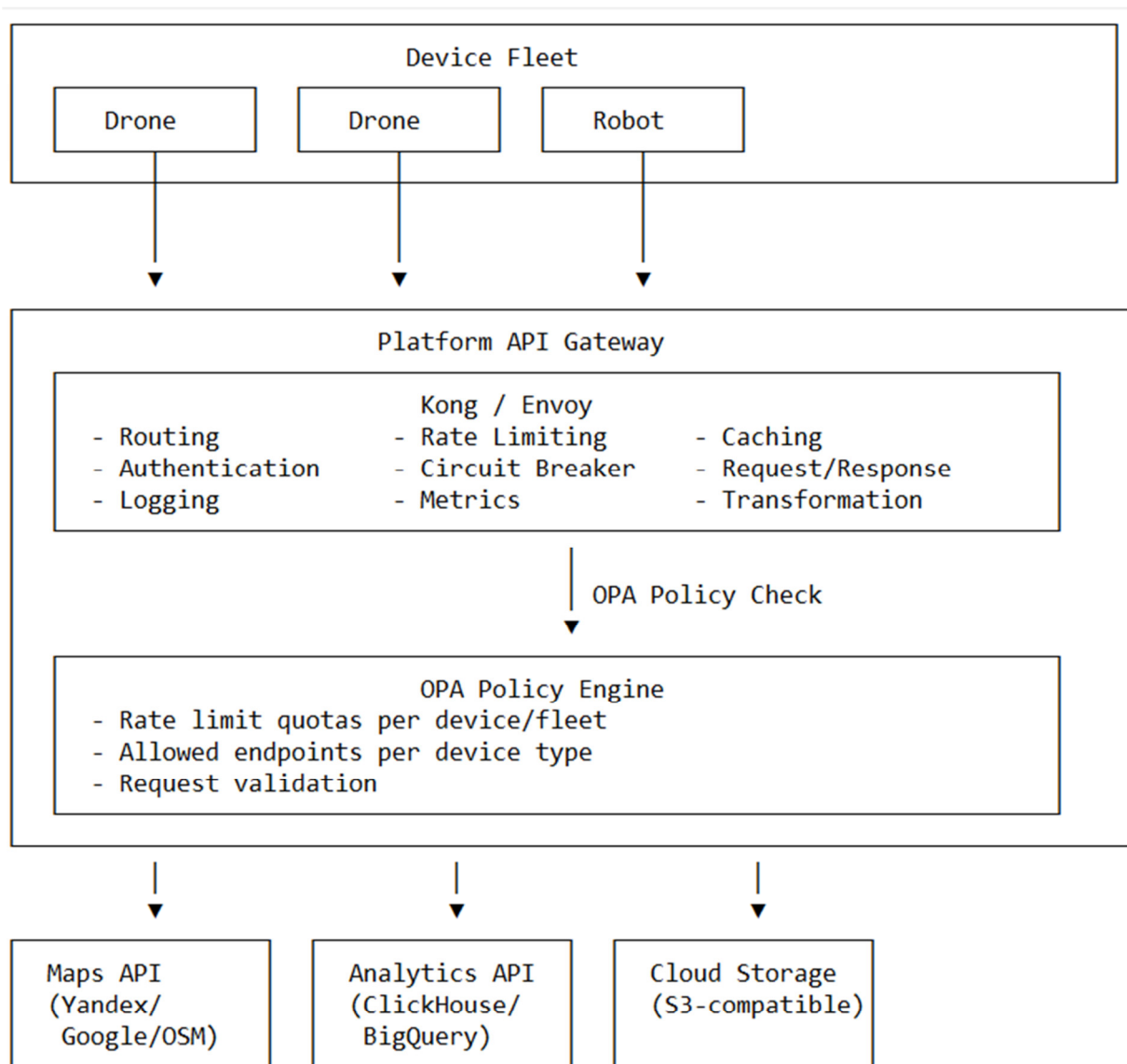
3. Краткое теоретическое обоснование

3.1. Проблемы интеграции с внешними API

При интеграции платформы управления устройствами с внешними сервисами возникают следующие вызовы:

Проблема	Описание	Решение
Квоты и лимиты	Внешние API имеют ограничения на количество запросов	Rate limiting, кэширование
Аутентификация	Различные методы (API key, OAuth2, mTLS)	Единый шлюз с адаптерами
Надежность	Внешние API могут быть недоступны	Circuit breaker, retry, fallback
Латентность	Запросы к внешним API могут быть медленными	Кэширование, async
Безопасность	Защита ключей и данных в транзите	mTLS, шифрование
Мониторинг	Отслеживание использования внешних API	Метрики, логи, трассировка

3.2. Архитектура платформенного шлюза



3.3. API Gateway: Kong vs Envoy vs Nginx

Характеристики	Kong	Envoy	Nginx
Плагины	Lua/Python/Go	Wasm/C++	Lua/C
Rate limiting	Встроенный	Через внешний сервис	Через модуль
OPA интеграция	Плагин	External auth filter	Требуется разработка

Характеристика	Kong	Envoy	Nginx
Производительность	Высокая	Очень высокая	Высокая
Сложность	Средняя	Высокая	Низкая

Для данной ЛР рекомендуется Kong из-за богатой экосистемы плагинов и хорошей документации.

3.4. Развертывание Kong в Kubernetes

```
# kong-values.yaml
image:
  repository: kong/kong gateway
  tag: "3.4"

env:
  database: "off" # In-memory mode для простоты
  prefix: /kong_prefix/
  log_level: info

proxy:
  enabled: true
  type: LoadBalancer
  http:
    enabled: true
    servicePort: 8000
  tls:
    enabled: true
    servicePort: 8443

admin:
  enabled: true
  type: ClusterIP
  http:
    enabled: true
    servicePort: 8001

plugins:
  - name: kong plugin opa
    enabled: true
  - name: kong plugin rate limiting
  - name: kong plugin proxy cache
  - name: kong plugin request transformer
  - name: kong plugin response transformer
```


- name: kong plugin statsd

```
ingressController:  
  enabled: true  
  installCRDs: true
```

3.5. Конфигурация Kong (через declarative config)

```
# kong-config.yaml  
_format_version: "3.0"  
_transform: true  
  
services:  
  # Сервис для картографического API  
  - name: maps service  
    url: https://api.maps.yandex.net/v1  
    connect_timeout: 5000  
    write_timeout: 10000  
    read_timeout: 10000  
    retries: 3  
    plugins:  
      - name: rate limiting  
        config:  
          minute: 100  
          hour: 5000  
          policy: local  
          limit_by: consumer  
      - name: proxy cache  
        config:  
          response_code: [200, 302]  
          request_method: ["GET"]  
          content_type: ["application/json"]  
          cache_ttl: 300  
          memory:  
            dictionary_name: kong_cache  
      - name: opa  
        config:  
          opa_url: "http://opa-service:8181/v1/data/kong/allow"  
          policy_enforcement: "all"  
      - name: statsd  
        config:  
          host: "prometheus-statsd-exporter:9125"  
          metrics:  
            - name: request_count  
              service: maps service  
  
  # Сервис для аналитики  
  - name: analytics service  
    url: https://api.analytics.example.com/v1  
    plugins:  
      - name: rate limiting  
        config:  
          minute: 50
```

```

        hour: 1000
      - name: request transformer
        config:
          add:
            headers:
              - "X-API-Key: ${ANALYTICS_API_KEY}"
      - name: opa
        config:
          opa_url: "http://opa-service:8181/v1/data/kong/allow"

# Сервис для облачного хранилища
- name: storage service
  url: https://storage.yandexcloud.net
  plugins:
    - name: request transformer
      config:
        add:
          headers:
            - "Authorization: AWS ${AWS_SIGNATURE}"

routes:
  # Маршруты для карт
  - name: geocode route
    service: maps service
    paths:
      - /api/v1/maps/geocode
    methods: ["GET"]
    strip_path: true
    plugins:
      - name: request transformer
        config:
          add:
            querystring:
              - "apikey:${YANDEX_MAPS_API_KEY}"

  - name: route route
    service: maps service
    paths:
      - /api/v1/maps/route
    methods: ["GET"]
    strip_path: true

  # Маршруты для аналитики
  - name: telemetry route
    service: analytics service
    paths:
      - /api/v1/telemetry
    methods: ["POST"]
    strip_path: true
    plugins:
      - name: response transformer
        config:
          add:
            headers:
              - "X-Request-ID:$request_id"

