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

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Принципы и практики Observability» для студентов
направления подготовки 09.04.04 Программная инженерия
Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

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

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

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

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

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

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

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

ВВЕДЕНИЕ

Дисциплина «Принципы и практики Observability» является частью образовательной программы магистратуры 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем») и изучается во втором семестре первого курса. Она направлена на формирование профессиональной компетенции ПК-10: «Способен проектировать, реализовывать и администрировать системы централизованного управления парками автономных аппаратов, включая диспетчеризацию задач, мониторинг состояния, управление конфигурациями, массовые обновления "по воздуху" и интеграцию со сторонними сервисами».

Современные инженерные системы, в том числе парки беспилотных летательных аппаратов (БПЛА) и робототехнических комплексов, генерируют огромные объёмы телеметрии. Для обеспечения надёжности, безопасности и эффективности таких систем недостаточно классического мониторинга на основе метрик: требуется полноценная наблюдаемость (observability), позволяющая не только фиксировать состояние, но и оперативно исследовать причины аномалий, анализировать производительность и управлять жизненным циклом устройств.

Данная дисциплина фокусируется на углубленных практиках наблюдаемости, основываясь на знаниях, полученных в первом семестре (базовые понятия SLO/SLI, работа с Prometheus и Grafana, метрики надёжности, архитектура высоконагруженных систем). Исключая дублирование, курс развивает навыки работы со структурированными логами (Grafana Loki), распределённой трассировкой (Jaeger, Grafana Tempo), корреляцией разнородных данных (метрики, логи, трейсы), унифицированным сбором телеметрии с использованием OpenTelemetry, а также учитывает специфику сбора данных с автономных устройств.

Лабораторный практикум состоит из восьми работ, каждая из которых имеет единую структуру:

- цель и задачи;
- теоретическое обоснование и архитектурные схемы;
- пример практического выполнения;
- 15 вариантов индивидуальных заданий трёх уровней сложности (базовый, средний, повышенный);
- контрольные вопросы;
- требования к отчёту и критерии оценивания.

Выполнение лабораторных работ предполагает использование доступного на территории Российской Федерации программного обеспечения:

- сбор метрик: Prometheus, VictoriaMetrics, OpenTelemetry Collector;
- логи: Grafana Loki, OpenSearch;
- трассировка: Grafana Tempo, Jaeger, SigNoz;
- визуализация: Grafana OSS (с опциональным упоминанием российской «Графини»);
- инструментирование: OpenTelemetry SDK (Python/Go);
- контейнеризация: Docker, Docker Compose;
- CI/CD: GitLab CI / Git□ub □ctions;
- симуляция устройств: Python (эмуляторы дронов и роботов);
- операционные системы (опционально): □stra Linux, Ред ОС.

Каждая лабораторная работа последовательно погружает обучающегося в экосистему наблюдаемости: от развёртывания инфраструктуры сбора метрик (работа №1) и написания кастомных экспортёров (№2) до построения полностековой платформы для роя дронов (№8). Особое внимание уделяется сквозной корреляции телеметрии и автоматизации контроля качества на всех этапах жизненного цикла, включая интеграцию в CI/CD-конвейер.

Оценочные средства предусматривают дифференцированный подход: для получения оценки «удовлетворительно» достаточно выполнить все критерии базового уровня; «хорошо» – среднего; «отлично» – повышенного. В зависимости от полноты выполнения каждого уровня оценка может снижаться. Бонусные баллы начисляются за реализацию симуляции телеметрии с устройства (БПЛА).

Настоящее методическое пособие предназначено для студентов магистратуры, преподавателей и инженеров, специализирующихся в области Site Reliability Engineering и управления парками автономных систем, и призвано сформировать практические компетенции, востребованные в высокотехнологичных отраслях промышленности.

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

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.

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.

