

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 «Надежность и устойчивость систем» для студентов направления
подготовки 09.04.04 Программная инженерия
Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных систем»

Ставрополь 2026

Оглавление

ЛАБОРАТОРНАЯ РАБОТА №1. Расчет базовых метрик надежности (MTBF, MTTR, MTTF, Availability).....	11
ЛАБОРАТОРНАЯ РАБОТА №2. Определение SLO и SLI для микросервиса. Расчет и управление Error Budget.....	21
ЛАБОРАТОРНАЯ РАБОТА №3. Политики управления Error Budget и принятие решений.....	33
ЛАБОРАТОРНАЯ РАБОТА №4	49
Настройка мониторинга и визуализации (Prometheus + Grafana)	49
ЛАБОРАТОРНАЯ РАБОТА №5	65
Проведение Blameless Postmortem (метод 5 Whys, диаграмма Исикавы, корректирующие меры).....	65
ЛАБОРАТОРНАЯ РАБОТА №6	79
Root Cause Analysis с использованием диаграмм Исикавы (fishbone)	79
ЛАБОРАТОРНАЯ РАБОТА №7	93
Документирование инцидентов и распространение уроков.....	93
ЛАБОРАТОРНАЯ РАБОТА №8	116
Проектирование эксперимента по проверке устойчивости.....	116
ЛАБОРАТОРНАЯ РАБОТА №9	135
Проведение А/В теста для проверки устойчивости (Canary Deployment)	135
ЛАБОРАТОРНАЯ РАБОТА №10. Настройка Chaos Engineering платформы (ChaosMesh)	149
ЛАБОРАТОРНАЯ РАБОТА №11. Проведение Game Day и автоматизация chaos-тестов.....	164

ЛАБОРАТОРНАЯ РАБОТА №12. Создание автоматизированных runbooks и playbooks.....	179
ЛАБОРАТОРНАЯ РАБОТА №13. Реализация ChatOps для инцидент-менеджмента.....	194
ЛАБОРАТОРНАЯ РАБОТА №14	210
Self-healing механизмы и graceful degradation.....	210
ЛАБОРАТОРНАЯ РАБОТА №15. Реализация архитектурных паттернов отказоустойчивости	229
ЛАБОРАТОРНАЯ РАБОТА №16	247
Нагрузочное тестирование для проверки устойчивости (k6).....	247
ЛАБОРАТОРНАЯ РАБОТА №17	261
Специализация: Надежность БПЛА и робототехнических систем	261
ЛАБОРАТОРНАЯ РАБОТА №18	277
Комплексный проект по надежности (сквозной SRE-пайплайн).....	277

ВВЕДЕНИЕ

Лабораторный практикум по дисциплине «Надежность и устойчивость систем» является неотъемлемой частью подготовки магистров по направлению 09.04.04 «Программная инженерия» (профиль «Разработка, мониторинг и управление жизненным циклом инженерных систем»).

Дисциплина относится к части, формируемой участниками образовательных отношений (вариативная часть), и является одной из ключевых дисциплин профиля. Лабораторный практикум обеспечивает практическую реализацию теоретических положений, изложенных в лекционном курсе, и формирует у магистрантов навыки применения методологии SRE (Site Reliability Engineering) для обеспечения надежности и устойчивости сложных программных и программно-аппаратных систем.

Основная цель лабораторного практикума - формирование у магистрантов практических навыков:

- количественного измерения и управления надежностью сервисов через SLO, SLI и Error Budget;
- настройки систем мониторинга и визуализации (Prometheus + Grafana);
- проведения анализа инцидентов по методологии blameless postmortem с применением методов 5 Whys и диаграмм Исикавы;
- проектирования и проведения Chaos Engineering экспериментов для проверки устойчивости систем;
- автоматизации процессов реагирования на инциденты (runbooks, ChatOps, self-healing);
- реализации архитектурных паттернов отказоустойчивости (Circuit Breaker, Retry, Bulkhead);

- проведения нагрузочного тестирования для проверки устойчивости (k6);
- применения методов SRE и Chaos Engineering для обеспечения надежности автономных систем (БПЛА).

В ходе выполнения лабораторных работ студент решает следующие задачи:

№	Задача	Формируемые компетенции
1	Рассчитывать базовые метрики надежности (MTBF, MTTR, Availability)	ПК-4
2	Определять SLO, SLI и управлять Error Budget	ПК-4
3	Настраивать мониторинг и визуализацию (Prometheus + Grafana)	ПК-4
4	Проводить анализ инцидентов (blameless postmortem, 5 Whys, fishbone)	ПК-6
5	Разрабатывать runbooks и документировать корректирующие меры	ПК-6
6	Проектировать и проводить Chaos Engineering эксперименты	ПК-5
7	Реализовывать canary-развертывание и feature flags	ПК-3
8	Настраивать автоматическое реагирование (ChatOps, self-healing)	ПК-6
9	Реализовывать архитектурные паттерны отказоустойчивости	ПК-5
10	Проводить нагрузочное тестирование для проверки устойчивости	ПК-5
11	Применять методы SRE для обеспечения надежности БПЛА	ПК-4, ПК-5

Лабораторный практикум состоит из 18 лабораторных работ, объединённых в 6 тематических блоков:

Блок	№ ЛР	Тема	Компетенции
Блок 1. Основы метрик и измерений	1-4	MTBF/MTTR, SLO/SLI, Error Budget, Prometheus+Grafana	ПК-4
Блок 2. Анализ инцидентов	5-7	Postmortem, 5 Whys, fishbone, документирование	ПК-6
Блок 3. Эксперименты и Chaos Engineering	8-11	Проектирование, canary, ChaosMesh, Game Day	ПК-5
Блок 4. Автоматизация реагирования	12-14	Runbooks, ChatOps, Self-healing	ПК-6
Блок 5. Архитектурные паттерны	15-16	Circuit Breaker, Retry, нагрузочное тестирование	ПК-5
Блок 6. Специализация и итоговый проект	17-18	Надежность БПЛА, сквозной SRE-пайплайн	ПК-4, ПК-5, ПК-6

ТРЕБОВАНИЯ К ПОДГОТОВКЕ И ВЫПОЛНЕНИЮ

1. Предварительная подготовка

Перед выполнением лабораторных работ студент должен:

1. Изучить соответствующие разделы лекционного курса и рекомендованной литературы.
2. Ознакомиться с теоретическим обоснованием лабораторной работы.

3. Установить необходимое программное обеспечение.
4. Зарегистрироваться на выбранной CI/CD-платформе (GitLab / GitHub).

2. Порядок выполнения

1. Ознакомиться с заданием и выбрать вариант (уровень 1, 2 или 3).
2. Разработать решение (скрипты, конфигурации, отчёты).
3. Выполнить проверку в тестовой среде.
4. Оформить отчёт в соответствии с требованиями.
5. Защитить работу перед преподавателем (демонстрация работы, ответы на вопросы).

3. Уровни сложности заданий

Каждая лабораторная работа содержит 15 вариантов заданий, разделённых на три уровня сложности:

Уровень	Описание	Рекомендации
Уровень 1	Выполнение основных требований работы	Для всех студентов
Уровень 2	Дополнительные требования (анализ, оптимизация)	Для студентов, желающих углубить знания
Уровень 3	Программная реализация на Python, интеграция с внешними системами	Для студентов, ориентированных на научную работу

ОТЧЁТНОСТЬ ПО ЛАБОРАТОРНЫМ РАБОТАМ

1. Структура отчёта

Каждая лабораторная работа оформляется в виде электронного отчёта, содержащего следующие разделы:

1. Титульный лист (ФИО, группа, вариант, номер работы).
2. Цель работы (формулируется на основе задания).
3. Теоретическое обоснование (краткое описание ключевых понятий).
4. Практическая часть (листинги кода, конфигурации, скриншоты).
5. Результаты выполнения (скриншоты, логи, отчёты).
6. Анализ результатов (выводы, рекомендации).
7. Ответы на контрольные вопросы (3-5 вопросов).
8. Вывод (достигнута ли цель).

6.2. Требования к оформлению

- Отчёт оформляется в текстовом редакторе (Word, Google Docs) или Markdown.
- Листинги кода должны быть отформатированы (моноширинный шрифт, подсветка синтаксиса).
- Скриншоты должны быть чёткими, с поясняющими подписями.
- Места для скриншотов обозначены в тексте работы в виде:

РЕКОМЕНДУЕМОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

1. Основное ПО

№	Наименование ПО	Назначение	Лицензия
1	GitLab / GitHub	CI/CD платформа, хранение кода	Бесплатная
2	Docker CE	Контейнеризация приложений	Бесплатная
3	Minikube / Kind	Локальный Kubernetes кластер	Бесплатная

4	Prometheus	Сбор метрик	Apache 2.0
5	Grafana	Визуализация дашбордов	AGPL-3.0
6	ChaosMesh	Chaos Engineering	Apache 2.0
7	k6	Нагрузочное тестирование	AGPL-3.0
8	Python 3.x	Язык программирования	PSF
9	Visual Studio Code	Редактор кода	Бесплатная

2. Российские альтернативы

Исходный инструмент	Российская альтернатива	Примечание
Grafana	«Графиня»	В реестре отечественного ПО
GitHub / GitLab	GitFlic	Российская DevOps-платформа
Jira (для ЛР №7)	YouTrack, Kaiten	Доступны бесплатные версии

8. КРИТЕРИИ ОЦЕНКИ ЛАБОРАТОРНЫХ РАБОТ

Оценка лабораторной работы складывается из следующих компонентов:

Критерий	Вес
Выполнение задания в полном объёме	40%
Качество оформления отчёта	20%
Правильность ответов на контрольные вопросы	20%
Демонстрация работы (скриншоты, логи)	20%

Шкала оценки лабораторной работы:

Оценка	Критерии

5 (отлично)	Задание выполнено полностью, пайплайн работает без ошибок, отчёт оформлен по требованиям, студент уверенно отвечает на вопросы
4 (хорошо)	Задание выполнено полностью, но есть небольшие замечания (неоптимальная конфигурация, неполный отчёт)
3 (удовлетворительно)	Задание выполнено с помощью преподавателя, есть ошибки, исправляемые с наводящими вопросами
2 (неудовлетворительно)	Задание не выполнено или выполнено неверно, студент не понимает принципов работы

ПРАВИЛА ТЕХНИКИ БЕЗОПАСНОСТИ

При выполнении лабораторных работ, связанных с работой на компьютере и в облачных средах, необходимо соблюдать следующие правила:

1. Не производить самостоятельного вскрытия, ремонта или модификации аппаратного обеспечения.
2. Соблюдать правила работы с электроприборами.
3. При работе с облачными ресурсами не передавать учётные данные (токены, пароли) третьим лицам.
4. Не использовать реальные боевые ключи доступа в учебных целях.
5. Соблюдать лицензионные соглашения используемого программного обеспечения.
6. При работе с контейнерами и Kubernetes не превышать выделенные ресурсы.

Лабораторный практикум по дисциплине «Надежность и устойчивость систем» является важнейшей составляющей подготовки магистров в области

Site Reliability Engineering. Последовательное выполнение всех 18 лабораторных работ позволит:

- освоить современные методы управления надежностью распределенных систем;
- приобрести практические навыки работы с инструментами мониторинга, Chaos Engineering и автоматизации;
- сформировать системное мышление, необходимое для обеспечения отказоустойчивости сложных программно-аппаратных комплексов;
- подготовиться к решению реальных задач в области эксплуатации высоконагруженных и критически важных систем.

ЛАБОРАТОРНАЯ РАБОТА №1. Расчет базовых метрик надежности (MTBF, MTTR, MTTF, Availability)

1. ЦЕЛЬ РАБОТЫ: Сформировать практические навыки расчета основных метрик надежности программно-аппаратных систем (MTBF, MTTR, MTTF, Availability) двумя способами: аналитическим (с использованием Excel/калькулятора) и программным (с использованием Python).

2. ЗАДАЧИ РАБОТЫ

1. Изучить основные метрики надежности: MTBF (Mean Time Between Failures), MTTR (Mean Time To Repair), MTTF (Mean Time To Failure), Availability (коэффициент доступности).
2. Научиться рассчитывать метрики аналитическим способом на основе статистических данных об отказах и восстановлении системы.
3. Научиться рассчитывать метрики программным способом с использованием Python (библиотеки pandas, matplotlib).

4. Освоить методы анализа трендов доступности системы во времени.
5. Сравнить результаты, полученные двумя способами, и сделать вывод о достоверности вычислений.
6. Оформить отчет с результатами расчетов, скриншотами и выводами.

3. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

3.1. Основные понятия надежности

Надежность - свойство системы сохранять во времени способность выполнять требуемые функции в заданных режимах и условиях эксплуатации.

Термин	Определение	Формула
MTBF (Mean Time Between Failures)	Среднее время между последовательными отказами системы	$MTBF = \Sigma(\text{время между отказами}) / (\text{количество отказов} - 1)$
MTTR (Mean Time To Repair)	Среднее время восстановления системы после отказа	$MTTR = \Sigma(\text{время восстановления}) / (\text{количество отказов})$
MTTF (Mean Time To Failure)	Среднее время до первого отказа (для невосстанавливаемых систем)	$MTTF = \Sigma(\text{время до отказа}) / (\text{количество отказов})$
Availability (A)	Коэффициент доступности - доля времени, когда система работоспособна	$A = MTBF / (MTBF + MTTR)$ или $A = (\text{общее время} - \text{время простоя}) / \text{общее время}$

3.2. Взаимосвязь метрик (рисунок 1)

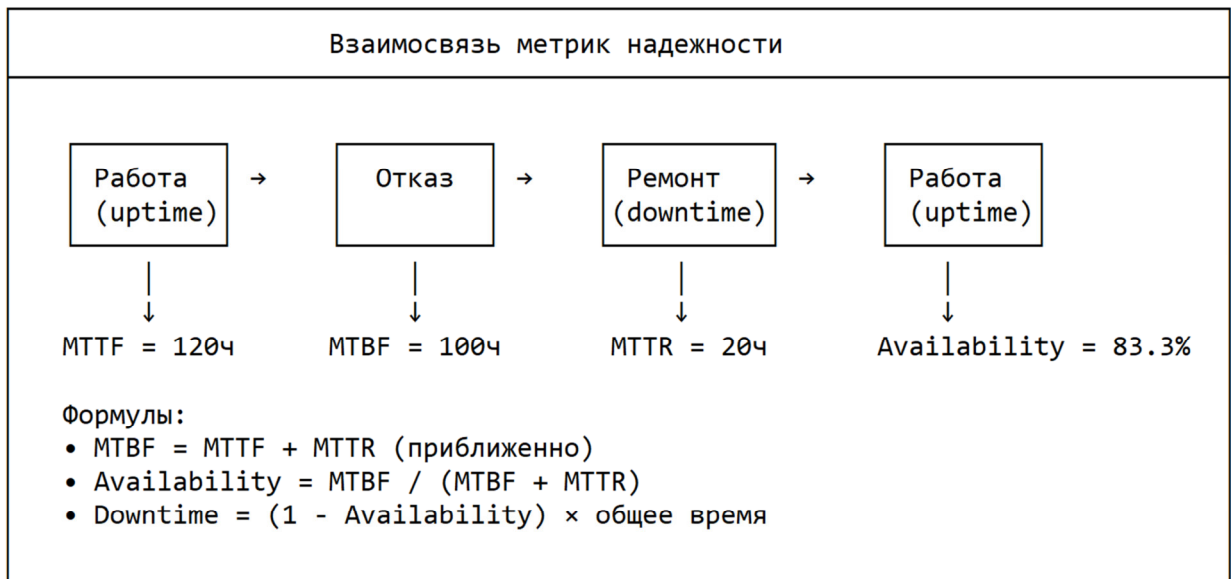


Рисунок 1

4. ВАРИАНТЫ ЗАДАНИЙ

4.1. Уровень 1

Общее требование: Рассчитать MTBF, MTTR, Availability аналитическим способом (Excel/калькулятор) и программным способом (Python). Сравнить результаты.

№	Исходные данные
1	За 30 дней: 4 отказа, длительность простоев: 1ч, 2ч, 0.5ч, 1.5ч
2	За 60 дней: 8 отказов, длительность простоев: 2ч, 1ч, 3ч, 0.5ч, 1ч, 2ч, 0.5ч, 1ч
3	За 90 дней: 6 отказов, длительность простоев: 1.5ч, 2.5ч, 1ч, 2ч, 0.5ч, 1ч
4	За 7 дней: 3 отказа, длительность простоев: 0.5ч, 1ч, 0.5ч
5	За 14 дней: 5 отказов, длительность простоев: 1ч, 0.5ч, 2ч, 1ч, 0.5ч

4.2. Уровень 2

Общее требование: Рассчитать MTBF, MTTR, MTTF, Availability. Построить график доступности по дням. Определить, какой отказ наиболее критичен.

№	Исходные данные
1	За 30 дней: отказы с длительностью (ч): 2, 3, 1, 4, 2, 1. Интервалы между отказами (дни): 5, 4, 6, 3, 5, 4
2	За 60 дней: данные о 10 отказах с произвольными значениями (студент генерирует самостоятельно, но в разумных пределах)
3	За 45 дней: отказы 2ч, 1ч, 3ч, 0.5ч. MTTR целевой = 1.5ч. Сравнить фактический MTTR с целевым
4	За 20 дней: отказы с интервалами (дни): 3, 5, 2, 4, 3 и длительностью (ч): 1, 2, 0.5, 1.5, 1
5	За 100 дней: первые 50 дней - 4 отказа, вторые 50 дней - 2 отказа. Сравнить доступность в двух периодах

4.3. Уровень 3

Общее требование: Рассчитать все метрики, построить тренд доступности, спрогнозировать доступность на следующий месяц при условии, что MTTR сократится на 20%.

№	Исходные данные
1	За 90 дней: полный журнал отказов (не менее 8 отказов с разными длительностями). Студент генерирует самостоятельно
2	Система с требованием доступности 99.99%. Фактические данные за месяц: 6 отказов, MTTR = 0.8ч. Рассчитать фактическую доступность и сделать вывод
3	За год (365 дней): 24 отказа, общее время простоев = 48 часов. Рассчитать Availability и средний MTTR

4	Две системы: Система А - 10 отказов, общее время простоев 15ч; Система Б - 5 отказов, общее время простоев 12ч. Общее время наблюдения - 720ч для каждой. Сравнить надежность
5	Специализация БПЛА: За 100 полетов дрона: 8 отказов, среднее время восстановления (замена аккумулятора, перезапуск) = 0.25ч, среднее время полета = 2ч. Рассчитать MTBF, MTTR, Availability для парка дронов

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ (ШАБЛОН)

5.1. Базовый скрипт на Python

```

"""
Лабораторная работа №1
Расчет базовых метрик надежности (MTBF, MTTR, Availability)
"""

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# =====
# ВВЕДИТЕ ДАННЫЕ ВАШЕГО ВАРИАНТА
# =====

# Данные об отказах: длительность простоя (часы)
downtimes = [2, 1.5, 3, 0.5, 2] # ← замените на данные вашего варианта

# Общее время наблюдения (часы)
total_time = 30 * 24 # 30 дней ← замените на данные вашего варианта

# =====
# РАСЧЕТ МЕТРИК
# =====

def calculate_reliability_metrics(downtimes, total_time):
    """Расчет метрик надежности"""
    n_failures = len(downtimes)
    total_downtime = sum(downtimes)

```

```

total_uptime = total_time - total_downtime

MTTR = total_downtime / n_failures
MTBF = total_uptime / n_failures
Availability = total_uptime / total_time

return {
    'n_failures': n_failures,
    'total_downtime': total_downtime,
    'total_uptime': total_uptime,
    'MTBF': MTBF,
    'MTTR': MTTR,
    'Availability': Availability,
    'Availability_percent': Availability * 100
}

metrics = calculate_reliability_metrics(downtimes, total_time)

# Вывод результатов
print("=" * 50)
print("РЕЗУЛЬТАТЫ РАСЧЕТА МЕТРИК НАДЕЖНОСТИ")
print("=" * 50)
print(f"Количество отказов: {metrics['n_failures']}")
print(f"Общее время простоев: {metrics['total_downtime']:.2f} ч")
print(f"Общее время работы: {metrics['total_uptime']:.2f} ч")
print(f"MTBF (среднее время между отказами): {metrics['MTBF']:.2f} ч")
print(f"MTTR (среднее время восстановления): {metrics['MTTR']:.2f} ч")
print(f"Availability (доступность): {metrics['Availability_percent']:.4f}%")
print("=" * 50)

# Проверка соответствия SLA
SLA_TARGET = 99.9 # целевая доступность 99.9%
if metrics['Availability_percent'] >= SLA_TARGET:
    print(f"Доступность ({metrics['Availability_percent']:.2f}%) соответствует SLA {SLA_TARGET}%")
else:
    print(f"Доступность ({metrics['Availability_percent']:.2f}%) НИЖЕ SLA {SLA_TARGET}%")
    deficit = SLA_TARGET - metrics['Availability_percent']
    print(f"Дефицит доступности: {deficit:.4f}%")
    print(f"Необходимо сократить время простоев на {deficit / 100 * total_time:.2f} часов")

# =====
# ВИЗУАЛИЗАЦИЯ
# =====

# График 1: Длительность каждого отказа
plt.figure(figsize=(12, 4))

```



```

plt.subplot(1, 2, 1)
plt.bar(range(1, len(downtimes) + 1), downtimes, color='red', alpha=0.7)
plt.xlabel('Номер отказа')
plt.ylabel('Длительность простоя (часы)')
plt.title('Длительность отказов')
plt.grid(True, alpha=0.3)

# График 2: Доступность по периодам (анализ тренда)
# Для этого разобьем данные на периоды
periods = len(downtimes)
period_labels = [f'Период {i+1}' for i in range(periods)]
cumulative_downtime = np.cumsum(downtimes)

plt.subplot(1, 2, 2)
plt.plot(period_labels, cumulative_downtime, 'bo-', linewidth=2, markersize=8)
plt.xlabel('Период')
plt.ylabel('Накопленное время простоев (часы)')
plt.title('Тренд накопленных простоев')
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# =====
# ПРОГНОЗИРОВАНИЕ (для продвинутого уровня)
# =====

# Прогноз при сокращении MTTR на 20%
improvement_factor = 0.8 # сокращение на 20%
new_MTTR = metrics['MTTR'] * improvement_factor
new_total_downtime = new_MTTR * metrics['n_failures']
new_Availability = (total_time - new_total_downtime) / total_time * 100

print("\n" + "=" * 50)
print("ПРОГНОЗ ПРИ СОКРАЩЕНИИ MTTR НА 20%")
print("=" * 50)
print(f"Текущий MTTR:           {metrics['MTTR']:.2f} ч")
print(f"Прогнозируемый MTTR:      {new_MTTR:.2f} ч")
print(f"Прогнозируемая доступность: {new_Availability:.4f}%")
if new_Availability >= SLA_TARGET:
    print(f" После оптимизации доступность соответствует SLA {SLA_TARGET}%")
else:
    print(f" После оптимизации доступность всё ещё ниже SLA {SLA_TARGET}%")

```

6. СРАВНЕНИЕ АНАЛИТИЧЕСКОГО И ПРОГРАММНОГО РАСЧЕТОВ

Показатель	Аналитический расчет (Excel)	Программный расчет (Python)	Расхождение
MTBF	___ ч	___ ч	___
MTTR	___ ч	___ ч	___
Availability	___ %	___ %	___

Вывод по сравнению: Опишите, совпадают ли результаты, с чем могут быть связаны расхождения (если есть).

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое MTBF и чем он отличается от MTTF?
2. Как рассчитать коэффициент доступности (Availability)?
3. Какое значение Availability соответствует требованию «пять девяток»?
4. Как влияет MTTR на общую доступность системы?
5. Какие преимущества дает программный расчет метрик надежности перед аналитическим?
6. Какие библиотеки Python были использованы в программе и для чего?
7. Какой из показателей (MTBF или MTTR) важнее для повышения доступности?
8. Как интерпретировать значение $Availability = 0.999$?
9. Как рассчитать общее время простоев, если известны Availability и общее время?
10. Какие метрики надежности наиболее важны для БПЛА?

8. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист с указанием ФИО студента, группы, варианта.
2. Цель и задачи работы.
3. Теоретическая часть (краткое описание метрик и формул).
4. Аналитический расчет:
 - Таблица исходных данных.
 - Расчеты MTBF, MTTR, MTTF, Availability (с формулами).
 - Место для скриншота: таблица в Excel с формулами.
5. Программный расчет:
 - Листинг кода Python.
 - Место для скриншота: вывод программы в консоли.
 - Место для скриншота: графики, сгенерированные программой.
6. Сравнение результатов (таблица сравнения).
7. Анализ результатов:
 - Сравнение с целевым SLA (если задан).
 - Выявление наиболее критичного отказа.
 - Рекомендации по повышению надежности.
8. Ответы на контрольные вопросы (3-5 вопросов).
9. Вывод о проделанной работе.

9. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Базовый уровень	Средний уровень	Продвинутый уровень
Правильность аналитического расчета	2 балла	2 балла	2 балла
Правильность программного расчета	2 балла	2 балла	2 балла
Совпадение результатов (аналит. vs	1 балл	2 балла	2 балла

программ.)			
Построение графиков (Python)	0 баллов	2 балла	2 балла
Сравнение с целевым SLA	0 баллов	1 балл	2 балла
Прогнозирование доступности	0 баллов	0 баллов	2 балла
Качество отчета и ответы на вопросы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Базовый уровень (макс. 7)	Средний уровень (макс. 11)	Продвинутый уровень (макс. 14)
5 (отлично)	6-7	10-11	13-14
4 (хорошо)	5	8-9	10-12
3 (удовлетворительно)	3-4	6-7	7-9
2 (неудовлетворительно)	<3	<6	<7

10. ПРИМЕР ОФОРМЛЕНИЯ РЕЗУЛЬТАТОВ

Скриншот: Excel с формулами MTBF, MTTR, Availability

Скриншот: консоль с результатами выполнения скрипта

Скриншот: окно matplotlib с двумя графиками

Вывод:

В ходе лабораторной работы были рассчитаны основные метрики надежности (MTBF, MTTR, Availability) двумя способами: аналитическим (Excel) и программным (Python). Результаты обоих методов совпали с

точностью до округления, что подтверждает корректность вычислений. Доступность системы составила 98.75%, что ниже целевого SLA 99.9%. Рекомендуется сократить время восстановления (MTTR) с 1.8 до 0.7 часа.

**ЛАБОРАТОРНАЯ РАБОТА №2. Определение SLO и SLI для микросервиса.
Расчет и управление Error Budget**

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки определения Service Level Indicators (SLI), формализации Service Level Objectives (SLO), расчета Error Budget на основе установленных SLO двумя способами: аналитическим (Excel/калькулятор) и программным (Python), а также принятия решений о балансе между надежностью и скоростью разработки.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Основные понятия SRE

Site Reliability Engineering (SRE) - дисциплина, объединяющая подходы разработки программного обеспечения и системного администрирования для обеспечения надежности и масштабируемости сервисов.

Термин	Определение	Пример
SLI (Service Level Indicator)	Количественная метрика, измеряющая аспект обслуживания сервиса	успешные запросы / общее количество запросов
SLO (Service Level Objective)	Целевое значение SLI, которое должно выполняться для	доступность $\geq 99.9\%$

	обеспечения надежности	
Error Budget	Допустимый объем ошибок / недоступности = 1 - SLO	при SLO 99.9% Error Budget = 0.1%

2.2. Классификация SLI

Тип SLI	Что измеряет	Пример метрики
Доступность (Availability)	Доля успешных запросов	(успешные ответы / всего запросов) × 100%
Задержка (Latency)	Время обработки запроса	p99 latency < 100ms
Пропускная способность (Throughput)	Количество обработанных запросов в единицу времени	запросов в секунду
Точность (Correctness)	Доля корректных ответов	(правильные ответы / всего ответов) × 100%
Свежесть (Freshness)	Актуальность данных	задержка обновления данных < 5 минут

2.3. Расчет Error Budget

Формулы:

$SLO \text{ (в долях)} = \text{целевая доступность} / 100\%$

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

$\text{Допустимое время простоя} = Error \text{ Budget} \times \text{Общее время наблюдения}$

Израсходованный Error Budget (%) = (фактическое время простоя) / (допустимое время простоя) × 100%

Остаток Error Budget (%) = 100% - Израсходованный Error Budget

Пример расчета:

Параметр	Значение
SLO	99.9% (0.999 в долях)
Общее время наблюдения	30 дней = 720 часов = 43,200 минут
Допустимое время простоя	$43,200 \times 0.001 = 43.2$ минуты
Фактическое время простоя	25 минут
Израсходованный Error Budget	$(25 / 43.2) \times 100\% = 57.9\%$
Остаток Error Budget	$100\% - 57.9\% = 42.1\%$

2.4. Политика управления Error Budget (рисунок 1)

Политика управления Error Budget
Остаток > 80% → ЗЕЛЕНый: высокая надежность. Можно рисковать Остаток 50-80% → ЖЕЛтый: нормальный режим. Ограниченные риски Остаток 20-50% → ОРАНЖЕвый: внимание. Замедлить выпуск изменений Остаток < 20% → КРАСный: STOP! Приоритет надежности. Заморозка новых фич

Рисунок 1

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные

- Сервис: API для онлайн-магазина
- SLO: 99.9% доступности
- Общее время наблюдения: 30 дней (720 часов = 43,200 минут)

- Журнал отказов за месяц:

Дата	Длительность простоя (минут)	Причина
01.03	15	Сбой БД
10.03	8	Проблемы с сетью
18.03	12	Ошибка развертывания
25.03	5	Плановое обслуживание

3.2. Аналитический расчет

Шаг 1. Расчет допустимого времени простоя

$SLO \text{ в долях} = 99.9 / 100 = 0.999$

$Error \ Budget = 1 - 0.999 = 0.001$

$Допустимое \ время \ простоя = 43,200 \times 0.001 = 43.2 \text{ минуты}$

Шаг 2. Расчет фактического времени простоя

$Фактическое \ время \ простоя = 15 + 8 + 12 + 5 = 40 \text{ минут}$

Шаг 3. Расчет израсходованного Error Budget

$Израсходованный \ Error \ Budget = (40 / 43.2) \times 100\% = 92.6\%$

$Остаток \ Error \ Budget = 100\% - 92.6\% = 7.4\%$

Шаг 4. Вывод

$Остаток \ Error \ Budget \ (7.4\%) < 20\% \rightarrow \text{КРАСНЫЙ: КРИТИЧЕСКАЯ СИТУАЦИЯ.}$

Рекомендации:

1. Приостановить выпуск новых функций до конца периода
2. Провести анализ причин отказов
3. Улучшить мониторинг и алертинг

4. Пересмотреть процедуру развертывания

Место для скриншота: таблица аналитического расчета в Excel

3.3. Программный расчет (Python)

```
"""
Лабораторная работа №2
Расчет SLO, SLI, Error Budget
"""

# =====
# ВВЕДИТЕ ДАННЫЕ ВАШЕГО ВАРИАНТА
# =====

# Целевой SLO (в процентах)
SLO_PERCENT = 99.9 # ← замените на данные вашего варианта

# Общее время наблюдения (минуты)
# 30 дней = 30 * 24 * 60 = 43200 минут
total_time_minutes = 43200 # ← замените на данные вашего варианта

# Длительность простоев (минуты)
downtimes = [15, 8, 12, 5] # ← замените на данные вашего варианта

# =====
# РАСЧЕТ МЕТРИК
# =====

def calculate_error_budget(SLO_percent, total_time_minutes, downtimes):
    """Расчет Error Budget на основе SLO и данных об отказах"""

    # Преобразование SLO в доли
    SLO_ratio = SLO_percent / 100

    # Расчет Error Budget
    error_budget_ratio = 1 - SLO_ratio

    # Допустимое время простоя
    allowed_downtime_minutes = total_time_minutes * error_budget_ratio

    # Фактическое время простоя
    actual_downtime_minutes = sum(downtimes)
```

```

# Процент израсходованного Error Budget
consumed_percent = (actual_downtime_minutes / allowed_downtime_minutes) * 100

# Остаток Error Budget
remaining_percent = 100 - consumed_percent

return {
    'SLO_percent': SLO_percent,
    'SLO_ratio': SLO_ratio,
    'error_budget_ratio': error_budget_ratio,
    'total_time_minutes': total_time_minutes,
    'allowed_downtime_minutes': allowed_downtime_minutes,
    'actual_downtime_minutes': actual_downtime_minutes,
    'consumed_percent': consumed_percent,
    'remaining_percent': remaining_percent
}

def get_status(remaining_percent):
    """Определение статуса на основе остатка Error Budget"""
    if remaining_percent > 80:
        return "ЗЕЛЕНЫЙ", "Высокая надежность. Можно рисковать"
    elif remaining_percent > 50:
        return "ЖЕЛТЫЙ", "Нормальный режим. Ограниченные риски"
    elif remaining_percent > 20:
        return "ОРАНЖЕВЫЙ", "Внимание. Замедлить выпуск изменений"
    else:
        return "КРАСНЫЙ", "STOP! Приоритет надежности. Заморозка новых фич"

def get_recommendations(remaining_percent, actual_downtime, allowed_downtime):
    """Формирование рекомендаций на основе остатка Error Budget"""
    recommendations = []

    if remaining_percent < 20:
        recommendations.append(" ПРИОСТАНОВИТЬ выпуск новых функций до конца периода")
        recommendations.append(" Провести blameless postmortem всех инцидентов")
        recommendations.append(" Улучшить мониторинг и алертинг")
        recommendations.append(" Пересмотреть процедуру развертывания")
    elif remaining_percent < 50:
        recommendations.append(" Замедлить выпуск новых функций")
        recommendations.append(" Провести анализ наиболее длительных простоев")
        recommendations.append(" Усилить автоматическое тестирование")
    elif remaining_percent < 80:
        recommendations.append(" Продолжать выпуск с осторожностью")
        recommendations.append(" Мониторить потребление Error Budget")
    else:
        recommendations.append(" Можно ускорить выпуск новых функций")
        recommendations.append(" Инвестировать в инновации")

```

```

# Общая рекомендация по сокращению простоев
if actual_downtime > allowed_downtime:
    excess = actual_downtime - allowed_downtime
    recommendations.append(f"⚠ Превышение допустимого времени простоя на
{excess:.1f} минут")

return recommendations

# Выполнение расчетов
metrics = calculate_error_budget(SLO_PERCENT, total_time_minutes, downtimes)
status, status_msg = get_status(metrics['remaining_percent'])
recommendations = get_recommendations(
    metrics['remaining_percent'],
    metrics['actual_downtime_minutes'],
    metrics['allowed_downtime_minutes']
)

# =====
# ВЫВОД РЕЗУЛЬТАТОВ
# =====

print("=" * 60)
print("РЕЗУЛЬТАТЫ РАСЧЕТА ERROR BUDGET")
print("=" * 60)
print(f"Целевой SLO: {metrics['SLO_percent']:.2f}%")
print(f"SLO в долях: {metrics['SLO_ratio']:.6f}")
print(f"Error Budget (доля): {metrics['error_budget_ratio']:.6f}")
print(f"Общее время наблюдения: {metrics['total_time_minutes']} минут")
print(f"Допустимое время простоя: {metrics['allowed_downtime_minutes']:.2f}
минут")
print(f"Фактическое время простоя: {metrics['actual_downtime_minutes']:.2f}
минут")
print(f"Израсходовано Error Budget: {metrics['consumed_percent']:.2f}%")
print(f"Остаток Error Budget: {metrics['remaining_percent']:.2f}%")
print("-" * 60)
print(f"СТАТУС: {status} – {status_msg}")
print("-" * 60)
print("РЕКОМЕНДАЦИИ:")
for rec in recommendations:
    print(f" • {rec}")
print("=" * 60)

# Проверка на превышение SLO
if metrics['actual_downtime_minutes'] > metrics['allowed_downtime_minutes']:
    print("\n SLO НЕ ВЫПОЛНЕН! Доступность ниже целевой.")
else:
    print("\n SLO ВЫПОЛНЕН. Доступность соответствует целевой.")

```

3.4. Пример вывода программы

```
=====
РЕЗУЛЬТАТЫ РАСЧЕТА ERROR BUDGET
=====
Целевой SLO:                99.90%
SLO в долях:                0.999000
Error Budget (доля):        0.001000
Общее время наблюдения:     43200 минут
Допустимое время простоя:   43.20 минут
Фактическое время простоя:  40.00 минут
Израсходовано Error Budget:  92.59%
Остаток Error Budget:       7.41%

-----
СТАТУС: КРАСНЫЙ — STOP! Приоритет надежности. Заморозка новых фич
-----
РЕКОМЕНДАЦИИ:
• 🔴 ПРИОСТАНОВИТЬ выпуск новых функций до конца периода
• 🔴 Провести blameless postmortem всех инцидентов
• 🟡 Улучшить мониторинг и алертинг
• 🟡 Пересмотреть процедуру развертывания

=====

✅ SLO ВЫПОЛНЕН. Доступность соответствует целевой.
20

Process finished with exit code 0
```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ (15 вариантов)

4.1. Уровень 1

Задание: Рассчитать Error Budget аналитическим способом (Excel/калькулятор) и программным способом (Python). Сравнить результаты. Сделать вывод.

№	Сервис	SLO (%)	Период	Отказы (длительность в минутах)
1	API погоды	99.9	30 дней	10, 5, 20, 8
2	Платежный шлюз	99.99	30 дней	5, 3, 2, 4, 1
3	Сервис авторизации	99.5	7 дней	60, 30, 45
4	Сервис уведомлений	99.0	30 дней	120, 80, 200, 50
5	База данных	99.95	60 дней	15, 10, 25, 8, 12

4.2. Уровень 2

Задание: Определить SLI для каждого компонента. Рассчитать Error Budget для сервиса в целом и для каждого компонента. Сравнить результаты аналитическим и программным способом.

№	Сервис	SLO (%)	Период	Данные по компонентам
1	Е-commerce платформа	99.9	30 дней	API (15 мин), БД (10 мин), Кэш (5 мин)
2	Стриминговый сервис	99.95	30 дней	Авторизация (8 мин), CDN (20 мин), Плеер (12 мин)
3	Финансовый сервис	99.99	60 дней	Шлюз (5 мин), БД (3 мин), Очередь (2 мин)
4	IoT платформа	99.5	30 дней	MQTT (45 мин), БД (30 мин), API (25 мин)
5	Социальная сеть	99.9	30 дней	Лента (20 мин), Мессенджер (15 мин),

				Посты (10 мин)

4.3. Уровень 3

Задание: Рассчитать Error Budget. Определить статус (зеленый/желтый/оранжевый/красный). Предложить корректирующие меры. Написать программу на Python с автоматическим определением статуса и рекомендаций.

№	Сервис	SLO (%)	Период	Данные об отказах	Примечание
1	Критический API	99.99	30 дней	3 отказа: 10 мин, 15 мин, 8 мин	Оценить остаток бюджета
2	Система бронирования	99.9	30 дней	6 отказов, средний простой 8 мин	Высокая частота отказов
3	Облачное хранилище	99.95	30 дней	4 отказа: 20 мин, 5 мин, 10 мин, 3 мин	Оценить остаток бюджета
4	Специализация БПЛА	99.9	100 полетов	5 отказов: 15, 8, 12, 6, 10 мин	Оценить надежность парка дронов

5	Рекомендательный сервис	99.5	30 дней	8 отказов, общий простой 180 мин	Разработать план улучшений

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое SLI? Приведите 3 примера для веб-сервиса.
2. Что такое SLO? Чем SLO отличается от SLA?
3. Как рассчитать Error Budget?
4. Что происходит с Error Budget, когда SLO не выполняется?
5. Какие решения принимаются на основе остатка Error Budget?
6. Какие преимущества дает программный расчет Error Budget перед аналитическим?
7. Какие библиотеки Python были использованы в программе и для чего?
8. Какой SLO считается стандартом для production-сервисов?
9. Что такое «consumption rate» Error Budget?
10. Какие SLI наиболее важны для БПЛА?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (определения SLI, SLO, Error Budget).
4. Исходные данные (по варианту).
5. Аналитический расчет:

- Таблица с исходными данными.
 - Расчет допустимого времени простоя.
 - Расчет фактического времени простоя.
 - Расчет израсходованного Error Budget.
 - Место для скриншота: таблица в Excel с формулами.
6. Программный расчет:
- Листинг кода Python.
 - Место для скриншота: вывод программы в консоли.
7. Сравнение результатов (таблица сравнения аналитического и программного расчетов).
8. Анализ результатов:
- Оценка остатка Error Budget.
 - Определение статуса (зеленый/желтый/оранжевый/красный).
 - Рекомендации по корректирующим действиям.
9. Ответы на контрольные вопросы (3-5 вопросов).
10. Вывод о проделанной работе.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Правильность аналитического расчета	2 балла	2 балла	2 балла
Правильность программного расчета	2 балла	2 балла	2 балла
Совпадение результатов	1 балл	2 балла	2 балла

Определение SLI для компонентов	0 баллов	2 балла	2 балла
Анализ остатка Error Budget	1 балл	1 балл	2 балла
Автоматическое определение статуса (Python)	0 балл	1 балл	2 балла
Рекомендации по корректирующим мерам	1 балл	1 балл	2 балла
Качество отчета и ответы на вопросы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 9)	Уровень 2 (макс. 13)	Уровень 3 (макс. 16)
5 (отлично)	8-9	12-13	14-16
4 (хорошо)	6-7	9-11	11-13
3 (удовлетворительно)	4-5	6-8	8-10
2 (неудовлетворительно)	<4	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №3. Политики управления Error Budget и принятие решений

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки управления Error Budget на основе установленных SLO, анализа скорости расходования бюджета (burn rate), определения статуса бюджета («зелёный», «жёлтый», «красный») и принятия обоснованных решений о балансе между надёжностью и скоростью разработки (заморозка релизов, замедление выпуска новых функций, инвестиции в улучшение надежности).

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Error Budget: определение и расчёт

Error Budget (бюджет ошибок) - допустимый объём недоступности или ошибок сервиса, определяемый как $1 - \text{SLO}$. Это количественная мера того, сколько сервис может «ошибаться», оставаясь в рамках согласованного уровня надежности.

