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 «Разработка геоприложений»

для студентов направления подготовки 09.03.04 Программная инженерия
«Разработка и сопровождение программного обеспечения»

Ставрополь, 2026

Содержание

Введение
Лабораторная работа №1. Развертывание инфраструктуры мониторинга на базе Prometheus и VictoriaMetrics
Лабораторная работа №2. Реализация кастомных метрик для отслеживания состояния автономных устройств
Лабораторная работа №3. Централизованный сбор и анализ логов с Loki и Promtail
Лабораторная работа №4. Распределенная трассировка микросервисного приложения (Jaeger/Tempo)
Лабораторная работа №5. OpenTelemetry Collector: унифицированный сбор телеметрии с edge-устройств
Лабораторная работа №6. Настройка алертинга и SLO-контроля для парка устройств
Лабораторная работа №7. Интеграция Observability в CI/CD конвейер (GitLab CI)
Лабораторная работа №8. Проектирование полностековой платформы наблюдаемости для роя дронов (комплексный проект)
Заключение

Введение

Современные системы управления роем автономных устройств (БПЛА, наземные роботы) предъявляют жёсткие требования к наблюдаемости (observability): необходимо в реальном времени отслеживать тысячи метрик, анализировать логи, выявлять аномалии и трассировать сложные микросервисные сценарии. Дисциплина направлена на формирование практических навыков построения промышленной инфраструктуры мониторинга с использованием открытых компонентов: Prometheus, VictoriaMetrics, Loki, Tempo, OpenTelemetry Collector, Grafana, Alertmanager, а также внедрение observability как код в CI/CD.

Лабораторный практикум включает 8 работ, от развёртывания базового стека до комплексного проекта, имитирующего реальный парк дронов. Каждая работа содержит вариативные задания трёх уровней сложности, что позволяет адаптировать курс под разный уровень подготовки.

Лабораторная работа №1. Развертывание инфраструктуры мониторинга на базе Prometheus и VictoriaMetrics

1. Цель работы

Освоить развертывание высокодоступной системы мониторинга для парка автономных устройств (БПЛА) с использованием Prometheus и долговременного хранилища VictoriaMetrics. Сформировать практические навыки конфигурирования сборщиков метрик, федерации и визуализации состояния роя дронов.

2. Формируемые компетенции и индикаторы

- **ИД-7пк-10**: Администрирует инфраструктуру платформы управления, обеспечивая её бесперебойную работу, масштабирование и защиту от НСД.
- **ИД-3пк-10**: Организует сбор, обработку, хранение и визуализацию потоковой телеметрии устройств в реальном времени.

3. Теоретическое обоснование

3.1. Ключевые концепции

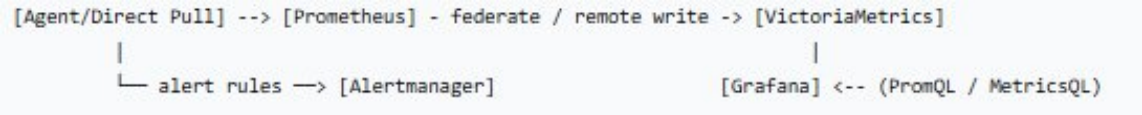
- **Prometheus** – pull-ориентированная time-series database (TSDB), оптимальная для краткосрочного хранения и оперативных алертов.
- **VictoriaMetrics** – совместимая с Prometheus TSDB, обеспечивающая эффективное долговременное хранение, высокую скорость ingest и сжатия.
- **Federation / Remote Write** – механизмы объединения данных для централизации мониторинга распределённых площадок (аэродромов, базовых станций).
- **Service Discovery** – автоматическое обнаружение целей (дронов) через файл, DNS или API диспетчера.

3.2. Сравнение Prometheus и VictoriaMetrics

Характеристика	Prometheus	VictoriaMetrics
Хранилище	Локальное TSDB, ограниченное объёмом диска	Оптимизированное слияние партиций, эффективная компрессия
Масштабирование	Федерация, Thanos/Cortex	Кластерная версия (vminsert, vmselect, vmstorage)
Потребление RAM	Высокое (индексы в памяти)	Низкое (адаптивная работа с памятью)

Характеристика	Prometheus	VictoriaMetrics
Запись	Pull + Remote Write приёмник	Принимает Remote Write от Prometheus, агентов, OTel Collector

3.3. Архитектурная схема



4. Задачи работы (для студента)

1. Развернуть Docker Compose со стеком: Prometheus, VictoriaMetrics, Grafana, Alertmanager.
2. Создать симулятор одного БПЛА, отдающий телеметрию (координаты, высота, заряд батареи, статус) в формате Prometheus на endpoint /metrics.
3. Настроить сбор метрик с симулятора Prometheus, передачу long-term в VictoriaMetrics посредством remote_write.
4. Построить Grafana-дашборд с панелями: заряд батареи, высота, uptime устройства.
5. Проверить сохранность метрик после перезапуска Prometheus.