Теоретическое обоснование

1. Ключевые концепции

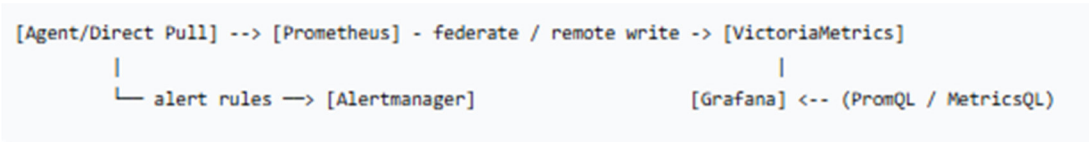
- Prometheus – pull-ориентированная TSDB, оптимальная для краткосрочного хранения и оперативных алертов.
- VictoriaMetrics – совместимая с Prometheus TSDB, обеспечивающая эффективное долговременное хранение, высокую скорость ingest и сжатия.

- Federation / Remote Write – механизмы объединения данных для централизации мониторинга распределённых площадок (аэродромов, базовых станций).
- Service Discovery – автоматическое обнаружение целей (дронов) через файл, DNS или □PI диспетчера.

2. Сравнение Pr□m□th□□s и Vict□riaM□trics

Характеристика	Prometheus	VictoriaMetrics
Хранилище	Локальное TSDB, ограниченное объёмом диска	Оптимизированное слияние партиций, эффективная компрессия
Масштабирование	Федерация, Thanos/Cortex	Кластерная версия (vminsert, vmselect, vmstorage)
Потребление R□M	Высокое (индексы в памяти)	Низкое (адаптивная работа с памятью)
Запись	Pull + Remote Write приёмник	Принимает Remote Write от Prometheus, агентов, OTel Collector

3. Архитектурная схема



Для эмуляции: скрипт на Python генерирует метрики в формате Prometheus на endpoint :8000/metrics.

Задачи работы

1. Развернуть Docker Compose со стеком: Prometheus, VictoriaMetrics, Grafana, lertmanager.
2. Создать симулятор одного БПЛА, отдающий телеметрию (координаты, высота, заряд батареи, статус).
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.

Задания для индивидуального выполнения

1. 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 lertmanager, 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).

2. 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.

5.3. 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 □□-пару Prometheus с синхронизацией алертов через □lertmanager 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)

6.1. Подготовка окружения

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"
```

*6.2. Конфигурация Pr□m□th□□s (*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']
```

*6.3. Симулятор дрона (*drone-sim/app.py*)*:

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

battery = Gauge('drone_battery_percent', 'Battery level')
altitude = Gauge('drone_altitude_meters', 'Current altitude')
longitude = Gauge('drone_longitude', 'Longitude')
latitude = Gauge('drone_latitude', 'Latitude')
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)
```

6.4. Результат

- Grafana: Data source Prometheus http://prometheus:9090 и VictoriaMetrics http://victoriametrics:8428.

Панели: drone_battery_percent, drone_altitude_meters, drone_up.

- При остановке Prometheus и перезапуске исторические метрики доступны через VictoriaMetrics data source.

7. Методические указания по выполнению

- При использовании □stra Linux / Ред ОС образы Docker работают стандартно; 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 R□M, следить за ресурсами ВМ (минимум 4 ГБ R□M).
- Типичная ошибка: забыть открыть порт 8428 в firewall (iptables) на □stra 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 отчету

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).
- Проверка 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

11. Материально-техническое обеспечение

- ОС: □stra Linux 1.7 / Ред ОС 7.3 / Ubuntu 22.04.

- Docker 24+, Docker Compose v2.
- Python 3.9+ (prometheus_client, psutil).
- Минимум 8 ГБ ОЗУ, 4 CPU.
- Образы: Prometheus, VictoriaMetrics, Grafana, python:3.11-slim.

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

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.

Формируемые компетенции и индикаторы

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

Теоретическое обоснование

1. Типы метрик

- Counter, Gauge, □istogram, Summary. Выбор типа зависит от характера параметра: заряд батареи – Gauge, количество успешных посадок – Counter, время манёвра – □istogram.
- Labels (drone_id, mission_type) позволяют агрегировать по парку.

2. Подходы

- Прямой □TTP-экспортёр (Prometheus client) – минимальные накладные расходы на edge.
- OpenTelemetry SDK – унификация, возможность отправлять метрики, логи, трейсы по одному протоколу (OTLP).

3. Схема