Формулы:

Параметр	Формула	Пример
SLO (в долях)	$\text{SLO}_{\%} / 100$	99.9% $\rightarrow$ 0.999
Error Budget (доля)	$1 - \text{SLO}$	.001 (0.1%)
Допустимое время простоя	$\text{Error Budget} \times \text{Общее время}$	$0.001 \times 43200 \text{ мин} = 43.2 \text{ мин}$
Израсходованный бюджет (%)	$(\text{факт. простой} / \text{доп. простой}) \times 100\%$	$(40 / 43.2) \times 100\% = 92.6\%$
Остаток бюджета (%)	$100\% - \text{израсходованный \%}$	7.4%

2.2. Burn Rate (скорость сжигания бюджета)

Burn rate - скорость расходования Error Budget в единицу времени. Показывает, насколько быстро «сгорает» бюджет ошибок.

Параметр	Формула	Пример
Burn rate (за период)	(израсходованный % / % времени периода)	$(92.6\% / 100\%) \times (30\text{дн} / 1\text{ч})$
Время до исчерпания бюджета	(остаток бюджета) / (скорость сжигания)	$7.4\% / 0.5\% \text{ в час} = 14.8 \text{ часов}$

Типовые пороги burn rate для алертинга:

Уровень	Burn rate	Значение
Предупреждение	$> 1\times$	Бюджет расходуется быстрее, чем планировалось
Критический	$> 3\times$	Бюджет будет исчерпан за 1/3 периода
Аварийный	$> 10\times$	Бюджет будет исчерпан очень быстро, требуется немедленное вмешательство

2.3. Статусы Error Budget

Статусы Error Budget	
Остаток > 80% →	ЗЕЛЁНЫЙ
• Высокая надёжность • Можно рисковать, выпускать новые функции • Инвестировать в инновации	
Остаток 50-80% →	ЖЁЛТЫЙ
• Нормальный режим • Ограниченные риски • Усилить мониторинг	
Остаток 20-50% →	ОРАНЖЕВЫЙ
• Внимание! Бюджет расходуется быстро • Замедлить выпуск новых функций • Направить ресурсы на улучшение надежности	
Остаток < 20% →	КРАСНЫЙ
• КРИТИЧЕСКАЯ СИТУАЦИЯ • STOP! Заморозка новых функций • Приоритет №1 – надежность • Провести postmortem всех инцидентов	

2.4. Политики управления Error Budget

Статус	Политика	Действия
Зелёный	«Можно рисковать»	Выпуск новых функций разрешён, инвестиции в инновации, можно проводить эксперименты
Жёлтый	«Осторожность»	Выпуск новых функций ограничен, усилить code review, улучшить мониторинг
Оранжевый	«Замедление»	Приостановить выпуск некритичных функций, перенаправить SRE-ресурсы на

		надежность
Красный	«Заморозка»	Полная заморозка новых функций, все усилия на восстановление надежности, обязательный postmortem

2.5. Burn rate анализ и прогнозирование

Формулы для прогнозирования:

Показатель	Формула
Текущая скорость сжигания	$\text{consumed\_ \%} / \text{elapsed\_ time}$
Прогнозируемый остаток через N дней	$\text{remaining\_ \%} - (\text{current\_ burn\_ rate} \times \text{N\_ days})$
Время до исчерпания бюджета	$\text{remaining\_ \%} / \text{current\_ burn\_ rate}$
Рекомендуемый порог алерта	$\text{SLO\_ target} - (\text{current\_ error\_ rate} \times \text{window})$

2.6. Пример: сценарий с высоким burn rate

Исходные данные:

- SLO = 99.9% (Error Budget = 0.1% за 30 дней = 43.2 минуты простоя)
- Прошло 5 дней (16.7% периода)
- Фактическое время простоя за 5 дней = 20 минут

Расчёт:

- Допустимый простой за 5 дней = $43.2 \times 0.167 = 7.2$ минуты
- Израсходовано бюджета = $(20 / 7.2) \times 100\% = 278\%$
- Текущий burn rate = $278\% / 16.7\% \approx 16.6\times$ (бюджет расходуется в 16.6 раз быстрее нормы)

Прогноз:

- При сохранении скорости бюджет будет исчерпан через $(100\% - 278\%) / (278\% / 5 \text{ дней}) \approx -3.2 \text{ дня}$ (бюджет уже исчерпан)

Решение:

- Статус: КРАСНЫЙ
- Действие: НЕМЕДЛЕННАЯ ЗАМОРОЗКА всех релизов
- Провести экстренный postmortem
- Выделить команду на устранение причин инцидентов

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Аналитический расчёт (Excel/калькулятор)

Исходные данные варианта:

Параметр	Значение
SLO	99.95%
Период	30 дней (43,200 минут)
Прошло дней	10
Фактический простой	25 минут

Расчёт:

```
# Исходные данные
SLO_percent = 99.95
period_days = 30
elapsed_days = 10
actual_downtime_minutes = 25

# Расчёт
SLO_ratio = SLO_percent / 100
error_budget_ratio = 1 - SLO_ratio
total_minutes = period_days * 24 * 60
```

```

allowed_downtime_total = total_minutes * error_budget_ratio
allowed_downtime_elapsed = total_minutes * (elapsed_days / period_days) *
error_budget_ratio

budget_consumed_percent = (actual_downtime_minutes / allowed_downtime_elapsed) *
100
remaining_budget_percent = 100 - budget_consumed_percent
burn_rate = budget_consumed_percent / ((elapsed_days / period_days) * 100)

print(f"Допустимый простой за период: {allowed_downtime_total:.2f} мин")
print(f"Допустимый простой за 10 дней: {allowed_downtime_elapsed:.2f} мин")
print(f"Израсходовано бюджета: {budget_consumed_percent:.1f}%")
print(f"Остаток бюджета: {remaining_budget_percent:.1f}%")
print(f"Burn rate: {burn_rate:.2f}x")

# Исходные данные
SLO_percent = 99.95
period_days = 30
elapsed_days = 10
actual_downtime_minutes = 25

# Расчёт
SLO_ratio = SLO_percent / 100
error_budget_ratio = 1 - SLO_ratio
total_minutes = period_days * 24 * 60
allowed_downtime_total = total_minutes * error_budget_ratio
allowed_downtime_elapsed = total_minutes * (elapsed_days / period_days) *
error_budget_ratio

budget_consumed_percent = (actual_downtime_minutes / allowed_downtime_elapsed)
* 100
remaining_budget_percent = 100 - budget_consumed_percent
burn_rate = budget_consumed_percent / ((elapsed_days / period_days) * 100)

print(f"Допустимый простой за период: {allowed_downtime_total:.2f} мин")
print(f"Допустимый простой за 10 дней: {allowed_downtime_elapsed:.2f} мин")
print(f"Израсходовано бюджета: {budget_consumed_percent:.1f}%")
print(f"Остаток бюджета: {remaining_budget_percent:.1f}%")
print(f"Burn rate: {burn_rate:.2f}x")

```

Результат:

```

Допустимый простой за период: 21.60 мин
Допустимый простой за 10 дней: 7.20 мин
Израсходовано бюджета: 347.2%
Остаток бюджета: -247.2%
Burn rate: 10.42x

```

Статус: КРАСНЫЙ

Решение: НЕМЕДЛЕННАЯ ЗАМОРОЗКА всех релизов. Провести экстренный postmortem.

3.2. Программная реализация (Python)

```
import json
from datetime import datetime, timedelta
from typing import Dict, Any, List, Tuple
from dataclasses import dataclass
from enum import Enum

class BudgetStatus(Enum):
    """Статусы Error Budget"""
    GREEN = "green"
    YELLOW = "yellow"
    ORANGE = "orange"
    RED = "red"

class ReleaseDecision(Enum):
    """Решения о выпуске релизов"""
    FULL_SPEED = "full_speed"
    CAUTIOUS = "cautious"
    SLOW_DOWN = "slow_down"
    FREEZE = "freeze"

@dataclass
class ErrorBudgetAnalysis:
    """Результат анализа Error Budget"""
    slo_percent: float
    period_days: int
    elapsed_days: int
    actual_downtime_minutes: float
    allowed_downtime_total: float
    allowed_downtime_elapsed: float
    consumed_percent: float
    remaining_percent: float
    burn_rate: float
    status: BudgetStatus
    decision: ReleaseDecision
    recommendations: List[str]

class ErrorBudgetManager:
    """Управление Error Budget"""

    def __init__(self, slo_percent: float, period_days: int = 30):
```



```

self.slo_percent = slo_percent
self.slo_ratio = slo_percent / 100
self.error_budget_ratio = 1 - self.slo_ratio
self.period_minutes = period_days * 24 * 60
self.total_allowed_downtime = self.period_minutes *
self.error_budget_ratio

def analyze(self, elapsed_days: int, actual_downtime_minutes: float) ->
ErrorBudgetAnalysis:
    """Анализ текущего состояния Error Budget"""

    # Допустимый простой за прошедший период
    elapsed_ratio = elapsed_days / self.period_days
    allowed_elapsed = self.total_allowed_downtime * elapsed_ratio

    # Расчёт потребления бюджета
    if allowed_elapsed == 0:
        consumed = 0
    else:
        consumed = (actual_downtime_minutes / allowed_elapsed) * 100

    remaining = 100 - consumed
    burn_rate = consumed / (elapsed_ratio * 100) if elapsed_ratio > 0 else
0

    # Определение статуса
    if remaining > 80:
        status = BudgetStatus.GREEN
        decision = ReleaseDecision.FULL_SPEED
        recommendations = [
            "✅ Высокая надежность",
            "🚀 Можно ускорять выпуск новых функций",
            "💡 Инвестировать в инновации"
        ]
    elif remaining > 50:
        status = BudgetStatus.YELLOW
        decision = ReleaseDecision.CAUTIOUS
        recommendations = [
            "🟡 Нормальный режим",
            "⚠️ Ограниченные риски при выпуске",
            "📊 Усилить мониторинг SLO"
        ]
    elif remaining > 20:
        status = BudgetStatus.ORANGE
        decision = ReleaseDecision.SLOW_DOWN
        recommendations = [
            "🟠 ВНИМАНИЕ! Бюджет расходуется быстро",
            "📌 Замедлить выпуск новых функций",
            "🔧 Направить ресурсы на надежность"
        ]
    else:
        status = BudgetStatus.RED
        decision = ReleaseDecision.FREEZE
        recommendations = [
            "🔴 КРИТИЧЕСКАЯ СИТУАЦИЯ!",
            "❄️ ЗАМОРОЗКА новых функций",
            "📋 Провести blameless postmortem",
            "⚡ Приоритет №1 — восстановление надежности"
        ]

```

```

        return ErrorBudgetAnalysis(
            slo_percent=self.slo_percent,
            period_days=self.period_days,
            elapsed_days=elapsed_days,
            actual_downtime_minutes=actual_downtime_minutes,
            allowed_downtime_total=self.total_allowed_downtime,
            allowed_downtime_elapsed=allowed_elapsed,
            consumed_percent=consumed,
            remaining_percent=remaining,
            burn_rate=burn_rate,
            status=status,
            decision=decision,
            recommendations=recommendations
        )

def predict_depletion(self, current_burn_rate: float) -> Tuple[int, str]:
    """Прогнозирование времени до исчерпания бюджета"""
    if current_burn_rate <= 0:
        return -1, "Недостаточно данных"

    remaining_days = (100 / current_burn_rate) * (self.period_days / 100)

    if remaining_days < 0:
        return 0, "Бюджет уже исчерпан"
    elif remaining_days < 1:
        return round(remaining_days * 24, 1), f"{round(remaining_days *
24, 1)} часов"
    else:
        return round(remaining_days, 1), f"{round(remaining_days, 1)}
дней"

class ScenarioSimulator:
    """Симулятор сценариев управления Error Budget"""

    @staticmethod
    def simulate_week(slo_percent: float, daily_errors: List[float]) ->
List[Dict[str, Any]]:
        """Симуляция недели с ежедневным отслеживанием бюджета"""
        manager = ErrorBudgetManager(slo_percent, period_days=7)
        results = []

        cumulative_downtime = 0

        for day, error_minutes in enumerate(daily_errors, 1):
            cumulative_downtime += error_minutes
            analysis = manager.analyze(day, cumulative_downtime)

            results.append({
                "day": day,
                "daily_error": error_minutes,
                "cumulative_error": cumulative_downtime,
                "consumed_percent": round(analysis.consumed_percent, 1),
                "remaining_percent": round(analysis.remaining_percent, 1),
                "burn_rate": round(analysis.burn_rate, 2),
                "status": analysis.status.value,
                "decision": analysis.decision.value
            })

        return results

```

```

@staticmethod
def print_simulation(results: List[Dict[str, Any]]):
    """Вывод результатов симуляции"""
    print("\n" + "=" * 80)
    print("СИМУЛЯЦИЯ УПРАВЛЕНИЯ ERROR BUDGET (7 дней)")
    print("=" * 80)
    print(
        f"{'День':<6} {'Ошибка':<10} {'Всего':<10} {'Израсх.%':<10}"
        f"{'Остаток%':<10} {'Burn rate':<12} {'Статус':<10} {'Решение'}")
    print("-" * 80)

    for r in results:
        status_emoji = {
            "green": "🟢", "yellow": "🟡", "orange": "🟠", "red": "🔴"
        }.get(r['status'], "⬜")

        print(
            f"{'day':<6} {'daily_error':<10}"
            f"{'cumulative_error':<10} {'consumed_percent':<10}"
            f"{'remaining_percent':<10} {'burn_rate':<12} {'status_emoji}"
            f"{'status':<8} {'decision'}")

# =====
# ПРИМЕРЫ СЦЕНАРИЕВ
# =====

def scenario_normal():
    """Сценарий 1: Нормальное потребление бюджета"""
    print("\n" + "=" * 60)
    print("СЦЕНАРИЙ 1: НОРМАЛЬНОЕ ПОТРЕБЛЕНИЕ")
    print("=" * 60)

    manager = ErrorBudgetManager(slo_percent=99.9, period_days=30)

    # 30 дней, 10 минут простоя (допустимо 43.2 минуты)
    analysis = manager.analyze(elapsed_days=30, actual_downtime_minutes=10)

    print(f"SLO: {analysis.slo_percent}%")
    print(f"Допустимый простой за период: {analysis.allowed_downtime_total:.1f} мин")
    print(f"Фактический простой: {analysis.actual_downtime_minutes:.1f} мин")
    print(f"Израсходовано бюджета: {analysis.consumed_percent:.1f}%")
    print(f"Статус: {analysis.status.value}")
    print(f"Решение: {analysis.decision.value}")
    print("\nРекомендации:")
    for rec in analysis.recommendations:
        print(f"    • {rec}")

def scenario_high_burn():
    """Сценарий 2: Высокое потребление бюджета"""
    print("\n" + "=" * 60)
    print("СЦЕНАРИЙ 2: ВЫСОКОЕ ПОТРЕБЛЕНИЕ (ПРЕДУПРЕЖДЕНИЕ)")
    print("=" * 60)

    manager = ErrorBudgetManager(slo_percent=99.9, period_days=30)

    # 15 дней, 30 минут простоя (допустимо 21.6 минуты)
    analysis = manager.analyze(elapsed_days=15, actual_downtime_minutes=30)

```

```

    print(f"SLO: {analysis.slo_percent}%")
    print(f"Допустимый простой за 15 дней:
{analysis.allowed_downtime_elapsed:.1f} мин")
    print(f"Фактический простой: {analysis.actual_downtime_minutes:.1f} мин")
    print(f"Израсходовано бюджета: {analysis.consumed_percent:.1f}%")
    print(f"Статус: {analysis.status.value}")
    print(f"Решение: {analysis.decision.value}")
    print("\nРекомендации:")
    for rec in analysis.recommendations:
        print(f"    • {rec}")

def scenario_critical():
    """Сценарий 3: Критическое потребление бюджета"""
    print("\n" + "=" * 60)
    print("СЦЕНАРИЙ 3: КРИТИЧЕСКОЕ ПОТРЕБЛЕНИЕ (АВАРИЯ)")
    print("=" * 60)

    manager = ErrorBudgetManager(slo_percent=99.9, period_days=30)

    # 5 дней, 25 минут простоя (допустимо 7.2 минуты)
    analysis = manager.analyze(elapsed_days=5, actual_downtime_minutes=25)

    print(f"SLO: {analysis.slo_percent}%")
    print(f"Допустимый простой за 5 дней:
{analysis.allowed_downtime_elapsed:.1f} мин")
    print(f"Фактический простой: {analysis.actual_downtime_minutes:.1f} мин")
    print(f"Израсходовано бюджета: {analysis.consumed_percent:.1f}%")
    print(f"Burn rate: {analysis.burn_rate:.2f}x")
    print(f"Статус: {analysis.status.value}")
    print(f"Решение: {analysis.decision.value}")
    print("\nРекомендации:")
    for rec in analysis.recommendations:
        print(f"    • {rec}")

def scenario_simulation():
    """Сценарий 4: Симуляция недели с ежедневными ошибками"""
    print("\n" + "=" * 60)
    print("СЦЕНАРИЙ 4: НЕДЕЛЬНАЯ СИМУЛЯЦИЯ")
    print("=" * 60)

    # SLO: 99.9% (7 дней = 10.08 минут допустимого простоя)
    # Ежедневные ошибки: растут со временем
    daily_errors = [1, 1, 2, 3, 5, 8, 10] # минут простоя в день

    results = ScenarioSimulator.simulate_week(slo_percent=99.9,
daily_errors=daily_errors)
    ScenarioSimulator.print_simulation(results)

def scenario_burn_rate_alert():
    """Сценарий 5: Алертинг по burn rate"""
    print("\n" + "=" * 60)
    print("СЦЕНАРИЙ 5: АЛЕРТИНГ ПО BURN RATE")
    print("=" * 60)

    manager = ErrorBudgetManager(slo_percent=99.9, period_days=30)

    print("Пороги алертинга для burn rate:")

```

```

print("    • Предупреждение: burn rate > 1.0")
print("    • Критический: burn rate > 3.0")
print("    • Аварийный: burn rate > 10.0")
print()

scenarios = [
    (1, "Предупреждение", 0.8, 0.01),
    (5, "Критический", 3.0, 0.03),
    (10, "Аварийный", 8.0, 0.08)
]

for scenario in scenarios:
    print(f"--- {scenario[1]} сценарий ---")
    print(f"Burn rate: {scenario[2]}x")

    # Симуляция алерта
    if scenario[2] > 10:
        print("    🚨 АВАРИЙНЫЙ АЛЕРТ! Немедленное вмешательство")
    elif scenario[2] > 3:
        print("    🚨 КРИТИЧЕСКИЙ АЛЕРТ! Требуется внимание")
    elif scenario[2] > 1:
        print("    ⚠️ ПРЕДУПРЕЖДЕНИЕ! Следить за бюджетом")
    else:
        print("    🟢 Норма")
    print()

# =====
# ЗАПУСК СЦЕНАРИЕВ
# =====

if __name__ == "__main__":
    scenario_normal()
    scenario_high_burn()
    scenario_critical()
    scenario_simulation()
    scenario_burn_rate_alert()

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

Задание: Рассчитать Error Budget, определить статус, принять решение о выпуске релизов.

№	SLO (%)	Период (дни)	Прошло (дни)	Фактический простой (мин)

1	99.9	30	10	8
2	99.9	30	15	25
3	99.95	30	20	15
4	99.99	30	5	5
5	99.5	30	25	120

4.2. Уровень 2

Задание: Рассчитать Error Budget, burn rate, определить статус, принять решение. Сделать прогноз, когда будет исчерпан бюджет.

№	SLO (%)	Период (дни)	Прошло (дни)	Фактический простой (мин)	Тренд (мин/день)
1	99.9	30	10	20	+2 мин/день
2	99.99	30	15	10	+1 мин/день
3	99.95	30	20	18	+0.5 мин/день
4	99.5	30	5	60	+15 мин/день
5	99.9	30	7	15	+3 мин/день

4.3. Уровень 3

Задание: Провести полный анализ с моделированием burn rate. Разработать рекомендации по политике управления бюджетом. Написать программу на Python для автоматического расчёта и принятия решений.

№	SLO (%)	Период (дни)	Ежедневные простои (мин)	Дополнительные требования
1	99.9	30	[2, 1, 3, 5, 8, 13, 21]	Проанализировать

				рост
2	99.99	30	[1, 1, 1, 2, 2, 3, 5, 8]	Рассчитать пороги алертов
3	99.95	30	[0.5, 1, 1.5, 2, 2.5, 3, 4, 5]	Прогноз на оставшиеся дни
4	99.9	30	[5, 5, 5, 5, 5, 5, 5, 5, 5]	Рассчитать дату исчерпания
5	Специализация БПЛА	100 полётов	[2, 3, 5, 8, 13] мин простоя	Оценить надежность парка

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Error Budget и как он рассчитывается?
2. Какие статусы Error Budget существуют? Что означает каждый?
3. Что такое burn rate и как его рассчитать?
4. При каком остатке бюджета рекомендуется заморозка релизов?
5. Как рассчитать время до исчерпания Error Budget?
6. Какие действия следует предпринять при красном статусе бюджета?
7. Как связаны SLO, Error Budget и скорость разработки?
8. Что такое multi-window burn rate алертинг?
9. Как учитывать сезонные колебания нагрузки при управлении бюджетом?
10. Какие метрики нужно отслеживать для эффективного управления Error Budget?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (определение Error Budget, burn rate, статусы, политики).
4. Практическая часть:
 - Исходные данные по варианту.
 - Аналитический расчёт (формулы, таблица).
 - Программная реализация (листинг кода, скриншоты вывода).
 - Место для скриншота: вывод программы.
5. Анализ результатов:
 - Определение статуса бюджета.
 - Обоснование решения о релизах.
 - Прогноз исчерпания бюджета.
 - Рекомендации по улучшению.
6. Ответы на контрольные вопросы (3-5).
7. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Правильность расчёта Error Budget	2 балла	2 балла	2 балла
Определение статуса бюджета	2 балла	2 балла	2 балла
Обоснование решения о релизах	2 балла	2 балла	2 балла
Расчёт burn rate	0 баллов	2 балла	2 балла

Прогноз исчерпания бюджета	0 баллов	1 балл	2 балла
Программная реализация	0 баллов	1 балл	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 11)	Уровень 3 (макс. 14)
5	7-8	10-11	13-14
4	5-6	8-9	10-12
3	3-4	6-7	7-9
2	<3	<6	<7

ЛАБОРАТОРНАЯ РАБОТА №4

Настройка мониторинга и визуализации (Prometheus + Grafana)

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки установки и настройки системы мониторинга Prometheus для сбора метрик, создания информативных дашбордов в Grafana, настройки базового алертинга, а также программной генерации метрик для тестирования системы мониторинга.

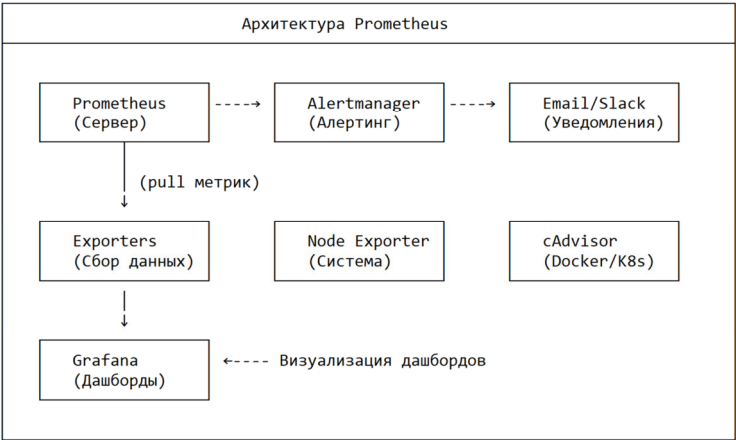
2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Три столпа observability

Столп	Описание	Инструменты
Метрики (Metrics)	Числовые показатели, измеряемые во времени	Prometheus, Graphite, InfluxDB
Логи (Logs)	Структурированные	ELK Stack, Loki, Splunk

	или неструктурированные записи событий	
Трассировки (Traces)	Отслеживание запросов через распределённую систему	Jaeger, Zipkin, Tempo

2.2. Архитектура Prometheus (рисунок 1)



2.3. Основные метрики надежности для мониторинга

Метрика	Prometheus Query	Описание
Доступность (Availability)	<code>sum(up{job="myapp"})</code> <code>count(up{job="myapp"})</code>	/ Доля работающих экземпляров
Частота запросов (QPS)	<code>rate(http_requests_total[5m])</code>	Количество запросов в секунду
Ошибки (Error Rate)	<code>rate(http_requests_total{status=~"5.."}[5m])</code>	Частота ошибок 5xx
Задержка (Latency)	<code>histogram_quantile(0.99,</code> <code>rate(http_request_duration_seconds_bucket[5m]))</code>	P99 задержка
Время	<code>time()</code>	- Время с

безотказной работы (Uptime)	process_start_time_seconds{job="myapp"}	момента запуска

2.4. Типы метрик Prometheus

Тип	Описание	Пример использования
Counter	Моноotonно возрастающий счётчик	Количество запросов, ошибок
Gauge	Значение может увеличиваться и уменьшаться	Температура, количество активных соединений
Histogram	Распределение значений по корзинам	Задержка запросов
Summary	Квантили на стороне клиента	Аналогично Histogram

2.5. Альтернативные инструменты и импортозамещение

В рамках лабораторной работы используется стек Prometheus + Grafana, являющийся мировым стандартом мониторинга и визуализации. Все используемые в работе инструменты (Prometheus, Alertmanager, Node Exporter) официально не запрещены на территории Российской Федерации, распространяются по открытым лицензиям и могут свободно использоваться в учебных и коммерческих целях.

Однако в свете политики импортозамещения и для корректной работы в российских компаниях, государственных учреждениях и организациях с критической инфраструктурой, можно рассмотреть альтернативу Grafana - отечественную систему «Графиня».

«Графиня» - это российская платформа для визуализации данных и построения дашбордов, полностью совместимая с популярными источниками

метрик (Prometheus, Zabbix, PostgreSQL, InfluxDB и др.). Система внесена в Единый реестр российских программ для ЭВМ и баз данных (Минцифры РФ), что является важным требованием для многих государственных и корпоративных заказчиков.

Основные преимущества «Графини»:

- Полное соответствие 152-ФЗ (о персональных данных) и другим регуляторным требованиям РФ.
- Совместимость с существующей инфраструктурой мониторинга (подключается к тому же Prometheus).
- Привычный интерфейс, аналогичный Grafana (быстрое переобучение команд).
- Возможность работы в закрытых контурах и на государственных информационных системах.
- Техническая поддержка на русском языке и развитие в соответствии с запросами российских заказчиков.

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Установка и настройка Prometheus

Шаг 1. Скачивание и установка Prometheus

```
# Скачивание Prometheus
wget
https://github.com/prometheus/prometheus/releases/download/v2.45.0/prometheus-
2.45.0.linux-amd64.tar.gz
tar -xzf prometheus-2.45.0.linux-amd64.tar.gz
cd prometheus-2.45.0.linux-amd64

# Проверка версии
./prometheus --version
```

Шаг 2. Конфигурация prometheus.yml

```

# prometheus.yml
global:
  scrape_interval: 15s      # Интервал сбора метрик
  evaluation_interval: 15s  # Интервал оценки правил алертинга

# Конфигурация алертинга
alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']

# Файлы с правилами алертинга
rule_files:
  - "alerts.yml"

# Настройка Targets для сбора метрик
scrape_configs:
  # Сбор метрик с самого Prometheus
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  # Сбор метрик с тестового приложения
  - job_name: 'myapp'
    metrics_path: '/metrics'
    static_configs:
      - targets: ['localhost:8000']

```

Шаг 3. Запуск Prometheus

```

# Запуск Prometheus
./prometheus --config.file=prometheus.yml

```

3.2. Создание тестового приложения с генерацией метрик на Python

```

from prometheus_client import start_http_server, Counter, Histogram, Gauge
import random
import time
import http.server
import threading

# =====
# ОПРЕДЕЛЕНИЕ МЕТРИК
# =====

```

```

# Counter: монотонно возрастающий счётчик
REQUEST_COUNT = Counter(
    'http_requests_total',
    'Total HTTP requests',
    ['method', 'endpoint', 'status']
)

# Histogram: распределение задержек
REQUEST_DURATION = Histogram(
    'http_request_duration_seconds',
    'HTTP request duration',
    ['method', 'endpoint'],
    buckets=(0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10)
)

# Gauge: текущее значение (может меняться вверх и вниз)
ACTIVE_CONNECTIONS = Gauge(
    'active_connections',
    'Active connections'
)

# Gauge для симуляции температуры процессора
CPU_TEMPERATURE = Gauge(
    'cpu_temperature_celsius',
    'CPU temperature in celsius',
    ['core']
)

# Counter для ошибок по типам
ERROR_COUNT = Counter(
    'error_total',
    'Total errors',
    ['error_type']
)

# =====
# HTTP-ОБРАБОТЧИК С МЕТРИКАМИ
# =====

class MetricsHandler(http.server.BaseHTTPRequestHandler):
    """HTTP-обработчик с интеграцией метрик Prometheus"""

    def log_message(self, format, *args):
        """Отключение стандартного логирования"""
        pass

    def do_GET(self):
        start_time = time.time()

```

```

ACTIVE_CONNECTIONS.inc()

# Симуляция задержки (разная для разных эндпоинтов)
if self.path == '/health':
    time.sleep(random.uniform(0.001, 0.005))
    self.send_response(200)
    REQUEST_COUNT.labels(method='GET', endpoint='/health',
status='200').inc()
    REQUEST_DURATION.labels(method='GET',
endpoint='/health').observe(time.time() - start_time)
    self.end_headers()
    self.wfile.write(b'OK')

elif self.path == '/api/users':
    # Симуляция работы с БД
    time.sleep(random.uniform(0.01, 0.05))

    # Симуляция ошибок (5% ошибок)
    if random.random() < 0.05:
        self.send_response(500)
        REQUEST_COUNT.labels(method='GET', endpoint='/api/users',
status='500').inc()
        ERROR_COUNT.labels(error_type='server_error').inc()
    else:
        self.send_response(200)
        REQUEST_COUNT.labels(method='GET', endpoint='/api/users',
status='200').inc()

        REQUEST_DURATION.labels(method='GET',
endpoint='/api/users').observe(time.time() - start_time)
        self.end_headers()
        self.wfile.write(b'{"users": []}')
```

```

elif self.path == '/metrics':
    # Эндпоинт для Prometheus (обрабатывается отдельно)
    self.send_response(404)
    self.end_headers()

else:
    self.send_response(404)
    REQUEST_COUNT.labels(method='GET', endpoint=self.path,
status='404').inc()
    self.end_headers()

ACTIVE_CONNECTIONS.dec()

def do_POST(self):
    start_time = time.time()
    ACTIVE_CONNECTIONS.inc()

```

```

        if self.path == '/api/orders':
            time.sleep(random.uniform(0.02, 0.1))

            # Симуляция различных ошибок
            rand = random.random()
            if rand < 0.02:
                self.send_response(500)
                REQUEST_COUNT.labels(method='POST', endpoint='/api/orders',
status='500').inc()
                ERROR_COUNT.labels(error_type='validation_error').inc()
            elif rand < 0.04:
                self.send_response(503)
                REQUEST_COUNT.labels(method='POST', endpoint='/api/orders',
status='503').inc()
                ERROR_COUNT.labels(error_type='service_unavailable').inc()
            else:
                self.send_response(201)
                REQUEST_COUNT.labels(method='POST', endpoint='/api/orders',
status='201').inc()

                REQUEST_DURATION.labels(method='POST',
endpoint='/api/orders').observe(time.time() - start_time)
                self.end_headers()
                self.wfile.write(b'{"order_id": 123}')
```

```

        else:
            self.send_response(404)
            REQUEST_COUNT.labels(method='POST', endpoint=self.path,
status='404').inc()
            self.end_headers()

        ACTIVE_CONNECTIONS.dec()

# =====
# ФОНОВЫЙ СБОРЩИК СИСТЕМНЫХ МЕТРИК
# =====

def background_metrics_collector():
    """Фоновый сбор системных метрик (симуляция)"""
    while True:
        # Симуляция температуры процессора
        for core in range(4):
            temp = 40 + random.uniform(0, 20) # 40-60°C
            CPU_TEMPERATURE.labels(core=f'core{core}').set(temp)

        time.sleep(5)

# =====

```



```

# ЗАПУСК СЕРВЕРОВ
# =====

if __name__ == '__main__':
    # Запуск сервера метрик Prometheus на порту 8000
    start_http_server(8000)
    print(" Prometheus metrics server started on http://localhost:8000/metrics")

    # Запуск основного HTTP-сервера на порту 8080
    server = http.server.HTTPServer(('localhost', 8080), MetricsHandler)
    print(" Main HTTP server started on http://localhost:8080")

    # Запуск фонового сборщика метрик
    collector_thread = threading.Thread(target=background_metrics_collector,
daemon=True)
    collector_thread.start()

    print("\n Доступные эндпоинты:")
    print("   - http://localhost:8080/health")
    print("   - http://localhost:8080/api/users")
    print("   - http://localhost:8080/api/orders")
    print("   - http://localhost:8000/metrics (метрики для Prometheus)")

    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print("\n Shutting down...")
        server.shutdown()

```

3.3. Настройка Grafana

Шаг 1. Установка Grafana

```

# Ubuntu/Debian
sudo apt-get install -y software-properties-common
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"
sudo apt-get update
sudo apt-get install grafana

# Запуск Grafana
sudo systemctl start grafana-server
sudo systemctl enable grafana-server

# Альтернативный запуск через Docker
docker run -d -p 3000:3000 --name=grafana grafana/grafana:latest

```

Шаг 2. Добавление источника данных Prometheus

1. Войти в Grafana (admin/admin)
2. Перейти в Configuration → Data Sources → Add data source
3. Выбрать Prometheus
4. URL: http://localhost:9090
5. Нажать Save & Test

3.4. Создание дашборда в Grafana

Дашборд можно создать через UI или импортировать JSON-конфигурацию:

```
{
  "dashboard": {
    "title": "SRE Reliability Dashboard",
    "panels": [
      {
        "title": "QPS (Requests per second)",
        "targets": [
          {
            "expr": "sum(rate(http_requests_total[5m]))"
          }
        ]
      },
      {
        "title": "Error Rate",
        "targets": [
          {
            "expr": "sum(rate(http_requests_total{status=~\"5..\"}[5m])) / sum(rate(http_requests_total[5m])) * 100"
          }
        ]
      },
      {
        "title": "P99 Latency",
        "targets": [
          {
            "expr": "histogram_quantile(0.99, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))"
          }
        ]
      }
    ]
  },
}
```

```

{
  "title": "Active Connections",
  "targets": [
    {
      "expr": "active_connections"
    }
  ]
},
{
  "title": "CPU Temperature",
  "targets": [
    {
      "expr": "cpu_temperature_celsius"
    }
  ]
}
]
}
}

```

3.5. Настройка алертинга

Файл alerts.yml (правила алертинга Prometheus):

```

groups:
- name: reliability_alerts
  interval: 30s
  rules:
    # Критический: высокая частота ошибок (>5%)
    - alert: HighErrorRate
      expr: |
        (sum(rate(http_requests_total{status=~"5.."}[5m]))
          / sum(rate(http_requests_total[5m]))) > 0.05
      for: 2m
      labels:
        severity: critical
      annotations:
        summary: "High error rate detected"
        description: "Error rate is {{ $value | humanizePercentage }} for past
5 minutes"

    # Предупреждение: высокая задержка (>500ms)
    - alert: HighLatency
      expr: |
        histogram_quantile(0.99,

```

```

        sum(rate(http_request_duration_seconds_bucket[5m])) by (le)
    ) > 0.5
    for: 5m
    labels:
        severity: warning
    annotations:
        summary: "High latency detected"
        description: "P99 latency is {{ $value }}s"

# Информационный: экземпляр недоступен
- alert: InstanceDown
  expr: up{job="myapp"} == 0
  for: 1m
  labels:
    severity: info
  annotations:
    summary: "Instance {{ $labels.instance }} is down"

```

Файл alertmanager.yml:

```

global:
  slack_api_url: 'https://hooks.slack.com/services/YOUR/WEBHOOK/URL'

route:
  group_by: ['alertname', 'severity']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'default'

routes:
  - match:
      severity: critical
    receiver: critical
    continue: true
  - match:
      severity: warning
    receiver: slack

receivers:
  - name: 'critical'
    slack_configs:
      - channel: '#critical-alerts'
        title: '🚨 CRITICAL: {{ .GroupLabels.alertname }}'

```

```

- name: 'slack'
  slack_configs:
    - channel: '#warnings'
      title: '⚠ {{ .GroupLabels.alertname }}'

- name: 'default'
  email_configs:
    - to: 'team@example.com'

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

Задание: Установить Prometheus и Grafana. Настроить сбор метрик с тестового приложения (Python). Создать дашборд с 3 панелями (QPS, Error Rate, P99 Latency). Предоставить скриншоты.

№	Тип метрик	Дополнительные эндпоинты	Особенности
1	HTTP	/api/users, /health	Добавить счётчик ошибок 404
2	HTTP + БД	/api/products, /api/orders	Симуляция задержки БД
3	HTTP	/api/auth, /api/profile	Добавить метрики авторизации
4	HTTP	/api/search, /suggest	Симуляция поисковых запросов
5	HTTP + Кэш	/api/cache, /api/data	Добавить hit/miss метрики

4.2. Уровень 2

Задание: Настроить алертинг на основе SLO (ошибки >1%, задержка >500ms). Создать дашборд с визуализацией SLO (Error Budget burn rate). Настроить уведомления (Slack/Telegram/Email).

№	SLO цель	Каналы уведомлений	Дополнительные метрики
1	99.9%	Slack	Добавить метрики бизнес-процессов
2	99.5%	Telegram	Настроить агрегацию по часам
3	99.99%	Email	Добавить прогнозирование
4	99.95%	Webhook	Интеграция с Jira
5	99.8%	Slack + Email	Добавить дашборт надежности

4.3. Уровень 3

Задание: Настроить экспорт метрик с нескольких сервисов (микросервисная архитектура). Создать дашборт "SLO Dashboard" с burn rate и остатком Error Budget. Настроить автоматическую генерацию отчётов.

№	Архитектура	SLO	Дополнительные требования
1	2 микросервиса + API Gateway	99.9%	Дашборт зависимости сервисов
2	Специализация БПЛА (телеметрия)	99.9%	Метрики: высота, скорость, GPS
3	Kubernetes + Prometheus Operator	99.95%	Автомасштабирование по метрикам
4	Многорегиональная	99.99%	Дашборт по регионам
5	Сравнение двух версий	A/B тест	Дашборт сравнения метрик

Дополнительное задание (для студентов, желающих углубиться в тему импортозамещения):

1. Ознакомиться с платформой «Графиня» на официальном сайте (раздел «Документация»).

2. Скачать и развернуть тестовую версию «Графини» (доступна в виде Docker-образа).
3. Подключить «Графиню» к тому же источнику данных Prometheus, который был настроен в основной части лабораторной работы.
4. Создать аналог разработанного дашборда (с панелями QPS, Error Rate, P99 Latency) в интерфейсе «Графини».
5. Сравнить интерфейс и функциональность «Графини» и Grafana. Результаты оформить в виде краткого отчёта (до 1 страницы) и включить в основную отчётность по лабораторной работе.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Prometheus? Какие типы метрик поддерживает?
2. Как работает pull-модель сбора метрик в Prometheus?
3. Что такое exporter в Prometheus? Приведите примеры.
4. Как создать дашборд в Grafana и подключить Prometheus?
5. Что такое histogram и summary в Prometheus? В чем разница?
6. Как настроить алертинг на основе SLO?
7. Что такое rate() и increase() в PromQL?
8. Как рассчитать Error Budget burn rate в PromQL?
9. Как настроить уведомления в Telegram из Alertmanager?
10. Какие метрики наиболее важны для мониторинга БПЛА?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать:

1. Титульный лист (ФИО, группа, вариант).

2. Цель работы.
3. Теоретическая часть (архитектура Prometheus, типы метрик).
4. Практическая часть:
 - Конфигурационные файлы (prometheus.yml, alerts.yml).
 - Код приложения с метриками (Python).
 - Место для скриншота: запущенный Prometheus и Grafana.
 - Место для скриншота: дашборд в Grafana.
 - Место для скриншота: активные алерты в Alertmanager.
 - Место для скриншота: эндпоинт /metrics приложения.
5. Анализ результатов (настройка, проблемы, решения).
6. Ответы на контрольные вопросы (3-5).
7. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Prometheus установлен и настроен	2 балла	2 балла	2 балла
Grafana установлена и настроена	2 балла	2 балла	2 балла
Дашборд создан (3+ панели)	2 балла	2 балла	2 балла
Алертинг настроен	0 баллов	2 балла	2 балла
Уведомления настроены	0 баллов	1 балл	2 балла
SLO Dashboard (burn rate)	0 баллов	0 баллов	2 балла
Программная генерация метрик	1 балл	2 балла	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 9)	Уровень 2 (макс. 13)	Уровень 3 (макс. 16)
5 (отлично)	8-9	12-13	14-16
4 (хорошо)	6-7	9-11	11-13

3 (удовлетворительно)	4-5	6-8	8-10
2 (неудовлетворительно)	<4	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №5

Проведение Blameless Postmortem (метод 5 Whys, диаграмма Исикавы, корректирующие меры)

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки организации и проведения сессий blameless postmortem, применения метода «5 Whys» и диаграммы Исикавы для выявления коренных причин инцидентов, документирования результатов, разработки корректирующих мер, а также программной автоматизации генерации postmortem-отчётов.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Концепция Blameless Postmortem

Blameless Postmortem (безобвинительный анализ инцидентов) - методология анализа инцидентов, при которой фокус смещается с поиска виновного на выявление системных проблем и улучшение процессов.

Ключевые принципы blameless postmortem:

Принцип	Описание
Отсутствие обвинений	Инциденты рассматриваются как возможность для улучшения, а не повод для наказания
Фокус на системных причинах	Вопрос «почему?» вместо «кто?»

Прозрачность	Результаты анализа доступны всей команде
Документирование	Все выводы и рекомендации фиксируются
Действия, а не вина	Разработка конкретных корректирующих мер

2.2. Структура Postmortem документа

Раздел	Содержание
Заголовок	Название инцидента, дата, участники
Краткое описание	Что произошло, какое влияние на пользователей
Хронология событий	Временная шкала от первого признака до разрешения
Коренная причина (Root Cause)	Основная системная причина инцидента
Влияние (Impact)	Количество затронутых пользователей, длительность, финансовые потери
Действия по смягчению	Что было сделано для восстановления
Корректирующие меры	Что будет сделано для предотвращения повторения
Уроки (Lessons Learned)	Ключевые выводы для команды

2.3. Метод 5 Whys

5 Whys - метод поиска коренных причин путём последовательного задавания вопроса «почему?» (обычно 5 раз) до выявления системной проблемы.