```

```

# Маршруты для хранилища
- name: upload route
  service: storage service
  paths:
    - /api/v1/storage/upload
  methods: ["PUT", "POST"]
  strip_path: true

consumers:
- username: drone-001
  custom_id: "DR-001-ABCD"
  plugins:
    - name: key auth
      config:
        key: "drone-key-001"

- username: drone-002
  custom_id: "DR-001-ABCD"
  plugins:
    - name: key auth
      config:
        key: "drone-key-002"

- username: platform service
  custom_id: "PLATFORM-API"
  plugins:
    - name: key auth
      config:
        key: "platform-key"

# Плагин аутентификации по умолчанию
plugins:
- name: key auth
  config:
    hide_credentials: true
    anonymous: null

```

3.6. OPA политики для API Gateway

```

# policies/kong.rego
package kong

import rego v1

# Проверка доступа к API
default allow := false

# Разрешение для авторизованных потребителей с правильными разрешениями
allow if {
  consumer := input.auth.consumer
  consumer != null
  route := input.attributes.request.route
  has_permission(consumer, route)
}

```

```

}

# Проверка rate limit для конкретного устройства
allow if {
  consumer := input auth.consumer
  route := input attributes.request.route
  rate_limit_exceeded(consumer, route) == false
}

# Проверка квоты запросов для устройства
rate_limit_exceeded(consumer, route) if {
  request_count := get_request_count(consumer, route,
time.now().minute)
  max_requests := get_max_requests_per_minute(consumer)
  request_count > max_requests
}

# Получение лимита запросов для устройства
get_max_requests_per_minute(consumer) := limit if {
  limits := {
    "drone-001": 60,
    "drone-002": 60,
    "drone-003": 60,
    "platform-service": 1000
  }
  limit := limits[consumer.username]
} else := 30 # default limit

# Проверка разрешений
has_permission(consumer, route) if {
  permissions := {
    "drone-001": ["geocode-route", "route-route", "telemetry-
route"],
    "drone-002": ["geocode-route", "route-route", "telemetry-
route"],
    "drone-003": ["geocode-route", "telemetry-route"],
    "platform-service": ["geocode-route", "route-route",
"telemetry-route", "upload-route"]
  }
  allowed := permissions[consumer.username]
  route name in allowed
}

# Запрет доступа к определенным зонам (geofencing)
allow if {
  route := input attributes.request.route
  route name == "route-route"
  origin := input attributes.request.query.origin
  destination := input attributes.request.query.destination
  allowed_zone(origin) and allowed_zone(destination)
}

allowed_zone(coord) if {
  # Проверка, что координаты находятся в разрешенной зоне
  lat := split(coord, ",")[0]
  lon := split(coord, ",")[1]
}

```

```

        number(lat) >= 55.70
        number(lat) <= 55.80
        number(lon) >= 37.50
        number(lon) <= 37.70
    }

    # Уведомление о нарушении
    violation[msg] if {
        not allow
        consumer := input auth consumer
        route := input attributes request.route
        msg := sprintf("Access denied for consumer %s to route %s",
[consumer.username, route.name])
    }
}

```

3.7. Адаптер для картографического сервиса

```

# service-adapters/maps_adapter.py
import asyncio
import aiohttp
import hashlib
import json
from typing import Optional, Dict, Any
from cachetools import TTLCache
from opentelemetry import trace
from opentelemetry.instrumentation.aiohttp_client import
AioHttpClientInstrumentor

tracer = trace.get_tracer(__name__)

class MapsServiceAdapter:
    """Адаптер для картографического сервиса с кэшированием и
    fallback"""

    def __init__(self, api_key: str, base_url: str =
"https://api.maps.yandex.net/v1"):
        self.api_key = api_key
        self.base_url = base_url
        # Кэш с TTL 5 минут, максимум 1000 записей
        self.cache = TTLCache(maxsize=1000, ttl=300)
        self.session = None
        AioHttpClientInstrumentor().instrument()

    async def _get_session(self) -> aiohttp.ClientSession:
        if self.session is None or self.session.closed:
            self.session = aiohttp.ClientSession(
                headers={"User-Agent": "Platform-API/1.0"},
                timeout=aiohttp.ClientTimeout(total=10)
            )
        return self.session

    async def geocode(self, address: str) -> Optional[Dict[str, Any]]:
        """Геокодирование адреса в координаты"""
        cache_key = f"geocode:{address}"

```

```

# Проверка кэша
if cache_key in self.cache:
    return self.cache[cache_key]

with tracer.start_as_current_span("maps.geocode") as span:
    span.set_attribute("address", address)

    try:
        session = await self._get_session()
        async with session.get(
            f"{self.base_url}/geocode",
            params={
                "apikey": self.api_key,
                "geocode": address,
                "format": "json"
            }
        ) as response:
            if response.status == 200:
                data = await response.json()
                self.cache[cache_key] = data
                span.set_attribute("success", True)
                return data
            else:
                span.set_attribute("error", f"HTTP
{response.status}")
                return await self._fallback_geocode(address)

    except asyncio.TimeoutError:
        span.set_attribute("error", "timeout")
        return await self._fallback_geocode(address)
    except Exception as e:
        span.set_attribute("error", str(e))
        return None

async def route(
    self,
    origin: str,
    destination: str,
    mode: str = "driving"
) -> Optional[Dict[str, Any]]:
    """Построение маршрута между точками"""
    cache_key = f"route:{origin}:{destination}:{mode}"

    if cache_key in self.cache:
        return self.cache[cache_key]

    with tracer.start_as_current_span("maps.route") as span:
        span.set_attributes({
            "origin": origin,
            "destination": destination,
            "mode": mode
        })

        try:
            session = await self._get_session()

```

```

        async with session.get(
            f"{self.base_url}/route",
            params={
                "apikey": self.api_key,
                "origin": origin,
                "destination": destination,
                "mode": mode
            }
        ) as response:
            if response.status == 200:
                data = await response.json()
                self.cache[cache_key] = data
                return data
            else:
                return None
    except Exception as e:
        span.set_attribute("error", str(e))
        return None

    async def _fallback_geocode(self, address: str) -> Dict[str, Any]:
        """Fallback при недоступности основного API"""
        # Использование OSM Nominatim как fallback
        try:
            session = await self._get_session()
            async with session.get(
                "https://nominatim.openstreetmap.org/search",
                params={
                    "q": address,
                    "format": "json",
                    "limit": 1
                }
            ) as response:
                if response.status == 200:
                    data = await response.json()
                    if data:
                        return {
                            "response": {
                                "GeoObjectCollection": {
                                    "featureMember": [{
                                        "GeoObject": {
                                            "Point": {
                                                "pos":
f"{data[0]['lon']} {data[0]['lat']}"
                                                }
                                            }
                                        }
                                    ]
                                }
                            }
                        }
                else:
                    return None
            except Exception:
                pass

        # Возврат заглушки
        return {
            "response": {
                "GeoObjectCollection": {

```

```

        "featureMember": []
    }
}

# Factory для создания адаптера
def get_maps_adapter() -> MapsServiceAdapter:
    import os
    return MapsServiceAdapter(
        api_key=os.environ.get("YANDEX_MAPS_API_KEY", ""),
        base_url=os.environ.get("MAPS_API_URL",
"https://api.maps.yandex.net/v1")
    )

```

3.8. Адаптер для аналитики

```

# service-adapters/analytics_adapter.py
import asyncio
import json
from typing import List, Dict, Any
from datetime import datetime
from opentelemetry import trace

tracer = trace.get_tracer(__name__)

class AnalyticsAdapter:
    """Адаптер для отправки телеметрии в аналитическую систему"""

    def __init__(self, api_url: str, api_key: str):
        self.api_url = api_url
        self.api_key = api_key
        self.batch_buffer = []
        self.batch_size = 100
        self.flush_interval = 10 # секунд
        self._flush_task = None

    async def start(self):
        """Запуск фоновой задачи для batch отправки"""
        self._flush_task = asyncio.create_task(self._periodic_flush())

    async def stop(self):
        """Остановка фоновой задачи"""
        if self._flush_task:
            await self.flush()
            self._flush_task.cancel()

    async def send_telemetry(self, device_id: str, metrics: Dict[str,
Any]):
        """Отправка телеметрии устройства"""
        with tracer.start_as_current_span("analytics.send") as span:
            span.set_attribute("device_id", device_id)

            event = {
                "device_id": device_id,