```
[Python Simulator] --/metrics (Pull)--> [Prometheus]
     или
[Python Simulator] --OTLP (Push)--> [OTel Collector] --> [Prometheus/VictoriaMetrics]
```

4. Задачи работы

1. Расширить симулятор дрона кастомными метриками (Counter миссий, istogram времени полёта).
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).

Варианты индивидуальных заданий

Базовый (1–5)

1. Добавить Gauge drone_speed и Counter mission_completed_total.
2. Создать istogram 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 □PI 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 P□4 через M□VLink.
15. Подключить VictoriaMetrics □nomaly Detection (через vmanomaly) к метрикам вибрации двигателя и выявлять деградацию.

Пример практического выполнения (базовый, вариант 1)

***drone_exporter.py*:**

```
from prometheus_client import start_http_server, Gauge, Counter
import time, random
speed = Gauge('drone_speed_mps', 'Current speed m/s')
missions = Counter('mission_completed_total', 'Completed missions')
...
speed.set(random.uniform(0,15))
missions.inc()
```

- avg(drone_speed_mps{drone_type="multirotor"})
- rate(mission_completed_total[5m])

Методические указания

Метки должны иметь низкую кардинальность; избегать drone_id как метку, если в парке >100 тысяч устройств.

Для тестирования гистограмм использовать sum(rate(maneuver_duration_seconds_bucket[1m])) в Prometheus.

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 □istogram 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?

Требования к отчету

1. Стандартные, плюс сравнение pull/push подходов, профилирование ресурсов экспортёра.
2. Критерии оценивания как в ЛР1.
3. МТО как в ЛР1, дополнительно OpenTelemetry SDK.

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

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

Компетенции: ИД-3пк-10.

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).

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.

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 (□-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 rduPilot, распарсив телеметрические сообщения.

15. Создать автомасштабируемый сбор логов с парка в 50 дронов с динамическими label.

Пример выполнения (базовый 1)

```
server:
  http_listen_port: 9080
positions:
  filename: /tmp/positions.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: {"timestamp":"...","drone_id":"drone-01","severity":"INFO","message":"takeoff"}. В Grafana добавляем Loki datasource, панель "Logs".

Методические указания

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

Контрольные вопросы

1. Зачем Loki не индексирует текст логов? Какие 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 аналогично ЛР1.

10. Критерии оценивания аналогично.

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

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.

Теоретическое обоснование

- Трассировка – сквозной контекст между сервисами. Span, Trace, контекст распространения (W3C Trace Context, Jaeger).
- Jaeger □ll-in-One удобен для разработки. Grafana Tempo использует объектное хранилище, масштабируется, требует OTel Collector.
- □ead-based sampling vs tail-based. В микросервисной среде flight-planner → drone-executor → telemetry-saver.

Задачи

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

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.

Пример выполнения (базовый 1)

docker-compose с jaeger-all-in-one:1.45. Запуск Flask: opentelemetry-instrument --traces_exporter jaeger_thrift --service_name flight-planner python app.py. В Jaeger UI видно трассу.

Методические указания

Использовать SDK opentelemetry-distro[otlp] для унификации. Проблема: не забыть установить переменные OTEL_E□PORTER_J□EGER_□GENT_□OST.

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?

Лабораторная работа №5: □penTelemetr□ □ollector: унифицированный сбор телеметрии с edge-устройств

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.

Формируемые компетенции

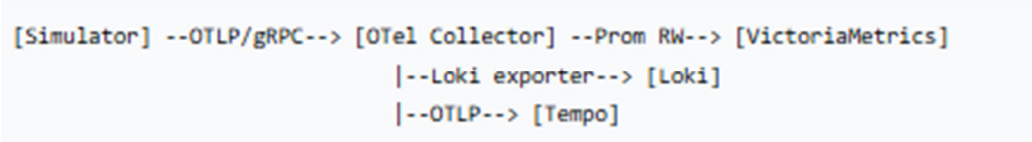
- ИД-3пк-10: Организует сбор потоковой телеметрии (координаты, высота, статус миссии) с симуляторов устройств.
- ИД-4пк-10: Разрабатывает конфигурацию Collector'a как код (Y□ML) для централизованного управления настройками сбора.