5. Задания для индивидуального выполнения

Базовый уровень (варианты 1–5)

1. Развернуть Prometheus + Grafana + VictoriaMetrics (single-node) согласно примеру. Настроить сбор метрик с одного симулятора, отобразить статус дрона.
2. То же, но вместо VictoriaMetrics использовать Prometheus с локальным хранилищем и retention 24h.
3. Добавить Alertmanager, правило на падение дрона (drone_up == 0).
4. Использовать VictoriaMetrics как remote write приёмник для трёх дронов (разные порты симуляторов).
5. Настроить базовую аутентификацию в Grafana и Prometheus (через nginx basic auth).

Средний уровень (варианты 6–10)

6. Реализовать федерацию: один Prometheus собирает метрики с дронов, второй Prometheus – агрегирует выборки и передаёт в VictoriaMetrics.

7. Настроить scrape_config с file_sd, динамически подхватывающий симуляторы из JSON-файла.
8. Настроить репликацию записи: Prometheus отправляет remote_write в два экземпляра VictoriaMetrics одновременно.
9. Развернуть кластер VictoriaMetrics (vminsert + vmstorage + vmselect) в Docker Compose.
10. Интегрировать сбор метрик системных ресурсов (node_exporter) самого диспетчерского сервера и коррелировать с метриками дронов.

Продвинутый уровень (варианты 11–15)

11. Внедрить OpenTelemetry Collector как прокси для приёма OTLP-метрик и экспорта в Prometheus remote write.
12. Настроить дедупликацию и downsampling в VictoriaMetrics, протестировать на данных трёхчасовой миссии.
13. Реализовать HA-пару Prometheus с синхронизацией алертов через Alertmanager cluster, проверить отсутствие дубликатов.
14. Настроить VictoriaMetrics Operator в Kubernetes (minikube) для управления парком симуляторов.
15. Собрать мониторинг самого мониторинга: Prometheus, VictoriaMetrics, Grafana должны отдавать свои метрики в центральный Prometheus, построить дашборд здоровья платформы.

6. Пример практического выполнения (базовый уровень, вариант 1)

docker-compose.yml

```yaml
version: '3.7'
services:
  prometheus:
    image: prom/prometheus:v2.45.0
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--enable-feature=remote-write-receiver'
  victoriametrics:
    image: victoriametrics/victoria-metrics:v1.93.0
    ports:
      - "8428:8428"
    command:
      - '-storageDataPath=/storage'
      - '-retentionPeriod=12w'
  grafana:
    image: grafana/grafana:10.0.0
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
  drone-simulator:
    build: ./drone-sim
    ports:
      - "8000:8000"
```

prometheus.yml

```yaml
global:
  scrape_interval: 5s
  evaluation_interval: 5s

remote_write:
  - url: http://victoriametrics:8428/api/v1/write

scrape_configs:
  - job_name: 'drone'
    static_configs:
      - targets: ['drone-simulator:8000']
```

drone-simulator/app.py

```python
from prometheus_client import start_http_server, Gauge
import time, math

battery = Gauge('drone_battery_percent', 'Battery level')
altitude = Gauge('drone_altitude_meters', 'Current altitude')
latitude = Gauge('drone_latitude', 'Latitude')
longitude = Gauge('drone_longitude', 'Longitude')
up = Gauge('drone_up', 'Drone connection status')

if __name__ == '__main__':
    start_http_server(8000)
    up.set(1)
    t = 0
    while True:
        battery.set(max(0, 100 - t*0.005))
        altitude.set(10 + 5*math.sin(t/10))
        latitude.set(55.7558 + t*0.00001)
        longitude.set(37.6173 + t*0.00001)
        t += 1
        time.sleep(1)
```

Результат: В Grafana подключаем источники данных Prometheus (http://prometheus:9090) и VictoriaMetrics (http://victoriametrics:8428). Строим панели по метрикам drone_battery_percent, drone_altitude_meters. Останавливаем Prometheus – исторические данные доступны через VictoriaMetrics.

7. Методические указания по выполнению

- При использовании Astra Linux / Ред ОС убедиться, что docker включён и пользователь в группе docker.
- В случае проблем с DNS внутри compose, использовать network_mode: host для отладки.
- Remote write VictoriaMetrics может потреблять до 1 ГБ RAM, минимальные требования 4 ГБ RAM.
- Типичная ошибка: забыть открыть порт 8428 в firewall (iptables) на Astra Linux.

8. Контрольные вопросы

1. Чем pull-модель Prometheus отличается от push-модели? Как это применимо к дронам с нестабильной связью?
2. Зачем нужно долговременное хранилище VictoriaMetrics для парка БПЛА?
3. В чём отличие федерации от remote write?
4. Как обеспечить сохранность метрик при выходе из строя Prometheus?
5. Что такое metric relabeling и для чего может применяться в контексте телеметрии дронов?
6. Как настроить аутентификацию при записи в VictoriaMetrics?

7. Какие метрики обязательно мониторить для оценки надёжности парка?
8. Как Prometheus обнаруживает новые цели (service discovery) без перезапуска?
9. Почему для оперативного алертинга предпочтителен Prometheus, а не VictoriaMetrics?
10. Каким образом VictoriaMetrics реализует дедупликацию меток, если один дрон переподключился с другим IP?

9. Требования к отчёту

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.
- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).
- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №2. Реализация кастомных метрик для отслеживания состояния автономных устройств

1. Цель работы

Научиться разрабатывать экспортёры и инструментировать код симулятора БПЛА с использованием Prometheus-клиента и OpenTelemetry SDK для сбора ключевых показателей: заряд батареи, GPS, скорость, состояние миссии, вибрации.