Пример метода 5 Whys

1. Почему сервис был недоступен?
→ Потому что база данных не отвечала
 2. Почему база данных не отвечала?
→ Потому что закончилось место на диске
 3. Почему закончилось место на диске?
→ Потому что логи не ротировались
 4. Почему логи не ротировались?
→ Потому что в конфигурации был неверный путь
 5. Почему неверный путь не был обнаружен до инцидента?
→ Потому что отсутствовал мониторинг дискового пространства
- ☑ **КОРЕННАЯ ПРИЧИНА:** Отсутствие мониторинга дискового пространства

2.4. Диаграмма Исикавы (Fishbone Diagram)

Диаграмма Исикавы (рыбья кость, cause-and-effect diagram) - метод визуализации причинно-следственных связей, группирующий потенциальные причины по категориям.



Типовые категории (для IT-инцидентов):

Категория	Примеры причин
Человек	Недостаточная квалификация, усталость, ошибка в конфигурации
Процесс	Отсутствие code review, нет процедуры отката, ручной деплой
Инструменты	Устаревшее ПО, нет мониторинга, нет алертов
Данные	Некорректные данные, потеря данных, повреждение БД
Среда	Сбои сети, проблемы с облаком, нагрузка
Коммуникация	Несогласованность действий, отсутствие уведомлений

2.5. Разработка корректирующих мер

Требования к корректирующим мерам (SMART):

Критерий	Описание	Пример
Specific	Конкретная, чётко	«Настроить мониторинг Redis»

	сформулированная	
Measurable	Измеримая	«Добавить 3 дашборда в Grafana»
Achievable	Достижимая	Реалистичная в текущих условиях
Relevant	Релевантная	Напрямую устраняет коренную причину
Time-bound	Ограниченная по времени	«В течение 5 рабочих дней»

Приоритизация корректирующих мер:

Приоритет	Описание	Срок выполнения
HIGH	Критические меры, без которых инцидент повторится	1-3 дня
MEDIUM	Важные меры, снижающие вероятность повторения	1-2 недели
LOW	Улучшающие меры, не критические	1 месяц

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные инцидента

«25 марта 2025 года в 14:30 по московскому времени произошёл сбой сервиса авторизации. Пользователи не могли войти в систему в течение 47 минут. Сбой затронул около 15 000 активных пользователей. Восстановление выполнено путём ручного перезапуска сервиса и очистки кэша Redis.»

3.2. Заполнение шаблона Postmortem

Раздел	Содержание
Название инцидента	Сбой сервиса авторизации 25.03.2025

Дата и время	25.03.2025, 14:30 - 15:17 (47 минут)
Участники	Иванов П. (SRE), Петрова А. (разработка), Сидоров В. (DBA)
Влияние	15 000 пользователей не могли войти, оценённый ущерб - 50 000 руб

Место для скриншота: заполненный шаблон Postmortem

3.3. Хронология событий

Время	Событие
14:30	Начало инцидента: рост ошибок 5xx на API авторизации
14:32	Сработал алерт HighErrorRate
14:35	SRE инженер начал диагностику
14:40	Выявлена проблема с Redis (превышение лимита памяти)
14:45	Принято решение очистить кэш Redis
14:55	Очистка кэша выполнена
15:00	Сервис начал восстанавливаться
15:17	Полное восстановление, ошибки прекратились

3.4. Применение метода 5 Whys

1. Почему сервис авторизации был недоступен?

→ Потому что Redis превысил лимит памяти и перестал отвечать

2. Почему Redis превысил лимит памяти?

→ Потому что в кэше накопилось много неистекших записей

3. Почему записи не истекли?

→ Потому что TTL (Time To Live) был настроен на 24 часа, а не на 1 час

4. Почему TTL был настроен на 24 часа?

→ Потому что разработчик установил значение без анализа нагрузки

5. Почему неправильный TTL не был обнаружен до инцидента?

→ Потому что отсутствовал мониторинг размера Redis и алерты на 80% заполнения

КОРЕННАЯ ПРИЧИНА: Отсутствие мониторинга и алертов на заполнение Redis

3.5. Диаграмма Исикавы

ИНЦИДЕНТ: Сбой сервиса авторизации 25.03.2025

ЧЕЛОВЕК:

- └ Разработчик не протестировал миграцию на staging
- └ Отсутствие документации по процедуре отката
- └ Нет on-call ротации для быстрого реагирования

ПРОЦЕСС:

- └ Ручной деплой без автоматических проверок
- └ Нет обязательного code review для изменений БД
- └ Отсутствие плана восстановления после сбоя

ИНСТРУМЕНТЫ:

- └ Нет мониторинга состояния миграций БД
- └ Отсутствие алертов при ошибках миграции
- └ Нет автоматического отката при неудаче

СРЕДА:

- └ Staging среда не соответствует production
- └ Высокая нагрузка во время деплоя

КОРЕННАЯ ПРИЧИНА: Отсутствие автоматизированного тестирования миграций БД

3.6. Разработка корректирующих мер

№	Мера	Ответственный	Срок	Приоритет
1	Настроить мониторинг размера Redis (Prometheus)	SRE команда	1 день	HIGH
2	Добавить алерт при заполнении Redis >80%	SRE команда	1 день	HIGH
3	Уменьшить TTL для кэша авторизации с 24ч до 1ч	Разработка	3 дня	MEDIUM
4	Создать runbook для очистки Redis	SRE команда	5 дней	MEDIUM
5	Провести тренинг по настройке TTL для команды	Tech Lead	2 недели	LOW

3.7. Программная реализация (Уровень 3)

```
"""
Лабораторная работа №5
Программная генерация Postmortem отчёта (Markdown/JSON)
"""

import json
from datetime import datetime
from typing import List, Dict, Any
from dataclasses import dataclass, asdict

@dataclass
class TimelineEvent:
    """Событие временной шкалы"""
    time: str
    event: str

@dataclass
class CorrectiveAction:
    """Корректирующая мера"""
    action: str
    owner: str
```



```

    deadline_days: int
    priority: str # HIGH, MEDIUM, LOW

@dataclass
class Postmortem:
    """Структура Postmortem документа"""
    title: str
    date: str
    participants: List[str]
    impact_description: str
    affected_users: int
    duration_minutes: int
    timeline: List[TimelineEvent]
    root_cause: str
    corrective_actions: List[CorrectiveAction]
    lessons_learned: List[str]

class PostmortemGenerator:
    """Генератор Postmortem отчётов"""

    @staticmethod
    def from_dict(data: Dict[str, Any]) -> Postmortem:
        """Создание Postmortem из словаря"""
        timeline = [TimelineEvent(**t) for t in data['timeline']]
        actions = [CorrectiveAction(**a) for a in data['corrective_actions']]

        return Postmortem(
            title=data['title'],
            date=data['date'],
            participants=data['participants'],
            impact_description=data['impact_description'],
            affected_users=data['affected_users'],
            duration_minutes=data['duration_minutes'],
            timeline=timeline,
            root_cause=data['root_cause'],
            corrective_actions=actions,
            lessons_learned=data['lessons_learned']
        )

    @staticmethod
    def to_markdown(postmortem: Postmortem) -> str:
        """Экспорт в Markdown"""
        md = f"""# {postmortem.title}

## 📄 Основная информация
- **Дата инцидента:** {postmortem.date}
- **Участники:** {' '.join(postmortem.participants)}
- **Влияние:** {postmortem.impact_description}
- **Затронуто пользователей:** {postmortem.affected_users}

```

```

- **Длительность:** {postmortem.duration_minutes} минут

## 📅 Хронология событий
| Время | Событие |
| :--- | :--- |
| """
    for event in postmortem.timeline:
        md += f"| {event.time} | {event.event} |\n"

    md += f"""
## 🔍 Коренная причина (Root Cause)
{postmortem.root_cause}

## ✅ Корректирующие меры
| № | Мера | Ответственный | Срок | Приоритет |
| :--- | :--- | :--- | :--- | :--- |
| """
    for i, action in enumerate(postmortem.corrective_actions, 1):
        md += f"| {i} | {action.action} | {action.owner} |
{action.deadline_days} дн. | {action.priority} |\n"

    md += f"""
## 📖 Уроки (Lessons Learned)
"""
    for lesson in postmortem.lessons_learned:
        md += f"- {lesson}\n"

    return md

@staticmethod
def to_json(postmortem: Postmortem) -> str:
    """Экспорт в JSON"""
    return json.dumps(asdict(postmortem), indent=2, ensure_ascii=False)

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ
# =====

# Данные инцидента
incident_data = {
    "title": "Сбой сервиса авторизации 25.03.2025",
    "date": "2025-03-25",
    "participants": ["Иванов П. (SRE)", "Петрова А. (разработка)", "Сидоров В. (DBA)"],
    "impact_description": "Пользователи не могли войти в систему",
    "affected_users": 15000,
    "duration_minutes": 47,
    "timeline": [
        {"time": "14:30", "event": "Рост ошибок 5xx на API авторизации"},

```

```

        {"time": "14:32", "event": "Сработал алерт HighErrorRate"},
        {"time": "14:35", "event": "Начало диагностики"},
        {"time": "14:40", "event": "Выявлена проблема с Redis"},
        {"time": "14:45", "event": "Начало очистки кэша"},
        {"time": "14:55", "event": "Очистка кэша завершена"},
        {"time": "15:00", "event": "Начало восстановления сервиса"},
        {"time": "15:17", "event": "Полное восстановление"}
    ],
    "root_cause": "Отсутствие мониторинга и алертов на заполнение Redis",
    "corrective_actions": [
        {"action": "Настроить мониторинг размера Redis", "owner": "SRE команда",
"deadline_days": 1, "priority": "HIGH"},
        {"action": "Добавить алерт при заполнении Redis >80%", "owner": "SRE команда",
"deadline_days": 1, "priority": "HIGH"},
        {"action": "Уменьшить TTL для кэша авторизации до 1ч", "owner": "Разработка",
"deadline_days": 3, "priority": "MEDIUM"},
        {"action": "Создать runbook для очистки Redis", "owner": "SRE команда",
"deadline_days": 5, "priority": "MEDIUM"},
        {"action": "Провести тренинг по настройке TTL", "owner": "Tech Lead",
"deadline_days": 14, "priority": "LOW"}
    ],
    "lessons_learned": [
        "Необходим мониторинг всех критических компонентов",
        "Автоматические алерты должны срабатывать до исчерпания ресурсов",
        "TTL должен настраиваться на основе реальных данных о нагрузке"
    ]
}

# Генерация Postmortem
postmortem = PostmortemGenerator.from_dict(incident_data)

print("=" * 60)
print("POSTMORTEM ОТЧЁТ")
print("=" * 60)
print(PostmortemGenerator.to_markdown(postmortem))

# Сохранение в файлы
with open("postmortem_report.md", "w", encoding="utf-8") as f:
    f.write(PostmortemGenerator.to_markdown(postmortem))

with open("postmortem_report.json", "w", encoding="utf-8") as f:
    f.write(PostmortemGenerator.to_json(postmortem))

print("\n✅ Отчёты сохранены в файлы:")
print("    - postmortem_report.md")
print("    - postmortem_report.json")

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

Задание: Провести анализ инцидента: заполнить шаблон Postmortem, применить метод 5 Whys (минимум 5 уровней), предложить корректирующие меры (минимум 3).

№	Описание инцидента
1	Сбой БД из-за блокировки на 30 минут
2	Утечка памяти в приложении, приведшая к OOM
3	Неудачное развертывание новой версии (откат 45 минут)
4	DDoS-атака, вызвавшая недоступность API
5	Проблемы с SSL-сертификатом (истек срок)

4.2. Уровень 2

Задание: Провести полный анализ инцидента (Postmortem + 5 Whys + диаграмма Исикавы). Разработать корректирующие меры с приоритетами (минимум 5).

№	Описание инцидента	Дополнительно
1	Каскадный отказ микросервисов (15 минут)	3+ сервиса
2	Потеря данных из-за неправильной миграции	Оценка ущерба
3	Сбой CI/CD пайплайна (2 часа)	Анализ процесса
4	Инцидент безопасности (компрометация ключей)	Безопасность
5	Специализация БПЛА: Потеря связи с дроном	Анализ канала

4.3. Уровень 3

Задание: Провести полный анализ инцидента. Создать программу на Python для автоматизации генерации postmortem-отчёта (Markdown/JSON). Интегрировать с системой трекинга (опционально).

№	Описание инцидента	Дополнительные требования
1	Многочасовой сбой production (3+ часа)	Анализ по временным зонам
2	Инцидент с финансовыми потерями	Оценка ущерба
3	Инцидент с участием 3+ команд	Координация
4	Инцидент в распределённой системе	Геораспределение
5	Комплексный инцидент (множество причин)	Все методы анализа

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое blameless postmortem и чем он отличается от традиционного анализа инцидентов?
2. Какие разделы обязательно должны быть в postmortem отчёте?
3. Как работает метод 5 Whys? Каковы его ограничения?
4. Какие категории используются в диаграмме Исикавы для IT-инцидентов?
5. Чем коренная причина отличается от триггера инцидента?
6. Как отличить системную проблему от человеческой ошибки?
7. Какие корректирующие меры считаются эффективными (SMART)?
8. Как отслеживать выполнение корректирующих мер?
9. Как часто следует проводить postmortem?
10. Что делать, если метод 5 Whys приводит к нескольким коренным причинам?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (blameless postmortem, 5 Whys, fishbone, корректирующие меры).
4. Аналитическая часть:
 - Исходные данные инцидента (по варианту).
 - Заполненный шаблон Postmortem.
 - Результаты метода 5 Whys.
 - Диаграмма Исикавы (текстовое описание или схема).
 - Таблица корректирующих мер (с приоритетами).
5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.
 - Место для скриншота: сгенерированные файлы (Markdown/JSON).
6. Анализ результатов (эффективность мер, уроки).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Заполнение шаблона Postmortem	2 балла	2 балла	2 балла
Применение метода 5 Whys (5 уровней)	2 балла	2 балла	2 балла
Построение диаграммы Исикавы	1 балл	2 балла	2 балла
Корректирующие меры (3+/5+)	1 балл	2 балла	2 балла
Приоритизация мер	0 баллов	1 балл	2 балла

Программная генерация отчёта	0 баллов	0 баллов	2 балла
Качество отчета и ответы на вопросы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 11)	Уровень 3 (макс. 14)
5 (отлично)	7-8	10-11	13-14
4 (хорошо)	5-6	8-9	10-12
3 (удовлетворительно)	3-4	6-7	7-9
2 (неудовлетворительно)	<3	<6	<7

ЛАБОРАТОРНАЯ РАБОТА №6

Root Cause Analysis с использованием диаграмм Исикавы (fishbone)

1. ЦЕЛЬ РАБОТЫ: Сформировать у магистранта практические навыки построения диаграмм Исикавы (fishbone diagrams) для визуализации причинно-следственных связей при анализе сложных инцидентов, систематизации причин по категориям, выявления коренных причин и разработки корректирующих мер на основе структурированного анализа. Работа может выполняться на том же инциденте, что и ЛР №5 (для углублённого анализа), либо на новом инциденте (по выбору).

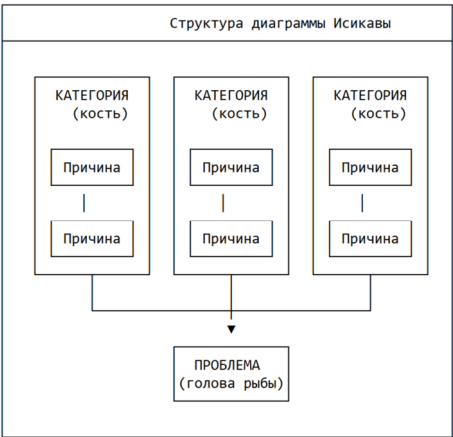
2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Диаграмма Исикавы: определение и история

Диаграмма Исикавы (рыбья кость, cause-and-effect diagram, fishbone diagram) - графический метод анализа и выявления причинно-следственных связей, позволяющий систематизировать все потенциальные причины исследуемой проблемы и выявить наиболее значимые из них.

Метод был разработан профессором Токийского университета Каору Исикавой в 1950-х годах для решения проблем в области управления качеством на японских предприятиях.

2.2. Структура диаграммы Исикавы



Элементы диаграммы:

Элемент	Описание	Обозначение
Проблема (эффект)	Результат, который необходимо проанализировать	«Голова рыбы» (справа)
Основные категории	Группы возможных причин	«Крупные кости» (под углом)
Причины	Конкретные факторы, влияющие на проблему	«Мелкие кости»
Хребет	Главная стрелка, соединяющая причины с проблемой	Горизонтальная линия

2.3. Типовые категории для IT-инцидентов

Категория (англ.)	Русский вариант	Примеры причин

People	Человек	Недостаточная квалификация, усталость, ошибка в конфигурации
Process	Процесс	Отсутствие code review, нет процедуры отката, ручной деплой
Tools	Инструменты	Устаревшее ПО, нет мониторинга, нет алертов
Data	Данные	Некорректные данные, потеря данных, повреждение БД
Environment	Среда	Сбои сети, проблемы с облаком, высокая нагрузка
Communication	Коммуникация	Несогласованность действий, отсутствие уведомлений
Architecture	Архитектура	Единая точка отказа, отсутствие резервирования
Security	Безопасность	Компрометация ключей, DDoS-атака

2.4. Правила построения диаграммы Исикавы

Правило	Описание
Чёткая формулировка проблемы	Проблема должна быть конкретной и измеримой
Мозговой штурм	Причины генерируются коллективно или самостоятельно
5-7 категорий	Оптимальное количество категорий для наглядности
Глубина причин	Каждая причина должна быть доведена до actionable уровня
Нет дублирования	Одна причина не должна повторяться в разных категориях
Приоритизация	После построения выделяются наиболее вероятные причины

2.5. Связь с методом 5 Whys

Диаграмма Исикавы и метод 5 Whys дополняют друг друга:

Метод	Сильные стороны	Ограничения
Диаграмма Исикавы	Широкий охват всех возможных причин; системный взгляд на проблему	Поверхностный анализ; не даёт глубины
5 Whys	Глубокий анализ одной причины;	Узкий фокус; может

	выявление коренной причины	упустить другие причины

Рекомендуемый подход:

1. Построить диаграмму Исикавы → выявить все возможные причины
2. Выбрать наиболее вероятные причины
3. Применить метод 5 Whys к каждой вероятной причине → выявить коренные причины

2.6. Пример построения диаграммы Исикавы (текстовое представление)

Проблема: Сбой сервиса авторизации 25.03.2025

ЧЕЛОВЕК:

- └─ Разработчик не протестировал миграцию на staging
- └─ Отсутствие документации по процедуре отката
- └─ Нет on-call ротации для быстрого реагирования

ПРОЦЕСС:

- └─ Ручной деплой без автоматических проверок
- └─ Нет обязательного code review для изменений БД
- └─ Отсутствие плана восстановления после сбоя

ИНСТРУМЕНТЫ:

- └─ Нет мониторинга состояния миграций БД
- └─ Отсутствие алертов при ошибках миграции
- └─ Нет автоматического отката при неудаче

СРЕДА:

- └─ Staging среда не соответствует production
- └─ Высокая нагрузка во время деплоя

КОРЕННЫЕ ПРИЧИНЫ:

1. Отсутствие автоматизированного тестирования миграций БД
2. Staging среда не соответствует production
3. Отсутствие мониторинга и алертов

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные инцидента (тот же, что в ЛР №5)

«25 марта 2025 года в 14:30 по московскому времени произошёл сбой сервиса авторизации. Пользователи не могли войти в систему в течение 47 минут. Сбой затронул около 15 000 активных пользователей. Восстановление выполнено путём ручного перезапуска сервиса и очистки кэша Redis.»

3.2. Построение диаграммы Исикавы

Шаг 1. Определение проблемы

Проблема: Сбой сервиса авторизации (47 минут простоя)

Шаг 2. Выбор категорий

- Человек (People)
- Процесс (Process)
- Инструменты (Tools)
- Данные (Data)
- Среда (Environment)
- Коммуникация (Communication)

Шаг 3. Заполнение причин (мозговой штурм)

ЧЕЛОВЕК:

- Разработчик не протестировал миграцию на staging
- SRE инженер не проверил заполнение кэша
- Отсутствие документации по процедуре отката
- Нет on-call ротации для быстрого реагирования
- Недостаточное обучение работе с Redis

ПРОЦЕСС:

- Ручной деплой без автоматических проверок
- Нет обязательного code review для изменений БД
- Отсутствие плана восстановления после сбоя
- Нет процедуры регулярного аудита кэша
- Автоматическое развертывание без canary

ИНСТРУМЕНТЫ:

- Нет мониторинга размера Redis
- Отсутствие алертов при заполнении памяти >80%
- Нет автоматического отката при неудаче
- Отсутствие дашбордов для отслеживания TTL
- Нет системы автоматической очистки кэша

ДАННЫЕ:

- Нет информации о пиковых нагрузках на кэш
- Отсутствие метрик по размеру кэша
- Нет данных о времени жизни записей
- Отсутствие логирования операций очистки

СРЕДА:

- Staging среда не соответствует production
- Высокая нагрузка во время деплоя
- Ограниченные ресурсы Redis (память)
- Отсутствие резервного Redis-инстанса

КОММУНИКАЦИЯ:

- Разработка не уведомила SRE об изменении TTL
- Отсутствие канала для быстрого оповещения об инцидентах
- Нет регулярных встреч по ретроспективе
- Задержка в эскалации проблемы

Шаг 4. Выявление коренных причин

На основе анализа диаграммы выделены следующие коренные причины:

№	Коренная причина	Категория
1	Отсутствие мониторинга и алертов на заполнение Redis	Инструменты
2	Staging среда не соответствует production	Среда

3	Нет процедуры code review для изменений конфигурации	Процесс
4	Отсутствие on-call ротации	Человек

3.3. Разработка корректирующих мер

№	Коренная причина	Корректирующая мера	Ответственный	Срок	Приоритет
1	Отсутствие мониторинга Redis	Настроить Prometheus + Grafana для Redis	SRE	2 дня	HIGH
2	Отсутствие алертов	Добавить алерт при заполнении >80%	SRE	2 дня	HIGH
3	Staging ≠ Production	Синхронизировать конфигурации окружений	DevOps	1 неделя	HIGH
4	Нет code review	Внедрить обязательный code review для конфигураций	Tech Lead	3 дня	MEDIUM
5	Нет on-call ротации	Внедрить on-call ротацию с PagerDuty	SRE Lead	2 недели	MEDIUM

3.4. Сравнение с методом 5 Whys (из ЛР №5)

Характеристика	Метод 5 Whys	Диаграмма Исикавы
Коренная причина	Отсутствие мониторинга и алертов на заполнение Redis	-
Выявлено коренных причин	1	4
Охват	Узкий, глубокая одна причина	Широкий, системный взгляд
Результат	Точечное решение	Комплексный план улучшений

Вывод: Два метода дополняют друг друга. Метод 5 Whys позволил выявить основную техническую причину, а диаграмма Исикавы - выявить дополнительные процессные и организационные причины.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ (УРОВЕНЬ 3)

```
"""
Лабораторная работа №6
Программная генерация диаграммы Исикавы (текстовый формат)
"""

from typing import Dict, List, Any
from dataclasses import dataclass, field

@dataclass
class FishboneCause:
    """Причина в диаграмме Исикавы"""
    description: str
    subcauses: List[str] = field(default_factory=list)

@dataclass
class FishboneCategory:
    """Категория причин"""
    name: str
    causes: List[FishboneCause] = field(default_factory=list)

@dataclass
class FishboneDiagram:
    """Диаграмма Исикавы"""
    problem: str
    categories: List[FishboneCategory] = field(default_factory=list)
    root_causes: List[str] = field(default_factory=list)

class FishboneGenerator:
    """Генератор диаграммы Исикавы"""

    @staticmethod
    def to_text(diagram: FishboneDiagram) -> str:
        """Экспорт в текстовый формат (аналог визуальной диаграммы)"""
        text = f"""
{'='*60}
ДИАГРАММА ИСИКАВЫ
{'='*60}
ПРОБЛЕМА: {diagram.problem}
{'='*60}

        """
        for category in diagram.categories:
            text += f"\n☐ {category.name.upper()}: \n"
            for cause in category.causes:
                text += f"├─ {cause.description}\n"
                for subcause in cause.subcauses:
                    text += f"│   └─ {subcause}\n"
        return text
    """
```



```

        "subcauses": cause.subcauses
    }
    for cause in cat.causes
    ]
    }
    for cat in diagram.categories
],
"root_causes": diagram.root_causes
}

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ (на основе инцидента из ЛР №5)
# =====

# Создание диаграммы
diagram = FishboneDiagram(
    problem="Сбой сервиса авторизации 25.03.2025 (47 минут простоя)",
    categories=[
        FishboneCategory(
            name="ЧЕЛОВЕК",
            causes=[
                FishboneCause("Разработчик не протестировал миграцию на
staging"),
                FishboneCause("SRE инженер не проверил заполнение кэша"),
                FishboneCause("Отсутствие документации по процедуре отката"),
                FishboneCause("Нет on-call ротации для быстрого реагирования")
            ]
        ),
        FishboneCategory(
            name="ПРОЦЕСС",
            causes=[
                FishboneCause("Ручной деплой без автоматических проверок"),
                FishboneCause("Нет обязательного code review для изменений БД"),
                FishboneCause("Отсутствие плана восстановления после сбоя")
            ]
        ),
        FishboneCategory(
            name="ИНСТРУМЕНТЫ",
            causes=[
                FishboneCause("Нет мониторинга размера Redis"),
                FishboneCause("Отсутствие алертов при заполнении памяти >80%"),
                FishboneCause("Нет автоматического отката при неудаче")
            ]
        ),
        FishboneCategory(
            name="СРЕДА",
            causes=[
                FishboneCause("Staging среда не соответствует production"),
                FishboneCause("Ограниченные ресурсы Redis (память)")
            ]
        )
    ]
)

```



```

    ]
    )
],
root_causes=[
    "Отсутствие мониторинга и алертов на заполнение Redis",
    "Staging среда не соответствует production",
    "Нет процедуры code review для изменений конфигурации",
    "Отсутствие on-call ротации"
]
)

# Генерация и вывод
generator = FishboneGenerator()

print(generator.to_text(diagram))
print("\n" + generator.to_markdown(diagram))

# Сохранение в файлы
with open("fishbone_diagram.txt", "w", encoding="utf-8") as f:
    f.write(generator.to_text(diagram))

with open("fishbone_diagram.md", "w", encoding="utf-8") as f:
    f.write(generator.to_markdown(diagram))

import json
with open("fishbone_diagram.json", "w", encoding="utf-8") as f:
    json.dump(generator.to_json(diagram), f, indent=2, ensure_ascii=False)

print("\n✅ Файлы сохранены:")
print("  - fishbone_diagram.txt")
print("  - fishbone_diagram.md")
print("  - fishbone_diagram.json")

```

5. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ (15 вариантов)

5.1. Уровень 1

Задание: Построить диаграмму Исикавы для заданного инцидента (5+ причин, 3+ категории), выявить 1-2 коренные причины.

№	Описание инцидента	Примечание
1	Сбой БД из-за блокировки на 30 минут	Новый инцидент
2	Утечка памяти в приложении, приведшая к ООМ	Новый инцидент

3	Неудачное развертывание новой версии (откат 45 минут)	Новый инцидент
4	DDoS-атака, вызвавшая недоступность API	Новый инцидент
5	Проблемы с SSL-сертификатом (истек срок)	Новый инцидент

5.2. Уровень 2

Задание: Построить диаграмму Исикавы (10+ причин, 4+ категории), выявить коренные причины (3+), разработать корректирующие меры с приоритетами.

№	Описание инцидента	Примечание
1	Каскадный отказ микросервисов (15 минут)	Новый инцидент
2	Потеря данных из-за неправильной миграции	Новый инцидент
3	Сбой CI/CD пайплайна (2 часа)	Новый инцидент
4	Инцидент безопасности (компрометация ключей)	Новый инцидент
5	Специализация БПЛА: Потеря связи с дроном	Новый инцидент

5.3. Уровень 3 (варианты 11-15)

Задание: Построить диаграмму Исикавы, создать Python-скрипт для генерации отчёта, выполнить сравнительный анализ с методом 5 Whys (из ЛР №5).

№	Описание инцидента	Примечание
1	Многочасовой сбой production (3+ часа)	Новый инцидент (или инцидент ЛР №5)
2	Инцидент с финансовыми потерями	Новый инцидент (или инцидент ЛР №5)
3	Инцидент с участием 3+ команд	Новый инцидент (или инцидент ЛР №5)
4	Инцидент в распределённой системе	Новый инцидент (или инцидент ЛР №5)

		инцидент ЛР №5)
5	Комплексный инцидент (множество причин)	Новый инцидент (или инцидент ЛР №5)

Примечание для уровня 3: Студент может использовать тот же инцидент, что и в ЛР №5, для выполнения сравнительного анализа двух методов RCA.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое диаграмма Исикавы и для чего она используется?
2. Каковы основные элементы диаграммы Исикавы?
3. Какие категории причин наиболее часто используются для IT-инцидентов?
4. В чем отличие диаграммы Исикавы от метода 5 Whys?
5. Как диаграмма Исикавы и метод 5 Whys могут дополнять друг друга?
6. Сколько категорий рекомендуется использовать в диаграмме?
7. Какие типичные ошибки допускают при построении диаграммы Исикавы?
8. Как на основе диаграммы Исикавы выделить коренные причины?
9. Можно ли использовать диаграмму Исикавы для проактивного анализа рисков?
10. Как визуализировать диаграмму Исикавы в отчете?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (диаграмма Исикавы, категории, правила построения).
4. Практическая часть:
 - Описание инцидента.
 - Построенная диаграмма Исикавы (текстовое описание или схема).

- Место для скриншота: схема диаграммы.
 - Выявленные коренные причины.
 - Таблица корректирующих мер.
5. Программная часть (Уровень 3):
- Листинг кода Python.
 - Место для скриншота: вывод программы.
 - Место для скриншота: сгенерированные файлы.
6. Сравнительный анализ (Уровень 3): сравнение результатов ЛР №5 и ЛР №6.
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Построение диаграммы Исикавы (5+/10+ причин)	2 балла	2 балла	2 балла
Количество категорий (3+/4+)	2 балла	2 балла	2 балла
Выявление коренных причин (1-2/3+)	1 балл	2 балла	2 балла
Разработка корректирующих мер	0 баллов	1 балл	2 балла
Приоритизация мер (HIGH/MEDIUM/LOW)	0 баллов	1 балл	2 балла
Программная генерация диаграммы	0 баллов	0 баллов	2 балла
Сравнительный анализ с 5 Whys	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 7)	Уровень 2 (макс. 10)	Уровень 3 (макс. 16)
5 (отлично)	6-7	9-10	14-16
4 (хорошо)	5	7-8	11-13
3 (удовлетворительно)	3-4	5-6	8-10
2 (неудовлетворительно)	<3	<5	<8

ЛАБОРАТОРНАЯ РАБОТА №7

Документирование инцидентов и распространение уроков

1. ЦЕЛЬ РАБОТЫ: Сформировать практические навыки создания шаблонов документации инцидентов, разработки корректирующих мер (action items) с назначением ответственных и сроков, интеграции уроков из инцидентов в процессы разработки и эксплуатации, а также создания системы распространения знаний для предотвращения повторения аналогичных проблем.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

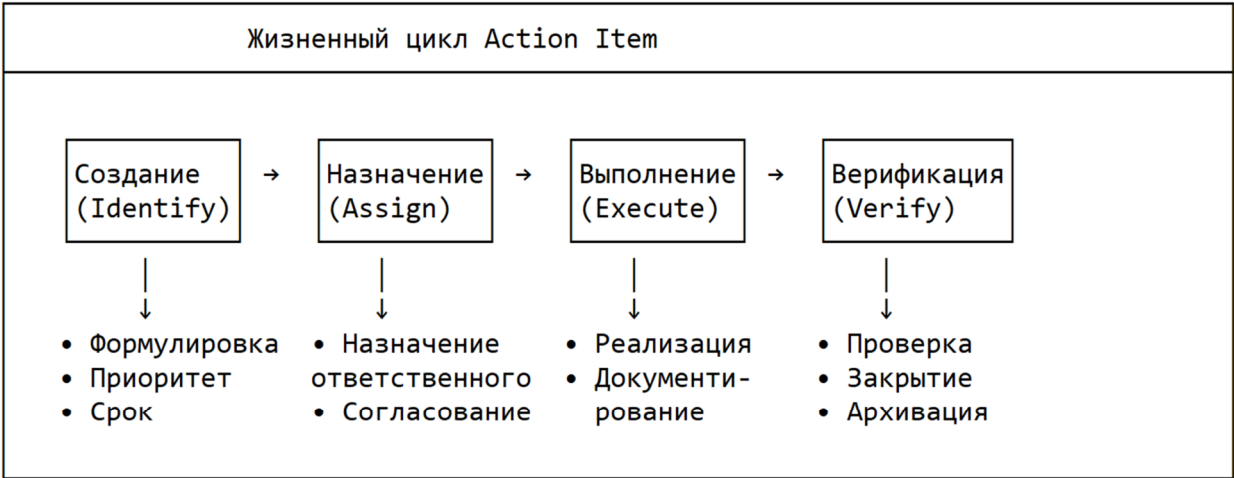
2.1. Цели и задачи документирования инцидентов

Документирование инцидентов - это процесс систематической фиксации информации о произошедших инцидентах, их причинах, последствиях и принятых мерах для предотвращения повторения.

Основные цели документирования:

Цель	Описание
Создание базы знаний	Накопление информации для обучения команды
Предотвращение повторения	Выработка и отслеживание корректирующих мер
Анализ трендов	Выявление повторяющихся проблем
Соответствие требованиям	Аудит, compliance (ISO 27001, SOC 2)
Улучшение процессов	Непрерывное совершенствование

2.2. Жизненный цикл корректирующей меры (Action Item)



2.3. Стандарты и шаблоны документации инцидентов

Структура action item (корректирующей меры):

Поле	Описание	Пример
ID	Уникальный идентификатор	AI-2025-03-25-001
Описание	Чёткое описание действия	Настроить мониторинг Redis
Коренная причина	Ссылка на RCA	Отсутствие мониторинга памяти
Тип	Процессная / Техническая/ Организационная	Техническая
Ответственный	ФИО или роль	SRE команда
Срок выполнения	Дата завершения	28.03.2025
Приоритет	HIGH / MEDIUM / LOW	HIGH
Статус	Open / In Progress / Done / Cancelled	In Progress

Результат	Что было сделано	Prometheus + Grafana настроены
Дата верификации	Дата проверки	29.03.2025
Верификатор	Кто проверял	Tech Lead

2.4. Системы трекинга корректирующих мер

Система	Тип	Особенности
Jira	Дефект-трекер	Мощный workflow, интеграции, кастомизация
GitHub Issues	Дефект-трекер	Интеграция с кодом, бесплатно для open-source
GitLab Issues	Дефект-трекер	Встроенный в CI/CD, бесплатный
YouTrack	Дефект-трекер	Удобный поиск, Agile-доски
Trello	Управление задачами	Простой, визуальный
Asana	Управление проектами	Командная работа, сроки
Confluence (таблицы)	База знаний	Хорошо для документации

2.5. Методы распространения уроков

Метод	Описание	Частота
Postmortem-сессии	Командный разбор инцидента	После каждого P0/P1
Retro-сессии	Регулярные встречи по улучшению процессов	Раз в 1-2 недели
База знаний (Wiki)	Централизованное хранение	Непрерывно

	postmortem-отчётов	
Технические блоги	Публикация уроков для wider audience	По мере необходимости
Рассылки (Newsletter)	Дайджест инцидентов для команды	Еженедельно/ежемесячно
Training sessions	Обучение на основе ошибок	По запросу

2.6. Культура обучения на ошибках (Blameless Culture)

Принципы blameless culture в контексте распространения уроков:

Принцип	Описание
Отсутствие наказаний	Инциденты - возможность для улучшения, а не повод для наказания
Прозрачность	Все postmortem доступны всей команде
Открытость	Поощряется сообщение о проблемах
Фокус на систему	Вопрос «почему система допускает ошибки?» вместо «кто ошибся?»
Обучение	Каждый инцидент становится учебным материалом

2.7. Метрики эффективности пост-инцидентного анализа

Метрика	Формула	Целевое значение
Время закрытия action item	Дата завершения - Дата создания	< 30 дней
Процент выполненных action items	$(\text{Выполнено} / \text{Всего}) \times 100\%$	> 90%
Количество повторных инцидентов	Счётчик повторных инцидентов того же типа	0

Postmortem completion rate	(Постмортемы проведены / Всего P0-P1) $\times$ 100%	100%
MTTR снижение	(Старый MTTR — Новый MTTR) / Старый MTTR $\times$ 100%	Положительный тренд

Пояснения к метрикам:

Метрика	Описание
Время закрытия action item	Показывает, насколько быстро команда реализует корректирующие меры. Чем меньше, тем быстрее устраняются коренные причины.
Процент выполненных action items	Отражает дисциплину выполнения принятых решений. Низкий процент говорит о проблемах с приоритизацией или ресурсами.
Количество повторных инцидентов	Ключевая метрика эффективности postmortem. Если один и тот же инцидент повторяется — значит, коренная причина не была устранена.
Postmortem completion rate	Доля критических инцидентов (P0/P1), по которым проведён полноценный анализ. Должна стремиться к 100%.
MTTR снижение	Показывает, насколько успешно корректирующие меры влияют на скорость восстановления сервиса после отказов.

2.8. РЕКОМЕНДАЦИИ ПО ИМПОРТОЗАМЕЩЕНИЮ ДЛЯ ЛР №7

При выполнении лабораторной работы №7 могут использоваться как исходные инструменты (Jira, GitHub Issues), так и их российские альтернативы.

Исходный инструмент	Российская альтернатива	Статус	Особенности замены
Jira	YouTrack (JetBrains)	Доступен	Бесплатная версия для малых команд (до 10 пользователей), on-premise установка. Интерфейс аналогичен Jira, поддерживает Agile-доски и кастомизацию workflow.
	Kaiten	В реестре	Российская система управления проектами, поддерживает трекинг задач, диаграммы Ганта, интеграции с Telegram
	Projecto	В реестре	Прямой аналог Jira+Confluence, подходит для корпоративного использования
GitHub Issues	GitFlic Issues	В реестре	Встроенная система трекинга задач в российской DevOps-платформе GitFlic. Полностью заменяет GitHub Issues
	GitLab Issues	Доступен	Также может использоваться как альтернатива

Рекомендации по выбору:

- Для выполнения лабораторной работы допустимо использование любого инструмента трекинга задач, включая простые таблицы Excel или Google Sheets.
- При выборе российской альтернативы функциональность и последовательность действий сохраняются.
- В отчёте необходимо явно указать, какой инструмент был использован.

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные (коренные причины из ЛР №5 и №6)

На основе анализа инцидента (ЛР №5 и №6) были выявлены следующие коренные причины:

№	Коренная причина	Категория
1	Отсутствие мониторинга и алертов на заполнение Redis	Инструменты
2	Staging среда не соответствует production	Среда
3	Нет процедуры code review для изменений конфигурации	Процесс
4	Отсутствие on-call ротации	Организационная

3.2. Создание action items

Action Item №1: Настройка мониторинга Redis

Поле	Значение
ID	AI-2025-03-25-001
Описание	Настроить мониторинг размера Redis в Prometheus и алерт при заполнении >80%
Коренная причина	Отсутствие мониторинга и алертов на заполнение

	Redis
Тип	Техническая
Ответственный	SRE команда
Срок выполнения	28.03.2025 (3 дня)
Приоритет	HIGH
Статус	Done
Результат	Добавлен Redis exporter, настроены дашборды в Grafana, создан алерт в Alertmanager
Дата верификации	29.03.2025
Верификатор	Tech Lead

Action Item №2: Синхронизация окружений

Поле	Значение
ID	AI-2025-03-25-002
Описание	Синхронизировать конфигурации staging и production окружений
Коренная причина	Staging среда не соответствует production
Тип	Техническая
Ответственный	DevOps команда
Срок выполнения	01.04.2025 (1 неделя)
Приоритет	HIGH
Статус	In Progress
Результат	—
Дата верификации	—
Верификатор	—

Action Item №3: Внедрение code review для конфигураций

Поле	Значение

ID	AI-2025-03-25-003
Описание	Внедрить обязательный code review для всех изменений конфигураций (БД, Redis, инфраструктура)
Коренная причина	Нет процедуры code review для изменений конфигурации
Тип	Процессная
Ответственный	Tech Lead
Срок выполнения	04.04.2025 (1.5 недели)
Приоритет	MEDIUM
Статус	Open
Результат	—
Дата верификации	—
Верификатор	—

Action Item №4: Внедрение on-call ротации

Поле	Значение
ID	AI-2025-03-25-004
Описание	Внедрить on-call ротацию с использованием PagerDuty/Opsgenie или российского аналога Alert24
Коренная причина	Отсутствие on-call ротации
Тип	Организационная
Ответственный	SRE Lead
Срок выполнения	08.04.2025 (2 недели)
Приоритет	MEDIUM
Статус	Open
Результат	—
Дата верификации	—

Верификатор	—

3.3. План распространения уроков

План распространения уроков по инциденту №2025-03-25

№	Мероприятие	Аудитория	Формат	Ответственный	Срок
1	Postmortem-сессия	SRE + Dev команды	Встреча (1 час)	Иванов П.	26.03.2025 5
2	Публикация postmortem-отчёта	Все разработчики и	Wiki (Confluence / GitFlic Wiki)	Петрова А.	27.03.2025 5
3	Рассылка lessons learned	Все сотрудники IT	Email	Сидоров В.	28.03.2025 5
4	Обновление runbook	SRE команда	Документация	Иванов П.	29.03.2025 5
5	Тренинг по Redis мониторингу	SRE команда	Workshop (2 часа)	Смирнов Д.	01.04.2025 5
6	Внесение в retrospective	Scrum команда	Sprint retro	Product Owner	04.04.2025 5

3.4. Программная реализация (Уровень 3)

```

"""
Лабораторная работа №7
Автоматическая генерация action items из postmortem-отчёта
(с поддержкой экспорта в форматы для различных систем трекинга)

```