```



```

        "timestamp": datetime.utcnow().isoformat(),
        "metrics": metrics
    }

    self.batch_buffer.append(event)

    if len(self.batch_buffer) >= self.batch_size:
        await self.flush()

    async def flush(self):
        """Принудительная отправка буфера"""
        if not self.batch_buffer:
            return

        events = self.batch_buffer.copy()
        self.batch_buffer.clear()

        try:
            async with aiohttp.ClientSession() as session:
                async with session.post(
                    f"{self.api_url}/events",
                    headers={
                        "Authorization": f"Bearer {self.api_key}",
                        "Content-Type": "application/json"
                    },
                    json={"events": events}
                ) as response:
                    if response.status != 200:
                        # Возврат в буфер при ошибке
                        self.batch_buffer.extend(events)
        except Exception as e:
            self.batch_buffer.extend(events)

    async def query_analytics(
        self,
        device_id: str,
        start_time: datetime,
        end_time: datetime,
        metrics: List[str]
    ) -> List[Dict]:
        """Запрос аналитических данных по устройству"""
        with tracer.start_as_current_span("analytics.query") as span:
            span.set_attributes({
                "device_id": device_id,
                "start_time": start_time.isoformat(),
                "end_time": end_time.isoformat()
            })

        try:
            async with aiohttp.ClientSession() as session:
                async with session.post(
                    f"{self.api_url}/query",
                    headers={"Authorization": f"Bearer {self.api_key}"},
                    json={
                        "device_id": device_id,

```

```

        "start_time": start_time.isoformat(),
        "end_time": end_time.isoformat(),
        "metrics": metrics
    }
) as response:
    if response.status == 200:
        return await response.json()
    return []
except Exception:
    return []

async def _periodic_flush(self):
    """Периодическая отправка буфера"""
    while True:
        await asyncio.sleep(self.flush_interval)
        await self.flush()

```

3.9. Платформенный API (FastAPI)

```

# platform_api.py
from fastapi import FastAPI, HTTPException, Depends, Request
from fastapi.security import APIKeyHeader
import aiohttp
import os
from opentelemetry import trace
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor

app = FastAPI(title="Platform Integration API")
tracer = trace.get_tracer(__name__)

# Аутентификация
api_key_header = APIKeyHeader(name="X-API-Key")

# Сервисные адаптеры (singleton)
maps_adapter = None
analytics_adapter = None

@app.on_event("startup")
async def startup():
    global maps_adapter, analytics_adapter
    from service_adapters.maps_adapter import get_maps_adapter
    from service_adapters.analytics_adapter import AnalyticsAdapter

    maps_adapter = get_maps_adapter()
    analytics_adapter = AnalyticsAdapter(
        api_url=os.environ.get("ANALYTICS_API_URL",
            "http://analytics:8080"),
        api_key=os.environ.get("ANALYTICS_API_KEY", "")
    )
    await analytics_adapter.start()

@app.on_event("shutdown")
async def shutdown():

```

```

        if analytics_adapter:
            await analytics_adapter.stop()

@app.get("/api/v1/maps/geocode")
async def geocode(
    address: str,
    request: Request,
    api_key: str = Depends(api_key_header)
):
    """Геокодирование адреса"""
    with tracer.start_as_current_span("api.geocode") as span:
        span.set_attribute("address", address)

        # Проверка через OPA (вызывается в Kong)
        # Здесь только бизнес-логика

        result = await maps_adapter.geocode(address)

        if result is None:
            raise HTTPException(status_code=502, detail="Maps service
unavailable")

        return result

@app.get("/api/v1/maps/route")
async def get_route(
    origin: str,
    destination: str,
    mode: str = "driving",
    api_key: str = Depends(api_key_header)
):
    """Построение маршрута"""
    result = await maps_adapter.route(origin, destination, mode)

    if result is None:
        raise HTTPException(status_code=502, detail="Maps service
unavailable")

    return result

@app.post("/api/v1/telemetry")
async def send_telemetry(
    device_id: str,
    metrics: dict,
    api_key: str = Depends(api_key_header)
):
    """Отправка телеметрии устройства"""
    await analytics_adapter.send_telemetry(device_id, metrics)
    return {"status": "accepted"}

@app.get("/api/v1/analytics/{device_id}")
async def get_device_analytics(
    device_id: str,
    start_time: str,
    end_time: str,
    api_key: str = Depends(api_key_header)

```

```

):
    """Получение аналитики устройства"""
    from datetime import datetime
    start = datetime.fromisoformat(start_time)
    end = datetime.fromisoformat(end_time)

    result = await analytics_adapter.query_analytics(
        device_id, start, end, ["battery", "distance", "errors"]
    )

    return {"data": result}

```

```
FastAPIInstrumentor.instrument_app(app)
```

3.10. Метрики и мониторинг

```

# prometheus-metrics.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: platform metrics
data:
  metrics-config.yaml: |
    metrics:
      - name: platform_api_requests_total
        type: counter
        labels: ["service", "endpoint", "status"]

      - name: platform_api_latency_seconds
        type: histogram
        labels: ["service", "endpoint"]
        buckets: [0.1, 0.5, 1, 2, 5, 10]

      - name: platform_external_api_calls_total
        type: counter
        labels: ["target", "status"]

      - name: platform_cache_hits_total
        type: counter
        labels: ["cache_name"]

      - name: platform_rate_limit_violations_total
        type: counter
        labels: ["consumer", "route"]

      - name: platform_active_consumers
        type: gauge
        labels: ["consumer_type"]

  alerts:
    - name: HighExternalAPILatency
      expr: histogram_quantile(0.95,
platform_external_api_calls_latency_bucket) > 5

```

```

    for: 5m
    severity: warning
    message: "External API {{ $labels.target }} latency above 5s"

- name: HighExternalAPIErrorRate
  expr: rate(platform_external_api_calls_total{status="error"}[5m]) >
0.1
  for: 5m
  severity: warning
  message: "External API {{ $labels.target }} error rate above 10%"

- name: RateLimitViolation
  expr: rate(platform_rate_limit_violations_total[5m]) > 10
  for: 2m
  severity: warning
  message: "High rate limit violations for {{ $labels.consumer }}"

```

3.11. Дашборд интеграций в Grafana

```

{
  "dashboard": {
    "title": "Platform Integrations Dashboard",
    "panels": [
      {
        "title": "External API Calls by Service",
        "type": "timeseries",
        "targets": [
          {
            "expr": "rate(platform_external_api_calls_total[5m])"
          }
        ]
      },
      {
        "title": "Cache Hit Ratio",
        "type": "stat",
        "targets": [
          {
            "expr": "platform_cache_hits_total / platform_cache_total"
          }
        ]
      },
      {
        "title": "Rate Limit Violations",
        "type": "piechart",
        "targets": [
          {
            "expr": "sum by (consumer)
(platform_rate_limit_violations_total)"
          }
        ]
      },
      {
        "title": "External API Latency (P95)",

```

```

        "type": "timeseries",
        "targets": [
            {
                "expr": "histogram_quantile(0.95,
platform_external_api_calls_latency_bucket)"
            }
        ]
    },
    {
        "title": "Active Consumers",
        "type": "stat",
        "targets": [
            {
                "expr": "platform_active_consumers"
            }
        ]
    },
    {
        "title": "OPA Decisions",
        "type": "table",
        "targets": [
            {
                "expr": "opa_decision_total"
            }
        ]
    }
]
}

```

4. Контрольные вопросы

1. Какие основные проблемы возникают при интеграции платформы с внешними API?
2. Какую роль выполняет API Gateway в архитектуре платформы?
3. Как реализовать rate limiting для защиты от превышения квот внешних API?
4. Какие стратегии кэширования эффективны для картографических API?
5. Как работает mTLS и зачем он нужен для безопасного взаимодействия?
6. Как OPA интегрируется с API Gateway для управления политиками доступа?
7. Как реализовать fallback при недоступности внешнего API?
8. Какие метрики необходимо отслеживать для мониторинга интеграций?