2. Формируемые компетенции

- **ИД-3пк-10**: Организует сбор, обработку, хранение и визуализацию потоковой телеметрии устройств в реальном времени с системой оповещения об аномалиях.

3. Теоретическое обоснование

3.1. Типы метрик

- **Counter** – только увеличивается (количество успешных миссий).

- **Gauge** – может увеличиваться и уменьшаться (заряд батареи, высота).
- **Histogram** – распределение значений (время выполнения манёвра).
- **Summary** – квантили на стороне клиента (задержки команд).

Метки (labels) – drone_id, mission_type, firmware_version – позволяют агрегировать по парку.

3.2. Подходы к инструментации

- Прямой HTTP-экспортёр (Prometheus client) – минимальные накладные расходы на edge.
- OpenTelemetry SDK – унификация метрик, логов, трейсов, возможность отправки по OTLP.

4. Задачи работы

1. Расширить симулятор дрона кастомными метриками (Counter миссий, Histogram времени полёта, Gauge скорости).
2. Настроить Prometheus на сбор обогащённых метрик.
3. Реализовать второй экземпляр симулятора, отличающийся меткой drone_type.
4. Построить дашборд с агрегированными показателями (средняя скорость по флоту, суммарное время полёта).

5. Варианты индивидуальных заданий

Базовый (1–5)

1. Добавить Gauge drone_speed и Counter mission_completed_total.
2. Создать Histogram maneuver_duration_seconds для замера времени поворота.
3. Реализовать метрику drone_gnss_sats (число спутников) и отследить пропадание сигнала.
4. Добавить метку flight_mode (auto/manual/rtl) и панель распределения по режимам.
5. Написать простой экспортёр для ROS-совместимого робота (опционально).

Средний (6–10)

6. Использовать клиент OpenTelemetry Python для эмуляции метрик и отправки через OTLP на collector.
7. Реализовать Summary battery_drain_rate с квантилями, визуализировать p95.
8. Разработать метрику payload_weight и настроить PromQL-запрос, показывающий дроны с перегрузом.
9. Создать метрики, основанные на вычислении из двух исходных (Recording Rules).

10. Интегрировать симулятор с Redis, кеширующий последнее состояние дрона, и экспортировать через отдельный exporter.

Продвинутый (11–15)
11. Написать динамический Service Discovery для Prometheus на основе API реестра дронов.
12. Реализовать логику расчёта остаточного времени полёта на основе разряда (Gauge прогноза) с linear prediction в PromQL.
13. Внедрить метрики на основе событий шины MQTT (симулятор публикует, агент конвертирует в OpenMetrics).
14. Создать собственный Prometheus exporter на Go для телеметрии PX4 через MAVLink.
15. Подключить VictoriaMetrics Anomaly Detection (через vmanomaly) к метрикам вибрации двигателя и выявлять деградацию.

6. Пример выполнения (базовый, вариант 1)

Расширенный код симулятора:

```
from prometheus_client import start_http_server, Gauge, Counter
import random, time

speed = Gauge('drone_speed_mps', 'Current speed m/s', ['drone_id'])
missions = Counter('mission_completed_total', 'Completed missions', ['drone_type'])

if __name__ == '__main__':
    start_http_server(8000)
    drone_id = "drone-01"
    drone_type = "multirotor"
    missions.labels(drone_type=drone_type).inc(0)  # инициализация
    while True:
        speed.labels(drone_id=drone_id).set(random.uniform(0,15))
        if random.random() > 0.99:
            missions.labels(drone_type=drone_type).inc()
        time.sleep(1)
```

PromQL запросы для дашборда:

- Средняя скорость по флоту: avg(drone_speed_mps)
- Суммарное число миссий по типам: sum by (drone_type) (mission_completed_total)

7. Методические указания

- Метки должны иметь низкую кардинальность; избегать drone_id как метку, если в парке >100 тысяч устройств (лучше использовать внешний mapping).
- Для тестирования гистограмм использовать sum(rate(maneuver_duration_seconds_bucket[1m])).

8. Контрольные вопросы

1. Почему значение заряда батареи следует моделировать Gauge, а не Counter?

2. Как метки влияют на объём хранимых данных в Prometheus?
3. Чем Histogram отличается от Summary? Что предпочтительнее для измерения задержки выполнения команд?
4. Какие преимущества даёт OpenTelemetry SDK перед прямым Prometheus-клиентом?
5. Как обеспечить сбор метрик с дрона, временно теряющего связь?
6. Можно ли агрегировать метрики с разных дронов без использования PromQL?
7. Какие метрики ROS робота вы бы добавили для наблюдаемости?
8. Что такое Recording Rules и зачем они нужны?
9. Как настроить алерт на основе прогноза остатка батареи?
10. Как защитить endpoint /metrics от несанкционированного доступа?

9. Требования к отчёту

Стандартные (схема, листинги, скриншоты), плюс сравнение pull/push подходов, профилирование ресурсов экспортёра.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №3. Централизованный сбор и анализ логов с Loki и Promtail

1. Цель работы

Научиться централизованно собирать, структурировать и анализировать логи с симуляторов БПЛА и микросервисов диспетчера, реализовать связь логов с метриками через Grafana.