Теоретическое обоснование

3.1. Архитектура C□ll□ct□r: R□c□iv□r → Pr□c□ss□r → Exp□rt□r.

- Receiver: OTLP (gRPC/□TTP), Prometheus, □ost Metrics.
- Processor: batch, memory_limiter, attributes (вставка drone_id).
- Exporter: Prometheus Remote Write, Loki, Jaeger/Tempo, Logging.
- Pipeline могут быть раздельными: metrics, logs, traces.

3.2. Схема



Задачи работы

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.

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.

Пример практического выполнения (базовый 1)

otel-collector-config.□aml:

```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 данные через opentelemetry.exporter.otlp.proto.grpc.metric_exporter. Результат: в логах collector'а видны принятые метрики, в Grafana – точки.

Методические указания

- Использовать последнюю версию otel/opentelemetry-collector-contrib.

- Для □stra Linux проверить совместимость gRPC библиотек.
- Типичная ошибка: порт 4317 занят; изменить endpoint.

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 □PI 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?

Лабораторная работа №6: Настройка алертинга и SL□-контроля для парка устройств

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

Компетенции: ИД-3пк-10.

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).

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 PI диспетчера).
- Интегрировать lertmanager для отправки в Telegram или OpsGenie.

Варианты

Базовый (1–5)

1. Алерт на падение drone_up == 0.
2. Алерт по уровню батареи < 20%.
3. Алерт на высокую задержку PI диспетчера (p99 > 500ms).
4. Настройка lertmanager 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.

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 с receiver 'telegram'. Сообщение приходит при остановке симулятора.

Методические указания

- Проверить правильность for (период ожидания), чтобы избежать ложных срабатываний при кратковременных сбоях связи.

- При создании SLO оценить реалистичную доступность канала.

Контрольные вопросы

1. Чем SLI отличается от SLO и SL□?
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 □lertmanager 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?

Лабораторная работа №7: Интеграция □bservabilit□в □I/□D конвейер (GitLab □I)

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

Компетенции: ИД-4пк-10, ИД-7пк-10.

Теоретическое обоснование

- Концепция "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 □PI 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.

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 PI 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 PI 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 Git ub ctions 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. □/B тестирование алгоритмов управления дроном с оценкой через телеметрию.

15. Внедрение OP□/Rego-правил для проверки compliance конфигураций алертов.

Пример выполнения (базовый 1)

.gitlab-ci.□ml:

```
stages:
  - validate

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

В репозитории лежит prometheus.yml. При пуше происходит проверка, ошибка приводит к failed pipeline.

Методические указания

- При локальном тестировании GitLab Runner в □stra 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 S□□) нужно прикреплять к метрикам?
9. Как откатить неудачный деплой на основе данных observability?
10. Как обеспечить безопасность доступа к □PI Prometheus из CI?

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

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.

Компетенции и индикаторы

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

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, lertmanager.
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.

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 -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 □/B тестированием.

15. Разработка "российской" витрины на "Графине" с дублированием ключевых индикаторов.

Пример выполнения (базовый 1)

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

Фрагмент dashb□ard JSON:

```
{
  "panels": [
    {
      "type": "geomap",
      "targets": [
        {
          "datasource": "Prometheus",
          "expr": "drone_latitude and drone_longitude"
        }
      ]
    }
  ]
}
```

Методические указания

Использовать переменные dashboard для переключения между дронами.

Обратить внимание на суммарное потребление ресурсов (8 ГБ R□M минимум).

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?

Требования к отчету

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

10. Критерии оценивания (аналогично, с повышенными весами за интеграцию).