9. Как обрабатывать ошибки внешних API с circuit breaker?
10. Как обеспечить идемпотентность запросов к внешним API?
11. Как настроить request/response transformation в Kong?
12. Какие методы аутентификации поддерживает Kong?
13. Как реализовать geofencing через OPA?
14. Какие форматы кэширования оптимальны для картографических данных?
15. Как организовать мониторинг использования внешних API (cost tracking)?

5. Задание для индивидуального выполнения (15 вариантов)

5.1. Общие требования для всех вариантов

1. Независимо от варианта, необходимо:
2. Развернуть API Gateway (Kong/Envoy/Nginx) согласно спецификации
3. Настроить маршрутизацию к заданным внешним API
4. Реализовать rate limiting с квотами для разных типов потребителей
5. Настроить кэширование для GET-запросов
6. Реализовать аутентификацию через API key
7. Интегрировать OPA для управления политиками доступа
8. Создать адаптер для конкретного внешнего сервиса
9. Продемонстрировать работу интеграций на примерах

5.2. Варианты заданий

Вар.	API Gateway	Внешние API (2+)	Особенность

Вар.	API Gateway	Внешние API (2+)	Особенность
1	Kong	Яндекс.Карты + OpenStreetMap	Fallback при недоступности
2	Kong	Google Maps + Places API	Кэширование геокодирования
3	Kong	OpenWeatherMap + AirQuality	Rate limiting по минутам
4	Kong	Telegram Bot + Slack	Webhook обработка
5	Kong	S3 + PostgreSQL	Загрузка и хранение данных
6	Envoy	ClickHouse + Prometheus	Агрегация метрик
7	Envoy	Kafka + Redis	Потоковая обработка
8	Kong	OSM + Routing API	mTLS аутентификация
9	Kong	Yandex.Cloud + AWS	Multi-cloud
10	Kong	GitHub + GitLab	CI/CD webhooks
11	Kong	Stripe + PayPal	Платежная интеграция
12	Kong	Twilio + SendGrid	Уведомления

Вар.	API Gateway	Внешние API (2+)	Особенность
3	Nginx	Docker Hub + GHCR	Container registry
4	Kong	Jira + ServiceNow	Трекинг инцидентов
5	Envoy	All (3+)	Полная интеграция

5.3. Настройка OPA в Kong через плагин

lua

```
-- kong/plugins/opa/handler.lua
local BasePlugin = require "kong.plugins.base_plugin"
local http = require "resty.http"
local cJSON = require "cjson"

local OPAHandler = BasePlugin:extend()

function OPAHandler:new()
    OPAHandler.super.new(self, "opa")
end

function OPAHandler:access(config)
    OPAHandler.super.access(self)

    local consumer = kong.client.get_consumer()
    local request = kong.request
    local route = kong.router.get_route()

    local body = {
        input = {
            auth = {
                consumer = consumer and {
                    username = consumer.username,
                    id = consumer.id
                } or nil
            },
        },
        attributes = {
            request = {
                method = request.get_method(),
                path = request.get_path(),
                headers = request.get_headers(),
                query = request.get_query(),
            },
        },
    }
```

```

        route = route and {
            name = route.name,
            id = route.id
        } or nil
    }
}
}

local httpc = http.new()
local res, err = http:request_uri(config.opa_url, {
    method = "POST",
    body = cJSON.encode(body),
    headers = {
        ["Content-Type"] = "application/json"
    }
})

if res and res.status == 200 then
    local data = cJSON.decode(res.body)
    if data.result == true then
        return
    end
end

-- Отказ в доступе
return kong.response.exit(403, {
    message = "Access denied by policy engine"
})

end

return OPAHandler

```

5.4. Docker Compose для развертывания

```

# docker-compose.yml
version: '3.8'

services:
  kong-database:
    image: postgres:15
    environment:
      POSTGRES_DB: kong
      POSTGRES_USER: kong
      POSTGRES_PASSWORD: kong
    volumes:
      - kong_db_data:/var/lib/postgresql/data

  kong-migrations:
    image: kong:3.4
    command: "kong migrations bootstrap"
    environment:
      KONG_DATABASE: postgres

```

```

    KONG_PG_HOST: kong database
    KONG_PG_DATABASE: kong
    KONG_PG_USER: kong
    KONG_PG_PASSWORD: kong
  depends_on:
    - kong database

kong:
  image: kong 3.4
  environment:
    KONG_DATABASE: postgres
    KONG_PG_HOST: kong database
    KONG_PG_USER: kong
    KONG_PG_PASSWORD: kong
    KONG_PROXY_ACCESS_LOG: /dev/stdout
    KONG_ADMIN_ACCESS_LOG: /dev/stdout
    KONG_PROXY_ERROR_LOG: /dev/stderr
    KONG_ADMIN_ERROR_LOG: /dev/stderr
    KONG_ADMIN_LISTEN: 0.0.0.0:8001
    KONG_PLUGINS: bundled opa
  ports:
    - "8000:8000"
    - "8001:8001"
  depends_on:
    - kong migrations
  volumes:
    - ./kong/plugins:/usr/local/share/lua/5.1/kong/plugins

opa:
  image: openpolicyagent/opa:latest
  command: "run --server --set=decision_logs.console=true"
  volumes:
    - ./policies:/policies
  ports:
    - "8181:8181"

konga:
  image: pantsel/konga:latest
  environment:
    NODE_ENV: production
  ports:
    - "1337:1337"

platform-api:
  build: ./platform api
  environment:
    YANDEX_MAPS_API_KEY: ${YANDEX_MAPS_API_KEY}
    ANALYTICS_API_URL: "http://analytics:8080"
  ports:
    - "8080:8080"
  depends_on:
    - kong

prometheus:
  image: prom/prometheus:latest
  volumes:

```

```

- ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
ports:
- "9090:9090"

grafana:
  image: grafana/grafana:latest
  environment:
    GF_SECURITY_ADMIN_PASSWORD: admin
  volumes:
    - ./grafana/dashboards:/etc/grafana/provisioning/dashboards
  ports:
    - "3000:3000"
  depends_on:
    - prometheus

volumes:
  kong_db_data:

```

6. Требования к оформлению отчета

6.1. Структура отчета

1. Титульный лист
2. Цель и задачи работы
3. Ответы на контрольные вопросы
4. Теоретическая часть (API Gateway, интеграционные паттерны)
5. Схема архитектуры интеграционной платформы
6. Конфигурация API Gateway
7. ОРА политики
8. Код адаптеров (maps, analytics, storage)
9. Результаты выполнения:
10. Скриншоты успешных запросов через шлюз
11. Скриншот rate limiting при превышении квоты
12. Скриншот работы кэша
13. Скриншот ОРА запрета доступа
14. Метрики и дашборды
15. Заключение
16. Список использованных источников

7. Критерии оценивания

Критерий	Макс. баллов
Развертывание API Gateway	15
Настройка маршрутизации и rate limiting	15
Реализация кэширования	10
Интеграция OPA для политик доступа	15
Разработка адаптеров внешних API	15
mTLS настройка	10
Демонстрация работы	10
Ответы на контрольные вопросы	10
Итого	100

Детальные критерии

Оценка	Баллы	Требования
Отлично	90-100	Полная интеграция с 2+ внешними API, rate limiting, кэширование, OPA, mTLS работают
Хорошо	75-89	Интеграция работает, но нет mTLS или неполное кэширование
Удовлетворительно	60-74	Только базовая маршрутизация,

Оценка	Баллы	Требования
		без ОРА
Неудовлетворительно	<60	Интеграция не работает

ВВЕДЕНИЕ

Методические указания по организации и проведению самостоятельной работы студентов по дисциплине «Инфраструктура как код и платформенная инженерия» предназначены для обучающихся по направлению подготовки 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»).

Актуальность дисциплины

Современная индустрия разработки программного обеспечения претерпевает фундаментальные изменения в подходах к управлению инфраструктурой. Традиционные методы администрирования уступают место подходу «Инфраструктура как код» (Infrastructure as Code, IaC), при котором инфраструктура описывается декларативными конфигурациями, хранящимися в системах контроля версий. Дальнейшим развитием этого подхода стала платформенная инженерия (Platform Engineering) — дисциплина, занимающаяся созданием внутренних платформ разработчика, обеспечивающих self-service доступ к инфраструктурным ресурсам.