2. Компетенции

- **ИД-3пк-10**

3. Теоретическое обоснование

- **Loki** – горизонтально-масштабируемая система агрегации логов, не индексирующая содержимое, а использующая метки как в Prometheus.
- **Promtail** – агент для чтения логов, парсинга и отправки в Loki.

- **Pipeline stages**: parsing (json, regex), labels, timestamp.
- **Корреляция логов и метрик** через dashboard links в Grafana (запрос Loki на основе временного окна и drone_id).

4. Задачи

- Настроить Promtail для сбора логов симулятора (stdout/stderr в Docker).
- Разработать структурированный лог (JSON) в коде дрона с полями timestamp, drone_id, severity, message, trace_id.
- Настроить Loki datasource в Grafana, создать панель журнала событий.
- Организовать корреляцию: из графика метрики батареи в Grafana перейти к логам за тот же период.

5. Варианты

Базовый (1–5)

1. Сбор логов одного дрона в текстовом формате, label job=drone.
2. Парсинг JSON-логов и извлечение поля severity в отдельный label.
3. Настройка Loki с хранением в файловой системе, retention 7 дней.
4. Фильтрация через LogQL: {job="drone"} |= "ERROR".
5. Отображение логов на дашборде с временной синхронизацией с метриками.

Средний (6–10)

6. Использование pipeline для добавления label drone_model из имени файла.
7. Настройка multi-tenancy Loki для разделения логов разных заказчиков (X-Scope-OrgID).
8. Агрегация логов: подсчёт rate ошибок за 5 мин через LogQL метрики.
9. Создание алерта в Loki ruler на частые перезагрузки миссии.
10. Интеграция логов с трассировкой (добавление trace_id в лог и связь в Grafana).

Продвинутый (11–15)

11. Развернуть Loki в микросервисном режиме (distributor, ingester, querier).
12. Организовать сбор логов с OpenTelemetry Collector (filelog receiver) и экспорт в Loki.
13. Настроить Geo-визуализацию: парсить координаты из логов и отображать на карте в Grafana.
14. Применить Loki к логам автопилота ArduPilot, распарсив телеметрические сообщения.
15. Создать автомасштабируемый сбор логов с парка в 50 дронов с динамическими label.

6. Пример выполнения (базовый 1)

docker-compose дополнение:

```yaml
loki:
  image: grafana/loki:2.9.0
  ports:
    - "3100:3100"
promtail:
  image: grafana/promtail:2.9.0
  volumes:
    - ./promtail-config.yml:/etc/promtail/config.yml
    - /var/run/docker.sock:/var/run/docker.sock
  command: -config.file=/etc/promtail/config.yml
```

promtail-config.yml:

```yaml
clients:
  - url: http://loki:3100/loki/api/v1/push
scrape_configs:
  - job_name: drone_logs
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
    relabel_configs:
      - source_labels: [__meta_docker_container_name]
        target_label: container
    pipeline_stages:
      - json:
          expressions:
            severity: severity
            message: message
      - labels:
          severity:
```

Симулятор пишет JSON логи:

```python
import json, sys, time
while True:
    log = {"timestamp": time.time(), "drone_id": "drone-01", "severity": "INFO", "message": "takeof
f"}
    sys.stdout.write(json.dumps(log) + "\n")
    sys.stdout.flush()
    time.sleep(2)
```

7. Методические указания

При больших объёмах логи переходят в режим "streaming", ограничивать время запроса. В Astra Linux установить права Docker socket.

8. Контрольные вопросы

1. Зачем Loki не индексирует текст логов? Преимущества и недостатки?
2. Что такое LogQL и чем он похож на PromQL?
3. Как обеспечить низкую задержку доставки логов с дрона в условии нестабильной связи?
4. Как настроить retention в Loki?
5. Какие ограничения на количество уникальных меток в Loki?

6. Чем Promtail отличается от Fluent Bit?
7. Как связать лог и конкретную точку на графике метрики в Grafana?
8. В чём разница между stream и index в Loki?
9. Можно ли на основе логов создавать метрики и алерты?
10. Как организовать сбор логов с микросервисов диспетчера (Golang, Python) в Kubernetes?

9. Отчёт

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.
- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).
- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №4. Распределенная трассировка микросервисного приложения (Jaeger/Tempo)

1. Цель работы

Получить навыки инструментирования микросервисов управления полётными заданиями для распределённой трассировки, анализа задержек и поиска узких мест при обработке миссий дронов.

2. Компетенции

- **ИД-3пк-10**

3. Теоретическое обоснование

- **Трассировка** – сквозной контекст между сервисами. Span, Trace, распространение контекста (W3C Trace Context, Jaeger).

- **Jaeger All-in-One** удобен для разработки. **Grafana Tempo** использует объектное хранилище, требует OTel Collector.
- **Head-based vs tail-based sampling**. В микросервисной среде: flight-planner → drone-executor → telemetry-saver.