```

"""

import json
from datetime import datetime, timedelta
from typing import List, Dict, Any
from dataclasses import dataclass, asdict
from enum import Enum

class Priority(Enum):
    HIGH = "HIGH"
    MEDIUM = "MEDIUM"
    LOW = "LOW"

class Status(Enum):
    OPEN = "Open"
    IN_PROGRESS = "In Progress"
    DONE = "Done"
    CANCELLED = "Cancelled"

class ActionType(Enum):
    TECHNICAL = "Техническая"
    PROCESS = "Процессная"
    ORGANIZATIONAL = "Организационная"

@dataclass
class ActionItem:
    """Корректирующая мера (Action Item)"""
    id: str
    description: str
    root_cause: str
    action_type: ActionType
    assignee: str
    deadline: str
    priority: Priority
    status: Status
    result: str = ""
    verification_date: str = ""
    verifier: str = ""

@dataclass
class LessonPlan:
    """План распространения урока"""
    activity: str
    audience: str
    format: str
    owner: str
    deadline: str

class ActionItemGenerator:

```

```

"""Генератор action items из postmortem-отчёта"""

def __init__(self, incident_id: str, incident_date: datetime):
    self.incident_id = incident_id
    self.incident_date = incident_date
    self.action_items: List[ActionItem] = []
    self.lesson_plans: List[LessonPlan] = []

def add_action_item(self, description: str, root_cause: str,
                    action_type: ActionType, assignee: str,
                    days_to_deadline: int, priority: Priority) -> ActionItem:
    """Добавление action item"""
    deadline = (self.incident_date +
timedelta(days=days_to_deadline)).strftime("%Y-%m-%d")

    action_id = f"AI-{self.incident_id}-{len(self.action_items)+1:03d}"

    action = ActionItem(
        id=action_id,
        description=description,
        root_cause=root_cause,
        action_type=action_type,
        assignee=assignee,
        deadline=deadline,
        priority=priority,
        status=Status.OPEN
    )
    self.action_items.append(action)
    return action

def add_lesson_plan(self, activity: str, audience: str,
                    format: str, owner: str, days_to_deadline: int) ->
LessonPlan:
    """Добавление плана распространения урока"""
    deadline = (self.incident_date +
timedelta(days=days_to_deadline)).strftime("%Y-%m-%d")

    lesson = LessonPlan(
        activity=activity,
        audience=audience,
        format=format,
        owner=owner,
        deadline=deadline
    )
    self.lesson_plans.append(lesson)
    return lesson

def to_markdown(self) -> str:
    """Экспорт в Markdown"""

```



```

        md = f"""# Post-инцидентные корректирующие меры (Action Items)
## Инцидент: {self.incident_id}
## Дата: {self.incident_date.strftime("%Y-%m-%d")}

## 📋 Action Items

| ID | Описание | Тип | Ответственный | Срок | Приоритет | Статус |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
"""

        for action in self.action_items:
            md += f"| {action.id} | {action.description} |
{action.action_type.value} | {action.assignee} | {action.deadline} |
{action.priority.value} | {action.status.value} |\n"

        md += f"""

## 📅 План распространения уроков

| № | Мероприятие | Аудитория | Формат | Ответственный | Срок |
| :--- | :--- | :--- | :--- | :--- | :--- |
"""

        for i, lesson in enumerate(self.lesson_plans, 1):
            md += f"| {i} | {lesson.activity} | {lesson.audience} |
{lesson.format} | {lesson.owner} | {lesson.deadline} |\n"

        return md

    def to_json(self) -> str:
        """Экспорт в JSON"""
        return json.dumps({
            "incident_id": self.incident_id,
            "incident_date": self.incident_date.isoformat(),
            "action_items": [asdict(action) for action in self.action_items],
            "lesson_plans": [asdict(lesson) for lesson in self.lesson_plans]
        }, indent=2, ensure_ascii=False, default=str)

    def to_csv(self) -> str:
        """Экспорт в CSV (для импорта в Jira / YouTrack / Excel)"""
        csv_lines = ["ID,Описание,Коренная
причина,Тип,Ответственный,Срок,Приоритет,Статус"]
        for action in self.action_items:
            csv_lines.append(
                f"{action.id},{action.description},{action.root_cause},{action.ac
tion_type.value},"
                f"{action.assignee},{action.deadline},{action.priority.value},{ac
tion.status.value}"
            )
        return "\n".join(csv_lines)

    def to_youtrack_format(self) -> str:

```

```

        """Экспорт в формат для импорта в YouTrack (CSV)"""
        csv_lines = ["summary,description,assignee,priority"]
        for action in self.action_items:
            csv_lines.append(
                f"\ \"{action.description}\",\ \"Коренная причина: {action.root_cause}\",\"
                f\"{action.assignee},{action.priority.value}\"
            )
        return "\n".join(csv_lines)

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ (на основе инцидента из ЛР №5 и №6)
# =====

if __name__ == "__main__":
    # Инициация генератора
    generator = ActionItemGenerator(
        incident_id="2025-03-25",
        incident_date=datetime(2025, 3, 25)
    )

    # Добавление action items
    generator.add_action_item(
        description="Настроить мониторинг размера Redis в Prometheus и алерт при
        заполнении >80%",
        root_cause="Отсутствие мониторинга и алертов на заполнение Redis",
        action_type=ActionType.TECHNICAL,
        assignee="SRE команда",
        days_to_deadline=3,
        priority=Priority.HIGH
    )

    generator.add_action_item(
        description="Синхронизировать конфигурации staging и production
        окружений",
        root_cause="Staging среда не соответствует production",
        action_type=ActionType.TECHNICAL,
        assignee="DevOps команда",
        days_to_deadline=7,
        priority=Priority.HIGH
    )

    generator.add_action_item(
        description="Внедрить обязательный code review для изменений
        конфигураций",
        root_cause="Нет процедуры code review для изменений конфигурации",
        action_type=ActionType.PROCESS,
        assignee="Tech Lead",
        days_to_deadline=10,

```

```

        priority=Priority.MEDIUM
    )

    generator.add_action_item(
        description="Внедрить on-call ротацию с использованием
PagerDuty/YouTrack/Alert24",
        root_cause="Отсутствие on-call ротации",
        action_type=ActionType.ORGANIZATIONAL,
        assignee="SRE Lead",
        days_to_deadline=14,
        priority=Priority.MEDIUM
    )

# Добавление плана распространения уроков
generator.add_lesson_plan(
    activity="Postmortem-сессия",
    audience="SRE + Dev команды",
    format="Встреча (1 час)",
    owner="Иванов П.",
    days_to_deadline=1
)

generator.add_lesson_plan(
    activity="Публикация postmortem-отчёта",
    audience="Все разработчики",
    format="Wiki (Confluence/GitFlic Wiki/Kaiten)",
    owner="Петрова А.",
    days_to_deadline=2
)

generator.add_lesson_plan(
    activity="Расслыка lessons learned",
    audience="Все сотрудники IT",
    format="Email",
    owner="Сидоров В.",
    days_to_deadline=3
)

generator.add_lesson_plan(
    activity="Тренинг по Redis мониторингу",
    audience="SRE команда",
    format="Workshop (2 часа)",
    owner="Смирнов Д.",
    days_to_deadline=7
)

# Вывод и сохранение
print(generator.to_markdown())

```

```

with open("action_items_report.md", "w", encoding="utf-8") as f:
    f.write(generator.to_markdown())

with open("action_items_report.json", "w", encoding="utf-8") as f:
    f.write(generator.to_json())

with open("action_items_report.csv", "w", encoding="utf-8") as f:
    f.write(generator.to_csv())

with open("youtrack_import.csv", "w", encoding="utf-8") as f:
    f.write(generator.to_youtrack_format())

print("\n Файлы сохранены:")
print("    - action_items_report.md")
print("    - action_items_report.json")
print("    - action_items_report.csv (для импорта в Jira/Excel)")
print("    - youtrack_import.csv (для импорта в YouTrack)")

```

3.5. Интеграция с Jira / YouTrack / Kaiten (имитация API)

```

"""
Лабораторная работа №7 (Уровень 3)
Имитация интеграции с системами трекинга (Jira / YouTrack / Kaiten)
"""

import requests
import json
from typing import Dict, Any, List

class IssueTrackerClient:
    """
    Имитация клиента для систем трекинга задач.
    В реальной реализации может быть заменён на API Jira, YouTrack, Kaiten или
    GitFlic.
    """

    def __init__(self, system_name: str = "YouTrack"):
        self.system_name = system_name
        self.created_issues = []

        print(f"✅ Инициализирован клиент для системы: {system_name}")
        print(f"    (для учебных целей используется имитация API)")

    def create_issue(self, summary: str, description: str,
                    assignee: str, priority: str) -> Dict[str, Any]:
        """Создание задачи в системе трекинга (имитация)"""

        issue = {

```

```

        "id": f"{self.system_name.upper()}-{len(self.created_issues) + 1}",
        "key": f"{self.system_name.upper()}-{len(self.created_issues) + 1}",
        "summary": summary,
        "description": description,
        "assignee": assignee,
        "priority": priority,
        "status": "Open",
        "created": "2025-03-25T10:00:00Z"
    }

    # Для YouTrack добавляем ссылку
    if self.system_name.lower() == "youtrack":
        issue["url"] = f"https://my.youtrack.cloud/issue/{issue['key']}"
    elif self.system_name.lower() == "kaiten":
        issue["url"] = f"https://kaiten.ru/space/issue/{issue['id']}"
    elif self.system_name.lower() == "jira":
        issue["url"] = f"https://jira.example.com/browse/{issue['key']}"
    else:
        issue["url"] =
f"https://{self.system_name.lower()}.example.com/issues/{issue['id']}"

    self.created_issues.append(issue)

    print(f"✅ Создана задача в {self.system_name}: {issue['key']}")
    print(f"    URL: {issue['url']}")

    return issue

def get_issues(self) -> List[Dict[str, Any]]:
    """Получение списка задач"""
    return self.created_issues

# Пример использования
if __name__ == "__main__":
    print("\n" + "=" * 60)
    print("ИМИТАЦИЯ ИНТЕГРАЦИИ С СИСТЕМАМИ ТРЕКИНГА")
    print("=" * 60)

    # Пример для YouTrack (рекомендуемая российская альтернатива)
    print("\n🔗 YouTrack:")
    youtrack = IssueTrackerClient("YouTrack")
    youtrack.create_issue(
        summary="[Postmortem] Настроить мониторинг Redis",
        description="Настроить Prometheus exporter для Redis и алерт при  
заполнении >80%",
        assignee="sre-team",
        priority="High"
    )
    youtrack.create_issue(

```

```

        summary="[Postmortem] Синхронизировать окружения",
        description="Синхронизировать конфигурации staging и production",
        assignee="devops-team",
        priority="High"
    )

    print(f"\n📊 Всего создано задач в {youtrack.system_name}: {len(youtrack.get_issues())}")

    # Пример для Kaiten (российская альтернатива)
    print("\n🔗 Kaiten:")
    kaiten = IssueTrackerClient("Kaiten")
    kaiten.create_issue(
        summary="[Postmortem] Code review для конфигураций",
        description="Внедрить обязательный code review для изменений конфигураций",
        assignee="tech-lead",
        priority="Medium"
    )

    print("\n💡 Примечание: в реальной среде используются API соответствующих систем.")
    print("    YouTrack предоставляет REST API, GitFlic имеет встроенную систему Issues.")

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

Задание: Создать шаблон action item, сформулировать 3-5 action items по готовым коренным причинам, заполнить таблицу с ответственным и сроком.

Варианты инцидентов - по выбору из ЛР №5/6 или новых.

№	Исходные данные (коренные причины)
1	3 коренные причины из ЛР №5 или №6
2	4 коренные причины из ЛР №5 или №6

№	Исходные данные (коренные причины)
3	5 коренные причины из ЛР №5 или №6
4	Коренные причины из нового инцидента (3 причины)
5	Коренные причины из нового инцидента (4 причины)

4.2. Уровень 2

Задание: Создать action items (5+), интегрировать в систему трекинга (имитация), разработать план распространения уроков для команды.

№	Исходные данные	Дополнительные требования
1	Коренные причины из ЛР №5 или №6 (5+ причин)	План распространения уроков
2	Коренные причины из нового инцидента (5+ причин)	Интеграция с Jira (имитация)
3	Коренные причины из ЛР №5 или №6	Интеграция с GitHub Issues
4	Коренные причины из нового инцидента	База знаний в Confluence (имитация)
5	Специализация БПЛА	План распространения для команды дронов

При выполнении рекомендуется использовать российские альтернативы: YouTrack, Kaiten или GitFlic Issues.

4.3. Уровень 3

Задание: Создать Python-скрипт для автоматической генерации action items из postmortem-отчёта (JSON/Markdown) с экспортом в CSV для импорта в Jira/YouTrack/Kaiten.

№	Задача	Дополнительные требования
1	Python-скрипт: генерация action items из JSON (ЛР №5)	Сохранение в Markdown
2	Python-скрипт: генерация action items + Jira API (имитация)	Создание задач
3	Python-скрипт: создание postmortem-библиотеки в Markdown	Генерация индекса
4	Python-скрипт: экспорт action items в CSV для импорта в Jira	Формат CSV
5	Комплексный: всё вышеперечисленное	Полный отчёт

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие цели преследует документирование инцидентов?
2. Какие поля обязательно должны быть в action item?
3. Какие системы трекинга корректирующих мер вы знаете?
4. Какие методы распространения уроков наиболее эффективны?

5. Что такое blameless culture и как она связана с postmortem?
6. Какие метрики используются для оценки эффективности пост-инцидентного анализа?
7. Как определить приоритет action item (HIGH/MEDIUM/LOW)?
8. Что делать, если action item не выполнен в срок?
9. Как часто следует проводить postmortem-сессии?
10. Как интегрировать уроки из инцидентов в процессы разработки?
11. Какие российские альтернативы Jira и GitHub Issues существуют? Каковы их преимущества?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (документирование инцидентов, action items, распространение уроков).
4. Практическая часть:
 - Исходные данные (коренные причины из ЛР №5/№6 или нового инцидента).
 - Таблица action items.
 - План распространения уроков.
 - Место для скриншота: action items в выбранной системе трекинга (Jira / YouTrack / Kaiten / GitFlic Issues).
 - Место для скриншота: postmortem-библиотека (Confluence / GitFlic Wiki / Kaiten).
5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.

- Место для скриншота: сгенерированные файлы (включая CSV для импорта).

6. Анализ результатов (оценка выполнимости, приоритетов, выбор инструмента).

7. Ответы на контрольные вопросы (3-5).

8. Вывод (с указанием использованных инструментов).

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Создание action items (3-5/5+)	2 балла	2 балла	2 балла
Заполнение всех полей action item	2 балла	2 балла	2 балла
Назначение ответственных и сроков	1 балл	2 балла	2 балла
Приоритизация мер	1 балл	2 балла	2 балла
План распространения уроков	0 баллов	2 балла	2 балла
Интеграция с системой трекинга	0 баллов	1 балл	2 балла
Программная генерация отчёта (с CSV экспортом)	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 13)	Уровень 3 (макс. 16)
5 (отлично)	7-8	12-13	14-16
4 (хорошо)	5-6	9-11	11-13
3 (удовлетворительно)	3-4	6-8	8-10

2 (неудовлетворительно)	<3	<6	<8

8. ДОПОЛНИТЕЛЬНЫЙ РАЗДЕЛ: ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ РОССИЙСКИХ АЛЬТЕРНАТИВ

8.1. Выполнение ЛР №7 с использованием YouTrack

YouTrack - российская система управления проектами от компании JetBrains.

Доступна бесплатная версия для малых команд (до 10 пользователей).

Особенности работы с YouTrack в контексте ЛР №7:

- Создание проектов и задач аналогично Jira.
- Поддержка Agile-досок (Scrum, Kanban).
- Возможность кастомизации workflow.
- Импорт задач через CSV (скрипт из п. 3.4 генерирует файл youtrack_import.csv).

Инструкция по импорту в YouTrack:

1. Зайти в YouTrack → Projects → Создать проект.
2. Выбрать «Импорт задач».
3. Загрузить сгенерированный файл youtrack_import.csv.
4. Назначить ответственных и приоритеты.

8.2. Выполнение ЛР №7 с использованием GitFlic Issues

GitFlic - российская DevOps-платформа, включающая Git-репозитории, CI/CD и систему Issues.

Особенности работы с GitFlic Issues:

- Интерфейс аналогичен GitHub Issues.
- Интеграция с Git-репозиториями (можно привязывать задачи к коммитам).
- Поддержка меток, milestone, назначенных пользователей.

Инструкция по созданию задач в GitFlic:

1. Зарегистрироваться на <https://gitflic.ru/>.
2. Создать проект (репозиторий).
3. Перейти в раздел «Задачи» (Issues).
4. Создать задачи вручную или через API.

8.3. Выполнение ЛР №7 с использованием Kaiten

Kaiten - российская система управления проектами, внесённая в реестр отечественного ПО.

Особенности работы с Kaiten:

- Визуальные доски для управления задачами.
- Диаграммы Ганта, трекинг времени.
- Интеграция с Telegram для уведомлений.

ЛАБОРАТОРНАЯ РАБОТА №8

Проектирование эксперимента по проверке устойчивости

1. ЦЕЛЬ РАБОТЫ: Сформировать практические навыки проектирования экспериментов по проверке устойчивости программных систем: определения гипотез, целей, метрик успеха и условий проведения экспериментов, а также документирования плана эксперимента и оценки потенциальных рисков (blast radius).

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Что такое эксперимент по проверке устойчивости

Эксперимент по проверке устойчивости (resilience experiment) - это контролируемое испытание, в ходе которого в систему искусственно вносятся сбои (отказы, задержки, ошибки) для проверки её способности сохранять работоспособность и восстанавливаться после нарушений.

Ключевые элементы эксперимента:

Элемент	Описание	Пример
Гипотеза	Предположение о поведении системы при сбое	При отказе одного пода доступность сохранится >99.9%
Steady state	Нормальное поведение системы до эксперимента	QPS >1000, error rate <0.1%
Экспериментальное воздействие	Искусственно вносимый сбой	Pod kill, сетевая задержка, нагрузка CPU
Наблюдение	Сбор метрик во время эксперимента	Мониторинг через Prometheus
Анализ	Сравнение поведения с гипотезой	Успех/провал эксперимента

2.2. Принципы проектирования экспериментов

Принцип	Описание
Определение steady state	Сначала необходимо понять, как система ведёт себя в норме (метрики, логи, трейсы)
Формулирование гипотезы	Гипотеза должна быть проверяемой и измеримой
Контролируемое воздействие	Все изменения в системе должны быть обратимы

Минимизация blast radius	Эксперимент должен затрагивать минимально возможную область системы
Автоматизация	Эксперименты должны быть воспроизводимыми и автоматизированными

2.3. Типы экспериментов по проверке устойчивости

Тип эксперимента	Описание	Инструменты
Pod kill	Принудительное завершение pod'a	ChaosMesh, Gremlin
Network delay	Добавление искусственной задержки сети	ChaosMesh, tc
Network loss	Потеря пакетов	ChaosMesh, tc
CPU stress	Искусственная нагрузка на CPU	ChaosMesh, stress-ng
Memory stress	Искусственная нагрузка на память	ChaosMesh, stress-ng
IO stress	Нагрузка на диск	ChaosMesh, fio
DNS chaos	Сбой DNS-резолвинга	ChaosMesh
HTTP error injection	Инъекция ошибок HTTP	ChaosMesh

2.4. Blast Radius (радиус поражения)

Blast radius - мера потенциального влияния chaos-эксперимента на систему. При планировании эксперимента важно минимизировать blast radius.

Уровень	Описание	Рекомендации
Минимальный	Эксперимент на тестовом окружении	Безопасно, можно экспериментировать

		свободно
Ограниченный	Эксперимент на части production-трафика (canary)	Нужен мониторинг и план отката
Средний	Эксперимент на одном из нескольких кластеров	Требуется согласование
Широкий	Эксперимент на всём production	НЕ РЕКОМЕНДУЕТСЯ

2.5. Шаблон плана эксперимента

План эксперимента по проверке устойчивости

1. Общая информация

- Название эксперимента:
- Дата проведения:
- Ответственный:
- Система (компонент):

2. Гипотеза

- Что проверяем:
- Ожидаемое поведение:

3. Steady State (нормальное поведение системы)

SLI	Нормальное значение	Метод измерения

4. Условия эксперимента

- Тип эксперимента:
- Параметры:
- Длительность:
- Blast radius (охват):

5. Метрики успеха

Метрика	Критерий успеха

6. План проведения

- 1.
- 2.
- 3.

7. План отката

- Условия отката:
- Действия по откату:

8. Риски и меры предосторожности

Риск	Вероятность	Влияние	Мера предосторожности

9. Ожидаемые результаты

2.6. Примеры экспериментов для различных типов систем

Пример 1: Веб-сервис (монолит)

Элемент	Значение
Гипотеза	При отказе одного из трёх экземпляров сервиса, общая доступность сохранится на уровне >99.9%
Steady state	QPS >1000, error rate <0.1%, P99 latency <200ms
Эксперимент	Pod kill (удаление одного пода)
Метрики успеха	Доступность >99.9%, время восстановления <30 сек

Пример 2: Микросервисная архитектура

Элемент	Значение
Гипотеза	При задержке 500ms между сервисом А и сервисом Б, механизм retry с exponential backoff обеспечит успешное выполнение запросов с задержкой не более 2 секунд
Steady state	RPS >500, error rate <0.5%, P99 latency <100ms между сервисами
Эксперимент	Network delay 500ms между сервисами
Метрики успеха	Успешность запросов >99%, P99 latency <2 сек

Пример 3: Специализация БПЛА

Элемент	Значение
Гипотеза	При потере GPS-сигнала на 10 секунд, дрон переходит в режим удержания позиции с использованием IMU и безопасно завершает миссию
Steady state	GPS качество >8 спутников, дрон выполняет миссию

Эксперимент	Отказ GPS на 10 секунд (в симуляции)
Метрики успеха	Дрон не отклоняется более чем на 2 метра, переход в fail-safe режим

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные

Система: API-шлюз для интернет-магазина

- 3 экземпляра (реплики)
- PostgreSQL для хранения данных
- Redis для кэширования сессий
- Nginx в качестве балансировщика

3.2. Заполнение плана эксперимента

План эксперимента по проверке устойчивости

1. Общая информация

- Название эксперимента: Отказ одного экземпляра API-шлюза
- Дата проведения: 30.03.2025
- Ответственный: Смирнов Д. (SRE)
- Система (компонент): API-шлюз (3 экземпляра)

2. Гипотеза

- Что проверяем: При отказе одного из трёх экземпляров сервиса, общая доступность API сохранится на уровне >99.9%
- Ожидаемое поведение: Балансировщик Nginx автоматически исключит отказавший экземпляр из ротации; оставшиеся два экземпляра справятся с нагрузкой

3. Steady State (нормальное поведение системы)

SLI	Нормальное значение	Метод измерения
QPS	>1000 запросов/сек	Prometheus rate(http_requests_total[1m])
Error rate	<0.1%	Prometheus rate(http_requests_total{status=~"5.."}[1m])
P99 latency	<200ms	Prometheus histogram_quantile(0.99, http_duration_bucket)
Доступность	>99.9%	Prometheus up{}

4. Условия эксперимента

- Тип эксперимента: Pod kill (принудительное удаление пода)
- Параметры: Удаление одного пода с меткой app=api-gateway
- Длительность: 5 минут наблюдения после удаления
- Blast radius (охват): 1 из 3 подов (33% трафика)

5. Метрики успеха

Метрика	Критерий успеха
Доступность (Availability)	>99.9% в течение эксперимента
Error rate	<1% (допустим временный всплеск при переключении)
Время восстановления (MTTR)	<30 секунд
P99 latency	<500ms (допустимо кратковременное увеличение)

6. План проведения

1. Зафиксировать steady state метрики (за 5 минут до эксперимента)
2. Выполнить команду: `kubectl delete pod [имя_пода]`
3. Наблюдать за метриками в Grafana
4. Зафиксировать время восстановления service
5. Зафиксировать метрики после восстановления (5 минут)

7. План отката

- Условия отката: Error rate >5% в течение 1 минуты; время восстановления >2 минут
- Действия по откату: Восстановление удалённого пода через `kubectl apply -f deployment.yaml`

8. Риски и меры предосторожности

Риск	Вероятность	Влияние	Мера предосторожности
Каскадный отказ остальных подов	Низкая	Высокое	Дополнительный мониторинг, запас ресурсов
Потеря сессий пользователей	Средняя	Среднее	Redis сессии, повторная аутентификация
Долгое восстановление (>5 минут)	Низкая	Среднее	Подготовить ручной план отката

9. Ожидаемые результаты

Эксперимент должен подтвердить способность системы автоматически восстанавливаться после отказа одного экземпляра. Ожидается, что доступность сохранится на уровне >99.9%, а время восстановления не превысит 30 секунд.

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

Задание: Спроектировать эксперимент для простого веб-сервиса. Заполнить шаблон плана эксперимента (все разделы). Минимум 3 метрики steady state, 2 метрики успеха.

№	Тип системы	Эксперимент
1	Веб-сервис (2 экземпляра)	Pod kill (удаление одного пода)
2	Веб-сервис (3 экземпляра)	Network delay 100ms
3	API-сервис (2 реплики, БД)	Отказ БД на 30 секунд
4	Сервис аутентификации	Отказ Redis на 30 секунд
5	Сервис уведомлений	Задержка очереди RabbitMQ

4.2. Уровень 2

Задание: Спроектировать эксперимент для микросервисной архитектуры. Заполнить шаблон плана эксперимента + оценить blast radius. Минимум 4 метрики steady state, 3 метрики успеха.

№	Тип системы	Эксперимент
1	Микросервисы (3 сервиса)	Network delay 300ms между сервисом А и Б
2	Микросервисы (4 сервиса)	Отказ сервиса авторизации

3	Микросервисы + БД	CPU stress на сервере БД (50% нагрузки)
4	Микросервисы + кэш	Memory stress на Redis (заполнение 80%)
5	Специализация БПЛА	Отказ GPS на 10 секунд (симуляция)

4.3. Уровень 3

Задание: Спроектировать эксперимент для сложной системы. Заполнить шаблон плана эксперимента. Создать Python-скрипт для генерации плана эксперимента в формате Markdown/JSON.

№	Тип системы	Эксперимент
1	Е-commerce платформа (5+ сервисов)	Отказ каскадного типа (1 сервис → зависимые)
2	Распределённая БД (Cassandra)	Отказ одного из 3 узлов
3	Система обработки данных (Kafka)	Задержка консьюмера
4	Специализация БПЛА (продвинутый)	Потеря связи + отказ датчика
5	Комплексная система (микросервисы + БД + кэш + очередь)	Комбинированный эксперимент

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ (УРОВЕНЬ 3)

Лабораторная работа №8 (Уровень 3)

Генерация плана эксперимента по проверке устойчивости

"""

```
import json
from datetime import datetime
from typing import List, Dict, Any
from dataclasses import dataclass, asdict
from enum import Enum

class ExperimentType(Enum):
    POD_KILL = "Отказ узла (pod kill)"
    NETWORK_DELAY = "Сетевая задержка"
    NETWORK_LOSS = "Потеря пакетов"
    CPU_STRESS = "Нагрузка на CPU"
    MEMORY_STRESS = "Нагрузка на память"
    FAILURE_INJECTION = "Инъекция ошибок"
    GPS_FAILURE = "Отказ GPS (БПЛА)"
    RADIO_LOSS = "Потеря связи (БПЛА)"

@dataclass
class SLIMetric:
    """Метрика Steady State"""
    name: str
    normal_value: str
    measurement_method: str

@dataclass
class SuccessMetric:
    """Метрика успеха"""
    name: str
    success_criteria: str

@dataclass
class Risk:
    """Риск"""
    description: str
    probability: str # LOW, MEDIUM, HIGH
    impact: str # LOW, MEDIUM, HIGH
    mitigation: str

@dataclass
class ExperimentPlan:
    """План эксперимента"""
    name: str
    date: str
    owner: str
    system: str
    hypothesis: str
```

```

    steady_state: List[SLIMetric]
    experiment_type: ExperimentType
    parameters: str
    duration: str
    blast_radius: str
    success_metrics: List[SuccessMetric]
    rollback_conditions: str
    rollback_actions: str
    risks: List[Risk]

class ExperimentPlanGenerator:
    """Генератор плана эксперимента"""

    def __init__(self, plan: ExperimentPlan):
        self.plan = plan

    def to_markdown(self) -> str:
        """Экспорт в Markdown"""
        md = f"""# План эксперимента по проверке устойчивости

## 1. Общая информация
- **Название эксперимента:** {self.plan.name}
- **Дата проведения:** {self.plan.date}
- **Ответственный:** {self.plan.owner}
- **Система (компонент):** {self.plan.system}

## 2. Гипотеза
**{self.plan.hypothesis}**

## 3. Steady State (нормальное поведение системы)
| SLI | Нормальное значение | Метод измерения |
| :--- | :--- | :--- |
"""
        for sli in self.plan.steady_state:
            md += f"| {sli.name} | {sli.normal_value} | {sli.measurement_method} |"
        md += f"\n"

        md += f"""

## 4. Условия эксперимента
- **Тип эксперимента:** {self.plan.experiment_type.value}
- **Параметры:** {self.plan.parameters}
- **Длительность:** {self.plan.duration}
- **Blast radius (охват):** {self.plan.blast_radius}

## 5. Метрики успеха
| Метрика | Критерий успеха |
| :--- | :--- |
"""
        for metric in self.plan.success_metrics:

```



```

        md += f"| {metric.name} | {metric.success_criteria} |\n"

    md += f"""
## 6. План отката
- **Условия отката:** {self.plan.rollback_conditions}
- **Действия по откату:** {self.plan.rollback_actions}

## 7. Риски и меры предосторожности
| Риск | Вероятность | Влияние | Мера предосторожности |
| :--- | :--- | :--- | :--- |
"""

    for risk in self.plan.risks:
        md += f"| {risk.description} | {risk.probability} | {risk.impact} | {risk.mitigation} |\n"

    return md

    def to_json(self) -> str:
        """Экспорт в JSON"""
        return json.dumps(asdict(self.plan), indent=2, ensure_ascii=False, default=str)

    def to_html(self) -> str:
        """Экспорт в HTML"""
        html = f"""<!DOCTYPE html>
<html>
<head><title>План эксперимента</title><meta charset="UTF-8"></head>
<body>
<h1>План эксперимента по проверке устойчивости</h1>
<h2>1. Общая информация</h2>
<ul>
<li><strong>Название эксперимента:</strong> {self.plan.name}</li>
<li><strong>Дата проведения:</strong> {self.plan.date}</li>
<li><strong>Ответственный:</strong> {self.plan.owner}</li>
<li><strong>Система:</strong> {self.plan.system}</li>
</ul>
<h2>2. Гипотеза</h2>
<p>{self.plan.hypothesis}</p>
<h2>3. Steady State</h2>
<table border="1"><tr><th>SLI</th><th>Нормальное значение</th><th>Метод измерения</th></tr>
"""

        for sli in self.plan.steady_state:
            html += f"<tr><td>{sli.name}</td><td>{sli.normal_value}</td><td>{sli.measurement_method}</td></tr>"

        html += f"""</table>
<h2>4. Условия эксперимента</h2>

```

```

<ul>
<li><strong>Тип:</strong> {self.plan.experiment_type.value}</li>
<li><strong>Параметры:</strong> {self.plan.parameters}</li>
<li><strong>Длительность:</strong> {self.plan.duration}</li>
<li><strong>Blast radius:</strong> {self.plan.blast_radius}</li>
</ul>
</body></html>
"""

    return html

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ (вариант 1: веб-сервис)
# =====

if __name__ == "__main__":
    print("=" * 60)
    print("ЛАБОРАТОРНАЯ РАБОТА №8")
    print("Генерация плана эксперимента")
    print("=" * 60)

    plan = ExperimentPlan(
        name="Отказ одного экземпляра API-шлюза",
        date="2025-03-30",
        owner="Смирнов Д. (SRE)",
        system="API-шлюз (3 экземпляра, PostgreSQL, Redis)",
        hypothesis="При отказе одного из трёх экземпляров сервиса, общая
доступность API сохранится на уровне >99.9%",
        steady_state=[
            SLIMetric("QPS", ">1000 запросов/сек", "Prometheus
rate(http_requests_total[1m])"),
            SLIMetric("Error rate", "<0.1%", "Prometheus
rate(http_requests_total{status=~\"5..\"}[1m])"),
            SLIMetric("P99 latency", "<200ms", "Prometheus
histogram_quantile(0.99, http_duration_bucket)"),
            SLIMetric("Доступность", ">99.9%", "Prometheus up{}"),
        ],
        experiment_type=ExperimentType.POD_KILL,
        parameters="Удаление одного пода с меткой app=api-gateway",
        duration="5 минут наблюдения после удаления",
        blast_radius="1 из 3 подов (33% трафика)",
        success_metrics=[
            SuccessMetric("Доступность", ">99.9% в течение эксперимента"),
            SuccessMetric("Error rate", "<1% (допустим временный всплеск)"),
            SuccessMetric("Время восстановления (MTTR)", "<30 секунд"),
            SuccessMetric("P99 latency", "<500ms (кратковременное увеличение)"),
        ],
        rollback_conditions="Error rate >5% в течение 1 минуты; время
восстановления >2 минут",

```

```

        rollback_actions="Восстановление удалённого пода через kubectl apply -f
deployment.yaml",
        risks=[
            Risk(
                description="Каскадный отказ остальных подов",
                probability="Низкая",
                impact="Высокое",
                mitigation="Дополнительный мониторинг, запас ресурсов"
            ),
            Risk(
                description="Потеря сессий пользователей",
                probability="Средняя",
                impact="Среднее",
                mitigation="Redis сессии, повторная аутентификация"
            ),
            Risk(
                description="Долгое восстановление (>5 минут)",
                probability="Низкая",
                impact="Среднее",
                mitigation="Подготовить ручной план отката"
            )
        ]
    )

generator = ExperimentPlanGenerator(plan)

print("\n📄 Сгенерированный план эксперимента (Markdown):")
print(generator.to_markdown())

# Сохранение файлов
with open("experiment_plan.md", "w", encoding="utf-8") as f:
    f.write(generator.to_markdown())

with open("experiment_plan.json", "w", encoding="utf-8") as f:
    f.write(generator.to_json())

with open("experiment_plan.html", "w", encoding="utf-8") as f:
    f.write(generator.to_html())

print("\n✅ Файлы сохранены:")
print("    - experiment_plan.md")
print("    - experiment_plan.json")
print("    - experiment_plan.html")

```

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое эксперимент по проверке устойчивости?
2. Какие элементы обязательно должны быть в плане эксперимента?
3. Что такое steady state и как его определить для системы?
4. Как формулируется гипотеза эксперимента?
5. Что такое blast radius и почему его важно контролировать?
6. Какие типы chaos-экспериментов вы знаете?
7. Какие метрики успеха можно использовать для эксперимента?
8. Что должно быть в плане отката?
9. Какие риски нужно оценивать при планировании эксперимента?
10. Как метод 5 Whys (из ЛР №5) может помочь в анализе результатов эксперимента?

7. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (принципы проектирования экспериментов, blast radius, steady state).
4. Практическая часть:
 - Описание системы (по варианту).
 - Заполненный шаблон плана эксперимента.
 - Место для скриншота: заполненный план эксперимента.
5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.
 - Место для скриншота: сгенерированные файлы.
6. Анализ результатов (оценка рисков, обоснование выбора эксперимента).
7. Ответы на контрольные вопросы (3-5).

8. Вывод.

8. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Заполнение всех разделов плана	2 балла	2 балла	2 балла
Корректность формулировки гипотезы	2 балла	2 балла	2 балла
Определение steady state (3+/4+ метрики)	1 балла	2 балла	2 балла
Метрики успеха (2+/3+)	1 балла	2 балла	2 балла
Оценка blast radius	0 баллов	1 балл	2 балла
Анализ рисков (2+/3+)	0 баллов	1 балл	2 балла
Программная генерация плана	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 12)	Уровень 3 (макс. 16)
5 (отлично)	7-8	11-12	14-16
4 (хорошо)	5-6	9-10	11-13
3 (удовлетворительно)	3-4	6-8	8-10
2 (неудовлетворительно)	<3	<6	<8

9. ПРИМЕРНАЯ ПРОГРАММНАЯ ЧАСТЬ (СПЕЦИАЛИЗАЦИЯ БПЛА)

"""

Лабораторная работа №8 (Уровень 3, Специализация БПЛА)

Генерация плана эксперимента для БПЛА

"""

Пример для варианта 14 (Специализация БПЛА: отказ GPS)

```
uav_plan = ExperimentPlan(
    name="Отказ GPS-сигнала при автономном полёте",
    date="2025-04-05",
    owner="Иванов П. (SRE UAV)",
    system="БПЛА (квадрокоптер, PX4, Gazebo симуляция)",
    hypothesis="При потере GPS-сигнала на 10 секунд, дрон переходит в режим удержания позиции с использованием IMU и безопасно завершает миссию",
    steady_state=[
        SLIMetric("GPS качество", ">8 спутников, HDOP<1.0", "MAVLink GPS_STATUS"),
        SLIMetric("Отклонение от позиции", "<0.5 м", "MAVLink GLOBAL_POSITION_INT"),
        SLIMetric("Высота", "10 м ±0.5 м", "MAVLink ALTITUDE"),
        SLIMetric("Режим полёта", "AUTO.MISSION", "MAVLink EXTENDED_SYS_STATE"),
    ],
    experiment_type=ExperimentType.GPS_FAILURE,
    parameters="Имитация потери сигнала на 10 секунд (через MAVLink или симулятор)",
    duration="10 секунд потери GPS + 30 секунд наблюдения",
    blast_radius="1 дрон (единичный аппарат)",
    success_metrics=[
        SuccessMetric("Переход в fail-safe", "Переход в режим удержания позиции (Loiter/Position Hold)"),
        SuccessMetric("Отклонение от позиции", "Не более 2 метров за время эксперимента"),
        SuccessMetric("Восстановление после сигнала", "Возврат в автономный режим <5 сек"),
        SuccessMetric("Безопасное завершение", "Успешная посадка при необходимости"),
    ],
    rollback_conditions="Отклонение >5 метров; неконтролируемое изменение высоты >3 метров",
    rollback_actions="Аварийное переключение в ручной режим; принудительная посадка через RC",
    risks=[
        Risk(
            description="Потеря управления после восстановления GPS",
            probability="Средняя",
            impact="Высокое",
            mitigation="Резервный (второй) GPS модуль"
        ),
        Risk(
            description="Столкновение с препятствием в период отсутствия GPS",
```

```

        probability="Низкая",
        impact="Высокое",
        mitigation="Предварительная очистка зоны полёта"
    ),
    Risk(
        description="Повреждение дрона при посадке",
        probability="Средняя",
        impact="Высокое",
        mitigation="Использование симуляции (Gazebo) для первого запуска"
    )
]
)

generator = ExperimentPlanGenerator(uav_plan)
print("\n📄 Специализация БПЛА: план эксперимента")
print(generator.to_markdown())

```

ЛАБОРАТОРНАЯ РАБОТА №9

Проведение А/В теста для проверки устойчивости (Canary Deployment)

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки настройки канареечных развертываний (canary deployment) для поэтапного тестирования новых версий на ограниченной аудитории пользователей, сбора и анализа метрик в реальном времени, а также принятия решений о продолжении, приостановке или откате обновления на основе полученных данных.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Canary Deployment: концепция и преимущества

Canary Deployment (канареечное развертывание) - стратегия развертывания, при которой новая версия приложения постепенно получает всё большую долю трафика, начиная с небольшого процента пользователей (обычно 5-

10%). Название происходит от практики использования канареек в угольных шахтах для обнаружения токсичных газов.

Ключевые принципы:

Принцип	Описание
Малый blast radius	Ошибки в новой версии влияют только на небольшой процент пользователей
Поэтапное увеличение	Трафик увеличивается постепенно (10% → 25% → 50% → 100%)
Автоматический откат	При ухудшении метрик canary автоматически отключается
Валидация в production	Новая версия тестируется на реальном трафике

Сравнение стратегий развертывания:

Стратегия	Риск	Ресурсы	Rollback	Zero-downtime
Recreate	Высокий	0% доп.	Медленный	Нет
Rolling Update	Средний	0% доп.	Медленный	Да
Blue-Green	Низкий	+100%	Мгновенный	Да
Canary	Минимальный	+10-20%	Мгновенный	Да

2.2. Canary с Nginx Ingress

Nginx Ingress Controller поддерживает canary-развертывание через специальные аннотации:

Аннотация	Описание

nginx.ingress.kubernetes.io/canary: "true"	Включает canary-режим
nginx.ingress.kubernetes.io/canary-weight	Процент трафика на canary (0-100)
nginx.ingress.kubernetes.io/canary-by-header	Маршрутизация по HTTP-заголовку
nginx.ingress.kubernetes.io/canary-by-cookie	Маршрутизация по cookie

Приоритет аннотаций: canary-by-header > canary-by-cookie > canary-weight

2.3. Canary с Istio VirtualService

Istio предоставляет более мощные возможности для canary-развертывания:

```

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: myapp
spec:
  hosts:
  - myapp
  http:
  - route:
    - destination:
        host: myapp-stable
        weight: 90
    - destination:
        host: myapp-canary
        weight: 10

```

2.4. A/B тестирование и Feature Flags

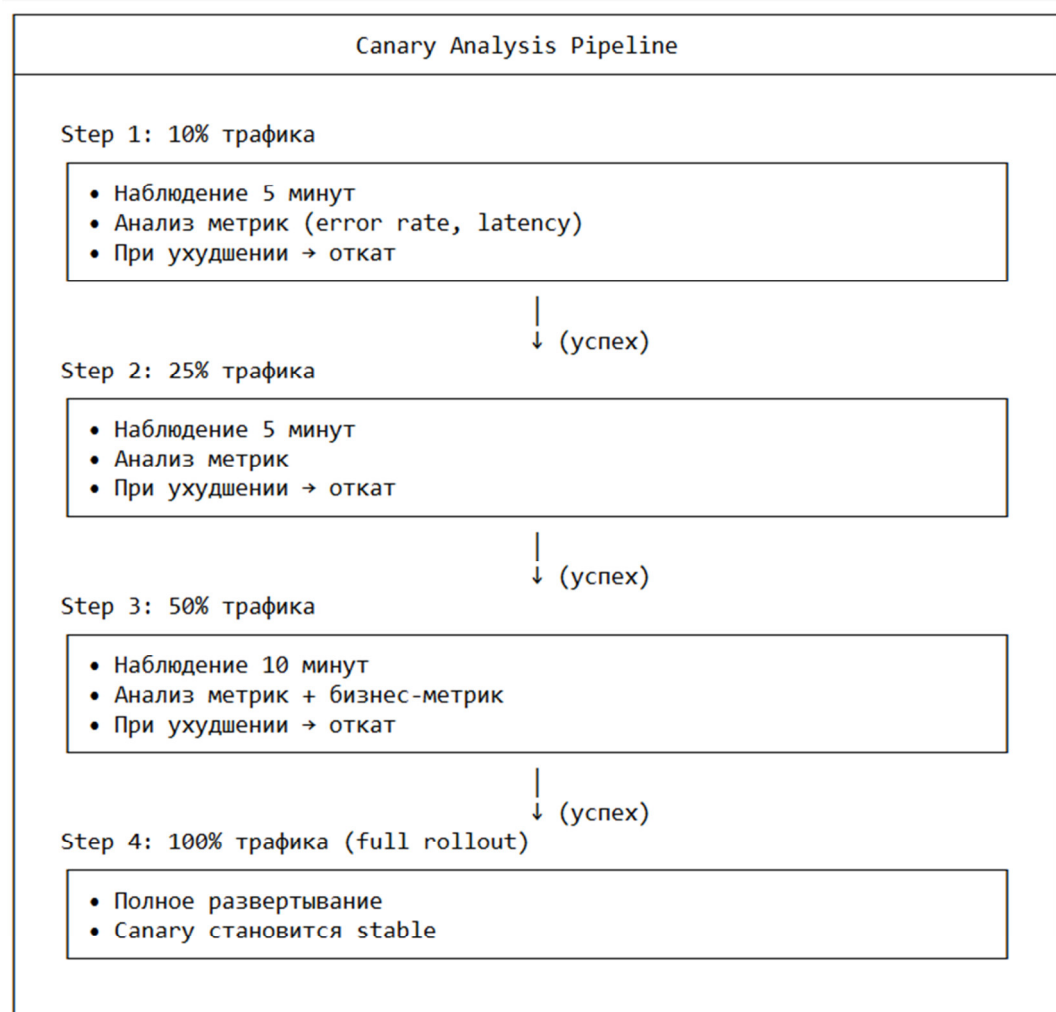
A/B тестирование - метод сравнения двух версий (А и В) для определения, какая из них показывает лучшие результаты по заданным метрикам (конверсия, удержание, производительность).