Настоящая дисциплина формирует у магистрантов компетенции, необходимые для проектирования, внедрения и эксплуатации современных

IaC-решений и платформенных сервисов, что является востребованным навыком на рынке труда.

Формируемая компетенция

В соответствии с учебным планом и образовательной программой, в результате освоения дисциплины формируется профессиональная компетенция:

Код	Формулировка компетенции
ПК-2	Способен применять принципы «Инфраструктура как код» и «Конфигурация как код» для автоматизированного, идиempotentного, безопасного и версионизируемого управления облачными, гибридными средами и конфигурациями парков автономных устройств с использованием современных инструментов

Индикаторы достижения компетенции

Индикатор	Формулировка
ИД-1 ПК-2	Описывает облачные ресурсы, сетевые компоненты и вычислительные мощности на декларативных языках конфигураций (HCL, YAML, JSON) с использованием инструментов класса IaC (Terraform, CloudFormation, Pulumi)
ИД-2 ПК-2	Разрабатывает идиempotentные сценарии управления конфигурациями (Ansible, Puppet, Chef) для настройки операционных систем, прикладного ПО и бортового программного обеспечения автономных устройств
ИД-3 ПК-	Применяет системы контроля версий (Git) для хранения,

Индикатор	Формулировка
2	версионирования и совместной работы над кодом инфраструктуры и конфигураций
ИД-4 ПК-2	Интегрирует процессы развертывания инфраструктуры и применения конфигураций в пайплайны непрерывной интеграции и доставки (CI/CD)
ИД-5 ПК-2	Обеспечивает безопасность управления инфраструктурой через код: внедряет сканирование кода на уязвимости, управляет секретами и реализует принцип минимальных привилегий
ИД-6 ПК-2	Реализует GitOps-подход с использованием инструментов (ArgoCD, Flux) для автоматической синхронизации состояния среды с Git-репозиторием и устранения дрейфа конфигураций

Целью настоящих методических указаний является оказание помощи студентам в организации эффективной самостоятельной работы при изучении дисциплины «Инфраструктура как код и платформенная инженерия».

Задачи методических указаний:

1. Определить виды, формы и объем самостоятельной работы студентов
2. Предоставить методические рекомендации по изучению теоретического материала
3. Предоставить методические указания по выполнению лабораторных работ
4. Предложить вопросы для самопроверки и собеседования

5. Установить критерии оценивания компетенций
6. Предоставить перечень необходимого программного обеспечения и литературы

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цель и задачи самостоятельной работы

Цель самостоятельной работы - формирование у студентов способности к самостоятельному освоению новых знаний, умений и навыков в области инфраструктуры как кода и платформенной инженерии, а также закрепление и углубление знаний, полученных в ходе аудиторных занятий.

Задачи самостоятельной работы:

№	Задача	Связь с индикатором ПК-2
	Изучение теоретических основ IaC и Platform Engineering	ИД-1
	Освоение инструментов управления конфигурациями	ИД-2
	Практика работы с системами контроля версий для кода инфраструктуры	ИД-3
	Интеграция IaC в CI/CD пайплайны	ИД-4
	Изучение практик безопасности управления инфраструктурой	ИД-5
	Освоение GitOps подхода	ИД-6
	Выполнение лабораторных работ и оформление отчетов	ИД-1 – ИД-6

№	Задача	Связь с индикатором ПК-2
	Подготовка к текущему контролю и промежуточной аттестации	ИД-1 – ИД-6

1.2. Объем самостоятельной работы

В соответствии с учебным планом, общая трудоемкость дисциплины составляет 3 зачетные единицы (108 академических часов), из которых:

Вид работы	Часов
Контактная работа (лекции)	18
Контактная работа (лабораторные работы)	36
Самостоятельная работа	54
Промежуточная аттестация (зачет с оценкой)	—
Итого	108

Распределение самостоятельной работы по темам:

№	Тема / Раздел	Часов
1	Модульный Terraform. Управление состоянием	5
2	Terragrunt и DRY-конфигурации	5
3	Policy as Code и OPA	5
4	Cost as Code и FinOps	5
5	Security as Code и SLSA	5

№	Тема / Раздел	Часов
6	Platform Engineering и Auto-DevOps	5
7	Observability of the Platform и SLO	10
8	Управление устройствами и OTA	14
Итого	54	

1.3. Виды самостоятельной работы

Вид работы	Описание	Трудоемкость (час/нед)
Проработка лекционного материала	Изучение конспектов лекций, дополнение материала из рекомендуемой литературы	2
Подготовка к лабораторным работам	Изучение методических указаний, подготовка программного обеспечения, предварительное ознакомление с инструментами	4
Выполнение лабораторных работ	Разработка конфигураций, написание кода, тестирование, отладка	6
Оформление отчетов	Подготовка отчетов по лабораторным работам в соответствии с требованиями	2
Самостоятельное	Изучение тем, вынесенных на	2

Вид работы	Описание	Трудоемкость (час/нед)
изучение тем	самостоятельную проработку (вопросы для собеседования)	
Подготовка к зачету	Повторение материала, выполнение итогового задания	4

1.4. Планируемые результаты самостоятельной работы

В результате выполнения самостоятельной работы студент должен:

Индикатор	Знать	Уметь	Владеть
ИД-1	Синтаксис HCL/YAML/JSON, архитектуру Terraform	Разрабатывать модульные конфигурации	Навыками работы с Terraform и Pulumi
ИД-2	Принципы идемпотентности, архитектуру Ansible	Писать плейбуки и роли	Навыками управления конфигурациями
ИД-3	Модели ветвления Git (GitFlow, GitHub Flow)	Разрешать конфликты в IaC-коде	Навыками командной работы с Git
ИД-4	Структуру CI/CD пайплайнов (GitLab CI/GitHub Actions)	Интегрировать Terraform и Ansible в CI/CD	Навыками настройки Quality Gates
ИД-5	Инструменты SAST/SCA для IaC	Настраивать сканирование	Навыками безопасного

Индикатор	Знать	Уметь	Владеть
	(Checkov, tfsec)	конфигураций	управления секретами
ИД-6	Архитектуру GitOps (ArgoCD, Flux)	Настраивать синхронизацию состояния	Навыками устранения дрейфа конфигураций

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Работа с лекционным материалом

Лекции являются основой теоретической подготовки. При работе с лекционным материалом рекомендуется:

Ведение конспекта. Записывайте основные определения, схемы и ключевые выводы. Используйте сокращения и символы для ускорения записи.

Маркировка непонятого. В процессе лекции или при последующем чтении конспекта отмечайте вопросы, требующие дополнительного изучения.

Дополнение материала. После лекции дополните конспект информацией из рекомендуемой литературы и онлайн-источников.

Составление интеллект-карт. Для сложных тем (архитектура Terraform, OPA Gatekeeper, GitOps) рекомендуется строить схемы и диаграммы.

Рекомендуемая структура конспекта лекции:

Тема лекции: _____

Дата: _____

1. Ключевые понятия:

- Термин 1: определение
- Термин 2: определение

2. Основные положения:

- 2.1. ...
- 2.2. ...

3. Схемы и диаграммы:

[вклеить / нарисовать]

4. Вопросы для самостоятельного изучения:

- Вопрос 1
- Вопрос 2

5. Список источников:

- [1] стр. ...
- [2] стр. ...