4. Задачи

- Написать три связанных микросервиса на Python (Flask) с OpenTelemetry инструментацией, передавать контекст.
- Развернуть Jaeger All-in-One, настроить экспорт span'ов.
- Проанализировать задержки, добавить кастомные теги (mission_id, drone_id).
- Настроить Tempo с OTel Collector для опытного варианта.

5. Варианты

Базовый

1. Автоматическая инструментация Flask-приложения через opentelemetry-instrument.
2. Ручное создание span'ов для этапов предполётной проверки.
3. Отправка в Jaeger, анализ trace с waterfall.
4. Добавление baggage (mission_id) для передачи бизнес-контекста.
5. Настройка урезанного Tempo с OTel collector (пример).

Средний

6. Реализовать асинхронную отправку span'ов через batch processor collector.
7. Создать span'ы на основе MQTT-сообщений.
8. Настроить Jaeger с Elasticsearch-бэкендом для production-подобного сценария.
9. Добавить кастомный sampler (probabilistic с учётом типа миссии).
10. Интегрировать трассировку с логами: выводить trace_id в структурированный лог.

Продвинутый

11. Развернуть Tempo с MinIO, настроить 5-секундный tail-based sampling ошибок.
12. Реализовать трассировку gRPC-взаимодействия между планировщиком и исполнителем.
13. Сравнить производительность Jaeger и Tempo на 1000 запросах в секунду.
14. Построить Service Graph в Grafana на основе Tempo.
15. Внедрить OpenTelemetry в ROS2 ноду и отследить путь команды до контроллера мотора.

6. Пример выполнения (базовый 1)

docker-compose сервис Jaeger:

```
jaeger:
  image: jaegertracing/all-in-one:1.45
  ports:
    - "16686:16686"  # UI
    - "4317:4317"    # OTLP gRPC
```

Запуск Flask приложения с инструментацией:

```
opentelemetry-instrument \
  --traces_exporter otlp \
  --service_name flight-planner \
  --exporter_otlp_endpoint jaeger:4317 \
  python app.py
```

Результат: в Jaeger UI видны трассы, включающие вызовы между сервисами.

7. Методические указания

- Использовать SDK opentelemetry-distro[otlp] для унификации.
- Проблема: не забыть установить переменные окружения OTEL_EXPORTER_OTLP_ENDPOINT.

8. Контрольные вопросы

1. Что такое span context и как он сериализуется?
2. Зачем нужен tail-based sampling в Tempo?
3. В чём отличия Jaeger и Tempo в архитектуре?
4. Как передать trace_id в логи для корреляции?
5. Какой формат контекста рекомендуется W3C?
6. Какие семантические конвенции OpenTelemetry применимы к миссиям БПЛА?
7. Как ограничить объем трейсов с дронов в канале с низкой пропускной способностью?
8. Можно ли трассировать запросы без изменения кода?
9. Что такое span kind (CLIENT, SERVER) и как они помогают в анализе?
10. Как настроить алерт на основе длительности трасс в Tempo?

. Требования к отчёту

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.
- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).

- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №5. OpenTelemetry Collector: унифицированный сбор телеметрии с edge-устройств

1. Цель работы

Освоить унифицированный подход к сбору метрик, логов и трейсов с симулированных автономных устройств (БПЛА) с использованием OpenTelemetry Collector, обеспечив централизованную конфигурацию сбора и снижение накладных расходов на edge.

2. Компетенции

- **ИД-3пк-10**, **ИД-4пк-10**

3. Теоретическое обоснование

- Архитектура Collector: **Receiver → Processor → Exporter**.
- Receiver: OTLP (gRPC/HTTP), Prometheus, Host Metrics.
- Processor: batch, memory_limiter, attributes (вставка drone_id).
- Exporter: Prometheus Remote Write, Loki, Jaeger/Tempo, Logging.

Схема:

```
[Simulator] --OTLP/gRPC--> [OTel Collector] --Prom RW--> [VictoriaMetrics]
                                |--Loki exporter--> [Loki]
                                |--OTLP--> [Tempo]
```

4. Задачи

1. Развернуть OTel Collector, настроить приём OTLP-метрик от симулятора дрона, экспорт в Prometheus и логирование.
2. Добавить атрибуты drone_id, firmware_version через processor attributes.
3. Настроить pipeline логов: парсинг JSON и отправка в Loki.
4. Проверить сквозную передачу трейсов от симулятора операций до Jaeger.

5. Варианты

Базовый (1–5)

1. OTel Collector с receiver OTLP (grpc), exporter logging.
2. Экспорт метрик в Prometheus и визуализация в Grafana.
3. Приём логов от симулятора через OTLP и вывод в stdout.
4. Настройка batch processor с таймаутом 2s.
5. Добавление атрибута environment=dev ко всем данным.

Средний (6–10)
6. Настроить attributes processor для извлечения drone_id из контекста и вставки во все сигналы.
7. Реализовать фильтрацию метрик: исключить process.* метрики дронов.
8. Обработка метрик hostmetrics приемником, экспорт в VictoriaMetrics.
9. Настроить memory_limiter processor для предотвращения OOM при пиковой нагрузке.
10. Интеграция OTel Collector с Kafka receiver для асинхронной доставки телеметрии.