11.МТО Мощный сервер/ПК (16 ГБ R□M, 8 CPU), Docker, □stra Linux/Ред ОС, VPN/доступ к реестру образов.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации и проведению самостоятельной работы по дисциплине «Принципы и практики Observability»

для студентов направления подготовки

09.04.04 Программная инженерия Направленность (профиль)

«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цель самостоятельной работы

1.2. Общая характеристика комплексного проекта

1.3. Формируемые компетенции и индикаторы

1.4. Трудоёмкость самостоятельной работы

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы для самостоятельного изучения

2.2. Рекомендуемая литература

2.3. Рекомендации по работе с литературой

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Структура лабораторного практикума

3.2. Порядок выполнения лабораторных работ

3.3. Рекомендации по подготовке к лабораторным работам

3.4. Рекомендации по выполнению комплексного проекта (ЛР №8)

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

5.2. Критерии оценивания комплексного проекта (ЛР №8)

5.3. Шкала перевода процента баллов в оценку

ВВЕДЕНИЕ

Методические указания для самостоятельной работы по дисциплине **«Принципы и практики □bservabilit□»** предназначены для студентов магистратуры, обучающихся по направлению подготовки **09.04.04 «Программная инженерия»** (профиль **«Разработка, мониторинг и управление жизненным циклом инженерных систем»**).

Дисциплина «Принципы и практики Observability» является одной из ключевых дисциплин вариативной части образовательной программы и направлена на формирование у магистрантов системных знаний и практических навыков в области обеспечения наблюдаемости распределённых программных систем, включая сбор, агрегацию, анализ и визуализацию телеметрических данных (метрик, логов, трейсов), а также интеграцию практик наблюдаемости в CI/CD-конвейеры для обеспечения качества, производительности и надёжности разрабатываемого программного обеспечения и парков автономных устройств.

Дисциплина реализуется во 2-м семестре 1-го курса и опирается на знания, полученные в 1-м семестре при изучении дисциплин:

- «Проектирование и оптимизация конвейеров CI/CD» (базовые принципы CI/CD, пайплайны);
- «Надежность и устойчивость систем» (метрики SLO/SLI, основы мониторинга Prometheus+Grafana);
- «Архитектура и проектирование высоконагруженных и отказоустойчивых систем» (микросервисная архитектура, распределённые системы).

Настоящие методические указания разработаны в соответствии с:

- Федеральным государственным образовательным стандартом высшего образования по направлению подготовки 09.04.04 «Программная инженерия»;
- Учебным планом направления подготовки 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»);
- Рабочей программой дисциплины «Принципы и практики Observability»;
- Положением об организации образовательного процесса на основе рейтинговой системы оценки знаний студентов в ФГАОУ ВО «СКФУ».

Целью самостоятельной работы является закрепление и углубление теоретических знаний о наблюдаемости (observability) распределённых систем, полученных на лекционных занятиях, а также формирование практических навыков развертывания инфраструктуры сбора телеметрии, инструментации приложений, настройки алертинга и интеграции observability в CI/CD-пайплайны.

Задачи самостоятельной работы:

В процессе выполнения самостоятельной работы студент должен решить следующие задачи:

1. Изучить теоретический материал по темам дисциплины в соответствии с учебным планом (8 тем лекций).
2. Подготовиться к выполнению лабораторных работ (№1–7) и осмыслению их результатов.
3. Выполнить комплексный проект по проектированию и реализации платформы наблюдаемости для парка автономных устройств (БПЛА/роботов) с обоснованием всех решений (Лабораторная работа №8).
4. Сформировать комплексный отчёт, включающий архитектурные схемы, дашборды, результаты анализа трасс и логов, а также защиту проекта.
5. Подготовиться к промежуточной аттестации (зачёту с оценкой) по дисциплине.

6. Развить навыки самостоятельного поиска и анализа научно-технической информации в области observability и SRE.

Настоящие методические указания призваны помочь студентам в организации самостоятельной работы по дисциплине «Принципы и практики Observability». Систематическое выполнение всех видов самостоятельной работы, своевременная подготовка к лабораторным занятиям и выполнение комплексного проекта позволят успешно освоить дисциплину и сформировать необходимые профессиональные компетенции.

1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1.1. Цель самостоятельной работы

Целью самостоятельной работы является закрепление и углубление теоретических знаний, полученных на лекциях, а также формирование практических навыков проектирования, развертывания и администрирования платформ наблюдаемости для высоконагруженных распределённых систем и парков автономных устройств.

1.2. Общая характеристика комплексного проекта

Комплексный проект выполняется студентом **индивидуально** в течение всего семестра параллельно с изучением дисциплины и выполнением лабораторных работ (№1–7). Защита проекта проводится на **8-м лабораторном занятии** (заключительном).