2.2. Работа с учебной и научной литературой

Рекомендации по работе с литературой:

Этап	Действие	Время
Ознакомительное чтение	Просмотр оглавления, аннотации, введения. Определение структуры источника	5-10 мин
Выборочное чтение	Изучение разделов, непосредственно связанных с изучаемой темой	20-30 мин
Углубленное чтение	Детальный анализ материала с	40-60 мин

Этап	Действие	Время
	конспектированием	
Критический анализ	Сравнение информации из разных источников, выявление противоречий	15-20 мин

Методы конспектирования литературы:

Метод	Описание	Когда применять
Тезисный	Запись основных положений краткими фразами	Для быстрого ознакомления
Плановый	Составление развернутого плана текста	Для структурирования материала
Цитатный	Выписка важных определений и формулировок	Для точного воспроизведения
Схематический	Построение схем, таблиц, диаграмм	Для визуализации связей

2.3. Работа с нормативной документацией

При изучении дисциплины необходимо ознакомиться со следующими нормативными документами:

Документ	Область применения	Ресурс

Документ	Область применения	Ресурс
ГОСТ Р ИСО/МЭК 12207-2010	Жизненный цикл ПО	Электронный фонд нормативно-технической документации
Приказ Минцифры № 000	Меры импортозамещения ПО	consultant.ru
Технические требования к IaC	Внутренние стандарты организации	Предоставляется преподавателем

2.4. Работа с онлайн-ресурсами

Для успешного освоения дисциплины рекомендуется использовать следующие типы онлайн-ресурсов:

Тип ресурса	Примеры	Назначение
Официальная документация	Terraform Docs, Kubernetes Docs	Изучение синтаксиса и возможностей инструментов
Профильные блоги	Habr (DevOps), Selectel Blog	Ознакомление с практическими кейсами
Видео-лекции	YouTube-каналы (HighLoad, DevOpsConf)	Визуализация сложных концепций
Интерактивные тренажеры	Killercoda, Katacoda	Практика без установки ПО

Инструмент	Способ установки	Проверка
OPA	winget install OpenPolicyAgent.OPA	opa version
Conftest	winget install OpenPolicyAgent.Conftest	conftest --version
Docker Desktop	С официального сайта	docker version
Minikube	winget install Kubernetes.minikube	minikube version
kubectrl	winget install Kubernetes.kubectrl	kubectrl version
Helm	winget install Helm.Helm	helm version
Git	winget install Git.Git	git --version

Рекомендации по настройке окружения:

1. Создайте отдельную директорию для лабораторных работ: ~/lab-infra-as-code/
2. Инициализируйте Git-репозиторий для версионирования ваших конфигураций
3. Используйте git tag для маркировки завершенных лабораторных работ
4. Для работы с Kubernetes используйте Minikube или Kind (локальный кластер)

3.2. Структура отчета по лабораторной работе

Отчет по каждой лабораторной работе должен содержать следующие разделы:

Раздел	Содержание	Объем

Раздел	Содержание	Объем
Титульный лист	Название работы, ФИО, группа, вариант, преподаватель	1 стр.
Цель работы	Формулировка цели из задания	0.25 стр.
Задачи работы	Перечень конкретных задач	0.5 стр.
Теоретическая часть	Краткое изложение основных понятий (своими словами)	1-2 стр.
Схема/архитектура	Диаграмма на языке PlantUML или Mermaid	0.5-1 стр.
Выполнение работы	Код, конфигурации, команды с комментариями	2-4 стр.
Результаты	Скриншоты выполнения, выводы команд	1-2 стр.
Анализ результатов	Обсуждение полученных результатов, проблемы и их решение	1 стр.
Выводы	Достигнуты ли цель и задачи, что нового освоено	0.5 стр.
Список источников	Литература и ссылки, использованные при выполнении	0.25 стр.

Пример оформления кода в отчете:

```
terraform
```

```
# main.tf
```

```
# Модуль для создания VPC в окружении dev
module "vpc" {
  source = "git::https://gitlab.ncfu.ru/modules/vpc.git?ref=v1.0.0"

  environment = "dev"
  vpc_cidr    = "10.0.0.0/16"
  subnet_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]
}
```

3.3. Рекомендации по работе с инструментами IaC

Terraform / OpenTofu:

bash

Рекомендуемый порядок команд при работе

terraform init # Инициализация (загрузка провайдеров и модулей)

terraform fmt -recursive # Форматирование кода (делать перед commit)

terraform validate # Проверка синтаксиса

terraform plan # Просмотр планируемых изменений

terraform apply -auto-approve # Применение (только после plan)

Terragrunt:

bash

При работе с Terragrunt

terragrunt run-all plan # Планирование во всех модулях

terragrunt run-all apply # Применение во всех модулях

terragrunt output # Получение outputs

Рекомендации по организации кода:

text

infrastructure/

├── modules/ # Переиспользуемые модули

| ├── network/

| | ├── main.tf

| | ├── variables.tf

```

| | | └─ outputs.tf
| | | └─ README.md
| | └─ compute/
| |   └─ ...
| └─ environments/
|   └─ dev/
|     └─ terragrunt.hcl
|     └─ network/
|       └─ terragrunt.hcl
|       └─ prod/
|       └─ ...
└─ .gitlab-ci.yml

```

3.4. Рекомендации по работе с Kubernetes

Для выполнения лабораторных работ рекомендуется использовать локальный кластер:

Инструмент	Особенность	Ресурсы
Minikube	Полноценный кластер, поддержка аддонов	2-4 GB RAM
Kind	Легковесный, быстрый запуск	1-2 GB RAM
K3d	K3s в Docker, очень быстрый	1-2 GB RAM

Установка и запуск Minikube:

```
bash
```

```
# Установка Minikube
```

```
minikube start --cpus=4 --memory=8192 --driver=docker
```

```
# Проверка
```

```
kubectl get nodes
```

```
# Установка аддонов
```

```
minikube addons enable ingress
```

```
minikube addons enable metrics-server
```

```
# Для доступа к дашборду
```

```
minikube dashboard
```

Установка Gatekeeper (LP6):

```
bash
```

```
helm repo add gatekeeper https://open-policy-agent.github.io/gatekeeper/charts
```

```
helm install gatekeeper gatekeeper/gatekeeper --namespace gatekeeper-system --create-namespace
```

3.5. Рекомендации по работе с CI/CD

Для лабораторных работ, требующих CI/CD (LP5, LP11, LP12), рекомендуется:

Платформа	Особенность
GitLab SaaS	Бесплатный аккаунт, публичные репозитории
GitLab CE локально	Требует ресурсов (4+ GB RAM)
GitHub Actions	Бесплатно для публичных репозиториях

GitLab CI переменные, необходимые для работ:

Переменная	Назначение	Где задать
GITLAB_TOKEN	Для комментариев в MR	Settings → CI/CD → Variables

Переменная	Назначение	Где задать
KUBE_CONFIG	Конфигурация доступа к кластеру	Settings → CI/CD → Variables (protected)
INFRACOST_API_KEY	API ключ Infracost Cloud	Settings → CI/CD → Variables
COSIGN_KEY	Приватный ключ для подписи образов	Settings → CI/CD → Variables (protected)

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

4.1. Раздел 1: Infrastructure as Code

Что такое Infrastructure as Code (IaC)? В чем отличие декларативного и императивного подходов?

Terraform vs Pulumi vs CloudFormation. Сравнение инструментов, области применения.

Terraform State. Для чего нужен state-файл? Какие проблемы возникают при работе в команде?

Блокировки состояния. Как работают блокировки в S3 backend? Чем отличаются от блокировок в PostgreSQL?

Модули Terraform. Какова рекомендуемая структура модуля? Для чего нужны locals?

Terragrunt. Как Terragrunt решает проблему дублирования конфигураций? Что такое dependency?

Принцип DRY в IaC. Как Terragrunt и модули помогают соблюдать DRY?

Backend в Terraform. Какие типы backend'ов существуют? Какой выбрать для командной работы?

Terraform Workspaces. Когда использовать workspaces, а когда отдельные конфигурации?

Terraform Registry. Как опубликовать модуль в частном регистре? Для чего нужны теги?

4.2. Раздел 2: Policy as Code

Что такое Policy as Code? Какие проблемы решает подход? Примеры использования.

Open Policy Agent (OPA). Архитектура OPA, компоненты, integration points.

Язык Rego. В чем отличие `:=`, `==` и `=`? Обязательность `if` в OPA 1.0.

Conftest. Как Conftest интегрируется с CI/CD? Как вернуть exit code при violation?

OPA Gatekeeper. В чем отличие Gatekeeper от "чистого" OPA в Kubernetes?

ConstraintTemplate vs Constraint. Как они связаны? Что такое enforcementAction?

Режимы Gatekeeper. Чем отличаются `deny`, `dryrun` и `warn`?

Политики безопасности для Pod. Как запретить привилегированные контейнеры через Gatekeeper?

Terraform plan JSON. Структура JSON. Какие поля необходимы для проверки политик?