Продвинутый (11–15)
11. Развернуть Collector как sidecar-контейнер с симулятором в одном поде Docker Compose.
12. Реализовать два pipeline: один для метрик с высокой частотой, второй для логов с низким приоритетом.
13. Настроить tail_sampling processor для трейсов, сохраняющий только ошибочные миссии.
14. Сконфигурировать OTel Collector Operator в Kubernetes, автоматически инжектирующий сбор телеметрии с подов дронов.
15. Имитация потери связи: буферизация данных на Collector'е через persistent queue (file storage extension) с последующей отправкой при восстановлении.

6. Пример выполнения (базовый 1)

otel-collector-config.yaml

```yaml
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
exporters:
  logging:
    loglevel: debug
  prometheusremotewrite:
    endpoint: http://victoriametrics:8428/api/v1/write
service:
  pipelines:
    metrics:
      receivers: [otlp]
      exporters: [logging, prometheusremotewrite]
```

Симулятор отправляет метрики через OTLP:

```python
from opentelemetry import metrics
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import OTLPMetricExporter
from opentelemetry.sdk.metrics import MeterProvider

exporter = OTLPMetricExporter(endpoint="otel-collector:4317", insecure=True)
provider = MeterProvider(metric_readers=[PeriodicExportingMetricReader(exporter)])
metrics.set_meter_provider(provider)
meter = metrics.get_meter(__name__)
battery = meter.create_gauge("drone_battery_percent")
battery.set(85, {"drone_id": "drone-01"})
```

7. Методические указания

- Использовать последнюю версию otel/opentelemetry-collector-contrib.
- Для Astra Linux проверить совместимость gRPC библиотек.
- Типичная ошибка: порт 4317 занят; изменить endpoint.

8. Контрольные вопросы

1. Какие преимущества OTel Collector перед разрозненными агентами?
2. Чем отличаются processor attributes и resource?
3. Зачем нужен batch processor?
4. Как организовать отказоустойчивый приём телеметрии с дронов?
5. Какие форматы сериализации поддерживает OTLP?
6. Можно ли обогатить телеметрию геоданными из внешнего API в Collector?
7. Что такое Connector в архитектуре Collector?
8. Как настроить шифрование (TLS) при передаче OTLP?

9. Какие существуют альтернативные протоколы для edge устройств кроме OTLP?
10. Как мониторить сам Collector?

. Требования к отчёту

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.
- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).
- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2
Качество отчёта и ответы	2	2	2
Бонус: симуляция	+1	+1	+1

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
телеметрии			

Лабораторная работа №6. Настройка алертинга и SLO-контроля для парка устройств

1. Цель работы

Научиться определять показатели уровня обслуживания (SLI), строить SLO, настраивать алерты на сжигание бюджета ошибок и управлять надёжностью парка автономных устройств.

2. Компетенции

- **ИД-3пк-10**

3. Теоретическое обоснование

- **SLI** – конкретная метрика (доступность дрона = drone_up / общее время).
- **SLO** – целевой уровень (99.9% доступности за месяц).
- **Error budget** – допустимое время недоступности.
- **Burn rate** – скорость расходования бюджета ошибок, позволяющая предсказывать нарушение SLO.
- **Multiwindow, Multi-Burn-Rate алерты** (по книге Google SRE).

4. Задачи

- Создать recording rules для SLI: drone:availability:ratio.

- Настроить алерты: мгновенный (burn rate > 14.4 за 1 час) и долгосрочный.
- Реализовать синтетический мониторинг (blackbox-проверку API диспетчера).
- Интегрировать Alertmanager для отправки в Telegram или OpsGenie.

5. Варианты

Базовый (1–5)

1. Алерт на падение drone_up == 0.
2. Алерт по уровню батареи < 20%.
3. Алерт на высокую задержку API диспетчера (p99 > 500ms).
4. Настройка Alertmanager с маршрутизацией по severity.
5. Создание Recording Rule job:drone_missions:rate5m.

Средний (6–10)
6. Вычислить SLI доступности дрона, настроить алерт на burn rate > 1 (fast burn).
7. Использовать Prometheus правило для прогноза остатка заряда (predict_linear) и алерт на аварийную посадку.
8. Настроить SLO-дашборд в Grafana с бюджетом ошибок и burn rate.
9. Реализовать multi-channel алертинг: warning в Telegram, critical в OpsGenie.
10. Добавить ингибирование алертов: при глобальном отказе связи не слать алерты отдельных дронов.

Продвинутый (11–15)
11. Создать сложный алерт на основе нескольких SLI (латентность + доступность) с комбинированным burn rate.
12. Интегрировать SLO-контроль в релизный пайплайн: автоматический откат, если бюджет ошибок превышен.
13. Настроить anomaly detection от VictoriaMetrics (vmanomaly) и алерты на аномалии в поведении вибрации.
14. Реализовать политику автоматического масштабирования парка на основе predict_linear бюджета ошибок.
15. Провести Game Day: симуляция аварии и проверка срабатывания алертов и реакции.