Тема проекта: «Проектирование и реализация платформы наблюдаемости для парка автономных устройств (БПЛА / робототехнических комплексов)»

Результат проекта: Комплексный отчёт, включающий:

- Архитектурную схему платформы наблюдаемости (C4 model - уровни Context и Containers)
- Конфигурационные файлы для развертывания стека (Docker Compose / Kubernetes)

- Дашборды в Grafana для визуализации телеметрии (метрики, логи, трассы)
- Настроенные правила алертинга и SLO-контроля
- Примеры трасс и логов с корреляцией данных
- Презентацию для защиты (10–12 слайдов)

1.3. Формируемые компетенции и индикаторы

Компетенция	**Индикаторы**
ПК-10. Способен проектировать, реализовывать и администрировать системы централизованного управления парками автономных аппаратов, включая диспетчеризацию задач, мониторинг состояния, управление конфигурациями, массовые обновления "по воздуху" и интеграцию со сторонними сервисами	**ИД-3** Организует сбор, обработку, хранение и визуализацию потоковой телеметрии устройств в реальном времени с системой оповещения об аномалиях.
	ИД-4 Разрабатывает систему централизованного управления конфигурациями устройств с контролем версий и автоматическим применением настроек.
	ИД-7 Администрирует инфраструктуру платформы управления, обеспечивая её бесперебойную работу, масштабирование и защиту от НСД.

1.4. Трудоёмкость самостоятельной работы

Вид работы	**Часы**
Изучение теоретического материала (8 тем)	16
Подготовка к лабораторным работам (№1–7)	28
Выполнение комплексного проекта (Лабораторная работа №8)	22
Подготовка к зачёту с оценкой	8
Итого	**74** *(в соответствии с РПД)*

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы для самостоятельного изучения

№	Тема	Литература	Часы
1	От мониторинга к Observability: три столпа наблюдаемости (Metrics, Logs, Traces)	[1, гл. 1-2]	2
2	Структурированные логи и управление событиями (Loki, OpenSearch)	[1, гл. 3]	2
3	Распределенная трассировка (Distributed Tracing): Trace, Span, Context Propagation	[2, гл. 4-5]	2

№	Тема	Литература	Часы
4	Jaeger и Tempo: визуализация и анализ трасс	[2, гл. 6]	2
5	Корреляция данных: связь метрик, логов и трейсов (Golden Signals, Correlation ID)	[2, гл. 7]	2
6	OpenTelemetry: унифицированный стандарт сбора телеметрии	[3]	2
7	Observability для парков автономных устройств (Fleet Management)	[4]	2
8	SRE-практики на основе Observability: SLI/SLO, Error Budget, автоматическое реагирование	[5]	2
Итого			**16**

2.2. Рекомендуемая литература

Основная литература

1. Majors, C. Observability Engineering: □chieving Production Excellence / C. Majors, L. Fong-Jones, G. Miranda. — O'Reilly Media, 2022. — 384 с. *(Доступно по лицензии CC BY-ND:* https://github.com/observability-engineering*)*

2. Beyer, B. Site Reliability Engineering: □ow Google Runs Production Systems / B. Beyer, C. Jones, J. Petoff, N. R. Murphy. — O'Reilly Media, 2016. — 552 с. *(Доступно по лицензии CC BY-NC-ND:* https://sre.google/books*)*

3. OpenTelemetry Documentation. — Режим доступа: https://opentelemetry.io/docs/ (свободный)

4. Grafana Loki Documentation. — Режим доступа: https://grafana.com/docs/loki/ (свободный)

5. Jaeger Documentation. — Режим доступа: https://www.jaegertracing.io/docs/ (свободный)

Дополнительная литература

6. Ньюмен, С. Создание микросервисов / С. Ньюмен. — 2-е изд. — СПб.: Питер, 2024. — 622 с.