Quality Gate. Как организовать автоматическую блокировку non-compliant изменений в CI/CD?

4.3. Раздел 3: Platform Engineering

Что такое Platform Engineering? Отличие от DevOps. Что такое Internal Developer Platform (IDP)?

Golden Paths и Paved Roads. Что это такое? Зачем нужны?

Auto-DevOps. Как организовать reusable шаблоны CI/CD в GitLab? Механизм include.

Эфемерные раннеры. Преимущества перед постоянными. Как настроить в Kubernetes?

Preview Environments. Зачем нужны? Как организовать изолированные окружения по Pull Request?

Уникальные идентификаторы. CI_COMMIT_REF_SLUG vs CI_ENVIRONMENT_SLUG. Как использовать?

GitOps. В чем отличие GitOps от CI/CD? Компоненты GitOps (operator, source controller).

ArgoCD vs Flux. Сравнение инструментов. Sync window, auto-sync, prune.

Дрейф конфигураций. Причины возникновения. Как GitOps борется с дрейфом?

Service Level Objectives для платформы. Чем отличаются SLO платформы от SLO приложений?

4.4. Раздел 4: Безопасность и FinOps

1. SLSA (Supply-chain Levels for Software Artifacts). Уровни SLSA, требования к каждому уровню.
2. Sigstore экосистема. Fulcio, Rekor, Cosign. Key-based vs Keyless signing.
3. SBOM (Software Bill of Materials). Форматы CycloneDX и SPDX. Зачем нужен SBOM?
4. Cosign. Как подписать контейнерный образ? Как проверить подпись перед деплоем?
5. HashiCorp Vault. Механизм аутентификации через Kubernetes Service Account.
6. Vault Agent Injector. Как работает sidecar инжектор? Аннотации для Pod.
7. Секреты в Kubernetes. Проблемы стандартных Secrets. Чем Vault безопаснее?

8. FinOps. Принципы FinOps. Что такое Cost as Code? Роль инженеров в управлении затратами.
9. Infracost. Как Infracost оценивает стоимость? Что такое Guardrails? Portefeuille types.
10. Karpenter. Отличие от Cluster Autoscaler. Консолидация: WhenEmpty, WhenUnderutilized.
- 4.5. Раздел 5: Управление устройствами
11. GitOps для конфигураций устройств. Архитектура. Как ОРА валидирует конфигурации БПЛА?
12. OTA-обновления (Over-The-Air). Стратегии: Canary, Rolling, Blue-Green. Преимущества и недостатки.
13. Canary обновление. Как выбрать процент? Как автоматизировать откат при ошибках?
14. Сбор телеметрии с устройств. Вызовы: ограниченные ресурсы, нестабильная связь, масштаб.
15. Буферизация данных. Как организовать persistent buffer на edge-устройстве (SQLite, disk queue)?
16. OpenTelemetry Collector на edge. Ограничения памяти, сжатие, retry, backoff.
17. Архитектура платформы управления устройствами. Компоненты: registry, OTA coordinator, telemetry.
18. API Gateway для устройств. Kong, Envoy. Зачем нужен? Rate limiting, аутентификация.
19. Интеграция с внешними API. Картографические сервисы, аналитика. Паттерны: adapter, cache, fallback.
20. Метрики и алерты для парка устройств. Какие SLI/SLO определять? Burn rate.

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Формируемая компетенция

Код	Наименование	Индикаторы
ПК-2	Способен применять принципы «Инфраструктура как код» и «Конфигурация как код»	ИД-1, ИД-2, ИД-3, ИД-4, ИД-5, ИД-6

5.2. Критерии оценивания самостоятельной работы

Индикатор	Вид самостоятельной работы	Критерии оценки
ИД-1	Разработка Terraform модулей, Terragrunt конфигураций	Корректность синтаксиса, использование locals, структура модуля
ИД-2	Написание Ansible плейбуков (при необходимости)	Идемпотентность, обработка ошибок, модульность
ИД-3	Работа с Git (ветки, теги, MR)	Наличие тегов для версий, понятные commit messages
ИД-4	Интеграция IaC в CI/CD	Работающий пайплайн, прохождение Quality Gates
ИД-5	Настройка секретов, сканирование уязвимостей	Отсутствие секретов в коде, прохождение SAST
ИД-6	Настройка ArgoCD/Flux для синхронизации	Автоматическое применение, корректная синхронизация

5.3. Шкала оценивания

Оценка	Баллы (за семестр)	Характеристика
Отлично	90-100	Студент полностью освоил компетенцию ПК-2: демонстрирует глубокое понимание принципов IaC и Platform Engineering, самостоятельно выполняет лабораторные работы, активно участвует в обсуждениях, отличное качество отчетов
Хорошо	75-89	Студент в целом освоил компетенцию, но допускает отдельные неточности; лабораторные работы выполнены, но есть небольшие недочеты
Удовлетворительно	60-74	Компетенция освоена частично; студент испытывает трудности при самостоятельном выполнении работ; отчеты неполные
Неудовлетворительно	<60	Компетенция не освоена; студент не может выполнить

Оценка	Баллы (за семестр)	Характеристика
		лабораторные работы без помощи; отчеты отсутствуют или не соответствуют требованиям

5.4. Порядок начисления баллов

Балльно-рейтинговая система:

Вид работы	Кол-во	Макс. баллов за единицу	Итого
Лабораторные работы (ЛР1-18)	18	5	90
Активность на лекциях (доклады, вопросы)	—	—	5
Промежуточное тестирование (при наличии)	1	5	5
Итого			100

Соответствие баллов оценке:

Оценка	Баллы (при зачете с оценкой)
Отлично	90 – 100
Хорошо	75 – 89
Удовлетворительно	60 – 74

Оценка	Баллы (при зачете с оценкой)
Неудовлетворительно	0 – 59

Условия допуска к зачету:

Выполнены и защищены не менее 12 лабораторных работ (из 18)

Суммарный балл за лабораторные работы не менее 50 баллов

6. ПЕРЕЧЕНЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

№	Наименование ПО	Лицензия	Статус в РФ	Назначение
1	Terraform (≥ 1.5)	Open Source (MPL 2.0)	Свободное ПО	Управление инфраструктурой как код
2	OpenTofu	Open Source (MPL 2.0)	Свободное ПО (форк Terraform)	Альтернатива Terraform
3	Terragrunt	Open Source (MIT)	Свободное ПО	DRY-конфигурации для Terraform
4	OPA (Open Policy Agent)	Open Source (Apache 2.0)	Свободное ПО	Policy Engine
5	Conftest	Open Source (Apache 2.0)	Свободное ПО	Тестирование политик в CI/CD
6	Gatekeeper	Open Source (Apache 2.0)	Свободное ПО	Admission controller для K8s

№	Наименование ПО	Лицензия	Статус в РФ	Назначение
7	Infracost	Open Source (Apache 2.0)	Свободное ПО	Cost as Code
8	Cosign (Sigstore)	Open Source (Apache 2.0)	Свободное ПО	Подпись контейнерных образов
9	Syft	Open Source (Apache 2.0)	Свободное ПО	Генерация SBOM
10	Trivy	Open Source (Apache 2.0)	Свободное ПО	Сканирование уязвимостей
11	Docker Desktop	Freemium	Доступен	Контейнеризация
12	Minikube	Open Source (Apache 2.0)	Свободное ПО	Локальный Kubernetes кластер
13	Kind	Open Source (Apache 2.0)	Свободное ПО	Kubernetes in Docker
14	K3d	Open Source (MIT)	Свободное ПО	K3s в Docker
15	Helm	Open Source (Apache 2.0)	Свободное ПО	Пакетный менеджер для K8s
16	kubectl	Open Source	Свободное	Управление

№	Наименование ПО	Лицензия	Статус в РФ	Назначение
		(Apache 2.0)	ПО	кластером K8s
17	Git	Open Source (GPL v2)	Свободное ПО	Система контроля версий
18	GitLab CE	Open Source (MIT)	Свободное ПО	CI/CD платформа
19	VS Code	Open Source (MIT)	Свободное ПО	Среда разработки
20	Python (≥ 3.9)	Open Source (PSF)	Свободное ПО	Скрипты автоматизации
21	Go (≥ 1.21)	Open Source (BSD)	Свободное ПО	Альтернатива для разработки
22	Prometheus	Open Source (Apache 2.0)	Свободное ПО	Сбор метрик
23	Grafana	Open Source (AGPL v3)	Свободное ПО	Визуализация
24	OpenTelemetry Collector	Open Source (Apache 2.0)	Свободное ПО	Сбор телеметрии
25	PostgreSQL	Open Source (PostgreSQL License)	Свободное ПО	БД для backend состояний

Примечание: Все перечисленное ПО является открытым или свободно распространяемым. Для работы не требуется приобретение коммерческих лицензий.

7. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

7.1. Основная литература

№	Библиографическое описание	Режим доступа
1	Брикман, Д. Terraform: инфраструктура на уровне кода / Д. Брикман. — СПб.: Питер, 2021. — 336 с. — ISBN 978-5-4461-1234-5	Печатное издание / ЭБС «Лань»
2	Хохель, Б. Ansible для DevOps: управление конфигурациями и автоматизация развертывания / Б. Хохель. — М.: ДМК Пресс, 2020. — 480 с. — ISBN 978-5-97060-123-4	Печатное издание
3	Рожков, М. Л. Git и GitHub для командной разработки / М. Л. Рожков. — М.: СОЛОН-Пресс, 2022. — 252 с. — ISBN 978-5-91359-123-4	Печатное издание
4	Васильев, А. Н. Системная интеграция программных средств: учебное пособие / А. Н. Васильев. — М.: ИНФРА-М, 2021. — 208 с. — ISBN 978-5-16-014567-8	ЭБС Znanium
5	Макаров, Е. В. Управление конфигурациями в CI/CD: практикум / Е. В. Макаров. — М.: Лаборатория знаний, 2023. — 184 с. — ISBN 978-5-93208-123-4	Печатное издание

7.2. Дополнительная литература

	Библиографическое описание	Режим доступа
	Лукс, С. Kubernetes в действии / С. Лукс. — М.: ДМК Пресс, 2021. — 672 с. — ISBN 978-5-97060-567-8	Печатное издание
	Лиггетт, Б. GitOps и Kubernetes / Б. Лиггетт. — М.: ДМК Пресс, 2022. — 304 с. — ISBN 978-5-93700-123-4	Печатное издание
	Бертон, Ш. ОТА-обновления встроенных систем / Ш. Бертон. — М.: Техносфера, 2022. — 312 с. — ISBN 978-5-94836-123-4	Печатное издание
	Нейлор, Д. DevSecOps: практики безопасности / Д. Нейлор. — М.: Альпина Паблишер, 2023. — 456 с. — ISBN 978-5-9614-1234-5	Печатное издание
	Site Reliability Engineering. Как Google управляет production-системами / Б. Бейер и др. — СПб.: Питер, 2019. — 552 с. — ISBN 978-5-4461-1188-6	Печатное издание / ЭБС

7.3. Ресурсы сети «Интернет»

№	Наименование ресурса	Адрес	Описание
1	Terraform Documentation	https://developer.hashicorp.com/terraform/docs	Официальная

№	Наименование ресурса	Адрес	Описание
			документация Terraform
	Terragrunt Documentation	https://terragrunt.gruntwork.io/docs	Официальная документация Terragrunt
	Open Policy Agent Documentation	https://www.openpolicyagent.org/docs	Официальная документация ОРА
	Conftest Documentation	https://www.conftest.dev/docs	Официальная документация Conftest
	Gatekeeper Documentation	https://open-policy-agent.github.io/gatekeeper	Документация ОРА Gatekeeper
	Infracost Documentation	https://www.infracost.io/docs	Официальная документация Infracost
	Sigstore / Cosign	https://docs.sigstore.dev	Документация Sigstore

№	Наименование ресурса	Адрес	Описание
	Karpenter Documentation	https://karpenter.sh/docs	Официальная документация Karpenter
	GitLab CI/CD Documentation	https://docs.gitlab.com/ee/ci	Документация GitLab CI/CD (зеркало)
0	ArgoCD Documentation	https://argo-cd.readthedocs.io	Документация ArgoCD
1	Flux Documentation	https://fluxcd.io/docs	Документация Flux
2	OpenTelemetry Documentation	https://opentelemetry.io/docs	Документация OpenTelemetry
3	Prometheus Documentation	https://prometheus.io/docs	Документация Prometheus
4	Grafana Documentation	https://grafana.com/docs	Документация Grafana
	Habr (DevOps /	https://habr.com/ru/hub/devops	Статьи на

№	Наименование ресурса	Адрес	Описание
5	SRE)		русском языке
6	Selectel Blog (IaC)	https://selectel.ru/blog	Статьи по Terraform и K8s
7	Научная электронная библиотека eLIBRARY.RU	https://www.elibrary.ru	Научные публикации
8	ЭБС «Лань»	https://e.lanbook.com	Учебная литература
9	ЭБС IPRbooks	https://www.iprbookshop.ru	Учебная литература
10	Книга «Pro Git» на русском	https://git-scm.com/book/ru/v2	Исчерпывающее руководство по Git

8. ТРЕБОВАНИЯ К ОТЧЕТНОСТИ

По окончании семестра студент предоставляет:

Пакет отчетов по лабораторным работам (ЛР1-ЛР18) в электронном виде (PDF или Markdown)

Ссылку на Git-репозиторий со всеми конфигурациями и кодом

Скриншоты/записи демонстраций по требованию преподавателя

Требования к оформлению отчетов:

Формат: PDF или Markdown (с возможностью просмотра без дополнительного ПО)

Шрифт: Times New Roman, 14 пт (для PDF), моноширинный для кода

Поля: левое — 30 мм, правое — 15 мм, верхнее/нижнее — 20 мм

Нумерация страниц: внизу по центру

Именование файлов: Фамилия_ЛР01_ВариантXX.pdf

ПРИЛОЖЕНИЕ А. Пример оформления отчета

(см. отдельный файл шаблона)

ПРИЛОЖЕНИЕ Б. Перечень лабораторных работ

№	Название лабораторной работы	Компетенции	Часов (СРС)
1	Организация модульного Terraform с частным регистром	ПК-2 (ИД-1, ИД-3)	4
2	Управление удаленным состоянием и блокировками	ПК-2 (ИД-1, ИД-3, ИД-4)	4
3	Внедрение Terragrunt для DRY-конфигураций	ПК-2 (ИД-1, ИД-3, ИД-4)	4
4	Написание политик на Rego для OPA	ПК-2 (ИД-4, ИД-5)	4
5	Интеграция OPA с Terraform	ПК-2 (ИД-4, ИД-5)	4

№	Название лабораторной работы	Компетенции	Часов (СРС)
	в CI/CD		
6	Policy as Code в Kubernetes: OPA Gatekeeper	ПК-2 (ИД-4, ИД-5, ИД-6)	2
7	Cost as Code с Infracost	ПК-2 (ИД-4, ИД-5)	2
8	FinOps: автоматическая оптимизация затрат (Karpenter)	ПК-2 (ИД-4, ИД-5)	2
9	Security as Code: управление секретами с Vault	ПК-2 (ИД-4, ИД-5)	2
10	1 SLSA-совместимая цепочка поставки (Sigstore)	ПК-2 (ИД-4, ИД-5, ИД-6)	2
11	1 Auto-DevOps шаблон как услуга	ПК-2 (ИД-4, ИД-6)	2
12	1 Preview-окружения по Pull Request	ПК-2 (ИД-4, ИД-6)	2
13	1 Наблюдаемость платформы (OpenTelemetry)	ПК-2 (ИД-4, ИД-6)	2
14	1 SLO и Error Budget платформы	ПК-2 (ИД-4, ИД-5, ИД-6)	2

№	Название лабораторной работы	Компетенции	Часов (СРС)
5	1 Управление конфигурациями устройств через Git + OPA	ПК-2 (ИД-2, ИД-3, ИД-4, ИД-5, ИД-6)	2
6	1 OTA-обновления как услуга платформы	ПК-2 (ИД-4, ИД-5, ИД-6)	2
7	1 Сбор телеметрии с устройств в платформу	ПК-2 (ИД-4, ИД-6)	2
8	1 Интеграция платформы с внешними API	ПК-2 (ИД-4, ИД-5, ИД-6)	2
Итого			54