6. Пример выполнения (базовый 1)

prometheus_rules.yml

```yaml
groups:
  - name: drone_alerts
    rules:
      - alert: DroneDown
        expr: drone_up == 0
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Дрон {{ $labels.drone_id }} недоступен"
```

alertmanager.yml

```yaml
route:
  group_by: ['alertname']
  receiver: 'telegram'
receivers:
  - name: 'telegram'
    telegram_configs:
      - bot_token: 'YOUR_TOKEN'
        chat_id: -123456
```

7. Методические указания

- Проверить правильность for (период ожидания), чтобы избежать ложных срабатываний при кратковременных сбоях связи.
- При создании SLO оценить реалистичную доступность канала.

8. Контрольные вопросы

1. Чем SLI отличается от SLO и SLA?
2. Почему не рекомендуется алертить на мгновенное превышение SLO?
3. Что такое error budget и как он используется при принятии решения о релизе?
4. Какие бывают burn rate и соответствующие окна?
5. Как избежать флаппинг алертов?
6. Как в Alertmanager реализовать дедупликацию?
7. Что такое synthetics monitoring и зачем он нужен для парка БПЛА?
8. Как настроить алерт на основе прогноза (predict_linear)?
9. Как обеспечить алертинг, если сам Prometheus недоступен?
10. Как организовать эскалацию алертов при отсутствии реакции дежурного?

. Требования к отчёту

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.

- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).
- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация данных	2	2	2
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №7. Интеграция Observability в CI/CD конвейер (GitLab CI)

1. Цель работы

Внедрить контроль наблюдаемости на этапах CI/CD: автоматическая валидация конфигураций, quality gates на основе SLO метрик, сравнение canary-релизов.

2. Компетенции

- **ИД-4пк-10**, **ИД-7пк-10**

3. Теоретическое обоснование

- **Observability as Code**: конфигурации Prometheus, Grafana dashboards, алерты хранятся в Git и проверяются в CI.
- **GitLab CI стадии**: validate, test, deploy, verify.
- **Quality gates** – использование API Prometheus/Loki для проверки метрик и логов в процессе canary deployment.
- Сравнение baseline (стабильная версия) с canary.

4. Задачи

- Настроить GitLab репозиторий с инфраструктурным кодом observability.
- Добавить этап promtool check config и promtool test rules.

- Реализовать smoke-тесты, после которых проверяется отсутствие новых алертов.
- Создать скрипт сравнения распределения latency (Kolmogorov-Smirnov) между версиями симулятора.

5. Варианты

Базовый (1–5)

1. Проверка синтаксиса prometheus.yml с помощью promtool в CI.
2. Валидация dashboard JSON в Grafana (dashboard-linter).
3. Тестирование правил алертинга (promtool test rules).
4. Проверка Loki конфигурации через logcli или dry-run.
5. Автоматический деплой дашборда в Grafana API из пайплайна.

Средний (6–10)

6. Создание quality gate: после деплоя дрона-симулятора проверять, что drone_battery_percent не падает ниже ожидаемого.
7. Интеграция с VictoriaMetrics API для проверки, что ingestion rate не упал.
8. Проверка отсутствия новых ERROR логов за 5 мин после деплоя.
9. Сравнение трасс: средняя длительность flight plan не увеличилась более чем на 10%.
10. Использование GitHub Actions для аналогичной проверки.

Продвинутый (11–15)

11. Полный пайплайн canary-анализа: автоматический постепенный перевод трафика с проверкой SLO.
12. Интеграция Grafana Tempo trace comparison в пайплайн.
13. Создание custom GitLab CI шаблонов для observability quality gates.
14. A/B тестирование алгоритмов управления дроном с оценкой через телеметрию.
15. Внедрение OPA/Rego-правил для проверки compliance конфигураций алертов.

6. Пример выполнения (базовый 1)

.gitlab-ci.yml

```yaml
stages:
  - validate

prometheus-lint:
  stage: validate
  image: prom/prometheus:latest
  script:
    - promtool check config prometheus.yml
```

7. Методические указания

- При локальном тестировании GitLab Runner в Astra Linux убедиться в доступе к /var/run/docker.sock.
- Использовать docker-compose up -d для тестовых стендов в CI.

8. Контрольные вопросы

1. Почему наблюдаемость должна быть частью CI/CD?
2. Какие инструменты проверки Prometheus-правил вы знаете?
3. Что такое Quality Gates в контексте observability?
4. Как проверить, что после релиза нет регрессии в задержках?
5. Какова роль Grafana dashboard provisioning в GitOps?
6. Что такое PromQL offset и как его можно использовать для сравнения версий?
7. Можно ли в CI запустить нагрузочное тестирование и проверить SLO?
8. Какие метаданные ревизии (commit SHA) нужно прикреплять к метрикам?
9. Как откатить неудачный деплой на основе данных observability?
10. Как обеспечить безопасность доступа к API Prometheus из CI?

. Требования к отчёту

Отчёт в формате PDF, содержащий:

- Схему развёрнутой инфраструктуры.
- Листинги docker-compose.yml, конфигурационных файлов, кода симулятора.
- Скриншоты дашбордов Grafana с отображением метрик (включая историческую выборку из VictoriaMetrics).
- Проверку восстановления данных после перезапуска Prometheus.
- Ответы на контрольные вопросы.