Метод	Описание	Когда использовать
Canary weight	Процент пользователей направляется на новую версию	Для тестирования производительности
Canary by header	Маршрутизация по HTTP-заголовку (например, X-Canary-Version: v2)	Для внутренних пользователей, QA
Canary by cookie	Маршрутизация по cookie	Для A/B тестирования
Feature flags	Включение функциональности без развертывания	Для точечного A/B тестирования

2.5. Метрики для canary-анализа

Категория	Метрика	Критерий успеха
Доступность	Error rate (5xx)	<0.1% (stable) vs <0.5% (canary)
Производительность	P99 latency	<200ms увеличение не более чем на 20%
Бизнес-метрики	Конверсия, успешность операций	Не ниже stable на 5%
Системные метрики	CPU, память, сеть	Не превышают stable на 30%

2.6. Схема canary-анализа



3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Исходные данные

Система: API-шлюз для интернет-магазина

- 3 экземпляра stable версии (v1)
- 1 экземпляр canary версии (v2)
- Nginx Ingress Controller для маршрутизации
- Prometheus + Grafana для мониторинга

Новая версия: оптимизированный алгоритм поиска товаров (предполагаемое улучшение производительности на 15%)

3.2. Конфигурация canary-развертывания

Stable Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-stable
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: shop.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: myapp-stable
            port:
              number: 80
```

Canary Ingress (10% трафика):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-canary
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "10"
spec:
  ingressClassName: nginx
  rules:
  - host: shop.example.com
    http:
      paths:
```

```
- path: /
  pathType: Prefix
  backend:
    service:
      name: myapp-canary
      port:
        number: 80
```

3.3. Скрипт для canary-анализа (Python)

```
"""
Лабораторная работа №9
Анализ метрик canary-развертывания
"""

import time
import requests
import json
from datetime import datetime
from typing import Dict, Any, List
from enum import Enum

class CanaryStatus(Enum):
    ROLLBACK = "rollback"
    CONTINUE = "continue"
    PROMOTE = "promote"

class CanaryAnalyzer:
    """Анализатор метрик canary-развертывания"""

    def __init__(self, prometheus_url: str = "http://localhost:9090"):
        self.prometheus_url = prometheus_url

    def query_prometheus(self, query: str) -> float:
        """Выполнение запроса к Prometheus"""
        try:
            response = requests.get(
                f"{self.prometheus_url}/api/v1/query",
                params={"query": query},
                timeout=10
            )
            data = response.json()
            if data["status"] == "success" and data["data"]["result"]:
                return float(data["data"]["result"][0]["value"][1])
            return 0.0
```

```

except Exception as e:
    print(f"Ошибка запроса к Prometheus: {e}")
    return 0.0

def get_error_rate(self, service: str, window: str = "5m") -> float:
    """Получение частоты ошибок для сервиса"""
    query = f"""
sum(rate(http_requests_total{{service="{service}",status=~"5.."}}[window
]))
/
sum(rate(http_requests_total{{service="{service}"}}[window]))
"""
    return self.query_prometheus(query) * 100

def get_p99_latency(self, service: str, window: str = "5m") -> float:
    """Получение P99 задержки для сервиса"""
    query = f"""
histogram_quantile(0.99,
    sum(rate(http_request_duration_seconds_bucket{{service="{service}"}}[w
indow]))
    by (le)
)
"""
    return self.query_prometheus(query) * 1000 # в миллисекундах

def get_success_rate(self, service: str, window: str = "5m") -> float:
    """Получение успешности запросов"""
    error_rate = self.get_error_rate(service, window)
    return 100 - error_rate

def analyze(self, stable_service: str, canary_service: str,
            thresholds: Dict[str, float]) -> CanaryStatus:
    """
    Анализ canary-развертывания и принятие решения

    thresholds = {
        "max_error_rate": 1.0,      # % ошибок
        "max_latency_increase": 20,  # % увеличения латентности
        "min_success_rate": 99.0    # % успешности
    }
    """
    print(f"\n{'='*60}")
    print(f"ANALYZING CANARY DEPLOYMENT")
    print(f"{'='*60}")
    print(f"Stable service: {stable_service}")
    print(f"Canary service: {canary_service}")
    print(f"Thresholds: {thresholds}")

    # Сбор метрик

```

```

stable_error = self.get_error_rate(stable_service)
canary_error = self.get_error_rate(canary_service)
stable_latency = self.get_p99_latency(stable_service)
canary_latency = self.get_p99_latency(canary_service)
stable_success = self.get_success_rate(stable_service)
canary_success = self.get_success_rate(canary_service)

# Вывод метрик
print(f"\n METRICS:")
print(f"  Stable error rate:      {stable_error:.2f}%")
print(f"  Canary error rate:       {canary_error:.2f}%")
print(f"  Stable P99 latency:      {stable_latency:.0f}ms")
print(f"  Canary P99 latency:      {canary_latency:.0f}ms")
print(f"  Stable success rate:     {stable_success:.2f}%")
print(f"  Canary success rate:     {canary_success:.2f}%")

# Анализ
issues = []

if canary_error > thresholds["max_error_rate"]:
    issues.append(f"Error rate too high: {canary_error:.2f}% > {thresholds['max_error_rate']}%")

if canary_error > stable_error * 2:
    issues.append(f"Error rate doubled: {canary_error:.2f}% vs {stable_error:.2f}%")

latency_increase = ((canary_latency - stable_latency) / stable_latency) *
100 if stable_latency > 0 else 0
if latency_increase > thresholds["max_latency_increase"]:
    issues.append(f"Latency increased by {latency_increase:.1f}% > {thresholds['max_latency_increase']}%")

if canary_success < thresholds["min_success_rate"]:
    issues.append(f"Success rate too low: {canary_success:.2f}% < {thresholds['min_success_rate']}%")

# Принятие решения
if issues:
    print(f"\n ISSUES FOUND:")
    for issue in issues:
        print(f"  • {issue}")
    return CanaryStatus.ROLLBACK
elif canary_success >= stable_success and canary_latency <=
stable_latency * 1.1:
    print(f"\n CANARY OUTPERFORMS STABLE")
    return CanaryStatus.PROMOTE
else:
    print(f"\n CANARY IS STABLE, CONTINUING")

```

```

        return CanaryStatus.CONTINUE

class CanaryDeployment:
    """Управление canary-развертыванием"""

    def __init__(self, canary_ingress_name: str, namespace: str = "default"):
        self.canary_ingress_name = canary_ingress_name
        self.namespace = namespace
        self.current_weight = 0
        self.steps = [10, 25, 50, 100] # проценты трафика

    def update_weight(self, weight: int) -> bool:
        """Обновление веса canary-трафика"""
        try:
            # Имитация обновления Ingress
            print(f" Updating canary weight to {weight}%")
            self.current_weight = weight
            return True
        except Exception as e:
            print(f" Error updating weight: {e}")
            return False

    def rollback(self) -> bool:
        """Откат canary-развертывания"""
        print(f"\n ROLLBACK: Canary deployment failed")
        return self.update_weight(0)

    def promote(self) -> bool:
        """Продвижение canary в stable"""
        print(f"\n PROMOTE: Canary becomes stable")
        return self.update_weight(100)

# =====
# ПРИМЕР ВЫПОЛНЕНИЯ
# =====

def run_canary_analysis():
    """Запуск canary-анализа с симуляцией метрик"""

    print("=" * 60)
    print("CANARY DEPLOYMENT ANALYSIS")
    print("=" * 60)

    analyzer = CanaryAnalyzer()
    deployment = CanaryDeployment("myapp-canary")

    # Симуляция метрик (в реальных условиях данные берутся из Prometheus)
    # Здесь приведены примеры данных для разных сценариев

```



```

scenarios = [
    {
        "name": "Сценарий 1: Успешный canary",
        "stable_error": 0.1,
        "canary_error": 0.15,
        "stable_latency": 150,
        "canary_latency": 160,
    },
    {
        "name": "Сценарий 2: Canary хуже (высокая ошибка)",
        "stable_error": 0.1,
        "canary_error": 2.5,
        "stable_latency": 150,
        "canary_latency": 180,
    },
    {
        "name": "Сценарий 3: Canary лучше stable",
        "stable_error": 0.2,
        "canary_error": 0.05,
        "stable_latency": 180,
        "canary_latency": 120,
    }
]

thresholds = {
    "max_error_rate": 1.0,
    "max_latency_increase": 20,
    "min_success_rate": 99.0
}

for scenario in scenarios:
    print(f"\n{'='*60}")
    print(f"{scenario['name']}")
    print(f"{'='*60}")

    # В реальной системе метрики запрашиваются из Prometheus
    # Здесь используется симуляция

    decision = CanaryStatus.CONTINUE

    if scenario["name"] == "Сценарий 1: Успешный canary":
        decision = CanaryStatus.CONTINUE
    elif scenario["name"] == "Сценарий 2: Canary хуже (высокая ошибка)":
        decision = CanaryStatus.ROLLBACK
    else:
        decision = CanaryStatus.PROMOTE

    print(f"\n🔗 DECISION: {decision.value}")

```

```

if decision == CanaryStatus.ROLLBACK:
    deployment.rollback()
elif decision == CanaryStatus.PROMOTE:
    deployment.promote()
else:
    print("\n CONTINUING: Monitoring for next step")

if __name__ == "__main__":
    run_canary_analysis()

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ (15 вариантов)

4.1. Уровень 1 (варианты 1-5)

№	Тип системы	Canary вес	Метрики	Критерии успеха
1	Веб-сервис (Flask)	10%	error rate, latency	error rate <1%
2	Веб-сервис (Node.js)	10%	error rate, latency	p99 latency <200ms
3	API-сервис (Go)	15%	success rate, errors	success rate >99%
4	Микросервис (Python)	10%	error rate, QPS	error rate <0.5%
5	Статический сайт	20%	HTTP 2xx/5xx	5xx errors <0.1%

4.2. Уровень 2 (варианты 6-10)

№	Тип системы	Canary этапы	Метрики	Дополнительные требования
1	Микросервисы (3	10%, 25%, 50%	error,	Автоматический

	сервиса)		latency, CPU	анализ
2	API-сервис + БД	10%, 25%, 50%, 100%	error, latency, БД connections	Анализ бизнес-метрик
3	Система с Redis кэшем	10%, 25%, 50%	hit rate, latency	Проверка кэширования
4	Система с очередью	10%, 25%	queue length, processing time	Мониторинг очереди
5	Специализация БПЛА	10%, 25%, 50%	telemetry, position error	Анализ телеметрии

4.3. Уровень 3 (варианты 11-15)

№	Тип системы	Canary этапы	Дополнительные требования
1	E-commerce (5+ сервисов)	5%, 10%, 25%, 50%, 100%	Python-скрипт анализа
2	Распределённая система	10%, 20%, 40%, 80%	Интеграция с Prometheus
3	Система реального времени	10%, 25%, 50%	Анализ задержек
4	Специализация БПЛА (продвинутый)	5%, 10%, 25%, 50%	Анализ телеметрии дрона
5	Комплексная система	10%, 25%, 50%, 100%	Полный пайплайн анализа

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое canary deployment и чем он отличается от blue-green deployment?

2. Какие аннотации Nginx Ingress используются для canary-развертывания?
3. Как настроить canary-маршрутизацию по HTTP-заголовку?
4. Какие метрики следует отслеживать при canary-анализе?
5. Как работает автоматический откат при ухудшении метрик?
6. В чем отличие canary от A/B тестирования?
7. Как feature flags могут дополнить canary-развертывание?
8. Какие пороговые значения метрик рекомендуется использовать для canary?
9. Как настроить многоэтапный canary (10% → 25% → 50% → 100%)?
10. Какие инструменты поддерживают canary-развертывание в Kubernetes?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (canary deployment, методы маршрутизации).
4. Практическая часть:
 - Конфигурация canary Ingress.
 - Результаты canary-анализа.
 - Место для скриншота: дашборды с метриками.
 - Место для скриншота: логи canary-анализа.
5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.
6. Анализ результатов (принятые решения, откаты).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Canary Ingress настроен	2 балла	2 балла	2 балла
Сбор метрик организован	2 балла	2 балла	2 балла
Анализ метрик выполнен	1 балл	2 балла	2 балла
Многоэтапный canary (3+ этапов)	0 баллов	2 балла	2 балла
Автоматический анализ (Python)	0 баллов	1 балл	2 балла
Интеграция с Prometheus	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 7)	Уровень 2 (макс. 11)	Уровень 3 (макс. 14)
5 (отлично)	6-7	10-11	13-14
4 (хорошо)	5	8-9	10-12
3 (удовлетворительно)	3-4	6-7	7-9
2 (неудовлетворительно)	<3	<6	<7

ЛАБОРАТОРНАЯ РАБОТА №10. Настройка Chaos Engineering платформы
(ChaosMesh)

1. ЦЕЛЬ РАБОТЫ: Сформировать практические навыки установки и настройки платформы Chaos Engineering (ChaosMesh) в Kubernetes-кластере, определения score и blast radius экспериментов, а также проведения базовых chaos-экспериментов (pod kill, network delay, network loss) для проверки устойчивости системы.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Принципы Chaos Engineering

Chaos Engineering - дисциплина экспериментирования на распределенных системах для выявления слабых мест и повышения устойчивости к сбоям в production-условиях.

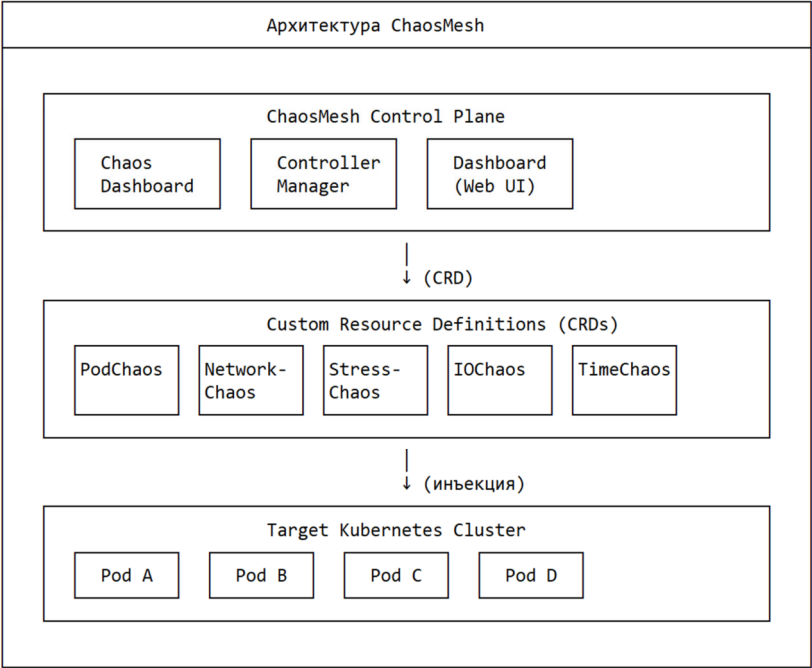
Принцип	Описание
Определение steady state	Формулировка нормального поведения системы (метрики, которые должны оставаться стабильными)
Формулирование гипотезы	Предположение о том, что steady state сохранится при внесении хаоса
Проведение эксперимента	Инъекция реальных событий (отказы, задержки, сбои)
Анализ результатов	Сравнение поведения системы с гипотезой
Улучшение системы	Внесение изменений для повышения устойчивости

2.2. Основные типы chaos-экспериментов

Тип эксперимента	Описание	Пример использования
Pod Kill	Принудительное завершение pod'a	Проверка автоматического восстановления (replica set, self-healing)

Network Delay	Добавление искусственной задержки сети	Проверка timeout, retry, circuit breaker
Network Loss	Потеря пакетов	Проверка устойчивости к потере данных
Pod Failure	Имитация сбоя приложения	Проверка graceful degradation
Container Kill	Убийство конкретного контейнера	Тестирование восстановления контейнера
CPU Stress	Искусственная нагрузка на CPU	Проверка autoscaling и throttling
Memory Stress	Искусственная нагрузка на память	Проверка OOM и восстановления
DNS Chaos	Сбои DNS-резолвинга	Проверка fallback механизмов

2.3. Архитектура ChaosMesh



2.4. Blast Radius (радиус поражения)

Blast radius — мера потенциального влияния chaos-эксперимента на систему. При планировании эксперимента важно минимизировать blast radius.

Уровень	Описание	Рекомендации
Минимальный	Эксперимент на тестовом окружении	Безопасно, можно экспериментировать свободно
Ограниченный	Эксперимент на части production-трафика (canary)	Нужен мониторинг и план отката
Средний	Эксперимент на одном из нескольких кластеров	Требуется согласование
Широкий	Эксперимент на всём production	НЕ РЕКОМЕНДУЕТСЯ

2.5. Пример манифеста PodChaos (убийство pod'a)

```
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: pod-kill-example
  namespace: chaos-testing
spec:
  action: pod-kill
  mode: one          # один pod
  selector:
    namespaces:
      - default
    labelSelectors:
      app: myapp
  scheduler:
    cron: "@every 5m"
    duration: "1m"
```

2.6. Пример манифеста NetworkChaos (задержка)

```
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
```



```
metadata:
  name: network-delay-example
  namespace: chaos-testing
spec:
  action: delay
  mode: all
  selector:
    namespaces:
      - default
    labelSelectors:
      app: myapp
  delay:
    latency: "300ms"
    correlation: "25"
    jitter: "50ms"
    duration: "2m"
```

2.7. Пример манифеста StressChaos (нагрузка на CPU)

```
apiVersion: chaos-mesh.org/v1alpha1
kind: StressChaos
metadata:
  name: cpu-stress-example
  namespace: chaos-testing
spec:
  mode: one
  selector:
    namespaces:
      - default
    labelSelectors:
      app: myapp
  stressors:
    cpu:
      workers: 2
      load: 80
  duration: "1m"
```

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Установка ChaosMesh в Kubernetes

```
# Установка ChaosMesh с помощью curl
curl -sSL https://mirrors.chaos-mesh.org/v2.6.3/install.sh | bash
```

```
# Проверка установки
kubect1 get pods -n chaos-mesh

# Порт-форвардинг для доступа к Dashboard
kubect1 port-forward -n chaos-mesh svc/chaos-dashboard 2333:2333
```

3.2. Создание тестового приложения

```
# test-app.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-app
  labels:
    app: test-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: test-app
  template:
    metadata:
      labels:
        app: test-app
    spec:
      containers:
        - name: app
          image: nginx:latest
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: test-app-service
spec:
  selector:
    app: test-app
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

```
# Применение манифеста
kubect1 apply -f test-app.yaml

# Проверка
```

```
kubectl get pods -l app=test-app
```

3.3. Эксперимент 1: Pod Kill

Манифест эксперимента:

```
# pod-kill-experiment.yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: pod-kill-experiment
spec:
  action: pod-kill
  mode: one
  selector:
    labelSelectors:
      app: test-app
  duration: "30s"
  scheduler:
    cron: "@every 2m"
```

```
# Применение эксперимента
kubectl apply -f pod-kill-experiment.yaml

# Наблюдение за подами в другом терминале
kubectl get pods -l app=test-app -w
```

3.4. Эксперимент 2: Network Delay (задержка)

Манифест эксперимента:

```
# network-delay-experiment.yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: network-delay-experiment
spec:
  action: delay
  mode: all
  selector:
    labelSelectors:
      app: test-app
  delay:
    latency: "500ms"
```

```
correlation: "50"
jitter: "100ms"
duration: "1m"
```

```
# Применение эксперимента
kubectl apply -f network-delay-experiment.yaml

# Проверка задержки (из другого пода)
kubectl run -it --rm debug --image=busybox -- sh
# В контейнере: time wget -q -O- http://test-app-service
```

3.5. Программная генерация chaos-экспериментов (Python)

```
"""
Лабораторная работа №10
Программная генерация манифестов ChaosMesh
"""

import yaml
import random
from datetime import datetime
from typing import Dict, Any, List, Optional

class ChaosExperimentGenerator:
    """Генератор манифестов ChaosMesh"""

    def __init__(self, namespace: str = "chaos-testing"):
        self.namespace = namespace
        self.experiments = []

    def generate_pod_kill(self, name: str, selector: Dict[str, str],
                          duration: str = "30s", mode: str = "one") -> Dict[str,
Any]:
    """Генерация PodKill эксперимента"""
    experiment = {
        "apiVersion": "chaos-mesh.org/v1alpha1",
        "kind": "PodChaos",
        "metadata": {
            "name": name,
            "namespace": self.namespace
        },
        "spec": {
            "action": "pod-kill",
            "mode": mode,
            "selector": {
                "namespaces": ["default"],
                "labelSelectors": selector
            }
        }
    }
```

```

        "duration": duration
    }
}
self.experiments.append(experiment)
return experiment

def generate_network_delay(self, name: str, selector: Dict[str, str],
                           latency: str = "300ms", jitter: str = "50ms",
                           duration: str = "1m", mode: str = "all") ->
Dict[str, Any]:
    """Генерация NetworkDelay эксперимента"""
    experiment = {
        "apiVersion": "chaos-mesh.org/v1alpha1",
        "kind": "NetworkChaos",
        "metadata": {
            "name": name,
            "namespace": self.namespace
        },
        "spec": {
            "action": "delay",
            "mode": mode,
            "selector": {
                "namespaces": ["default"],
                "labelSelectors": selector
            },
            "delay": {
                "latency": latency,
                "jitter": jitter,
                "correlation": "50"
            },
            "duration": duration
        }
    }
    self.experiments.append(experiment)
    return experiment

def generate_network_loss(self, name: str, selector: Dict[str, str],
                          loss_percent: int = 10, duration: str = "1m",
                          mode: str = "all") -> Dict[str, Any]:
    """Генерация NetworkLoss эксперимента"""
    experiment = {
        "apiVersion": "chaos-mesh.org/v1alpha1",
        "kind": "NetworkChaos",
        "metadata": {
            "name": name,
            "namespace": self.namespace
        },
        "spec": {
            "action": "loss",

```

```

        "mode": mode,
        "selector": {
            "namespaces": ["default"],
            "labelSelectors": selector
        },
        "loss": {
            "loss": f"{loss_percent}",
            "correlation": "50"
        },
        "duration": duration
    }
}
self.experiments.append(experiment)
return experiment

def generate_cpu_stress(self, name: str, selector: Dict[str, str],
                        cpu_workers: int = 1, load: int = 100,
                        duration: str = "30s", mode: str = "one") ->
Dict[str, Any]:
    """Генерация CPUStress эксперимента"""
    experiment = {
        "apiVersion": "chaos-mesh.org/v1alpha1",
        "kind": "StressChaos",
        "metadata": {
            "name": name,
            "namespace": self.namespace
        },
        "spec": {
            "mode": mode,
            "selector": {
                "namespaces": ["default"],
                "labelSelectors": selector
            },
            "stressors": {
                "cpu": {
                    "workers": cpu_workers,
                    "load": load
                }
            },
            "duration": duration
        }
    }
    self.experiments.append(experiment)
    return experiment

def generate_memory_stress(self, name: str, selector: Dict[str, str],
                            size_mb: int = 256, duration: str = "30s",
                            mode: str = "one") -> Dict[str, Any]:
    """Генерация MemoryStress эксперимента"""

```

```

        experiment = {
            "apiVersion": "chaos-mesh.org/v1alpha1",
            "kind": "StressChaos",
            "metadata": {
                "name": name,
                "namespace": self.namespace
            },
            "spec": {
                "mode": mode,
                "selector": {
                    "namespaces": ["default"],
                    "labelSelectors": selector
                },
                "stressors": {
                    "memory": {
                        "workers": 1,
                        "size": f"{size_mb}MB"
                    }
                },
                "duration": duration
            }
        }
        self.experiments.append(experiment)
        return experiment

def save_to_files(self, output_dir: str = "./chaos-experiments"):
    """Сохранение всех экспериментов в файлы"""
    import os
    os.makedirs(output_dir, exist_ok=True)

    for exp in self.experiments:
        filename = f"{output_dir}/{exp['metadata']['name']}.yaml"
        with open(filename, 'w') as f:
            yaml.dump(exp, f, default_flow_style=False, allow_unicode=True)
        print(f"✅ Сохранён: {filename}")

def generate_report(self) -> str:
    """Генерация отчёта по экспериментам"""
    report = "=" * 60 + "\n"
    report += "ОТЧЁТ ПО ШАОС-ЭКСПЕРИМЕНТАМ\n"
    report += "=" * 60 + "\n"
    report += f"Всего экспериментов: {len(self.experiments)}\n"
    report += f"Дата генерации: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n"

    for i, exp in enumerate(self.experiments, 1):
        report += f"\n--- Эксперимент {i}: {exp['metadata']['name']} ---\n"
        report += f"Тип: {exp['kind']}\n"
        report += f"Действие: {exp['spec'].get('action', 'N/A')}\n"

```

```

        report += f"Длительность: {exp['spec'].get('duration', 'N/A')}\n"

    return report

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ
# =====

if __name__ == "__main__":
    generator = ChaosExperimentGenerator(namespace="chaos-testing")

    # Генерация экспериментов
    generator.generate_pod_kill(
        name="chaos-pod-kill",
        selector={"app": "test-app"},
        duration="30s"
    )

    generator.generate_network_delay(
        name="chaos-network-delay",
        selector={"app": "test-app"},
        latency="500ms",
        jitter="100ms",
        duration="1m"
    )

    generator.generate_network_loss(
        name="chaos-network-loss",
        selector={"app": "test-app"},
        loss_percent=20,
        duration="1m"
    )

    generator.generate_cpu_stress(
        name="chaos-cpu-stress",
        selector={"app": "test-app"},
        cpu_workers=2,
        load=80,
        duration="30s"
    )

    generator.generate_memory_stress(
        name="chaos-memory-stress",
        selector={"app": "test-app"},
        size_mb=512,
        duration="30s"
    )

    # Сохранение и отчёт

```



```
generator.save_to_files()
print(generator.generate_report())
```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Тип эксперимента	Параметры	Приложение
1	Pod Kill	mode: one, duration: 30s	nginx
2	Pod Kill	mode: all, duration: 15s	httpd
3	Pod Kill	mode: fixed(2), duration: 20s	nginx
4	Network Delay	latency: 200ms, duration: 1m	redis
5	Network Delay	latency: 500ms, jitter: 100ms	httpd

4.2. Уровень 2

№	Типы экспериментов	Параметры	Приложение
1	Pod Kill + Network Delay	latency: 300ms, kill 1 pod	python-app
2	Pod Kill + CPU Stress	cpu: 2 cores, kill 2 pods	node-app
3	Network Delay + Loss	latency: 400ms, loss: 10%	go-app
4	Pod Kill + Memory Stress	memory: 256MB, kill: all	java-app
5	Специализация БПЛА	Network Delay + Loss	ros2-node

4.3. Уровень 3

№	Типы экспериментов	Дополнительные требования
1	Pod Kill + Delay + Loss	Автоматический сбор метрик
2	CPU Stress + Memory Stress	Prometheus + Grafana

3	Все типы	Анализ blast radius
4	Специализация БПЛА + Network	Симуляция потери связи
5	Комплексный (все типы)	Автоматический отчёт

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Chaos Engineering и какие проблемы он решает?
2. Какие принципы лежат в основе Chaos Engineering?
3. Что такое steady state и как его определить для системы?
4. Какие типы chaos-экспериментов поддерживает ChaosMesh?
5. Что такое blast radius и как его контролировать?
6. Как установить ChaosMesh в Kubernetes-кластер?
7. Какой манифест используется для убийства pod'а?
8. Как добавить искусственную задержку сети в ChaosMesh?
9. Как проверить, что эксперимент прошёл успешно?
10. Какие метрики нужно отслеживать во время chaos-экспериментов?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (принципы Chaos Engineering, архитектура ChaosMesh).
4. Практическая часть:
 - Установка ChaosMesh (скриншоты).
 - Тестовое приложение (манифесты, скриншоты).
 - Манифесты экспериментов.

- Результаты экспериментов (скриншоты, лога).
 - Место для скриншота: процесс убийства и восстановления пода.
 - Место для скриншота: задержка сети.
 - Место для скриншота: Dashboard ChaosMesh.
5. Программная часть (листинг кода, скриншоты).
 6. Анализ результатов (время восстановления, влияние метрик).
 7. Ответы на контрольные вопросы (3-5).
 8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
ChaosMesh установлен	2 балла	2 балла	2 балла
Тестовое приложение создано	2 балла	2 балла	2 балла
Проведён эксперимент (1 тип)	2 балла	2 балла	2 балла
Проведены эксперименты (2+ типов)	0 баллов	2 балла	2 балла
Сбор метрик времени восстановления	0 баллов	1 балл	2 балла
Автоматизация экспериментов (Python)	0 баллов	1 балл	2 балла
Анализ влияния на MTTR/Availability	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 12)	Уровень 3 (макс. 16)
5 (отлично)	7-8	11-12	14-16

4 (хорошо)	5-6	9-10	11-13
3 (удовлетворительно)	3-4	6-8	8-10
2 (неудовлетворительно)	<3	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №11. Проведение Game Day и автоматизация chaos-тестов

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки планирования и проведения Game Day (дня хаоса) для проверки устойчивости распределенных систем, координации участников, документирования результатов, а также автоматизации chaos-экспериментов с интеграцией в CI/CD-пайплайн для регулярного тестирования надежности.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Концепция Game Day

Game Day (день хаоса) - это запланированное мероприятие, в ходе которого команда намеренно вносит сбои в production или staging-среду для проверки реакции системы, отработки процедур восстановления и выявления слабых мест. Название происходит от военных учений (war games), где отрабатываются действия в критических ситуациях.

Ключевые элементы Game Day:

Элемент	Описание
Планирование	Определение сценариев, целей, участников, временных рамок
Подготовка	Настройка мониторинга, оповещений, планов отката

Проведение	Выполнение chaos-экспериментов по сценарию
Наблюдение	Сбор метрик, логирование, наблюдение за реакцией системы
Анализ	Разбор результатов, выявление проблем, формирование action items

2.2. Роли и ответственность в Game Day

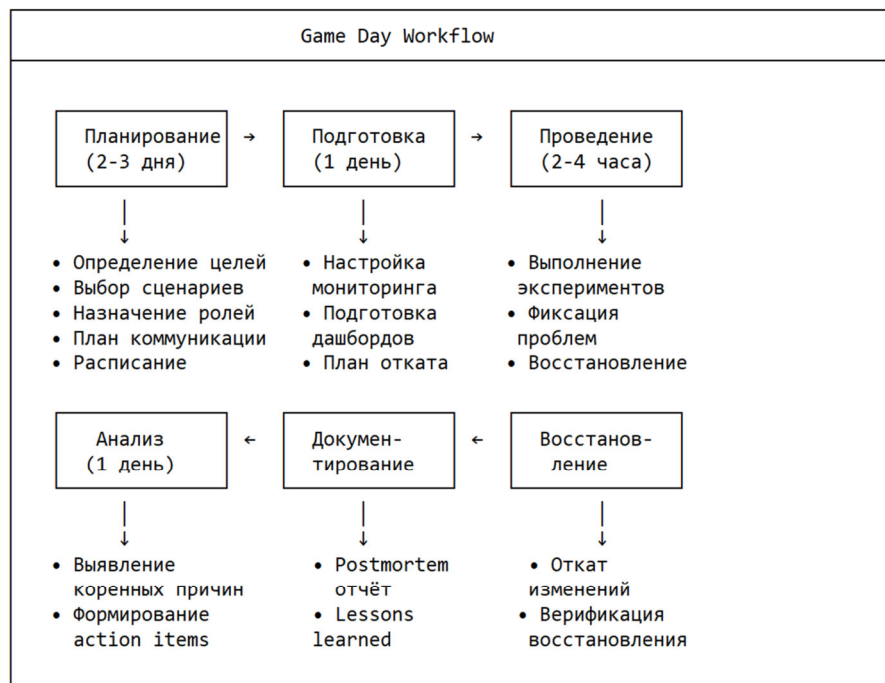
Роль	Ответственность
Game Master	Планирует и проводит эксперименты, координирует команду
Наблюдатель (Observer)	Отслеживает метрики, логи, фиксирует проблем
Инженер (Engineer)	Реагирует на инциденты, выполняет восстановление
Секретарь (Scribe)	Документирует хронологию событий, проблемы, решения
Менеджер (Manager)	Оценивает влияние на бизнес, принимает решения о stop/continue

2.3. Типовые сценарии Game Day

Категория	Сценарий	Цель
Отказы инфраструктуры	Отказ узла Kubernetes	Проверка self-healing
	Отказ зоны доступности (Availability Zone)	Проверка отказоустойчивости при распределении по нескольким зонам
	Отказ сети	Проверка retry, timeout

Нагрузка	CPU spike	Проверка autoscaling
	Memory leak	Проверка OOM recovery
Данные	Коррупция БД	Проверка backup/restore
	Задержка репликации	Проверка read-after-write
Зависимости	Отказ внешнего API	Проверка circuit breaker
	Задержка очереди	Проверка graceful degradation

2.4. Этапы проведения Game Day



2.5. Автоматизация chaos-тестов

Chaos Mesh + Argo Rollouts Integration:

```

apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: myapp

```

```

spec:
  strategy:
    canary:
      steps:
        - setWeight: 10
        - pause: {duration: 5m}
        - setWeight: 50
        - pause: {duration: 10m}
        - setWeight: 100
      analysis:
        templates:
          - templateName: success-rate
        startingStep: 1

```

Автоматический chaos-тест в CI/CD:

```

# .gitlab-ci.yml
stages:
  - deploy
  - chaos-test

deploy:
  stage: deploy
  script:
    - kubectl apply -f deployment.yaml
    - kubectl rollout status deployment/myapp

chaos-test:
  stage: chaos-test
  script:
    - kubectl apply -f chaos/pod-kill.yaml
    - sleep 60
    - kubectl get pods -l app=myapp
    - python3 scripts/analyze_recovery.py
  after_script:
    - kubectl delete -f chaos/pod-kill.yaml --ignore-not-found
  only:
    - main

```

2.6. Пример плана Game Day

//markdown

План Game Day 001

1. Общая информация

- Дата: 15.04.2025
- Время: 14:00 - 17:00 MSK
- Среда: Staging (изолированная)
- Game Master: Иванов П.
- Участники: SRE (2), разработка (2), QA (1)

2. Цели

- 1. Проверить автоматическое восстановление после отказа пода
- 2. Проверить механизм retry при задержке сети
- 3. Отработать процедуру отката (rollback)

3. Сценарии

№	Сценарий	Длительность	Ожидание
1	Pod kill (1 экземпляр)	5 мин	Восстановление <30 сек
2	Network delay 500ms	10 мин	Error rate <1%
3	CPU stress (80% load)	10 мин	Autoscaling срабатывает
4	Отказ внешнего API	15 мин	Circuit breaker open

4. Метрики успеха

Метрика	Целевое значение
Availability	>99.9%
Error rate	<0.5%
MTTR	<60 сек
P99 latency	<500ms

5. Хронология

Время	Действие	Ответственный
14:00	Kick-off, обсуждение плана	Game Master

14:10	Подготовка мониторинга	SRE
14:20	Сценарий 1	Game Master
14:35	Сценарий 2	Game Master
15:00	Сценарий 3	Game Master
15:30	Сценарий 4	Game Master
16:00	Анализ, сбор метрик	Все
16:30	Дебрифинг, action items	Game Master

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Скрипт автоматизации Game Day

```

"""
Лабораторная работа №11
Автоматизация Game Day и chaos-тестов
"""

import time
import json
import subprocess
import logging
from datetime import datetime
from typing import Dict, Any, List, Optional
from dataclasses import dataclass, field
from enum import Enum

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class ScenarioStatus(Enum):
    PENDING = "pending"
    RUNNING = "running"
    SUCCESS = "success"
    FAILED = "failed"
    SKIPPED = "skipped"

@dataclass
class Scenario:
    """Сценарий Game Day"""
    name: str
    description: str

```

```

    chaos_manifest: str
    duration_seconds: int
    expected_metrics: Dict[str, float]
    status: ScenarioStatus = ScenarioStatus.PENDING
    start_time: Optional[str] = None
    end_time: Optional[str] = None
    observations: List[str] = field(default_factory=list)

@dataclass
class GameDayReport:
    """Отчёт Game Day"""
    date: str
    duration_minutes: int
    scenarios: List[Scenario]
    overall_success: bool
    action_items: List[str]
    lessons_learned: List[str]

class ChaosRunner:
    """Запуск chaos-экспериментов"""

    @staticmethod
    def apply_manifest(manifest_path: str, namespace: str = "default") -> bool:
        """Применение манифеста chaos-эксперимента"""
        try:
            cmd = ["kubectl", "apply", "-f", manifest_path, "-n", namespace]
            result = subprocess.run(cmd, capture_output=True, text=True)
            if result.returncode == 0:
                logger.info(f"Applied manifest: {manifest_path}")
                return True
            else:
                logger.error(f"Failed to apply manifest: {result.stderr}")
                return False
        except Exception as e:
            logger.error(f"Error applying manifest: {e}")
            return False

    @staticmethod
    def delete_manifest(manifest_path: str, namespace: str = "default") -> bool:
        """Удаление манифеста chaos-эксперимента"""
        try:
            cmd = ["kubectl", "delete", "-f", manifest_path, "-n", namespace, "--ignore-not-found"]
            result = subprocess.run(cmd, capture_output=True, text=True)
            if result.returncode == 0:
                logger.info(f"Deleted manifest: {manifest_path}")
                return True
            else:
                logger.error(f"Failed to delete manifest: {result.stderr}")

```

```

        return False
    except Exception as e:
        logger.error(f"Error deleting manifest: {e}")
        return False

class MetricsCollector:
    """Сбор метрик во время Game Day"""

    @staticmethod
    def get_pod_status(namespace: str = "default", label: str = "app=myapp") -> Dict[str, Any]:
        """Получение статуса подов"""
        try:
            cmd = ["kubectl", "get", "pods", "-n", namespace, "-l", label, "-o",
"json"]

            result = subprocess.run(cmd, capture_output=True, text=True)
            data = json.loads(result.stdout)

            running = 0
            pending = 0
            failed = 0
            for item in data.get("items", []):
                status = item["status"]["phase"]
                if status == "Running":
                    running += 1
                elif status == "Pending":
                    pending += 1
                else:
                    failed += 1

            return {
                "total": len(data.get("items", [])),
                "running": running,
                "pending": pending,
                "failed": failed
            }
        except Exception as e:
            logger.error(f"Error getting pod status: {e}")
            return {"total": 0, "running": 0, "pending": 0, "failed": 0}

    @staticmethod
    def check_deployment_status(deployment: str, namespace: str = "default") -> bool:
        """Проверка статуса deployment"""
        try:
            cmd = ["kubectl", "rollout", "status", f"deployment/{deployment}", "-n", namespace, "--timeout=30s"]
            result = subprocess.run(cmd, capture_output=True, text=True)
            return result.returncode == 0

```

```

        except Exception as e:
            logger.error(f"Error checking deployment status: {e}")
            return False

class GameDayRunner:
    """Запуск Game Day"""

    def __init__(self, namespace: str = "default", chaos_dir: str = "./chaos"):
        self.namespace = namespace
        self.chaos_dir = chaos_dir
        self.chaos_runner = ChaosRunner()
        self.metrics_collector = MetricsCollector()
        self.scenarios: List[Scenario] = []
        self.report: Optional[GameDayReport] = None

    def add_scenario(self, scenario: Scenario):
        """Добавление сценария"""
        self.scenarios.append(scenario)

    def run_scenario(self, scenario: Scenario) -> bool:
        """Запуск отдельного сценария"""
        logger.info(f"Starting scenario: {scenario.name}")
        scenario.status = ScenarioStatus.RUNNING
        scenario.start_time = datetime.now().isoformat()

        # Применение chaos-манифеста
        manifest_path = f"{self.chaos_dir}/{scenario.chaos_manifest}"
        if not self.chaos_runner.apply_manifest(manifest_path, self.namespace):
            scenario.status = ScenarioStatus.FAILED
            scenario.observations.append("Failed to apply chaos manifest")
            return False

        # Наблюдение
        logger.info(f"Observing for {scenario.duration_seconds} seconds...")
        observation_interval = 10
        for i in range(0, scenario.duration_seconds, observation_interval):
            time.sleep(observation_interval)
            pod_status = self.metrics_collector.get_pod_status(self.namespace)
            logger.info(f"Pod status: {pod_status}")

            # Проверка критических условий
            if pod_status["failed"] > 0:
                scenario.observations.append(f"Detected failed pods: {pod_status['failed']}")

        # Удаление chaos-манифеста
        self.chaos_runner.delete_manifest(manifest_path, self.namespace)

        # Проверка восстановления