7. Клеппман, М. Проектирование систем, интенсивно работающих с данными / М. Клеппман. — Электронное издание (перевод сообщества Vonng). — Режим доступа: https://github.com/Vonng/ddia

2.3. Рекомендации по работе с литературой

- При изучении теоретического материала рекомендуется вести конспект в виде структурированных заметок с выделением ключевых терминов (Trace, Span, Context Propagation, Correlation ID, Golden Signals и др.).
- Для каждой темы составить глоссарий (не менее 5 терминов).
- Выполнять практические примеры из литературы в тестовой среде (Docker, Docker Compose).
- Использовать официальную документацию как основной источник для выполнения лабораторных работ.
- При работе с российскими ОС (□stra Linux, Ред ОС) проверять совместимость образов Docker и сетевых настроек.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Структура лабораторного практикума

Лабораторный практикум состоит из **8 работ**, объединённых в 3 тематических блока:

Блок	**№ ЛР**	**Тема**	**Компетенции**

Блок	№ ЛР	Тема	Компетенции
Блок 1. Основы сбора и визуализации данных		Развертывание инфраструктуры мониторинга на базе Prometheus и VictoriaMetrics	ИД-7пк-10, ИД-3пк-10
		Реализация кастомных метрик для отслеживания состояния автономных устройств	ИД-3пк-10
Блок 2. Логи и распределенная трассировка		Централизованный сбор и анализ логов с Loki и Promtail	ИД-3пк-10
		Распределенная трассировка микросервисного приложения (Jaeger/Tempo)	ИД-3пк-10
Блок 3. Унификация и автоматизация		OpenTelemetry Collector: унифицированный сбор телеметрии с edge-устройств	ИД-3пк-10 ИД-4пк-10
		Настройка алертинга и SLO-контроля для парка устройств	ИД-3пк-10
		Интеграция Observability в CI/CD конвейер (GitLab CI)	ИД-4пк-10, ИД-7пк-10

Блок	№ ЛР	Тема	Компетенции
		Проектирование полностековой платформы наблюдаемости для роя дронов (комплексный проект)	Все индикаторы ПК-10

3.2. Порядок выполнения лабораторных работ

1. Ознакомиться с теоретическим обоснованием работы из методических указаний к лабораторным работам.
2. Выбрать уровень сложности (базовый / средний / продвинутый) по согласованию с преподавателем.
3. Выбрать вариант задания в соответствии с номером, выданным преподавателем.
4. Разработать решение (код, конфигурации, docker-compose, дашборды).
5. Выполнить проверку в тестовой среде (Docker, локальный кластер).
6. Оформить отчёт по установленной форме (Markdown/PDF).
7. Защитить работу перед преподавателем (демонстрация работающих дашбордов, трасс, логов + ответы на вопросы).

3.3. Рекомендации по подготовке к лабораторным работам

- Изучить соответствующий раздел теоретического материала перед выполнением работы.
- Подготовить тестовую среду (установленные Docker, Docker Compose, Python 3.9+, Git, браузер).
- Ознакомиться с примерами выполнения из методических указаний по лабораторным работам.

- При возникновении вопросов - консультироваться с преподавателем.
- Для работ с симуляцией БПЛА заранее ознакомиться с принципами работы эмуляторов.

3.4. Рекомендации по выполнению комплексного проекта (ЛР №8)

Комплексный проект выполняется на протяжении всего семестра параллельно с лабораторными работами. Каждая лабораторная работа добавляет новый артефакт в проект.

Требования к проекту:

- Выбрать сценарий: парк автономных устройств (3+ симулированных дрона) или микросервисная система с 3+ сервисами.
- Развернуть полный стек observability: Prometheus/VictoriaMetrics, Loki, Tempo/Jaeger, Grafana.
- Настроить OpenTelemetry Collector для унифицированного сбора телеметрии.
- Реализовать симулятор устройства (Python), отправляющий метрики, логи и трассы.
- Создать дашборд в Grafana для визуализации состояния парка устройств (карта с треками, индикаторы батарей, статус миссий).
- Настроить алерты на аномалии (потеря связи, превышение порогов).
- Подготовить презентацию (10–12 слайдов) и защитить проект.