10. Критерии оценивания

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
Развертывание инфраструктуры	2	2	2
Сбор и визуализация	2	2	2

Критерий	Базовый (макс. 7)	Средний (макс. 11)	Продвинутый (макс. 14)
данных			
Настройка алертинга и SLO-контроля	1	2	2
Корреляция данных	0	2	2
Программная инструментация	0	1	2
Интеграция в CI/CD	0	0	2
Качество отчёта и ответы	2	2	2
Бонус: симуляция телеметрии	+1	+1	+1

Лабораторная работа №8. Проектирование полностековой платформы наблюдаемости для роя дронов (комплексный проект)

1. Цель работы

Спроектировать и реализовать интегрированную платформу наблюдаемости для роя из 5–10 симулированных дронов, объединяющую метрики, логи, трассировку, алертинг и SLO-контроль, с демонстрацией реального состояния миссии.

2. Компетенции

- Все индикаторы ПК-10: ИД-3, ИД-4, ИД-7.

3. Теоретическое обоснование

- Архитектура платформы: Edge (симуляторы) → OTel Collector → Backend (Prometheus/Mimir, Loki, Tempo) → Grafana.
- Единый дашборд: карта с треками, индикаторы батарей, статус миссий, график бюджета ошибок, журнал событий.
- Многотенантность (разные клиенты-операторы) и разграничение доступа.

4. Задачи

1. Создать docker-compose, поднимающий рой из N дронов с уникальными ID и типами.
2. Развернуть полный стек observability с OTel Collector, VictoriaMetrics, Loki, Tempo, Alertmanager.
3. Написать сценарий миссии: взлёт, путь, RTL при низкой батарее.
4. Создать общий Grafana dashboard с переменной drone_id для фильтрации.
5. Настроить алерты на критические события и SLO-дашборд по доступности роя.
6. Провести демонстрацию: симуляция отказа одного дрона и анализ цепочки телеметрии.

5. Варианты

Базовый (1–5)

1. Рой из 3 одинаковых дронов с метриками и логами, общий dashboard.
2. Разные модели дронов (квадрокоптер, беспилотный автомобиль) с общим стеком.
3. Интеграция карты (Grafana Geomap) с треками дронов.
4. Использование Grafana "State Timeline" для миссий.
5. Настройка резервного копирования конфигураций в Git.

Средний (6–10)

6. Реализация многотенантности (Loki и Tempo с X-Scope-OrgID).
7. Создание автоматического скейлинга симуляторов через Makefile.
8. Настройка канареечного обновления прошивки дрона с проверкой телеметрии.
9. Интеграция OTel Collector с внешним Kafka-кластером для буферизации.
10. Разработка операторского пульта на основе Grafana с кнопками для отправки команд.

Продвинутый (11–15)

11. Полноценный проект с Kubernetes (minikube) и оператором Prometheus для роя.
12. Внедрение спайк-детектора на основе машинного обучения (vmanomaly) и автоматическое перепланирование миссии.
13. Создание цифрового двойника роя: данные поступают в Grafana, обратная связь через MQTT.
14. CI/CD пайплайн обновления конфигураций роя с автоматическим A/B тестированием.
15. Разработка "российской" витрины на "Графине" с дублированием ключевых индикаторов.

6. Пример выполнения (базовый 1)

Интеграция всех предыдущих работ. **docker-compose** включает 3 сервиса симуляторов, OTel Collector, VictoriaMetrics, Loki, Tempo, Alertmanager, Grafana. Ключевой dashboard: карта (Geomap) с треками, timeseries батарей, таблица алертов, Logs panel из Loki.

Фрагмент dashboard JSON (панель карты):

```
{
  "panels": [
    {
      "type": "geomap",
      "targets": [
        {
          "datasource": "Prometheus",
          "expr": "drone_latitude{drone_id=~\"$drone_id\"} and drone_longitude{drone_id=~\"$drone_id
\"}"
        }
      ]
    }
  ]
}
```

Переменная drone_id**:** label_values(drone_up, drone_id)

7. Методические указания

- Использовать переменные dashboard для переключения между дронами.
- Обратить внимание на суммарное потребление ресурсов (8 ГБ RAM минимум).

8. Контрольные вопросы

1. Какие компоненты обязательны для полностековой observability платформы?
2. Как обеспечить корреляцию метрик, логов, трейсов в едином UI?
3. Зачем нужна многотенантность в платформе управления парком?
4. Какие подходы к визуализации роя на карте вы знаете?
5. Как организовать secure доступ к Grafana через reverse proxy?
6. Как реализовать автоматическое обнаружение новых дронов (service discovery) в рое?
7. Что такое OpenTelemetry Semantic Conventions для устройств IoT?
8. Как построить алерты на уровне роя (групповые)?
9. Какие методы аномалий наиболее эффективны для вибраций?
10. Какие ограничения накладывает низкая пропускная способность канала связи с роем и как их учесть в архитектуре?

9. Требования к отчёту

Расширенный отчёт, включающий полный архитектурный документ, диаграммы последовательности, анализ покрытия наблюдаемости, предложения по усовершенствованию.

10. Критерии оценивания

Критерий	Макс. балл
Развёртывание роя и стека observability	3
Корректная работа метрик, логов, трейсов	3
Реализация SLO и алертов	2
Качество дашбордов и UX	2
Документация и воспроизводимость	2
Защита (ответы на вопросы)	2
Бонус (продвинутые функции)	+2