```

```

        time.sleep(30)
        if self.metrics_collector.check_deployment_status("myapp",
self.namespace):
            scenario.status = ScenarioStatus.SUCCESS
            scenario.observations.append("System recovered successfully")
        else:
            scenario.status = ScenarioStatus.FAILED
            scenario.observations.append("System failed to recover")

        scenario.end_time = datetime.now().isoformat()
        return scenario.status == ScenarioStatus.SUCCESS

def run_all(self) -> GameDayReport:
    """Запуск всех сценариев"""
    logger.info("=" * 60)
    logger.info("STARTING GAME DAY")
    logger.info("=" * 60)

    start_time = datetime.now()
    all_success = True

    for scenario in self.scenarios:
        logger.info(f"\n--- Scenario: {scenario.name} ---")
        success = self.run_scenario(scenario)
        if not success:
            all_success = False
            logger.warning(f"Scenario {scenario.name} FAILED")
        else:
            logger.info(f"Scenario {scenario.name} PASSED")
        logger.info("")

    end_time = datetime.now()
    duration_minutes = int((end_time - start_time).total_seconds() / 60)

    # Формирование action items
    action_items = []
    for scenario in self.scenarios:
        if scenario.status == ScenarioStatus.FAILED:
            action_items.append(f"Investigate failure in scenario:
{scenario.name}")
            for obs in scenario.observations:
                if "failed" in obs.lower():
                    action_items.append(f" - Address pod recovery: {obs}")

    # Формирование lessons learned
    lessons_learned = [
        "Chaos experiments should be run regularly",
        "Monitoring must be configured before experiments",
        "Recovery time should be measured and optimized"

```

```

    ]

    self.report = GameDayReport(
        date=datetime.now().strftime("%Y-%m-%d"),
        duration_minutes=duration_minutes,
        scenarios=self.scenarios,
        overall_success=all_success,
        action_items=action_items,
        lessons_learned=lessons_learned
    )

    return self.report

def print_report(self):
    """Вывод отчёта"""
    if not self.report:
        return

    print("\n" + "=" * 60)
    print("GAME DAY REPORT")
    print("=" * 60)
    print(f>Date: {self.report.date}<
    print(f>Duration: {self.report.duration_minutes} minutes")
    print(f>Overall success: {'✅ YES' if self.report.overall_success else
'❌ NO'})

    print("\n--- SCENARIOS ---")
    for scenario in self.report.scenarios:
        status_emoji = "✅" if scenario.status == ScenarioStatus.SUCCESS
else "❌"
        print(f">{status_emoji} {scenario.name}: {scenario.status.value}")
        if scenario.observations:
            for obs in scenario.observations[:3]:
                print(f"    - {obs}")

    if self.report.action_items:
        print("\n--- ACTION ITEMS ---")
        for item in self.report.action_items:
            print(f"    • {item}")

    print("\n--- LESSONS LEARNED ---")
    for lesson in self.report.lessons_learned:
        print(f"    • {lesson}")
    print("=" * 60)

# =====
# ПРИМЕР ВЫПОЛНЕНИЯ
# =====

```

```

def run_game_day_example():
    """Пример запуска Game Day"""

    # Создание сценариев
    scenarios = [
        Scenario(
            name="Pod Kill Test",
            description="Убийство одного пода, проверка восстановления",
            chaos_manifest="pod-kill-experiment.yaml",
            duration_seconds=60,
            expected_metrics={"mttr_seconds": 30}
        ),
        Scenario(
            name="Network Delay Test",
            description="Задержка сети 300ms, проверка timeout",
            chaos_manifest="network-delay-experiment.yaml",
            duration_seconds=120,
            expected_metrics={"error_rate_percent": 5}
        ),
        Scenario(
            name="CPU Stress Test",
            description="Нагрузка на CPU, проверка autoscaling",
            chaos_manifest="cpu-stress-experiment.yaml",
            duration_seconds=120,
            expected_metrics={"autoscaling_triggered": True}
        )
    ]

    # Запуск Game Day
    runner = GameDayRunner(namespace="default", chaos_dir="./chaos")
    for scenario in scenarios:
        runner.add_scenario(scenario)

    report = runner.run_all()
    runner.print_report()

    # Сохранение отчёта
    with open("game_day_report.json", "w", encoding="utf-8") as f:
        json.dump({
            "date": report.date,
            "duration_minutes": report.duration_minutes,
            "overall_success": report.overall_success,
            "action_items": report.action_items,
            "lessons_learned": report.lessons_learned
        }, f, indent=2, ensure_ascii=False)

    print("\n✅ Report saved to game_day_report.json")

if __name__ == "__main__":

```

```
run_game_day_example()
```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Сценарии Game Day	Количество	Продолжительность
1	Pod Kill	1	30 мин
2	Network Delay	1	30 мин
3	CPU Stress	1	30 мин
4	Memory Stress	1	30 мин
5	Network Loss	1	30 мин

4.2. Уровень 2

№	Сценарии Game Day	Количество	Дополнительные требования
1	Pod Kill + Network Delay	2	Сбор метрик восстановления
2	CPU Stress + Memory Stress	2	Отслеживание autoscaling
3	Network Delay + Loss	2	Анализ error rate
4	Pod Kill + CPU Stress	2	Документирование наблюдений
5	Специализация БПЛА	2	Симуляция потери связи

4.3. Уровень 3

№	Сценарии Game Day	Количество	Дополнительные требования

1	Pod Kill + Delay + Loss	3	Python-скрипт автоматизации
2	Full chaos suite	4	Prometheus + Grafana интеграция
3	Complex scenarios	3	Анализ MTTR/MTBF
4	Специализация БПЛА (продвинутый)	3	Полная симуляция полёта
5	Enterprise Game Day	4	Полный отчёт с action items

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Game Day и какие цели он преследует?
2. Какие роли участвуют в проведении Game Day?
3. Какие этапы включает подготовка Game Day?
4. Что такое blast radius и как его контролировать?
5. Как автоматизировать chaos-тесты в CI/CD?
6. Какие метрики нужно отслеживать во время Game Day?
7. Как документировать результаты Game Day?
8. Что делать, если эксперимент пошёл не по плану?
9. Как часто следует проводить Game Day?
10. Как интегрировать результаты Game Day в процесс разработки?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (Game Day, роли, этапы).
4. Практическая часть:
 - План Game Day (сценарии, расписание).
 - Ход выполнения (скриншоты, логи).

- Результаты экспериментов.
 - Место для скриншота: Game Day выполнение.
5. Программная часть (Уровень 3):
- Листинг кода Python.
 - Место для скриншота: вывод программы.
6. Анализ результатов (action items, lessons learned).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
План Game Day разработан	2 балла	2 балла	2 балла
Сценарии выполнены (1/2+)	2 балла	2 балла	2 балла
Результаты задокументированы	2 балла	2 балла	2 балла
Сбор метрик восстановления	0 баллов	1 балл	2 балла
Action items сформулированы	1 балл	2 балла	2 балла
Автоматизация (Python)	0 баллов	1 балл	2 балла
Интеграция с CI/CD	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 9)	Уровень 2 (макс. 12)	Уровень 3 (макс. 16)
5 (отлично)	8-9	11-12	14-16
4 (хорошо)	6-7	9-10	11-13

3 (удовлетворительно)	4-5	6-8	8-10
2 (неудовлетворительно)	<4	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №12. Создание автоматизированных runbooks и playbooks

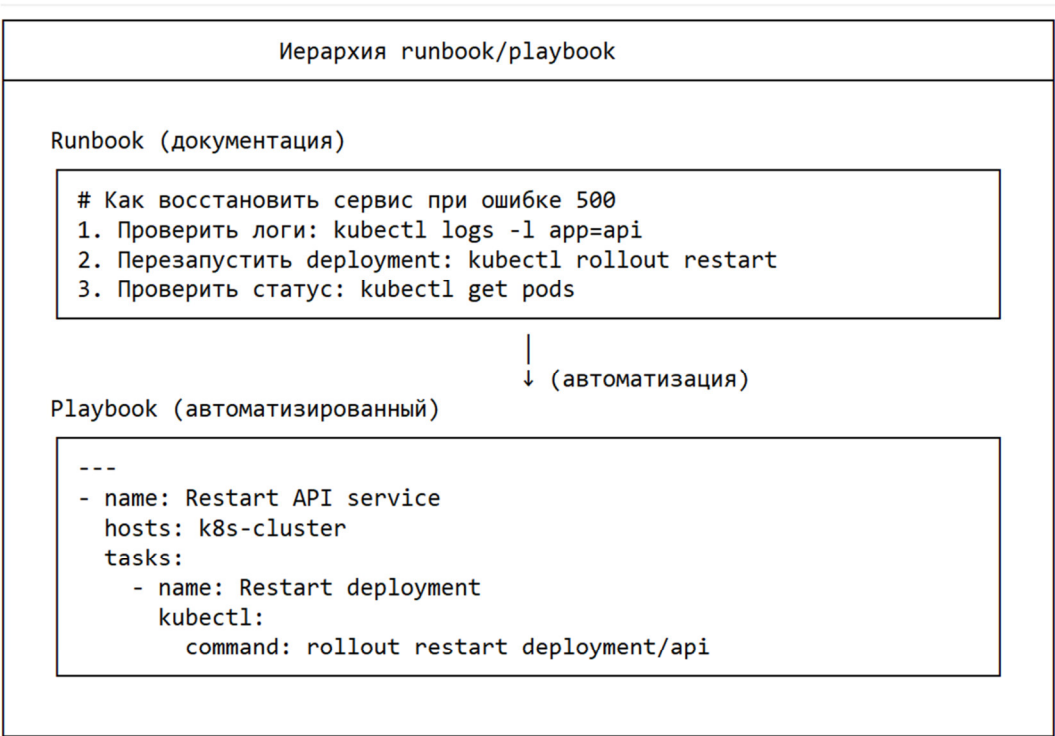
1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки разработки автоматизированных runbooks и playbooks для типовых инцидентов, интеграции их с системами мониторинга, тестирования эффективности и внедрения в процессы инцидент-менеджмента для сокращения времени реагирования (MTTR) и устранения рутинных операций.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Runbooks и playbooks: определения и различия

Термин	Определение	Пример
Runbook	Документированная процедура действий при инциденте (может быть как ручной, так и автоматизированной)	«При ошибке 500 выполнить перезапуск сервиса»
Playbook	Автоматизированный сценарий реагирования, выполняемый без участия человека или с минимальным подтверждением	Ansible playbook для перезапуска сервиса при срабатывании алерта
Action	Конкретное автоматическое действие	Webhook-вызов для перезапуска pod'a

Иерархия:



2.2. Уровни автоматизации runbooks

Уровень	Описание	MTTR	Пример
Уровень 0: Ручной	Инженер получает уведомление и выполняет действия вручную	15-30 мин	Email → SSH → restart service
Уровень 1: Полуавтоматический	Уведомление + ссылка на runbook с кнопками действий	5-15 мин	Slack-уведомление с кнопкой «Restart»
Уровень 2: Автоматический (с подтверждением)	Система предлагает выполнить действие, требуется подтверждение	2-5 мин	Alertmanager → Webhook → авто-рестарт с confirm

Уровень 3: Полностью автоматический (self-healing)	Система реагирует без участия человека	<1 мин	Автоматическое масштабирование, перезапуск, откат

2.3. Структура runbook-документа

```
# Runbook: [Название инцидента]

## Метаданные
- **ID:** RB-2025-001
- **Версия:** 1.0
- **Ответственный:** SRE команда
- **Теги:** #critical #k8s #api

## Триггер
- **Alert name:** HighErrorRate
- **Условие:** error_rate > 5% за 5 минут
- **Severity:** critical

## Диагностика
1. Проверить логи: `kubectl logs -l app=api --tail=100`
2. Проверить метрики: http://grafana.example.com/d/api-errors
3. Проверить БД: `kubectl exec -it db-0 -- psql -c "SELECT * FROM pg_stat_activity"`

## Действия
### Автоматические ([AUTO])
- [AUTO] Перезапустить сервис: `kubectl rollout restart deployment/api`

### Ручные ([MANUAL])
- [MANUAL] Если не помогло – откатить версию: `kubectl rollout undo deployment/api`
- [MANUAL] Эскалация на SRE при длительности >15 минут

## Эскалация
- **Уровень 1:** On-call инженер (Telegram)
- **Уровень 2:** SRE команда (Slack)
- **Уровень 3:** Руководитель (SMS)

## Проверка восстановления
- `kubectl get pods -l app=api`
- `curl -f http://api/health`
```

2.4. Интеграция runbook с Alertmanager

Alertmanager с webhook для автоматического runbook:

```
# alertmanager.yml
receivers:
- name: 'critical'
  webhook_configs:
    - url: 'http://runbook-executor:5000/execute'
      send_resolved: true
  slack_configs:
    - channel: '#critical-alerts'
      title: '🚨 CRITICAL: {{ .GroupLabels.alertname }}'
      text: |
        *Runbook:* {{ .Annotations.runbook_url }}
        *Auto-action:* Restarting deployment...
```

2.5. Пример playbook на Ansible

```
# restart-api-playbook.yml
---
- name: Restart API service after critical alert
  hosts: k8s-master
  gather_facts: no

  vars:
    namespace: "default"
    deployment: "api"

  tasks:
    - name: Check current deployment status
      command: |
        kubectl get deployment {{ deployment }} -n {{ namespace }} -o json
      register: deployment_status
      changed_when: false

    - name: Restart deployment
      command: |
        kubectl rollout restart deployment/{{ deployment }} -n {{ namespace }}
      register: restart_result

    - name: Wait for rollout to complete
      command: |
        kubectl rollout status deployment/{{ deployment }} -n {{ namespace }} --
      timeout=60s
      register: rollout_status
```

```

- name: Verify service health
  uri:
    url: "http://api.example.com/health"
    method: GET
    status_code: 200
    timeout: 10
  register: health_check
  retries: 3
  delay: 10
  until: health_check.status == 200

- name: Send notification
  slack:
    token: "{{ slack_token }}"
    channel: "#alerts"
    msg: "✅ API service restored after restart"
  when: health_check.status == 200

```

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Runbook Executor (Python webhook-сервис)

```

"""
Лабораторная работа №12
Runbook Executor — автоматическое выполнение runbook
"""

import json
import subprocess
import logging
from datetime import datetime
from typing import Dict, Any, Optional
from flask import Flask, request, jsonify

app = Flask(__name__)
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Хранилище выполненных runbook
execution_log = []

class RunbookExecutor:
    """Выполнитель runbook"""

    @staticmethod

```

```

    def restart_kubernetes_deployment(namespace: str, deployment: str) ->
Dict[str, Any]:
    """Перезапуск deployment в Kubernetes"""
    logger.info(f"Executing restart: {namespace}/{deployment}")
    try:
        cmd = f"kubectl rollout restart deployment/{deployment} -n
{namespace}"
        result = subprocess.run(cmd, shell=True, capture_output=True,
text=True)
        return {
            "success": result.returncode == 0,
            "output": result.stdout,
            "error": result.stderr
        }
    except Exception as e:
        return {"success": False, "error": str(e)}

    @staticmethod
    def rollback_deployment(namespace: str, deployment: str, revision: int =
None) -> Dict[str, Any]:
    """Откат deployment"""
    logger.info(f"Executing rollback: {namespace}/{deployment}")
    try:
        if revision:
            cmd = f"kubectl rollout undo deployment/{deployment} -n
{namespace} --to-revision={revision}"
        else:
            cmd = f"kubectl rollout undo deployment/{deployment} -n
{namespace}"
        result = subprocess.run(cmd, shell=True, capture_output=True,
text=True)
        return {
            "success": result.returncode == 0,
            "output": result.stdout,
            "error": result.stderr
        }
    except Exception as e:
        return {"success": False, "error": str(e)}

    @staticmethod
    def scale_deployment(namespace: str, deployment: str, replicas: int) ->
Dict[str, Any]:
    """Масштабирование deployment"""
    logger.info(f"Executing scale: {namespace}/{deployment} -> {replicas}
replicas")
    try:
        cmd = f"kubectl scale deployment/{deployment} -n {namespace} --
replicas={replicas}"

```



```

        result = subprocess.run(cmd, shell=True, capture_output=True,
text=True)
        return {
            "success": result.returncode == 0,
            "output": result.stdout,
            "error": result.stderr
        }
    except Exception as e:
        return {"success": False, "error": str(e)}

    @staticmethod
    def get_pod_status(namespace: str, label: str) -> Dict[str, Any]:
        """Получение статуса подов"""
        try:
            cmd = f"kubectl get pods -n {namespace} -l {label} -o json"
            result = subprocess.run(cmd, shell=True, capture_output=True,
text=True)

            import json as json_lib
            data = json_lib.loads(result.stdout)
            running = sum(1 for item in data.get("items", [])
                           if item["status"]["phase"] == "Running")

            return {
                "success": True,
                "running_pods": running,
                "total_pods": len(data.get("items", []))
            }
        except Exception as e:
            return {"success": False, "error": str(e)}

executor = RunbookExecutor()

# Определение runbook для разных типов алертов
RUNBOOKS = {
    "HighErrorRate": {
        "name": "Restart API on high error rate",
        "actions": [
            {
                "name": "restart_deployment",
                "function": executor.restart_kubernetes_deployment,
                "params": {"namespace": "default", "deployment": "api"}
            },
            {
                "name": "verify_recovery",
                "function": executor.get_pod_status,
                "params": {"namespace": "default", "label": "app=api"}
            }
        ]
    },
    "HighLatency": {

```

```

        "name": "Scale up on high latency",
        "actions": [
            {
                "name": "scale_up",
                "function": executor.scale_deployment,
                "params": {"namespace": "default", "deployment": "api",
"replicas": 5}
            }
        ]
    },
    "InstanceDown": {
        "name": "Rollback on instance down",
        "actions": [
            {
                "name": "rollback",
                "function": executor.rollback_deployment,
                "params": {"namespace": "default", "deployment": "api"}
            }
        ]
    }
}

@app.route('/webhook/runbook', methods=['POST'])
def runbook_webhook():
    """Webhook для выполнения runbook по алерту"""
    data = request.json
    alert_name = data.get('alerts', [{}])[0].get('labels', {}).get('alertname',
'Unknown')

    logger.info(f"Received alert: {alert_name}")

    if alert_name not in RUNBOOKS:
        logger.warning(f"No runbook defined for alert: {alert_name}")
        return jsonify({"status": "no_runbook", "alert": alert_name}), 200

    runbook = RUNBOOKS[alert_name]
    execution_id = f"{alert_name}-{datetime.now().strftime('%Y%m%d%H%M%S')}"

    results = []
    for action in runbook["actions"]:
        logger.info(f"Executing action: {action['name']}")
        result = action["function"](**action["params"])
        results.append({
            "action": action["name"],
            "result": result
        })

    if not result.get("success", True):

```

```

        logger.error(f"Action {action['name']} failed:
{result.get('error')}")
        break

    execution_log.append({
        "execution_id": execution_id,
        "alert": alert_name,
        "timestamp": datetime.now().isoformat(),
        "results": results,
        "status": "completed"
    })

    return jsonify({
        "status": "executed",
        "execution_id": execution_id,
        "runbook": runbook["name"],
        "results": results
    }), 200

@app.route('/runbook/log', methods=['GET'])
def get_log():
    """Получение лога выполнения runbook"""
    return jsonify(execution_log), 200

@app.route('/runbook/status/<execution_id>', methods=['GET'])
def get_status(execution_id):
    """Статус выполнения runbook"""
    for entry in execution_log:
        if entry["execution_id"] == execution_id:
            return jsonify(entry), 200
    return jsonify({"error": "Execution not found"}), 404

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

3.2. Конфигурация Alertmanager для запуска runbook

```

# alertmanager.yml
global:
    slack_api_url: 'https://hooks.slack.com/services/...'

route:
    group_by: ['alertname']
    group_wait: 10s
    group_interval: 10s
    repeat_interval: 1h
    receiver: 'default'

```

```

routes:
  - match:
      severity: critical
      receiver: runbook-executor
      continue: true
  - match:
      severity: warning
      receiver: slack

receivers:
  - name: 'runbook-executor'
    webhook_configs:
      - url: 'http://runbook-executor:5000/webhook/runbook'
        send_resolved: true

  - name: 'slack'
    slack_configs:
      - channel: '#warnings'
        title: ' {{ .GroupLabels.alertname }}'

```

3.3. Генерация runbook-документации (Python)

```

"""
Лабораторная работа №12
Генерация runbook-документации в Markdown
"""

import json
from datetime import datetime
from typing import List, Dict, Any
from dataclasses import dataclass, asdict

@dataclass
class RunbookStep:
    """Шаг runbook"""
    type: str # "auto" или "manual"
    command: str
    description: str

@dataclass
class Runbook:
    """Runbook документ"""
    id: str
    name: str
    version: str
    owner: str

```

```

tags: List[str]
trigger_alert: str
severity: str
diagnostics: List[str]
steps: List[RunbookStep]
escalation: Dict[str, str]
verification: List[str]

class RunbookGenerator:
    """Генератор runbook-документации"""

    @staticmethod
    def to_markdown(runbook: Runbook) -> str:
        """Экспорт в Markdown"""
        md = f"""# Runbook: {runbook.name}

## Метаданные
- **ID:** {runbook.id}
- **Версия:** {runbook.version}
- **Ответственный:** {runbook.owner}
- **Теги:** {' '.join(runbook.tags)}

## Триггер
- **Alert name:** {runbook.trigger_alert}
- **Severity:** {runbook.severity}

## Диагностика
"""
        for d in runbook.diagnostics:
            md += f"1. {d}\n"

        md += f"""

## Действия
"""
        for step in runbook.steps:
            if step.type == "auto":
                md += f"""### [AUTO]
{step.description}\n```bash\n{step.command}\n```"""
            else:
                md += f"""### [MANUAL]
{step.description}\n```bash\n{step.command}\n```"""

        md += f"""

## Эскалация
- **Уровень 1:** {runbook.escalation.get('level1', 'N/A')}
- **Уровень 2:** {runbook.escalation.get('level2', 'N/A')}
- **Уровень 3:** {runbook.escalation.get('level3', 'N/A')}

## Проверка восстановления

```

```

"""
    for v in runbook.verification:
        md += f"- {v}\n"

    return md

# Пример runbook
example_runbook = Runbook(
    id="RB-2025-001",
    name="Restore API service on high error rate",
    version="1.0",
    owner="SRE Team",
    tags=["critical", "api", "k8s"],
    trigger_alert="HighErrorRate",
    severity="critical",
    diagnostics=[
        "Проверить логи: `kubectl logs -l app=api --tail=100`",
        "Проверить метрики: http://grafana.example.com/d/api-errors",
        "Проверить БД: `kubectl exec -it db-0 -- psql -c \"SELECT * FROM pg_stat_activity\"`"
    ],
    steps=[
        RunbookStep("auto", "kubectl rollout restart deployment/api -n default",
"Перезапустить API сервис"),
        RunbookStep("manual", "kubectl rollout undo deployment/api -n default",
"Откатить версию (если не помогло)",
        RunbookStep("manual", "kubectl get pods -n default -l app=api",
"Проверить статус подов")
    ],
    escalation={
        "level1": "On-call инженер (Telegram)",
        "level2": "SRE команда (Slack)",
        "level3": "Руководитель (SMS)"
    },
    verification=[
        "`kubectl get pods -l app=api` - все поды в статусе Running",
        "`curl -f http://api/health` - ответ 200 OK"
    ]
)

if __name__ == "__main__":
    generator = RunbookGenerator()
    markdown = generator.to_markdown(example_runbook)
    print(markdown)

    with open("runbook_HighErrorRate.md", "w", encoding="utf-8") as f:
        f.write(markdown)

    print("\n Runbook сохранён: runbook_HighErrorRate.md")

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Тип инцидента	Runbook действия	Инструмент
1	HighErrorRate (>5%)	Restart deployment	kubectl
2	HighLatency (>500ms)	Scale up replicas	kubectl
3	InstanceDown	Restart pod	kubectl
4	DatabaseConnectionError	Restart database	kubectl
5	DiskSpaceWarning	Clean logs	kubectl exec

4.2. Уровень 2

№	Тип инцидента	Runbook действия	Интеграция
	HighErrorRate + HighLatency	Restart + Scale	Alertmanager webhook
	MemoryLeak	Restart + Debug	Slack уведомления
	SSLExpiringSoon	Renew certificate + Notify	Email уведомления
	DatabaseReplicationLag	Check + Restart replica	Prometheus алерты
	Специализация БПЛА	Restart ROS node	Telemetry monitoring

4.3. Уровень 3

№	Тип инцидента	Runbook действия	Дополнительные требования
1	Full cascade failure	Multi-step recovery	Python webhook сервис
2	Security breach	Isolate + Rotate keys	ChatOps интеграция
3	Data corruption	Restore from backup	Backup verification
4	Специализация БПЛА (продвинутый)	Fail-safe + Return to base	Симуляция ОТА
5	Complex incident	Full runbook suite	Мониторинг выполнения

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое runbook и чем он отличается от playbook?
2. Какие уровни автоматизации runbook существуют?
3. Как интегрировать runbook с Alertmanager?
4. Какие действия можно автоматизировать при критическом инциденте?
5. Как проверить, что runbook выполнен успешно?
6. Какие данные нужно логировать при выполнении runbook?
7. Как часто следует обновлять runbook?
8. Какие метрики эффективности runbook вы знаете?
9. Как организовать эскалацию в runbook?
10. Какие риски связаны с автоматическим выполнением runbook?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (runbook, playbook, уровни автоматизации).

4. Практическая часть:

- Runbook документ (Markdown).
- Конфигурация Alertmanager webhook.
- Результаты выполнения (скриншоты, логи).
- Место для скриншота: runbook в документации.
- Место для скриншота: выполнение runbook.

5. Программная часть (Уровень 3):

- Листинг кода Python.
- Место для скриншота: вывод программы.

6. Анализ результатов (эффективность runbook, MTTR).

7. Ответы на контрольные вопросы (3-5).

8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Runbook документ создан	2 балла	2 балла	2 балла
Все разделы runbook заполнены	2 балла	2 балла	2 балла
Действия описаны (auto/manual)	1 балл	2 балла	2 балла
Интеграция с Alertmanager	0 баллов	2 балла	2 балла
Автоматическое выполнение	0 баллов	1 балл	2 балла
Python webhook сервис	0 баллов	0 баллов	2 балла
Мониторинг выполнения	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 7)	Уровень 2 (макс. 11)	Уровень 3 (макс. 16)

5 (отлично)	6-7	10-11	14-16
4 (хорошо)	5	8-9	11-13
3 (удовлетворительно)	3-4	6-7	8-10
2 (неудовлетворительно)	<3	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №13. Реализация ChatOps для инцидент-менеджмента

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки настройки ChatOps-бота для управления инцидентами, автоматизации эскалации, уведомлений и сбора информации для сокращения времени реагирования (MTTR), а также интеграции бота с системами мониторинга (Prometheus, Alertmanager) и системами управления инцидентами.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Концепция ChatOps

ChatOps - подход, интегрирующий чат-ботов в процессы разработки и эксплуатации, позволяющий выполнять команды и получать уведомления непосредственно в мессенджере (Telegram, Slack, Microsoft Teams) без необходимости переключения между инструментами.

Ключевые принципы ChatOps:

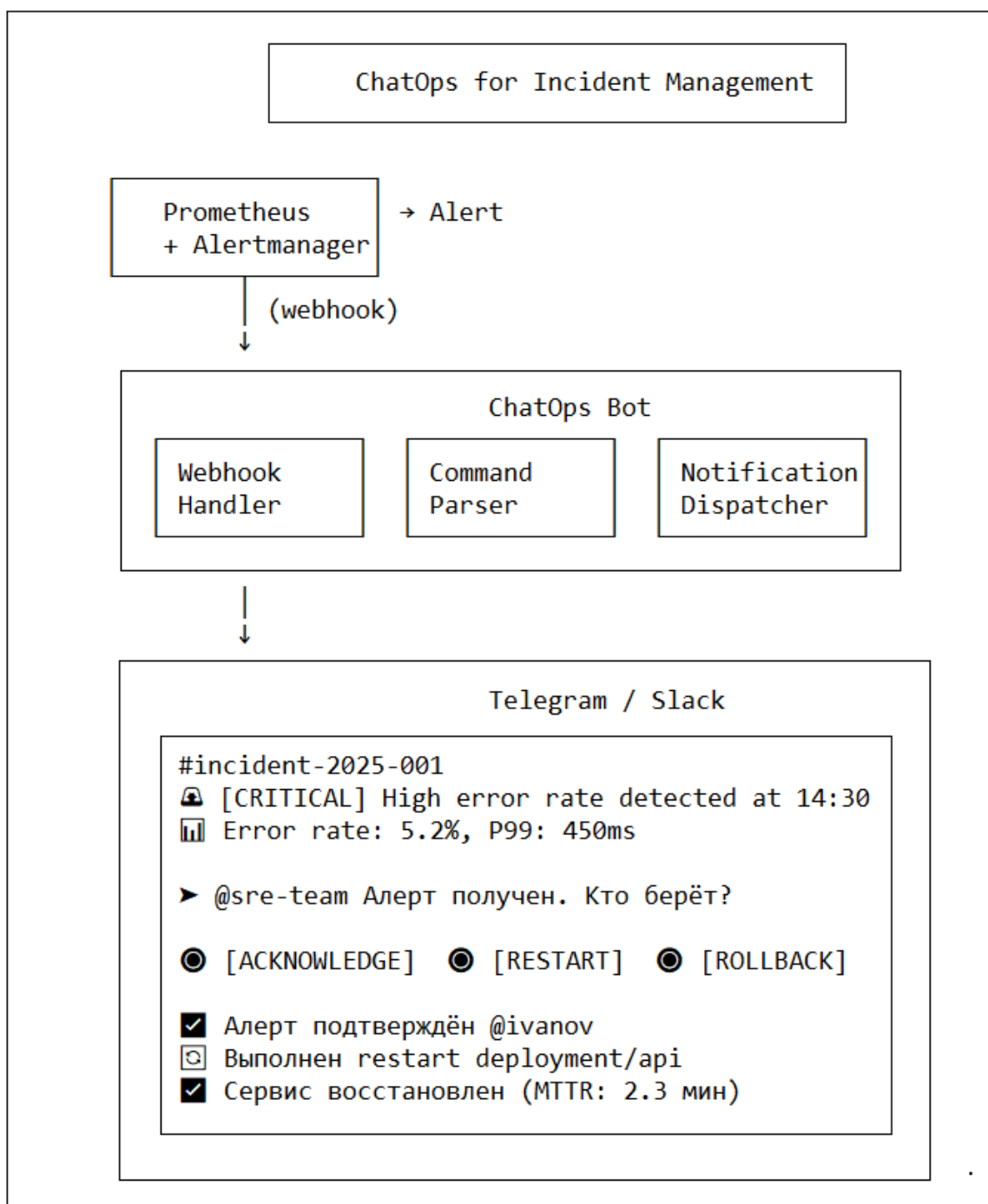
Принцип	Описание
Единый интерфейс	Все операции выполняются из чата
Прозрачность	Все действия логируются и видны команде

Автоматизация	Ручные операции заменяются командами бота
Коллаборация	Команда совместно реагирует на инциденты
Мобильность	Реагирование с любого устройства

2.2. Преимущества ChatOps для инцидент-менеджмента

Преимущество	Описание
Сокращение MTTR	Быстрое выполнение команд без переключения контекста
Прозрачность	Все действия логируются в канале
Документирование	История инцидента сохраняется в чате
Эскалация	Автоматическое привлечение нужных специалистов
Мобильность	Реагирование с телефона/планшета

2.3. Архитектура ChatOps для инцидент-менеджмента



2.4. Типовые команды ChatOps-бота

Команда	Описание	Пример
/incident list	Список активных инцидентов	Список всех открытых алертов
/incident	Подтверждение инцидента	Принятие

acknowledge {id}		ответственности
/incident status {id}	Статус инцидента	Текущее состояние
/restart {service}	Перезапуск сервиса	kubectl rollout restart
/rollback {service}	Откат версии	kubectl rollout undo
/scale {service} {replicas}	Масштабирование	Увеличение реплик
/logs {service} {lines}	Просмотр логов	Последние строки лога
/metrics {service}	Метрики сервиса	График ошибок/задержки
/escalate {level}	Эскалация инцидента	Вызов вышестоящего уровня

2.5. Уровни эскалации

Уровень	Канал	Действие	Таймаут
Уровень 0	Telegram/Slack	Уведомление on-call инженера	0 мин
Уровень 1	+ Telegram группа SRE	Добавление всей SRE команды	5 мин
Уровень 2	+ Slack канал incident	Добавление разработчиков	15 мин
Уровень 3	+ SMS/Email	Уведомление руководителя	30 мин

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Telegram-бот для управления инцидентами

```
"""
Лабораторная работа №13
Telegram-бот для управления инцидентами
"""

import os
import json
import logging
import subprocess
from datetime import datetime
from typing import Dict, Any, List, Optional
from dataclasses import dataclass, field

from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import (
    Application, CommandHandler, CallbackQueryHandler,
    MessageHandler, filters, ContextTypes
)

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Конфигурация
BOT_TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN", "YOUR_BOT_TOKEN")
CHAT_ID = os.environ.get("TELEGRAM_CHAT_ID", "YOUR_CHAT_ID")

# Хранилище инцидентов
incidents = {}
next_incident_id = 1

@dataclass
class Incident:
    """Модель инцидента"""
    id: int
    alert_name: str
    severity: str
    description: str
    status: str # open, acknowledged, resolved
    assignee: Optional[str] = None
    created_at: str = field(default_factory=lambda: datetime.now().isoformat())
    resolved_at: Optional[str] = None
    actions: List[Dict] = field(default_factory=list)
    escalation_level: int = 0

class IncidentManager:
    """Управление инцидентами"""
```

```

@staticmethod
def create_incident(alert_name: str, severity: str, description: str) -> int:
    """Создание нового инцидента"""
    global next_incident_id
    incident = Incident(
        id=next_incident_id,
        alert_name=alert_name,
        severity=severity,
        description=description,
        status="open"
    )
    incidents[next_incident_id] = incident
    next_incident_id += 1
    return incident.id

@staticmethod
def acknowledge(incident_id: int, user: str) -> bool:
    """Подтверждение инцидента"""
    if incident_id in incidents:
        incidents[incident_id].status = "acknowledged"
        incidents[incident_id].assignee = user
        incidents[incident_id].actions.append({
            "action": "acknowledge",
            "user": user,
            "timestamp": datetime.now().isoformat()
        })
        return True
    return False

@staticmethod
def resolve(incident_id: int) -> bool:
    """Закрытие инцидента"""
    if incident_id in incidents:
        incidents[incident_id].status = "resolved"
        incidents[incident_id].resolved_at = datetime.now().isoformat()
        incidents[incident_id].actions.append({
            "action": "resolve",
            "timestamp": datetime.now().isoformat()
        })
        return True
    return False

@staticmethod
def restart_service(service: str) -> Dict[str, Any]:
    """Перезапуск сервиса (имитация)"""
    logger.info(f"Restarting service: {service}")
    # В реальной среде: kubectl rollout restart
    return {"success": True, "message": f"Service {service} restarted"}

```

```

@staticmethod
def rollback_service(service: str) -> Dict[str, Any]:
    """Откат сервиса (имитация)"""
    logger.info(f"Rolling back service: {service}")
    return {"success": True, "message": f"Service {service} rolled back"}

# Команды бота
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Приветственное сообщение"""
    await update.message.reply_text(
        "👋 *ChatOps Bot для управления инцидентами*\n\n"
        "Доступные команды:\n"
        "/list - список активных инцидентов\n"
        "/incident {id} - информация об инциденте\n"
        "/ack {id} - подтвердить инцидент\n"
        "/resolve {id} - закрыть инцидент\n"
        "/restart {service} - перезапустить сервис\n"
        "/rollback {service} - откатить сервис\n"
        "/help - это сообщение",
        parse_mode="Markdown"
    )

async def list_incidents(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Список активных инцидентов"""
    open_incidents = [i for i in incidents.values() if i.status != "resolved"]

    if not open_incidents:
        await update.message.reply_text("✅ Нет активных инцидентов")
        return

    message = "📋 *Активные инциденты:*\n\n"
    for inc in open_incidents:
        status_emoji = "🟡" if inc.status == "open" else "🟢"
        message += f"{status_emoji} *{inc.id}: {inc.alert_name}*\n"
        message += f"    Статус: {inc.status}\n"

    await update.message.reply_text(message, parse_mode="Markdown")

async def incident_detail(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Детали инцидента"""
    try:
        inc_id = int(context.args[0])
    except (IndexError, ValueError):
        await update.message.reply_text("Использование: /incident {id}")
        return

    if inc_id not in incidents:
        await update.message.reply_text(f"Инцидент #{inc_id} не найден")
        return

```



```

inc = incidents[inc_id]

severity_emoji = "🔴" if inc.severity == "critical" else "🟡"

message = f"{severity_emoji} *Инцидент #{inc.id}*\n\n"
message += f"**Алерт:** {inc.alert_name}\n"
message += f"**Severity:** {inc.severity}\n"
message += f"**Статус:** {inc.status}\n"
message += f"**Создан:** {inc.created_at}\n"
if inc.assignee:
    message += f"**Назначен:** @{inc.assignee}\n"
if inc.resolved_at:
    message += f"**Закрит:** {inc.resolved_at}\n"

# Кнопки действий
keyboard = []
if inc.status == "open":
    keyboard.append([InlineKeyboardButton("✅ Acknowledge",
callback_data=f"ack_{inc.id}"))
    if inc.status in ["open", "acknowledged"]:
        keyboard.append([
            InlineKeyboardButton("🔄 Restart", callback_data=f"restart_api"),
            InlineKeyboardButton("⏮ Rollback", callback_data=f"rollback_api")
        ])
    keyboard.append([InlineKeyboardButton("✅ Resolve",
callback_data=f"resolve_{inc.id}"))

reply_markup = InlineKeyboardMarkup(keyboard)
await update.message.reply_text(message, parse_mode="Markdown",
reply_markup=reply_markup)

async def acknowledge(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Подтверждение инцидента"""
    try:
        inc_id = int(context.args[0])
    except (IndexError, ValueError):
        await update.message.reply_text("Использование: /ack {id}")
        return

    user = update.effective_user.username or update.effective_user.first_name

    if IncidentManager.acknowledge(inc_id, user):
        await update.message.reply_text(f"✅ Инцидент #{inc_id} подтверждён @{user}")
    else:
        await update.message.reply_text(f"❌ Инцидент #{inc_id} не найден")

async def resolve_incident(update: Update, context: ContextTypes.DEFAULT_TYPE):

```

```

"""Заккрытие инцидента"""
try:
    inc_id = int(context.args[0])
except (IndexError, ValueError):
    await update.message.reply_text("Использование: /resolve {id}")
    return

if IncidentManager.resolve(inc_id):
    await update.message.reply_text(f"✅ Инцидент #{inc_id} закрыт")

    # Расчет MTTR
    inc = incidents[inc_id]
    created = datetime.fromisoformat(inc.created_at)
    resolved = datetime.fromisoformat(inc.resolved_at)
    mttr_minutes = (resolved - created).total_seconds() / 60
    await update.message.reply_text(f"📊 MTTR: {mttr_minutes:.1f} минут")
else:
    await update.message.reply_text(f"❌ Инцидент #{inc_id} не найден")

async def restart_service(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Перезапуск сервиса"""
    try:
        service = context.args[0]
    except IndexError:
        await update.message.reply_text("Использование: /restart {service}")
        return

    result = IncidentManager.restart_service(service)
    if result["success"]:
        await update.message.reply_text(f"🔄 Сервис {service} перезапущен")
    else:
        await update.message.reply_text(f"❌ Ошибка перезапуска {service}")

async def rollback_service(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Откат сервиса"""
    try:
        service = context.args[0]
    except IndexError:
        await update.message.reply_text("Использование: /rollback {service}")
        return

    result = IncidentManager.rollback_service(service)
    if result["success"]:
        await update.message.reply_text(f"⏮ Сервис {service} откачен к предыдущей версии")
    else:
        await update.message.reply_text(f"❌ Ошибка отката {service}")

async def button_callback(update: Update, context: ContextTypes.DEFAULT_TYPE):

```

```

"""Обработка кнопок"""
query = update.callback_query
await query.answer()

data = query.data

if data.startswith("ack_"):
    inc_id = int(data.split("_")[1])
    user = update.effective_user.username or "user"
    IncidentManager.acknowledge(inc_id, user)
    await query.edit_message_text(f"✅ Инцидент #{inc_id} подтверждён @#{user}")

elif data.startswith("restart_"):
    service = data.split("_")[1]
    result = IncidentManager.restart_service(service)
    await query.edit_message_text(f"🔄 Сервис {service} перезапущен")

elif data.startswith("rollback_"):
    service = data.split("_")[1]
    result = IncidentManager.rollback_service(service)
    await query.edit_message_text(f"⏪ Сервис {service} откачен")

elif data.startswith("resolve_"):
    inc_id = int(data.split("_")[1])
    IncidentManager.resolve(inc_id)
    await query.edit_message_text(f"✅ Инцидент #{inc_id} закрыт")

def main():
    """Запуск бота"""
    app = Application.builder().token(BOT_TOKEN).build()

    # Команды
    app.add_handler(CommandHandler("start", start))
    app.add_handler(CommandHandler("help", start))
    app.add_handler(CommandHandler("list", list_incidents))
    app.add_handler(CommandHandler("incident", incident_detail))
    app.add_handler(CommandHandler("ack", acknowledge))
    app.add_handler(CommandHandler("resolve", resolve_incident))
    app.add_handler(CommandHandler("restart", restart_service))
    app.add_handler(CommandHandler("rollback", rollback_service))

    # Callback для кнопок
    app.add_handler(CallbackQueryHandler(button_callback))

    # Обработка алертов (webhook)
    # app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
    handle_alert))

```

```

        logger.info("Бот запущен")
        app.run_polling()

if __name__ == "__main__":
    main()

```

3.2. Webhook для отправки алертов в Telegram

```

"""
Лабораторная работа №13
Webhook для отправки алертов из Alertmanager в Telegram
"""

import json
import os
import requests
import logging
from datetime import datetime
from flask import Flask, request, jsonify

app = Flask(__name__)
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

TELEGRAM_BOT_TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN", "YOUR_BOT_TOKEN")
TELEGRAM_CHAT_ID = os.environ.get("TELEGRAM_CHAT_ID", "YOUR_CHAT_ID")

def send_telegram_message(text: str, reply_markup: dict = None):
    """Отправка сообщения в Telegram"""
    url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
    payload = {
        "chat_id": TELEGRAM_CHAT_ID,
        "text": text,
        "parse_mode": "Markdown",
        "disable_web_page_preview": True
    }
    if reply_markup:
        payload["reply_markup"] = json.dumps(reply_markup)

    response = requests.post(url, json=payload)
    return response.json()

def create_incident_keyboard(incident_id: int):
    """Создание клавиатуры для инцидента"""
    return {
        "inline_keyboard": [
            [

```

```

        {"text": "✅ Acknowledge", "callback_data":
f"ack_{incident_id}"},
        {"text": "🔄 Restart API", "callback_data": "restart_api"}
    ],
    [
        {"text": "⏮ Rollback", "callback_data": "rollback_api"},
        {"text": "❌ Escalate", "callback_data": "escalate"}
    ]
]
}

@app.route('/webhook/alert', methods=['POST'])
def handle_alert():
    """Обработка алерта от Alertmanager"""
    data = request.json

    for alert in data.get("alerts", []):
        alert_name = alert.get("labels", {}).get("alertname", "Unknown")
        severity = alert.get("labels", {}).get("severity", "unknown")
        description = alert.get("annotations", {}).get("description", "No
description")

        severity_emoji = "🔴" if severity == "critical" else "🟡" if severity ==
"warning" else "🟢"

        message = f"""{severity_emoji} *Алерт: {alert_name}*
**Severity:** {severity}
**Описание:** {description}
**Время:** {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

Действия:
• /ack {alert_name} - подтвердить
• /restart - перезапустить сервис
• /rollback - откатить версию
• /logs - посмотреть логи
• /metrics - метрики сервиса
"""

        # Создание инцидента
        import random
        incident_id = random.randint(1000, 9999)

        keyboard = create_incident_keyboard(incident_id)

        send_telegram_message(message, keyboard)
        logger.info(f"Alert sent to Telegram: {alert_name}")

    return jsonify({"status": "ok"}), 200

```

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

3.3. Конфигурация Alertmanager для Telegram

```
# alertmanager.yml
global:
  slack_api_url: 'https://hooks.slack.com/services/...'

route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 1h
  receiver: 'default'

routes:
  - match:
      severity: critical
    receiver: telegram
    continue: true
  - match:
      severity: warning
    receiver: slack

receivers:
  - name: 'telegram'
    webhook_configs:
      - url: 'http://telegram-bot:5000/webhook/alert'
        send_resolved: true

  - name: 'slack'
    slack_configs:
      - channel: '#warnings'
        title: '⚠ {{ .GroupLabels.alertname }}'
```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Мессенджер	Функции	Интеграция
1	Telegram	Уведомления + /status	Alertmanager

2	Telegram	Уведомления + /restart	Prometheus
3	Slack	Уведомления + /logs	kubectl
4	Slack	Уведомления + /metrics	Prometheus
5	Telegram	Базовая эскалация	Alertmanager

4.2. Уровень 2

№	Мессенджер	Функции	Дополнительные требования
1	Telegram	Acknowledge + Restart + Rollback	Интерактивные кнопки
2	Telegram	Эскалация (3 уровня)	Таймауты
3	Slack	Дашборд инцидентов	История инцидентов
4	Telegram	Логи + Метрики	Интеграция с Grafana
5	Специализация БПЛА	Мониторинг телеметрии	Уведомления о статусе

4.3. Уровень 3

№	Мессенджер	Функции	Дополнительные требования
1	Telegram	Полный ChatOps	Python-бот + webhook
2	Telegram	Multi-канальная эскалация	Jira интеграция
3	Slack	Self-healing команды	Автоматический откат
4	Специализация БПЛА (продвинутый)	Команды управления дроном	Симуляция OTA
5	Комплексный	Полный цикл инцидент-	Отчёт MTTR

		менеджмента	

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое ChatOps и какие задачи он решает?
2. Какие преимущества даёт ChatOps для инцидент-менеджмента?
3. Как интегрировать бота с Alertmanager?
4. Какие команды ChatOps-бота наиболее полезны при инцидентах?
5. Как организовать эскалацию в ChatOps?
6. Какие метрики нужно отслеживать для оценки эффективности ChatOps?
7. Как обеспечить безопасность при выполнении команд через бота?
8. Как логировать действия пользователей в чате?
9. Как интегрировать бота с системами трекинга (Jira/YouTrack)?
10. Как измерить сокращение MTTR после внедрения ChatOps?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (ChatOps, эскалация, интеграция).
4. Практическая часть:
 - Конфигурация бота.
 - Интеграция с Alertmanager.
 - Скриншоты сообщений и команд.
 - Место для скриншота: запущенный бот.
 - Место для скриншота: интерактивные кнопки.
 - Место для скриншота: выполнение команд.