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

1. Чем наблюдаемость (observability) отличается от мониторинга?
2. Какие три столпа наблюдаемости выделяют? Охарактеризуйте каждый.
3. Что такое структурированное логирование? Какие форматы используются?
4. В чём разница между Loki и OpenSearch для сбора логов?

5. Что такое Trace, Span и Context Propagation?

6. Как работает распределенная трассировка в микросервисной архитектуре?

7. Какие инструменты распределенной трассировки вы знаете? Сравните Jaeger и Tempo.

8. Что такое Correlation ID и как он связывает логи и трейсы?

9. Что такое Golden Signals? Перечислите их.

10. Для чего нужен OpenTelemetry Collector? Каковы его основные компоненты (Receiver, Processor, Exporter)?

11. Как работает OTLP-протокол?

12. Что такое SLI, SLO, SL□ и Error Budget?

13. Как рассчитать Error Budget на основе SLO?

14. Как настроить алертинг в Prometheus на основе SLO?

15. Какие особенности сбора телеметрии с автономных устройств (БПЛА/роботов)?

16. Какие протоколы используются для передачи телеметрии с edge-устройств?

17. Как можно коррелировать данные от разных устройств в одном дашборде?

18. Как интегрировать observability в CI/CD пайплайн?

19. Что такое Quality Gate на основе observability-данных?

20. Как провести blameless postmortem с использованием трасс и логов?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

Оценка	**Критерии**
5 (отлично)	Работа выполнена в полном объёме,

Оценка	Критерии
	код/конфигурации работают без ошибок, студент демонстрирует понимание всех этапов, отвечает на дополнительные вопросы, отчёт оформлен по требованиям, дашборды и трассы соответствуют заданию.
4 (хорошо)	Работа выполнена полностью, но есть небольшие замечания (неоптимальная конфигурация, неполный отчёт). Студент отвечает на большинство вопросов.
3 (удовлетворительно)	Работа выполнена с помощью преподавателя, есть ошибки, которые студент исправляет с наводящими вопросами. Отчёт оформлен минимально.
2 (неудовлетворительно)	Работа не выполнена или выполнена неверно, студент не понимает принципов наблюдаемости, не может объяснить свои действия.

5.2. Критерии оценивания комплексного проекта (ЛР №8)

Критерий	Вес, %	5 (отлично)	4 (хорошо)	3 (удовлетворительно)	2 (неудовлетворительно)
Полнота	20	Prometheus + Loki + Tempo	3 из 4 компон	2 компонента,	Менее 2 компонентов

Критерий	Вес, %	5 (отлично)	4 (хорошо)	3 (удовлетворительно)	2 (неудовлетворительно)
реализации стека observability		+ Grafana, полная корреляция	ентов, корреляция частичная	корреляции нет	
Сбор телеметрии с устройств (3+)	20	3+ устройства, метрики+логи+трейсы	2 устройства, неполные данные	1 устройство, только метрики	Нет устройств
Дашборды в Grafana	5	4+ панели, наличие карты/списка устройств, корреляция	3 панели, корреляции нет	2 панели, минимально	Нет дашбордов
Алертинг и SLO	5	Алерты на аномалии, SLO-дашборд, error budget	Алерты есть, SLO не рассчитан	Только один алерт	Нет алертов
Качест		Отчёт	Отчёт с	Отчёт	Отчёт

Критерий	Вес, %	5 (отлично)	4 (хорошо)	3 (удовлетворительно)	2 (неудовлетворительно)
во отчёта и презентации	5	полный, презентация готова, скриншоты есть	небольшими недочётами	минимальный, презентация отсутствует	отсутствует
Ответы на вопросы защиты	5	Уверенные, полные ответы, понимание trade-offs	Хорошие ответы, неполные обоснования	Минимальные ответы, нет понимания	Нет ответов

7. ЗАКЛЮЧЕНИЕ

Данные методические указания призваны помочь студентам в организации самостоятельной работы по дисциплине «Принципы и практики Observability». Систематическое выполнение всех видов самостоятельной работы, своевременная подготовка к лабораторным занятиям и выполнение комплексного проекта позволят успешно освоить дисциплину и сформировать необходимые профессиональные компетенции в области обеспечения наблюдаемости распределенных систем и парков автономных устройств.