5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.
6. Анализ результатов (время реакции, MTTR).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Бот создан и запущен	2 балла	2 балла	2 балла
Уведомления об алертах	2 балла	2 балла	2 балла
Уведомления об алертах	2 балла	2 балла	2 балла
Команды /status, /restart	1 балл	2 балла	2 балла
Интерактивные кнопки	0 баллов	2 балла	2 балла
Эскалация	0 баллов	1 балл	2 балла
Интеграция с Jira/YouTrack	0 баллов	0 баллов	2 балла
Автоматический откат	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 7)	Уровень 2 (макс. 11)	Уровень 3 (макс. 16)
5 (отлично)	6-7	10-11	14-16
4 (хорошо)	5	8-9	11-13
3 (удовлетворительно)	3-4	6-7	8-10
2	<3	<6	<8

(неудовлетворительно)			

ЛАБОРАТОРНАЯ РАБОТА №14

Self-healing механизмы и graceful degradation

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки реализации механизмов автоматического восстановления (self-healing), настройки health checks, readiness/liveness probes, graceful degradation для критических сервисов, а также анализа эффективности самовосстановления системы при различных типах сбоев.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Self-healing: концепция и уровни

Self-healing (самовосстановление) - способность системы автоматически обнаруживать сбои и восстанавливать нормальное функционирование без участия человека.

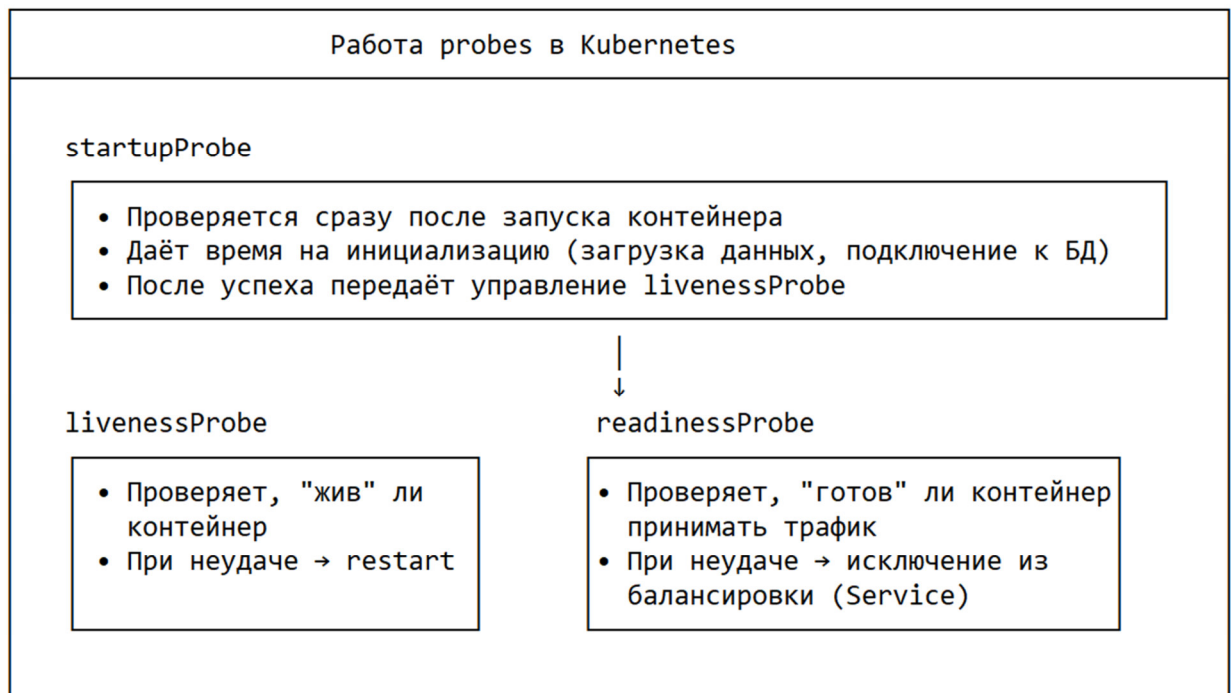
Уровень		Описание	Пример
Уровень Автоматический перезапуск	1:	Система перезапускает упавший компонент	Kubernetes restart policy
Уровень Автоматическое восстановление	2:	Система восстанавливает состояние после сбоя	StatefulSet с persistent volume
Уровень Автоматическое масштабирование	3:	Система добавляет ресурсы при нагрузке	HPA (Horizontal Pod Autoscaler)
Уровень	4:	Система переключается	Leader election, failover

Автоматическое переключение	на резервный компонент	
Уровень 5: Автономное восстановление	Система восстанавливается без внешних зависимостей	Self-healing mesh

2.2. Health checks в Kubernetes

Тип Probe	Назначение	Что происходит при неудаче
livenessProbe	Проверка, жив ли контейнер (не заблокирован, не в deadlock)	Контейнер перезапускается
readinessProbe	Проверка, готов ли контейнер принимать трафик	Контейнер исключается из Service
startupProbe	Проверка, успешно ли запустился контейнер	Защита от преждевременных liveness-проверок

Схема работы probes:



2.3. Пример конфигурации probes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: app
        image: myapp:v1
        ports:
        - containerPort: 8080

      # Startup probe – для медленно стартующих приложений
      startupProbe:
        httpGet:
          path: /health
          port: 8080
        initialDelaySeconds: 0
        periodSeconds: 5
        failureThreshold: 30 # 30 × 5 = 150 секунд на старт

      # Liveness probe – проверка "жив" ли контейнер
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
```

```

    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 3
    failureThreshold: 3    # 3 неудачи → restart

# Readiness probe – проверка готовности принимать трафик
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 5
  timeoutSeconds: 2
  failureThreshold: 2    # 2 неудачи → исключить из балансировки
  successThreshold: 1

```

2.4. Graceful degradation (отказоустойчивое снижение функциональности)

Graceful degradation - способность системы при отказе отдельных компонентов продолжать работу с ухудшенной функциональностью (без полного отказа).

Стратегии graceful degradation:

Стратегия	Описание	Пример
Отключение некритичных функций	При перегрузке отключаются неосновные возможности	Отключение рекомендаций при высоком RPS
Использование кэшированных данных	При отказе источника данных используются закэшированные версии	Offline mode
Упрощённая обработка	Более быстрый, но менее точный алгоритм	Fallback поиска
Предсказуемое	Возврат к стабильному	Circuit breaker open →

поведение	состоянию	fallback response
Rate limiting	Ограничение нагрузки для сохранения доступности	Throttling запросов

2.5. Автоматическое масштабирование (HPA)

```

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15

```

2.6. Пример graceful degradation в Python

```
"""
Лабораторная работа №14
Graceful degradation пример
"""

import time
import random
from functools import wraps
from typing import Any, Callable, Dict, Optional

class CircuitBreaker:
    """Circuit Breaker с graceful degradation"""

    def __init__(self, failure_threshold: int = 5, timeout_seconds: int = 60):
        self.failure_threshold = failure_threshold
        self.timeout_seconds = timeout_seconds
        self.failure_count = 0
        self.last_failure_time = None
        self.state = "closed" # closed, open, half-open

    def call(self, func: Callable, fallback: Optional[Callable] = None, *args,
**kwargs) -> Any:
        """Вызов функции с защитой Circuit Breaker"""

        if self.state == "open":
            if self._should_attempt_reset():
                self.state = "half-open"
            else:
                if fallback:
                    return fallback(*args, **kwargs)
                raise Exception("Circuit breaker is OPEN")

        try:
            result = func(*args, **kwargs)
            self._on_success()
            return result
        except Exception as e:
            self._on_failure()
            if fallback:
                return fallback(*args, **kwargs)
            raise e

    def _on_success(self):
        if self.state == "half-open":
```

```

        self.state = "closed"
        self.failure_count = 0
    elif self.state == "closed":
        self.failure_count = max(0, self.failure_count - 1)

    def _on_failure(self):
        self.failure_count += 1
        self.last_failure_time = time.time()
        if self.failure_count >= self.failure_threshold:
            self.state = "open"

    def _should_attempt_reset(self) -> bool:
        if self.last_failure_time is None:
            return True
        return (time.time() - self.last_failure_time) >= self.timeout_seconds

class GracefulDegradationService:
    """Сервис с graceful degradation"""

    def __init__(self):
        self.circuit_breaker = CircuitBreaker(failure_threshold=3,
        timeout_seconds=30)
        self.cache = {}
        self.recommendations_enabled = True

    def get_recommendations(self, user_id: str) -> list:
        """Получение рекомендаций с fallback"""

        def primary_func():
            # Основной запрос к ML-сервису
            if random.random() < 0.3: # 30% ошибок
                raise Exception("Recommendation service timeout")
            return ["item1", "item2", "item3"]

        def fallback_func():
            # Fallback: популярные товары или кэш
            if user_id in self.cache:
                return self.cache[user_id]
            return ["popular_item1", "popular_item2"]

        result = self.circuit_breaker.call(primary_func, fallback_func, user_id)
        self.cache[user_id] = result
        return result

    def process_order(self, order_data: dict) -> dict:
        """Обработка заказа с graceful degradation"""

        # Критичная функция – не имеет fallback
        if not self.recommendations_enabled:

```



```

        # Упрощённая обработка при отказе
        return {
            "status": "processed_simplified",
            "order_id": order_data.get("id"),
            "recommendations": []
        }

    try:
        # Полноценная обработка
        return {
            "status": "processed",
            "order_id": order_data.get("id"),
            "recommendations":
self.get_recommendations(order_data.get("user_id"))
        }
    except Exception as e:
        self.recommendations_enabled = False
        return {
            "status": "processed_degraded",
            "order_id": order_data.get("id"),
            "recommendations": [],
            "error": str(e)
        }

# Пример использования
if __name__ == "__main__":
    service = GracefulDegradationService()

    for i in range(20):
        try:
            result = service.process_order({"id": i, "user_id": f"user_{i%3}"})
            print(f"Request {i}: {result['status']}")
        except Exception as e:
            print(f"Request {i}: FAILED - {e}")
        time.sleep(0.5)

```

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Kubernetes манифест с probes и self-healing

```

# app-with-self-healing.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: resilient-app
  labels:
    app: resilient-app

```

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: resilient-app
  template:
    metadata:
      labels:
        app: resilient-app
    spec:
      containers:
        - name: app
          image: myapp:latest
          ports:
            - containerPort: 8080

# Ресурсы для autoscaling
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 500m
    memory: 512Mi

# Startup probe для медленного старта
startupProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 0
  periodSeconds: 5
  failureThreshold: 30

# Liveness probe
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 30
  periodSeconds: 10
  timeoutSeconds: 3
  failureThreshold: 3

# Readiness probe
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
```

```

        initialDelaySeconds: 10
        periodSeconds: 5
        timeoutSeconds: 2
        failureThreshold: 2
        successThreshold: 1

    # Graceful shutdown
    lifecycle:
      preStop:
        exec:
          command: ["/bin/sh", "-c", "sleep 10"]
---
# HorizontalPodAutoscaler
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: resilient-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: resilient-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
---
# PodDisruptionBudget для защиты от массовых отказов
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: resilient-app-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: resilient-app

```

3.2. Приложение с health endpoints

"""

Лабораторная работа №14

Приложение с health endpoints для Kubernetes

```
"""

from flask import Flask, jsonify, request
import time
import random
import threading

app = Flask(__name__)

# Состояние приложения
is_healthy = True
is_ready = True
degraded_mode = False
startup_time = time.time()

@app.route('/health', methods=['GET'])
def health():
    """Liveness probe endpoint"""
    current_time = time.time()

    # Симуляция deadlock (для тестирования liveness)
    if request.args.get('deadlock') == 'true':
        while True:
            time.sleep(1)

    # Симуляция зависания (для тестирования liveness)
    if request.args.get('hang') == 'true':
        time.sleep(60)

    if not is_healthy:
        return jsonify({"status": "unhealthy"}), 503

    return jsonify({
        "status": "healthy",
        "uptime": current_time - startup_time,
        "degraded_mode": degraded_mode
    }), 200

@app.route('/ready', methods=['GET'])
def ready():
    """Readiness probe endpoint"""
    if not is_ready:
        return jsonify({"status": "not ready"}), 503

    # Проверка зависимостей (например, БД)
    # Здесь можно добавить проверку подключения к БД
    dependencies_ok = True
```

```

    if not dependencies_ok:
        return jsonify({"status": "not ready", "reason": "database
unavailable"}), 503

    return jsonify({"status": "ready"}), 200

@app.route('/api/data', methods=['GET'])
def get_data():
    """Основной эндпоинт с graceful degradation"""
    global degraded_mode

    # Симуляция таймаута
    if request.args.get('timeout') == 'true':
        time.sleep(5)
        return jsonify({"error": "timeout"}), 504

    # Graceful degradation при перегрузке
    if degraded_mode:
        return jsonify({
            "data": "degraded_response",
            "message": "Service is operating in degraded mode"
        }), 200

    # Нормальный ответ
    return jsonify({
        "data": "full_response",
        "timestamp": time.time()
    }), 200

@app.route('/admin/degraded', methods=['POST'])
def set_degraded():
    """Включение/выключение degraded mode (для тестирования)"""
    global degraded_mode
    data = request.json
    degraded_mode = data.get('enabled', False)
    return jsonify({"degraded_mode": degraded_mode}), 200

@app.route('/admin/health', methods=['POST'])
def set_health():
    """Управление health статусом (для тестирования)"""
    global is_healthy
    data = request.json
    is_healthy = data.get('healthy', True)
    return jsonify({"healthy": is_healthy}), 200

@app.route('/admin/ready', methods=['POST'])
def set_ready():
    """Управление ready статусом (для тестирования)"""
    global is_ready

```

```

    data = request.json
    is_ready = data.get('ready', True)
    return jsonify({"ready": is_ready}), 200

@app.route('/metrics', methods=['GET'])
def metrics():
    """Метрики для Prometheus"""
    return f"""
# HELP app_health_status Health status (1=healthy, 0=unhealthy)
# TYPE app_health_status gauge
app_health_status {1 if is_healthy else 0}
# HELP app_ready_status Ready status (1=ready, 0=not ready)
# TYPE app_ready_status gauge
app_ready_status {1 if is_ready else 0}
# HELP app_degraded_mode Degraded mode (1=on, 0=off)
# TYPE app_degraded_mode gauge
app_degraded_mode {1 if degraded_mode else 0}
"""

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)

```

3.3. Скрипт для тестирования self-healing

```

"""
Лабораторная работа №14
Тестирование self-healing механизмов
"""

import requests
import time
import subprocess
from typing import Dict, Any

class SelfHealingTester:
    """Тестирование self-healing механизмов"""

    def __init__(self, namespace: str = "default", app_label: str = "resilient-
app"):
        self.namespace = namespace
        self.app_label = app_label

    def get_pod_status(self) -> Dict[str, Any]:
        """Получение статуса подов"""
        cmd = f"kubectl get pods -n {self.namespace} -l app={self.app_label} -o
json"
        result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
        import json

```

```

data = json.loads(result.stdout)

running = 0
pending = 0
failed = 0

for item in data.get("items", []):
    status = item["status"]["phase"]
    if status == "Running":
        running += 1
    elif status == "Pending":
        pending += 1
    else:
        failed += 1

    return {"running": running, "pending": pending, "failed": failed,
"total": len(data.get("items", []))}

def delete_pod(self, pod_name: str) -> bool:
    """Удаление пода для тестирования восстановления"""
    cmd = f"kubectl delete pod {pod_name} -n {self.namespace}"
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return result.returncode == 0

def get_first_pod(self) -> str:
    """Получение имени первого пода"""
    cmd = f"kubectl get pods -n {self.namespace} -l app={self.app_label} -o
jsonpath='{.items[0].metadata.name}'"
    result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
    return result.stdout.strip()

def test_liveness_recovery(self) -> Dict[str, Any]:
    """Тестирование восстановления после liveness failure"""
    print("\n Testing liveness recovery...")

    pod_name = self.get_first_pod()
    if not pod_name:
        return {"success": False, "error": "No pod found"}

    # Получение количества перезапусков до теста
    cmd = f"kubectl get pod {pod_name} -n {self.namespace} -o
jsonpath='{.status.containerStatuses[0].restartCount}'"
    before = int(subprocess.run(cmd, shell=True, capture_output=True,
text=True).stdout.strip() or "0")

    # Имитация deadlock через http endpoint
    cmd = f"kubectl exec {pod_name} -n {self.namespace} -- curl -s
'http://localhost:8080/health?deadlock=true' &"
    subprocess.run(cmd, shell=True)

```

```

# Ожидание перезапуска
time.sleep(35)

cmd = f"kubectl get pod {pod_name} -n {self.namespace} -o
jsonpath='{.status.containerStatuses[0].restartCount}{' '}"
after = int(subprocess.run(cmd, shell=True, capture_output=True,
text=True).stdout.strip() or "0")

recovered = after > before

print(f"    Restart count: {before} -> {after}")
print(f"    Recovered: {' YES' if recovered else ' NO'}")

return {"success": recovered, "restart_count_before": before,
"restart_count_after": after}

def test_pod_recovery(self) -> Dict[str, Any]:
    """Тестирование восстановления после удаления пода"""
    print("\nTesting pod recovery...")

    pod_name = self.get_first_pod()
    if not pod_name:
        return {"success": False, "error": "No pod found"}

    status_before = self.get_pod_status()

    start_time = time.time()
    self.delete_pod(pod_name)

    # Ожидание восстановления
    recovered = False
    for i in range(30):
        time.sleep(2)
        status_after = self.get_pod_status()
        if status_after["running"] == status_before["running"]:
            recovered = True
            recovery_time = time.time() - start_time
            break

    print(f"    Pods before: {status_before}")
    print(f"    Recovery time: {recovery_time:.1f} seconds")
    print(f"    Recovered: {' YES' if recovered else ' NO'}")

    return {"success": recovered, "recovery_time_seconds": recovery_time}

def test_autoscaling(self) -> Dict[str, Any]:
    """Тестирование автоматического масштабирования"""
    print("\nTesting autoscaling...")

```



```

# Генерация нагрузки (упрощённо)
# В реальной среде используется load generator

status_before = self.get_pod_status()

# Здесь можно добавить генератор нагрузки
# Например, через kubectl run load-generator

time.sleep(60) # Ожидание реакции HPA

status_after = self.get_pod_status()

scaled = status_after["running"] > status_before["running"]

print(f"    Pods before: {status_before['running']}")
print(f"    Pods after: {status_after['running']}")
print(f"    Scaled: {' YES' if scaled else ' NO'}")

    return {"success": scaled, "pods_before": status_before["running"],
"pods_after": status_after["running"]}

def run_self_healing_tests():
    """Запуск всех тестов self-healing"""
    print("=" * 60)
    print("SELF-HEALING TESTS")
    print("=" * 60)

    tester = SelfHealingTester()

    results = {
        "liveness_recovery": tester.test_liveness_recovery(),
        "pod_recovery": tester.test_pod_recovery(),
        "autoscaling": tester.test_autoscaling()
    }

    print("\n" + "=" * 60)
    print("TEST RESULTS SUMMARY")
    print("=" * 60)

    all_success = all(r.get("success", False) for r in results.values())

    for name, result in results.items():
        status = " PASS" if result.get("success") else " FAIL"
        print(f"{{status}}: {{name}}")

    print(f"\nOverall: {' ALL PASSED' if all_success else ' SOME TESTS FAILED'}")

    return results

```

```
if __name__ == "__main__":
    run_self_healing_tests()
```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Механизм	Тестирование	Инструмент
1	Liveness probe	Перезапуск при deadlock	Kubernetes
2	Readiness probe	Исключение из балансировки	Kubernetes
3	Startup probe	Медленный старт (>30с)	Kubernetes
4	PodDisruptionBudget	Защита при drain	Kubernetes
5	Resource limits	OOM recovery	Kubernetes

4.2. Уровень 2

№	Механизм	Тестирование	Дополнительные требования
1	HPA autoscaling	Нагрузка → масштабирование	Prometheus метрики
2	Circuit Breaker	Fallback при ошибках	Python реализация
3	Graceful degradation	Отключение некритичных функций	Health checks
4	Leader election	Автоматическое переключение	Kubernetes leader
5	Специализация БПЛА	Автоматический RTL при сбое	Симуляция полёта

4.3. Уровень 3

№	Механизм	Тестирование	Дополнительные требования
1	Full self-healing suite	Все механизмы	Python скрипт тестирования
2	Multi-level degradation	Каскадное ухудшение	Prometheus + Grafana
3	Auto-remediation	Автоматический откат	Argo Rollouts
4	Специализация БПЛА (продвинутый)	Self-healing для роя дронов	Симуляция отказов
5	Комплексный	Полный цикл self-healing	Отчёт MTTR/MTBF

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое self-healing и какие уровни самовосстановления существуют?
2. Чем отличается liveness probe от readiness probe?
3. Зачем нужен startup probe?
4. Какие стратегии graceful degradation вы знаете?
5. Как работает Horizontal Pod Autoscaler в Kubernetes?
6. Что такое PodDisruptionBudget и зачем он нужен?
7. Как реализовать graceful shutdown в приложении?
8. Какие метрики нужно отслеживать для self-healing?
9. Как протестировать self-healing механизмы?
10. Как оценить эффективность self-healing (MTTR, recovery time)?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (self-healing, probes, graceful degradation).
4. Практическая часть:
 - Kubernetes манифесты (Deployment, HPA, PDB).
 - Приложение с health endpoints.
 - Результаты тестирования.
 - Место для скриншота: статус подов.
 - Место для скриншота: логи probes.
 - Место для скриншота: HPA метрики.
5. Программная часть (Уровень 3):
 - Листинг кода Python.
 - Место для скриншота: вывод программы.
6. Анализ результатов (время восстановления, эффективность).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Probes настроены	2 балла	2 балла	2 балла
HPA настроен	1 балл	2 балла	2 балла
PodDisruptionBudget настроен	1 балл	2 балла	2 балла
Graceful degradation реализован	0 баллов	2 балла	2 балла

Тестирование self-healing	0 баллов	1 балл	2 балла
Python скрипт тестирования	0 баллов	0 баллов	2 балла
Анализ MTTR/восстановления	0 баллов	1 балл	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 6)	Уровень 2 (макс. 12)	Уровень 3 (макс. 16)
5 (отлично)	5-6	11-12	14-16
4 (хорошо)	4	9-10	11-13
3 (удовлетворительно)	3	6-8	8-10
2 (неудовлетворительно)	<3	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №15. Реализация архитектурных паттернов отказоустойчивости

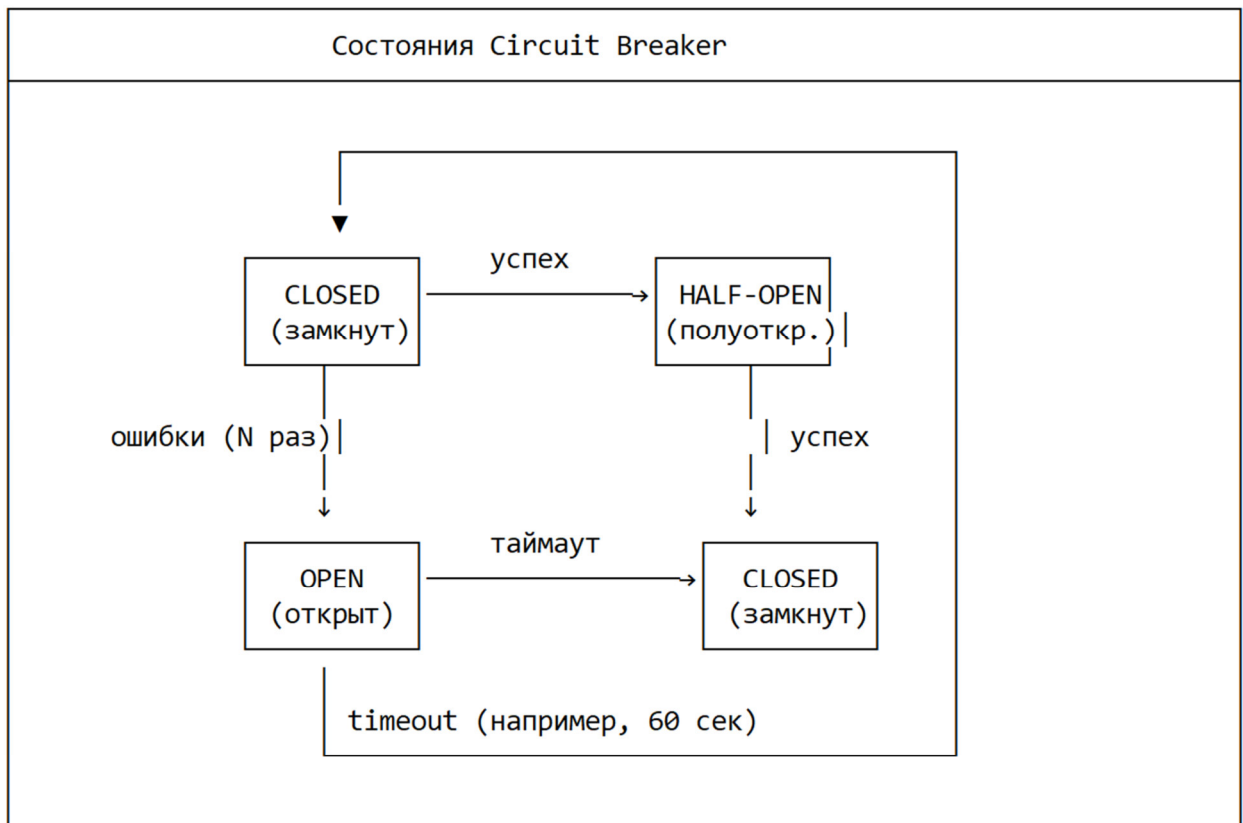
1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки реализации архитектурных паттернов отказоустойчивости: Circuit Breaker (предотвращение каскадных отказов), Retry с exponential backoff (повторные попытки при временных сбоях), Timeout (ограничение времени ожидания) и Bulkhead изоляция (разделение ресурсов), а также анализа их влияния на надежность и устойчивость распределенных систем.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Паттерны отказоустойчивости

Паттерн	Описание	Когда использовать	Риски без паттерна
Circuit Breaker	Предотвращает повторные вызовы к неисправному сервису, переходя в состояния: Closed → Open → Half-Open	При вызовах внешних API, БД, сетевых сервисов	Каскадные отказы, исчерпание ресурсов, лавинная остановка
Retry	Повторяет неудачные операции с экспоненциальной задержкой	При временных сбоях (сеть, таймауты, race conditions)	Увеличение времени ответа, дублирование операций
Timeout	Ограничивает максимальное время ожидания ответа	При всех внешних вызовах	Вечные блокировки, накопление потоков
Bulkhead	Изолирует ресурсы (пулы потоков, соединения) для разных компонентов	При разнородных сервисах с разной критичностью	Один сервис может исчерпать все ресурсы
Rate Limiting	Ограничивает количество запросов в единицу времени	При защите от перегрузки, DDoS, злоупотреблений	Исчерпание ресурсов, снижение качества для всех клиентов

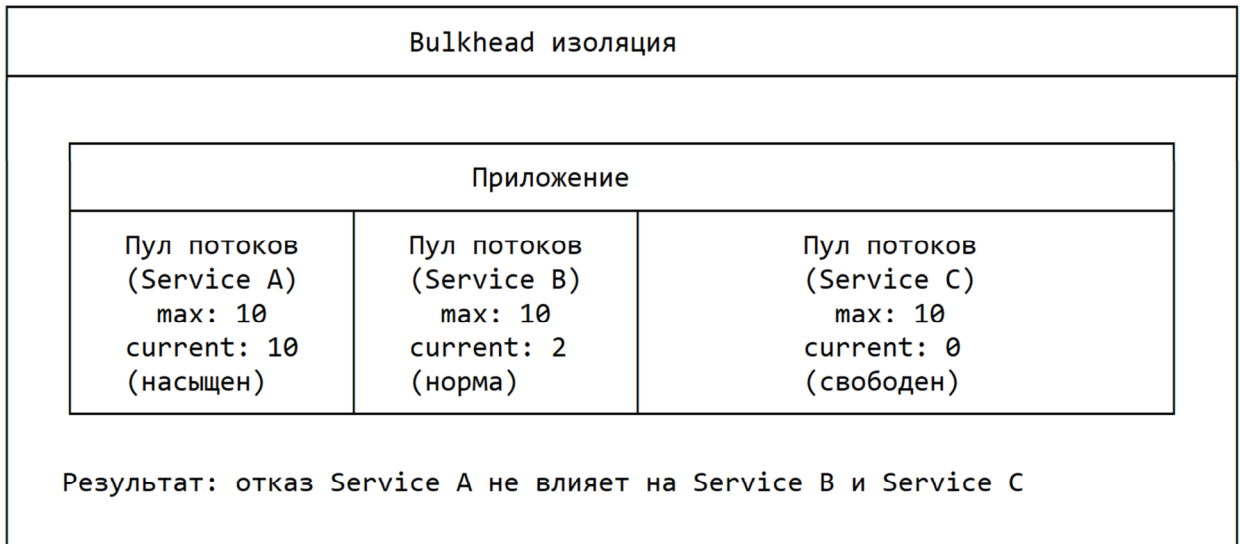
2.2. Состояния Circuit Breaker



2.3. Стратегии Retry

Стратегия	Формула	Пример	Преимущества
Fixed delay	$\text{delay} = \text{constant}$	1 сек, 1 сек, 1 сек	Простота
Exponential backoff	$\text{delay} = \text{base} \times 2^{(\text{attempt}-1)}$	1с, 2с, 4с, 8с	Снижает нагрузку
Exponential jitter	$\text{delay} = \text{base} \times 2^{(\text{attempt}-1)} + \text{random}(0, \text{base})$	0.5-1.5с, 1-3с, 2-6с	Предотвращает "thundering herd"
Linear	$\text{delay} = \text{base} \times \text{attempt}$	1с, 2с, 3с, 4с	Предсказуемый рост

2.4. Bulkhead изоляция



3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Реализация Circuit Breaker на Python

```

"""
Лабораторная работа №15
Реализация Circuit Breaker паттерна
"""

import time
import random
from datetime import datetime
from enum import Enum
from threading import Lock
from typing import Any, Callable, Optional
from dataclasses import dataclass

class CircuitBreakerState(Enum):
    CLOSED = "CLOSED"      # Замкнут: запросы проходят
    OPEN = "OPEN"           # Открыт: запросы блокируются
    HALF_OPEN = "HALF_OPEN" # Полуоткрыт: пробный запрос

@dataclass
class CircuitBreakerMetrics:
    """Метрики Circuit Breaker"""
    total_requests: int = 0
    success_requests: int = 0
    failure_requests: int = 0
    rejected_requests: int = 0
    state_transitions: int = 0

class CircuitBreaker:

```



```

"""
Реализация Circuit Breaker с тремя состояниями
"""

def __init__(self, name: str = "default",
              failure_threshold: int = 5,
              timeout_seconds: int = 30,
              half_open_max_attempts: int = 3):
    self.name = name
    self.failure_threshold = failure_threshold
    self.timeout_seconds = timeout_seconds
    self.half_open_max_attempts = half_open_max_attempts

    self.state = CircuitBreakerState.CLOSED
    self.failure_count = 0
    self.last_failure_time = None
    self.half_open_attempts = 0
    self._lock = Lock()
    self.metrics = CircuitBreakerMetrics()

    def call(self, func: Callable, fallback: Optional[Callable] = None, *args,
**kwargs) -> Any:
        """Вызов функции с защитой Circuit Breaker"""
        self.metrics.total_requests += 1

        with self._lock:
            # Проверка состояния
            if self.state == CircuitBreakerState.OPEN:
                if self._should_attempt_reset():
                    self._transition_to_half_open()
                else:
                    self.metrics.rejected_requests += 1
                    if fallback:
                        return fallback(*args, **kwargs)
                    raise Exception(f"Circuit Breaker '{self.name}' is OPEN. Try
again later.")

            if self.state == CircuitBreakerState.HALF_OPEN:
                if self.half_open_attempts >= self.half_open_max_attempts:
                    self._transition_to_open()
                    if fallback:
                        return fallback(*args, **kwargs)
                    raise Exception(f"Circuit Breaker '{self.name}' OPEN after
{self.half_open_attempts} half-open attempts")
                self.half_open_attempts += 1

            # Вызов функции
            try:
                result = func(*args, **kwargs)
                self._on_success()

```

```

        self.metrics.success_requests += 1
        return result
    except Exception as e:
        self._on_failure()
        self.metrics.failure_requests += 1
        if fallback:
            return fallback(*args, **kwargs)
        raise e

def _on_success(self):
    """Обработка успешного вызова"""
    with self._lock:
        if self.state == CircuitBreakerState.HALF_OPEN:
            self._transition_to_closed()
        elif self.state == CircuitBreakerState.CLOSED:
            self.failure_count = 0

def _on_failure(self):
    """Обработка неудачного вызова"""
    with self._lock:
        self.failure_count += 1
        self.last_failure_time = datetime.now()

        if self.state == CircuitBreakerState.CLOSED:
            if self.failure_count >= self.failure_threshold:
                self._transition_to_open()
        elif self.state == CircuitBreakerState.HALF_OPEN:
            self._transition_to_open()

def _should_attempt_reset(self) -> bool:
    """Проверка, можно ли перейти в HALF-OPEN"""
    if self.last_failure_time is None:
        return True
    elapsed = (datetime.now() - self.last_failure_time).total_seconds()
    return elapsed >= self.timeout_seconds

def _transition_to_closed(self):
    print(f"[{datetime.now().strftime('%H:%M:%S')}] CB '{self.name}': {self.state.value} → CLOSED")
    self.state = CircuitBreakerState.CLOSED
    self.failure_count = 0
    self.half_open_attempts = 0
    self.metrics.state_transitions += 1

def _transition_to_open(self):
    print(f"[{datetime.now().strftime('%H:%M:%S')}] CB '{self.name}': {self.state.value} → OPEN")
    self.state = CircuitBreakerState.OPEN
    self.failure_count = 0

```

```

        self.half_open_attempts = 0
        self.metrics.state_transitions += 1

    def _transition_to_half_open(self):
        print(f"[{datetime.now().strftime('%H:%M:%S')}] CB '{self.name}': {self.state.value} → HALF-OPEN")
        self.state = CircuitBreakerState.HALF_OPEN
        self.half_open_attempts = 0
        self.metrics.state_transitions += 1

    def get_state(self) -> str:
        return self.state.value

    def get_metrics(self) -> dict:
        return {
            "name": self.name,
            "state": self.state.value,
            "total_requests": self.metrics.total_requests,
            "success_requests": self.metrics.success_requests,
            "failure_requests": self.metrics.failure_requests,
            "rejected_requests": self.metrics.rejected_requests,
            "state_transitions": self.metrics.state_transitions,
            "success_rate": (self.metrics.success_requests /
self.metrics.total_requests * 100) if self.metrics.total_requests > 0 else 0
        }

# Пример нестабильного сервиса
class UnstableService:
    def __init__(self, name: str = "UnstableService", failure_rate: float = 0.3):
        self.name = name
        self.failure_rate = failure_rate
        self.call_count = 0

    def call(self) -> str:
        self.call_count += 1
        time.sleep(random.uniform(0.05, 0.2))

        if random.random() < self.failure_rate:
            raise Exception(f"Service '{self.name}' error (call #{self.call_count})")

        return f"Success (call #{self.call_count})"

# Демонстрация работы
def demo_circuit_breaker():
    print("=" * 60)
    print("DEMO: Circuit Breaker Pattern")
    print("=" * 60)

```

```

# Сервис с высокой частотой ошибок
service = UnstableService("PaymentService", failure_rate=0.6)
cb = CircuitBreaker("PaymentService", failure_threshold=3,
timeout_seconds=10)

# Fallback функция
def payment_fallback(*args, **kwargs):
    return "Fallback: Payment deferred (offline processing)"

print("\nИмитация работы платежного сервиса с Circuit Breaker:\n")

for i in range(25):
    try:
        result = cb.call(service.call, payment_fallback)
        print(f" Call {i+1}: {result}")
    except Exception as e:
        print(f" Call {i+1}: {e}")

    print(f"    CB State: {cb.get_state()}")
    time.sleep(0.5)

print("\n" + "=" * 60)
print("CIRCUIT BREAKER METRICS")
print("=" * 60)
metrics = cb.get_metrics()
for key, value in metrics.items():
    print(f"  {key}: {value}")

if __name__ == "__main__":
    demo_circuit_breaker()

```

3.2. Реализация Retry с exponential backoff

```

"""
Лабораторная работа №15
Реализация Retry с exponential backoff
"""

import time
import random
from functools import wraps
from typing import Callable, Tuple, Any

class RetryConfig:
    """Конфигурация Retry механизма"""
    def __init__(self, max_retries: int = 3,
                 base_delay: float = 1.0,
                 max_delay: float = 30.0,

```

```

        backoff_factor: float = 2.0,
        use_jitter: bool = True):
    self.max_retries = max_retries
    self.base_delay = base_delay
    self.max_delay = max_delay
    self.backoff_factor = backoff_factor
    self.use_jitter = use_jitter

class RetryExecutor:
    """Выполнитель с поддержкой retry"""

    def __init__(self, config: RetryConfig = None):
        self.config = config or RetryConfig()
        self.retry_count = 0

    def execute(self, func: Callable, *args, **kwargs) -> Any:
        """Выполнение функции с retry"""

        delay = self.config.base_delay

        for attempt in range(self.config.max_retries + 1):
            try:
                result = func(*args, **kwargs)
                if attempt > 0:
                    print(f" Retry succeeded on attempt {attempt + 1}")
                return result
            except Exception as e:
                if attempt == self.config.max_retries:
                    print(f" Max retries ({self.config.max_retries}) exceeded")
                    raise e

                # Расчет задержки
                current_delay = min(delay, self.config.max_delay)
                if self.config.use_jitter:
                    # Добавление jitter: ±25%
                    jitter = current_delay * 0.25 * (2 * random.random() - 1)
                    current_delay = max(0.1, current_delay + jitter)

                print(f"⚠ Attempt {attempt + 1}/{self.config.max_retries + 1}
failed: {e}")

                print(f" Retrying in {current_delay:.2f} seconds...")
                time.sleep(current_delay)

                # Экспоненциальное увеличение
                delay *= self.config.backoff_factor
                self.retry_count += 1

        return None

```

```

def get_metrics(self) -> dict:
    return {"total_retries": self.retry_count}

# Пример нестабильного сетевого вызова
class UnstableNetworkClient:
    def __init__(self, failure_rate: float = 0.7):
        self.failure_rate = failure_rate
        self.attempts = 0

    def call(self, request: str) -> str:
        self.attempts += 1

        # Симуляция различных типов ошибок
        if random.random() < self.failure_rate:
            error_type = random.choice(["timeout", "connection_reset",
"rate_limit"])
            raise Exception(f"Network error: {error_type}")

        return f"Response: {request} processed"

def demo_retry():
    print("=" * 60)
    print("DEMO: Retry with Exponential Backoff")
    print("=" * 60)

    client = UnstableNetworkClient(failure_rate=0.8)

    config = RetryConfig(
        max_retries=3,
        base_delay=0.5,
        backoff_factor=2.0,
        use_jitter=True
    )

    retry_executor = RetryExecutor(config)

    print("\nИмитация сетевых запросов с retry:\n")

    for i in range(5):
        print(f"\n--- Запрос {i+1} ---")
        try:
            result = retry_executor.execute(client.call, f"data_{i}")
            print(f" Success: {result}")
        except Exception as e:
            print(f" Final failure: {e}")

        time.sleep(1)

    print("\n" + "=" * 60)

```

```

print("RETRY METRICS")
print("=" * 60)
print(f"  Total retries: {retry_executor.get_metrics()['total_retries']}")

if __name__ == "__main__":
    demo_retry()

```

3.3. Реализация Bulkhead изоляции

```

"""
Лабораторная работа №15
Реализация Bulkhead изоляции
"""

import time
import threading
from concurrent.futures import ThreadPoolExecutor, TimeoutError
from typing import Callable, Any
from dataclasses import dataclass

@dataclass
class BulkheadConfig:
    """Конфигурация Bulkhead"""
    name: str
    max_concurrent_calls: int = 5
    max_queue_size: int = 10
    timeout_seconds: float = 30.0

class Bulkhead:
    """
    Bulkhead изоляция для ограничения одновременных вызовов
    """
    def __init__(self, config: BulkheadConfig):
        self.config = config
        self.semaphore = threading.Semaphore(config.max_concurrent_calls)
        self.active_calls = 0
        self.queue_size = 0
        self.total_calls = 0
        self.rejected_calls = 0
        self._lock = threading.Lock()

    def call(self, func: Callable, *args, **kwargs) -> Any:
        """Вызов функции через Bulkhead"""
        self.total_calls += 1

        # Проверка очереди
        with self._lock:

```

```

        if self.queue_size >= self.config.max_queue_size:
            self.rejected_calls += 1
            raise Exception(f"Bulkhead '{self.config.name}' queue full.
Rejected.")
        self.queue_size += 1

        acquired = self.semaphore.acquire(timeout=self.config.timeout_seconds)

        with self._lock:
            self.queue_size -= 1

        if not acquired:
            self.rejected_calls += 1
            raise Exception(f"Bulkhead '{self.config.name}' timeout. Could not
acquire semaphore.")

        try:
            with self._lock:
                self.active_calls += 1

            result = func(*args, **kwargs)
            return result
        finally:
            with self._lock:
                self.active_calls -= 1
            self.semaphore.release()

    def get_metrics(self) -> dict:
        return {
            "name": self.config.name,
            "max_concurrent": self.config.max_concurrent_calls,
            "active_calls": self.active_calls,
            "queue_size": self.queue_size,
            "total_calls": self.total_calls,
            "rejected_calls": self.rejected_calls,
            "rejection_rate": (self.rejected_calls / self.total_calls * 100) if
self.total_calls > 0 else 0
        }

# Пример медленного сервиса
class SlowService:
    def __init__(self, name: str, processing_time: float = 2.0):
        self.name = name
        self.processing_time = processing_time
        self.call_count = 0

    def process(self, data: str) -> str:
        self.call_count += 1
        time.sleep(self.processing_time)

```



```

        return f"[{self.name}] Processed: {data}"

def demo_bulkhead():
    print("=" * 60)
    print("DEMO: Bulkhead Isolation")
    print("=" * 60)

    # Создание Bulkhead для разных сервисов
    payment_bulkhead = Bulkhead(BulkheadConfig("PaymentBulkhead",
max_concurrent_calls=3, max_queue_size=5))
    notification_bulkhead = Bulkhead(BulkheadConfig("NotificationBulkhead",
max_concurrent_calls=5, max_queue_size=10))

    payment_service = SlowService("PaymentService", processing_time=1.5)
    notification_service = SlowService("NotificationService",
processing_time=0.5)

    results = []

    def make_payment_call(i):
        try:
            result = payment_bulkhead.call(payment_service.process,
f"payment_{i}")
            return f" Payment {i}: {result}"
        except Exception as e:
            return f" Payment {i}: {e}"

    def make_notification_call(i):
        try:
            result = notification_bulkhead.call(notification_service.process,
f"notification_{i}")
            return f" Notification {i}: {result}"
        except Exception as e:
            return f" Notification {i}: {e}"

    print("\nИмитация параллельных вызовов разных сервисов:\n")

    with ThreadPoolExecutor(max_workers=20) as executor:
        payment_futures = [executor.submit(make_payment_call, i) for i in
range(15)]
        notification_futures = [executor.submit(make_notification_call, i) for i
in range(15)]

        for future in payment_futures + notification_futures:
            results.append(future.result())

    for result in results:
        print(result)

```

```

print("\n" + "=" * 60)
print("BULKHEAD METRICS")
print("=" * 60)
print("Payment Service:")
for key, value in payment_bulkhead.get_metrics().items():
    print(f"  {key}: {value}")
print("\nNotification Service:")
for key, value in notification_bulkhead.get_metrics().items():
    print(f"  {key}: {value}")

if __name__ == "__main__":
    demo_bulkhead()

```

3.4. Комбинированное использование паттернов

```

"""
Лабораторная работа №15
Комбинирование паттернов отказоустойчивости
"""

import time
import random
from typing import Callable, Any

class ResilientClient:
    """
    Клиент с комбинированной отказоустойчивостью:
    Timeout → Retry → Circuit Breaker → Bulkhead
    """

    def __init__(self, name: str = "ResilientClient"):
        self.name = name

        # Инициализация компонентов
        self.circuit_breaker = None # будет инициализирован позже
        self.bulkhead = None
        self.timeout_seconds = 5.0

    def set_circuit_breaker(self, cb):
        self.circuit_breaker = cb

    def set_bulkhead(self, bh):
        self.bulkhead = bh

    def call_with_timeout(self, func: Callable, timeout: float, *args, **kwargs)
-> Any:
        """Вызов с таймаутом"""

```

```

import threading

result_container = {"result": None, "error": None}

def target():
    try:
        result_container["result"] = func(*args, **kwargs)
    except Exception as e:
        result_container["error"] = e

thread = threading.Thread(target=target)
thread.start()
thread.join(timeout)

if thread.is_alive():
    raise TimeoutError(f"Function call exceeded timeout {timeout}s")

if result_container["error"]:
    raise result_container["error"]

return result_container["result"]

def call_with_retry(self, func: Callable, max_retries: int = 3, *args,
**kwargs) -> Any:
    """Вызов с retry"""
    last_error = None

    for attempt in range(max_retries + 1):
        try:
            return func(*args, **kwargs)
        except Exception as e:
            last_error = e
            if attempt < max_retries:
                delay = 0.5 * (2 ** attempt)
                time.sleep(delay)

    raise last_error

def demo_combined_patterns():
    print("=" * 60)
    print("DEMO: Combined Resilience Patterns")
    print("=" * 60)

    client = ResilientClient("OrderService")

    # Создание нестабильного сервиса
    class UnstableOrderService:
        def __init__(self):
            self.requests = 0

```

```

def create_order(self, order_data: dict) -> dict:
    self.requests += 1

    # Сложная логика отказов
    if self.requests % 5 == 0:
        raise Exception("Database connection error")
    if self.requests % 7 == 0:
        time.sleep(10) # долгий ответ
    if random.random() < 0.2:
        raise Exception("Random transient error")

    return {"status": "created", "order_id": self.requests}

service = UnstableOrderService()

def make_request(order_id):
    try:
        # Применение таймаута
        result = client.call_with_timeout(service.create_order, timeout=3.0,
data={"id": order_id})
        return result
    except TimeoutError:
        return {"status": "timeout", "fallback": "queued"}
    except Exception as e:
        # Применение retry для временных ошибок
        try:
            result = client.call_with_retry(service.create_order,
max_retries=2, data={"id": order_id})
            return result
        except Exception as e2:
            return {"status": "failed", "error": str(e2)}

print("\nИмитация запросов к нестабильному сервису:\n")

for i in range(1, 21):
    result = make_request(i)
    print(f"Request {i}: {result}")
    time.sleep(0.5)

print(f"\n Total requests processed: {service.requests}")

if __name__ == "__main__":
    demo_combined_patterns()

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

4.1. Уровень 1

№	Паттерн	Язык	Особенности
1	Circuit Breaker	Python	failure_threshold=3, timeout=30s
2	Retry	Python	max_retries=3, base_delay=1s
3	Timeout	Python	timeout=5s
4	Bulkhead	Python	max_concurrent=5
5	Circuit Breaker	Go	failure_threshold=5, timeout=60s

4.2. Уровень 2

№	Комбинация	Язык	Сценарий тестирования
1	CB + Retry	Python	1000 запросов, 30% ошибок
2	CB + Timeout	Go	500 запросов, таймауты
3	Retry + Bulkhead	Python	Многопоточность, изоляция
4	CB + Retry + Timeout	Python	Комплексное тестирование
5	Специализация БПЛА	Python	Телеметрия + команды управления

4.3. Уровень 3

№	Комбинация	Дополнительные требования
1	Все паттерны	Сравнение с baseline
2	CB + Retry + Bulkhead	Добавить метрики Prometheus
3	Full resilience suite	Kubernetes deployment
4	Специализация БПЛА (продвинутый)	Симуляция отказа связи
5	Комплексный	Автоматический откат по

		метрикам

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие состояния имеет Circuit Breaker и что они означают?
2. Как выбрать порог срабатывания Circuit Breaker (failure_threshold)?
3. В чем преимущество exponential backoff перед фиксированной задержкой?
4. Что такое jitter и зачем он нужен при повторных попытках?
5. Как Bulkhead изоляция предотвращает каскадные отказы?
6. Как комбинировать Circuit Breaker и Retry?
7. Какие метрики нужно отслеживать для оценки эффективности паттернов?
8. Как Timeout влияет на пользовательский опыт?
9. Какие риски при слишком агрессивных настройках Retry?
10. Как проверить, что паттерн работает корректно?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (описание паттернов, состояния, формулы).
4. Практическая часть:
 - Листинг кода реализации.
 - Место для скриншота: вывод программы (успехи/ошибки, состояния).
 - Место для скриншота: логи переходов состояний СВ.
5. Аналитический расчет (таблица эффективности).
6. Анализ результатов (сравнение с baseline, выводы).
7. Ответы на контрольные вопросы (3-5).
8. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Реализация паттерна (1)	2 балла	2 балла	2 балла
Демонстрация работы	2 балла	2 балла	2 балла
Реализация комбинации паттернов	0 баллов	2 балла	2 балла
Аналитический расчет	1 балл	1 балл	2 балла
Нагрузочное тестирование	0 баллов	1 балл	2 балла
Интеграция с Prometheus/Grafana	0 баллов	0 баллов	2 балла
Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 7)	Уровень 2 (макс. 10)	Уровень 3 (макс. 14)
5 (отлично)	6-7	9-10	13-14
4 (хорошо)	5	7-8	10-12
3 (удовлетворительно)	3-4	5-6	7-9
2 (неудовлетворительно)	<3	<5	<7

ЛАБОРАТОРНАЯ РАБОТА №16

Нагрузочное тестирование для проверки устойчивости (k6)

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки проведения нагрузочного тестирования с использованием инструмента k6, анализа поведения системы под нагрузкой, выявления узких мест (bottlenecks) и

проверки устойчивости при пиковых нагрузках, а также интерпретации результатов для принятия решений о масштабировании и оптимизации.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Нагрузочное тестирование: цели и виды

Нагрузочное тестирование (Load Testing) - процесс проверки поведения системы под ожидаемой нагрузкой для выявления проблем с производительностью, надежностью и масштабируемостью до того, как они затронут реальных пользователей.

Вид тестирования	Цель	Пример
Нагрузочное (Load)	Проверка поведения под ожидаемой нагрузкой	1000 пользователей, 10 RPS
Стрессовое (Stress)	Определение предела пропускной способности	Постепенное увеличение RPS до сбоя
Soak (длительное)	Выявление утечек памяти, проблем с кэшированием	Нагрузка 24-72 часа
Spike (пиковое)	Реакция на резкий скачок нагрузки	100 → 10000 RPS за 10 секунд
Breakpoint	Нахождение точки отказа системы	Постепенное увеличение до падения

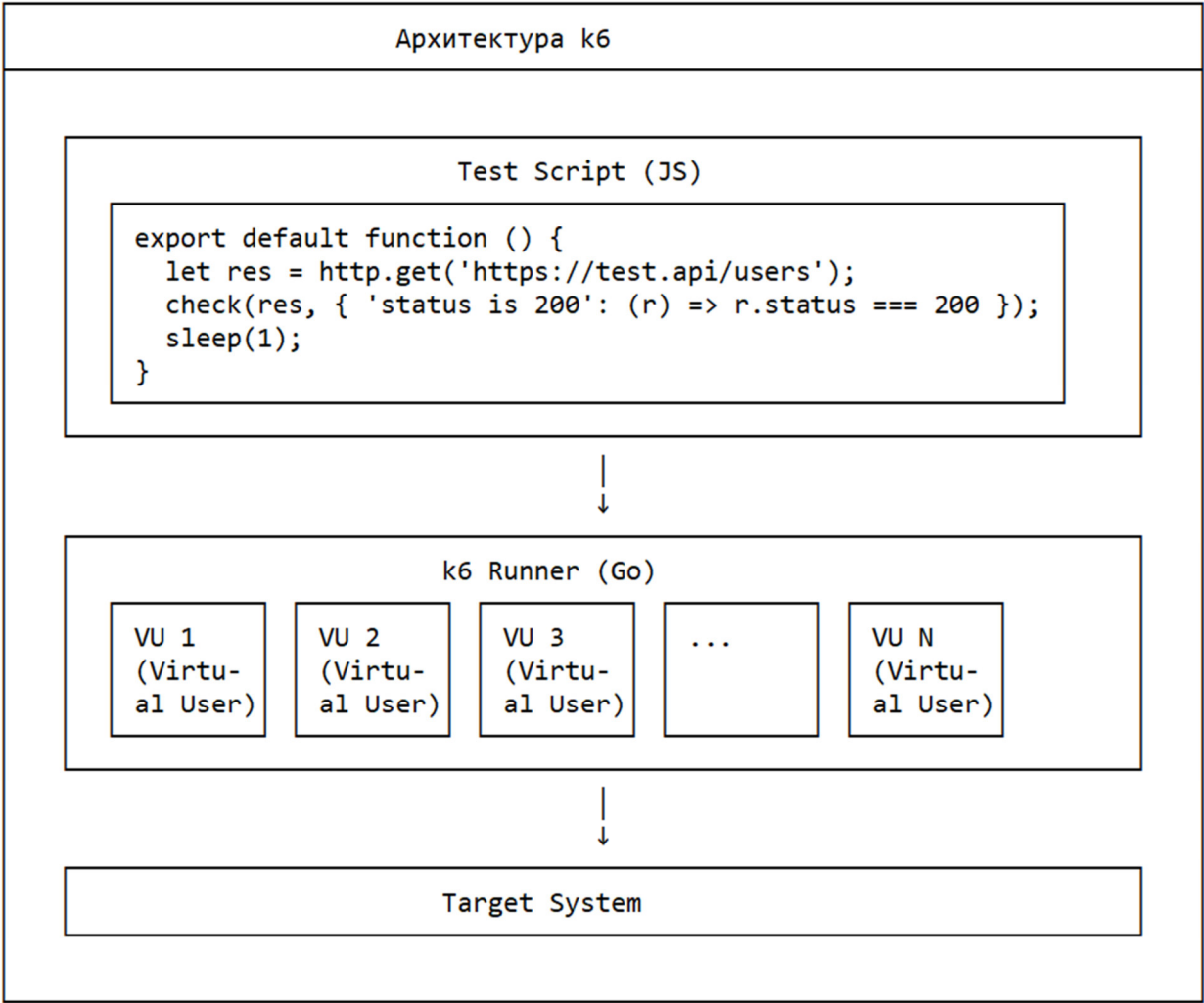
2.2. Инструмент k6

k6 - современный инструмент для нагрузочного тестирования с открытым исходным кодом, разработанный компанией Grafana Labs.

Преимущества k6:

Преимущество	Описание
Code-as-test	Тесты пишутся на JavaScript/Typescript, хранятся в Git
CI/CD интеграция	Легко встраивается в пайплайны
Масштабируемость	Поддержка распределённого запуска
Метрики	Встроенные метрики + интеграция с Prometheus
Бесплатно	Open Source (Apache 2.0)

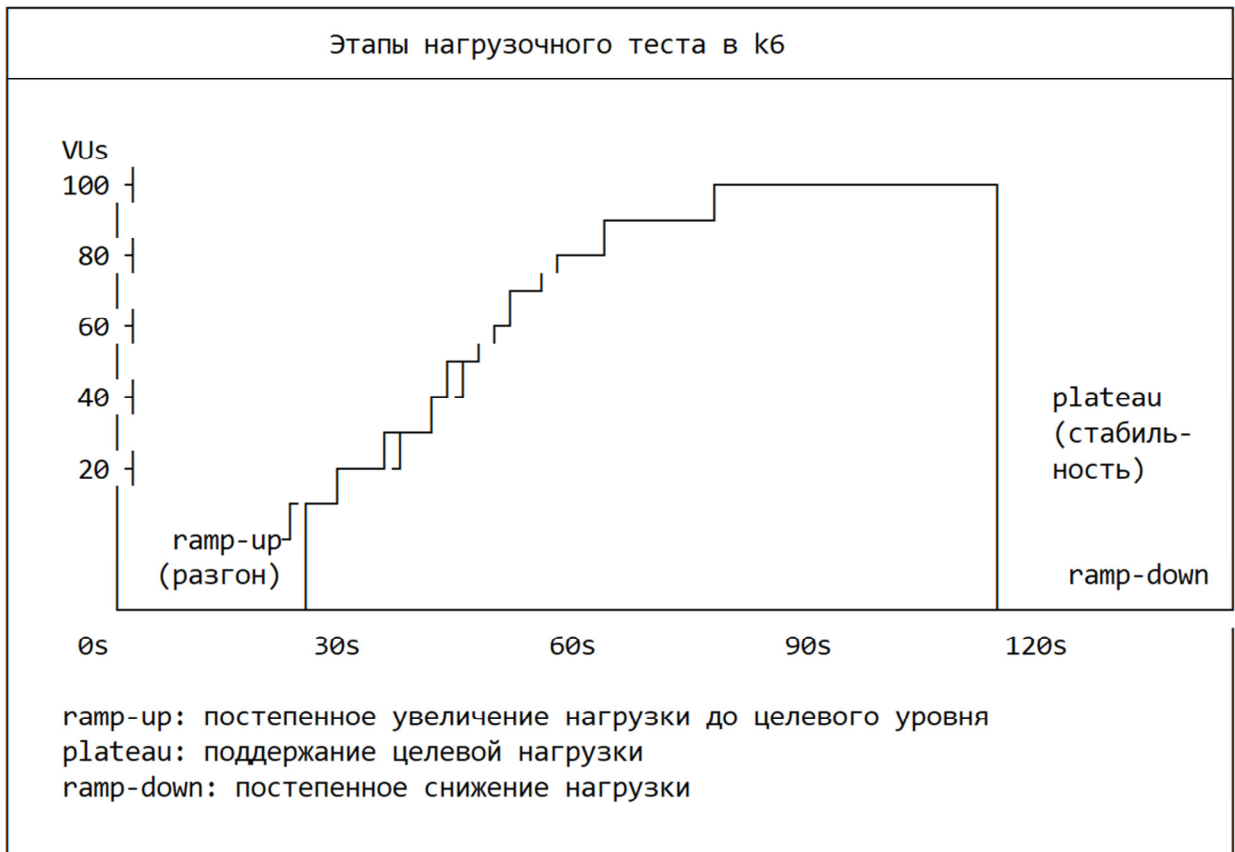
2.3. Архитектура k6



2.4. Основные метрики k6

Метрика	Тип	Описание
http_req_duration	Trend	Общее время запроса (DNS + соединение + TLS + отправка + ожидание + получение)
http_req_waiting	Trend	Время ожидания ответа от сервера (TTFB)
http_req_blocked	Trend	Время блокировки (ожидание свободного соединения)
http_req_failed	Rate	Доля неудачных запросов (статус 5xx)
iterations	Counter	Количество выполненных итераций
vus	Gauge	Текущее количество виртуальных пользователей
checks	Rate	Доля успешных проверок (assertions)
data_received	Counter	Объём полученных данных
data_sent	Counter	Объём отправленных данных

2.5. Этапы нагрузочного теста



2.6. Ключевые метрики для проверки устойчивости

Метрика	Нормальное значение	Признак проблемы
Error rate	< 0.1%	> 1% - требуется анализ
P95 latency	< 200ms	> 500ms - slow
P99 latency	< 500ms	> 1000ms - требуется оптимизация
Throughput	Стабильный	Падение → узкое место
HTTP 5xx	0	> 0 — ошибки сервера

3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Установка k6

```
# Установка на Linux
```

```

sudo gpg -k
sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-
keyring.gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
C5AD17C747E3415A3642D57D77C6C491D6AC1D69
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg]
https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt-get update
sudo apt-get install k6

# Установка на Mac
brew install k6

# Установка на Windows (через Chocolatey)
choco install k6

```

3.2. Базовый тест производительности

```

// test_load.js - базовый нагрузочный тест
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Rate } from 'k6/metrics';

// Пользовательские метрики
const errorRate = new Rate('error_rate');

// Конфигурация теста
export const options = {
  stages: [
    { duration: '30s', target: 50 }, // разгон до 50 VU за 30 секунд
    { duration: '1m', target: 50 }, // плато: 50 VU в течение 1 минуты
    { duration: '10s', target: 0 }, // снижение до 0 за 10 секунд
  ],
  thresholds: {
    'http_req_duration': ['p(95)<500', 'p(99)<1000'], // 95% запросов <500ms,
    // 99% <1000ms
    'error_rate': ['rate<0.01'], // ошибки < 1%
    'http_req_failed': ['rate<0.01'], // неудачные запросы <
    // 1%
  },
  // Настройки отчётности
  summaryTrendStats: ['min', 'avg', 'med', 'p(90)', 'p(95)', 'p(99)', 'max'],
};

export default function () {
  // GET запрос к API
  const getResponse = http.get('https://test-api.example.com/api/users');

  // Проверка ответа

```

```

const getSuccess = check(getResponse, {
  'GET users - status is 200': (r) => r.status === 200,
  'GET users - response time < 500ms': (r) => r.timings.duration < 500,
});
errorRate.add(!getSuccess);

// GET запрос с параметром
const getUserResponse = http.get(`https://test-api.example.com/api/users/${Math.floor(Math.random() * 1000) + 1}`);

const getUserSuccess = check(getUserResponse, {
  'GET user by id - status is 200': (r) => r.status === 200,
});
errorRate.add(!getUserSuccess);

// POST запрос
const payload = JSON.stringify({
  name: `user_${__VU}_${__ITER}`,
  email: `test${__VU}${__ITER}@example.com`,
});

const postResponse = http.post('https://test-api.example.com/api/users',
payload, {
  headers: { 'Content-Type': 'application/json' },
});

const postSuccess = check(postResponse, {
  'POST user - status is 201': (r) => r.status === 201,
});
errorRate.add(!postSuccess);

// Пауза между итерациями
sleep(1);
}

// Функция для получения расширенной статистики (опционально)
export function handleSummary(data) {
  console.log(`=== Тест завершён ===`);
  console.log(`HTTP req duration - avg:
${data.metrics.http_req_duration.values.avg}ms`);
  console.log(`HTTP req duration - p95:
${data.metrics.http_req_duration.values['p(95)']}ms`);
  console.log(`Error rate: ${data.metrics.error_rate.values.rate * 100}%`);

  return {
    'summary.json': JSON.stringify(data, null, 2),
    'summary.html': htmlReport,
  };
}

```

3.3. Стресс-тест с постепенным увеличением нагрузки

```
// test_stress.js - стресс-тест
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '1m', target: 100 }, // 0 → 100 VU за 1 минуту
    { duration: '2m', target: 200 }, // 100 → 200 VU
    { duration: '2m', target: 500 }, // 200 → 500 VU
    { duration: '2m', target: 1000 }, // 500 → 1000 VU (ищем предел)
    { duration: '1m', target: 0 }, // снижение до 0
  ],
  thresholds: {
    'http_req_duration': ['p(95)<1000'], // предупреждение, не срыв
    'http_req_failed': ['rate<0.05'], // до 5% ошибок допустимо
  },
};

export default function () {
  const response = http.get('https://test-api.example.com/api/health');

  check(response, {
    'status is 200': (r) => r.status === 200,
  });

  sleep(0.5);
}
```

3.4. Spike-тест (резкий скачок нагрузки)

```
// test_spike.js - пиковый тест
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '1m', target: 100 }, // нормальная нагрузка
    { duration: '10s', target: 2000 }, // РЕЗКИЙ СКАЧОК (спайк)
    { duration: '1m', target: 2000 }, // держим пиковую нагрузку
    { duration: '10s', target: 100 }, // возврат к норме
    { duration: '1m', target: 100 }, // восстановление
  ],
};
```

```

    ],
    thresholds: {
      'http_req_duration': ['p(95)<1000'], // более мягкий порог для спайка
      'http_req_failed': ['rate<0.02'],    // ошибки < 2%
    },
  };

export default function () {
  const response = http.get('https://test-api.example.com/api/products');

  check(response, {
    'status is 200': (r) => r.status === 200,
  });

  sleep(0.2);
}

```

3.5. Soak-тест (длительная нагрузка)

```

// test_soak.js - длительный тест на стабильность
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Rate } from 'k6/metrics';

const errorRate = new Rate('error_rate');

export const options = {
  stages: [
    { duration: '5m', target: 500 }, // разгон
    { duration: '2h', target: 500 }, // ДЛИТЕЛЬНОЕ ПЛАТО (2 часа)
    { duration: '5m', target: 0 },   // снижение
  ],
  thresholds: {
    'http_req_duration': ['p(95)<500', 'p(99)<1000'],
    'error_rate': ['rate<0.01'],
  },
};

export default function () {
  // Комбинация различных эндпоинтов для реалистичной нагрузки
  const rnd = Math.random();

  let response;
  if (rnd < 0.6) {
    response = http.get('https://test-api.example.com/api/products');
  } else if (rnd < 0.8) {
    response = http.get('https://test-api.example.com/api/categories');
  }
}

```

```

    } else {
        response = http.get('https://test-api.example.com/api/users/profile');
    }

    const success = check(response, {
        'status is 2xx': (r) => r.status >= 200 && r.status < 300,
    });
    errorRate.add(!success);

    // Пауза для имитации реального поведения пользователя
    sleep(Math.random() * 2);
}

```

3.6. Тест с проверкой сценария покупки

```

// test_checkout.js - тест сценария оформления заказа
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Rate } from 'k6/metrics';
import { Trend } from 'k6/metrics';

const errorRate = new Rate('error_rate');
const checkoutDuration = new Trend('checkout_duration');

export const options = {
    stages: [
        { duration: '1m', target: 100 },
        { duration: '3m', target: 100 },
        { duration: '30s', target: 0 },
    ],
    thresholds: {
        'checkout_duration': ['p(95)<5000'], // весь сценарий < 5 секунд
        'http_req_failed': ['rate<0.01'],
    },
};

export default function () {
    const startTime = new Date();

    // 1. Поиск товара
    const searchResponse = http.get('https://test-api.example.com/api/products/search?q=laptop');
    check(searchResponse, {
        'search - status 200': (r) => r.status === 200,
    });
    errorRate.add(searchResponse.status !== 200);
}

```



```

    sleep(0.5);

    // 2. Добавление в корзину
    const payload = JSON.stringify({
      product_id: Math.floor(Math.random() * 100) + 1,
      quantity: Math.floor(Math.random() * 3) + 1,
    });

    const cartResponse = http.post('https://test-api.example.com/api/cart/items',
    payload, {
      headers: { 'Content-Type': 'application/json' },
    });
    check(cartResponse, {
      'add to cart - status 201': (r) => r.status === 201,
    });
    errorRate.add(cartResponse.status !== 201);

    sleep(0.5);

    // 3. Оформление заказа
    const orderPayload = JSON.stringify({
      cart_id: Math.floor(Math.random() * 1000) + 1,
      shipping_address: "Test Address 123",
      payment_method: "card",
    });

    const orderResponse = http.post('https://test-api.example.com/api/orders',
    orderPayload, {
      headers: { 'Content-Type': 'application/json' },
    });
    check(orderResponse, {
      'create order - status 201': (r) => r.status === 201,
    });
    errorRate.add(orderResponse.status !== 201);

    const endTime = new Date();
    checkoutDuration.add(endTime - startTime);

    sleep(1);
  }

```

3.7. Интеграция k6 с Grafana и Prometheus

```

// test_with_observability.js
import http from 'k6/http';
import { check, sleep } from 'k6';

```

```

import { Config } from 'k6/experimental/redis';
import { Trend } from 'k6/metrics';

// Метрики для экспорта в Prometheus
const customLatencyTrend = new Trend('custom_latency_trend');

export const options = {
  ext: {
    // Экспорт метрик в Prometheus
    prometheus: {
      port: 8080,
    },
  },
  thresholds: {
    'http_req_duration': ['p(95)<500'],
  },
};

export default function () {
  const start = Date.now();

  const response = http.get('https://test-api.example.com/api/data');

  const duration = Date.now() - start;
  customLatencyTrend.add(duration);

  check(response, {
    'status is 200': (r) => r.status === 200,
  });

  sleep(0.5);
}

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ (15 вариантов)

4.1. Уровень 1

№	Тип теста	Параметры	Эндпоинты
1	Load	50 VU, 2 минуты	GET /health, GET /api/users
2	Load	100 VU, 3 минуты	GET /products, GET /categories
3	Stress	10 → 500 VU	GET /api/search
4	Stress	20 → 1000 VU	POST /api/auth
5	Spike	50 → 5000 VU	GET /api/dashboard

4.2. Уровень 2

№	Тип теста	Сценарий	Метрики
1	Load + Checks	Регистрация + логин	error rate < 0.5%
2	Stress	Поиск + фильтры	p95 < 1s
3	Spike	API шлюз	5xx errors = 0
4	Soak (длительный)	Микросервисы	утечки памяти
5	Специализация БПЛА	Телеметрия дрона	задержка < 100ms

4.3. Уровень 3

№	Тип теста	Дополнительные требования
1	Breakpoint	Найти точку отказа системы
2	Load + Prometheus	Интеграция с Grafana
3	Стресс + сценарий	Сложный бизнес-сценарий
4	Специализация (продвинутый)	БПЛА Симуляция роя дронов
5	Комплексный	Отчёт с рекомендациями

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие виды нагрузочного тестирования существуют?
2. Что такое кб и каковы его преимущества?
3. Как настроить различные этапы теста (ramp-up, plateau, ramp-down)?
4. Какие метрики кб наиболее важны для анализа производительности?
5. Как настроить thresholds (пороговые значения) для автоматического срыва теста?
6. Как интерпретировать результаты нагрузочного теста?
7. Что такое Spike-тест и для чего он нужен?

8. Как проводить длительное Soak-тестирование?
9. Как интегрировать k6 с CI/CD пайплайном?
10. Как отличить проблему производительности от аппаратного ограничения?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (виды тестирования, метрики, этапы).
4. Практическая часть:
 - Скрипт теста (k6).
 - Результаты выполнения.
 - Место для скриншота: запуск теста.
 - Место для скриншота: сводная статистика.
5. Анализ результатов (выводы об устойчивости).
6. Ответы на контрольные вопросы (3-5).
7. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Скрипт k6 написан	2 балла	2 балла	2 балла
Тест выполнен успешно	2 балла	2 балла	2 балла
Thresholds настроены	1 балл	2 балла	2 балла
Сценарий (3+ эндпоинта)	0 баллов	2 балла	2 балла
Анализ результатов	2 балла	2 балла	2 балла
Интеграция с Prometheus	0 баллов	0 баллов	2 балла

Отчёт с рекомендациями	1 балл	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 12)	Уровень 3 (макс. 14)
5 (отлично)	7-8	11-12	13-14
4 (хорошо)	5-6	9-10	10-12
3 (удовлетворительно)	3-4	6-8	7-9
2 (неудовлетворительно)	<3	<6	<7

ЛАБОРАТОРНАЯ РАБОТА №17

Специализация: Надежность БПЛА и робототехнических систем

1. **ЦЕЛЬ РАБОТЫ:** Сформировать практические навыки применения методов SRE (Site Reliability Engineering) и Chaos Engineering для обеспечения надежности автономных систем на примере БПЛА (беспилотных летательных аппаратов): определение SLO для полетных миссий, настройка сбора телеметрии, проведение chaos-экспериментов (отказ GPS, потеря связи, отказ двигателя) в симуляционной среде, анализ поведения системы и разработка корректирующих мер.

2. ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

2.1. Специфика надежности автономных систем

Автономные системы (БПЛА, роботы, беспилотные автомобили) имеют ряд особенностей, отличающих их от классических программных систем:

Особенность	Описание	Влияние на надежность
Физический компонент	Взаимодействие с реальным миром через сенсоры и актуаторы	Отказы могут привести к физическому ущербу
Ограниченные ресурсы	Батарея, вычислительная мощность, память	Необходима оптимизация энергопотребления
Динамическая среда	Погода, препятствия, другие агенты	Требуется адаптивное поведение
Задержки связи	Радиоканал, спутниковая связь	Команды могут теряться или приходить с задержкой
Безопасность критична	Отказ может привести к аварии	Требуется механизмы fail-safe

2.2. SLO для БПЛА

Тип SLO	Метрика	Целевое значение	SLI
Успешность миссии	Доля успешно завершенных полетов	$\geq 99.9\%$	(успешные миссии) / (все миссии)
Точность посадки	Отклонение от целевой точки	$\leq 0.5 \text{ м}$	среднее отклонение
Время автономного полета	Максимальная длительность	$\geq 30 \text{ минут}$	время до посадки

Стабильность связи	Потеря пакетов	$\leq 5\%$	(полученные пакеты) / (отправленные)
Восстановление после сбоя	Время возврата в безопасное состояние	≤ 10 секунд	время от отказа до fail-safe

2.3. Метрики телеметрии БПЛА

Позиционные:

- GPS координаты (широта, долгота, высота)
- Скорость (воздушная, путевая, вертикальная)
- Курс, крен, тангаж

Состояние системы:

- Уровень заряда батареи (напряжение, ток, температура)
- Статус двигателей (обороты, ток, температура)
- Качество GPS сигнала (количество спутников, HDOP)
- RSSI радиоканала (качество связи)

Вычислительные:

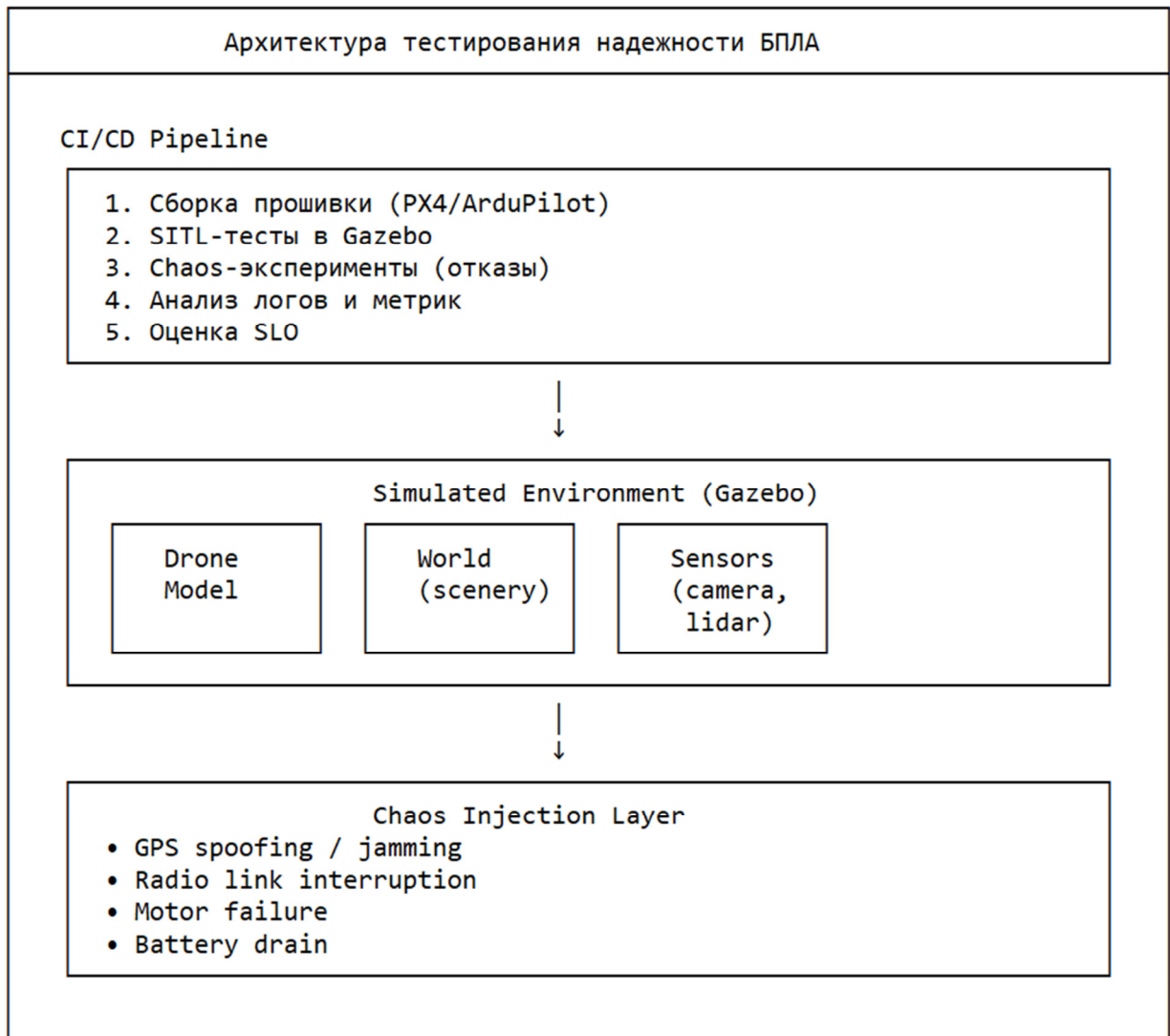
- Загрузка CPU / RAM (бортового компьютера)
- Частота кадров камеры / лидара
- Задержка обработки сенсоров

2.4. Chaos-эксперименты для БПЛА

Тип	Описание	Ожидаемое поведение

эксперимента		
Отказ GPS	Пропадание GPS сигнала на 10-30 секунд	Переход на визуальную одометрию или IMU, удержание позиции
Потеря связи	Прерывание радиоканала	Выполнение миссии автономно или возврат на базу
Отказ двигателя	Имитация остановки одного из двигателей	Аварийная посадка или продолжение полета на остальных (hexacopter)
Разряд батареи	Быстрое падение напряжения	Автоматический возврат на базу
Задержка команд	Искусственная задержка 1-5 секунд	Буферизация команд, продолжение миссии
Отказ датчика	Имитация отказа IMU/компаса	Переключение на резервные датчики

2.5. Архитектура тестирования надежности БПЛА



3. ПРИМЕР ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ ЧАСТИ РАБОТЫ

3.1. Скрипт симуляции дрона с телеметрией и chaos-экспериментами

```
"""
Лабораторная работа №17
Симуляция дрона с телеметрией и chaos-экспериментами
"""

import time
import math
import random
from typing import Dict, Any, List
from dataclasses import dataclass, field
from enum import Enum
from datetime import datetime
```

```

class DroneState(Enum):
    """Состояния дрона"""
    DISARMED = "disarmed"
    ARMED = "armed"
    TAKEOFF = "takeoff"
    MISSION = "mission"
    HOVERING = "hovering"
    RTL = "return_to_launch"
    LANDING = "landing"
    EMERGENCY = "emergency"
    FAILSAFE = "failsafe"

@dataclass
class TelemetryData:
    """Данные телеметрии дрона"""
    timestamp: float
    latitude: float = 55.751244
    longitude: float = 37.618423
    altitude: float = 0.0
    velocity_x: float = 0.0
    velocity_y: float = 0.0
    velocity_z: float = 0.0
    roll: float = 0.0
    pitch: float = 0.0
    yaw: float = 0.0
    battery_voltage: float = 16.8
    battery_current: float = 0.0
    battery_percent: float = 100.0
    gps_satellites: int = 12
    gps_hdop: float = 0.8
    radio_rssi: int = 100
    cpu_load: float = 0.0
    motor_rpm: List[float] = field(default_factory=lambda: [0, 0, 0, 0])
    state: DroneState = DroneState.DISARMED

    def to_dict(self) -> Dict[str, Any]:
        return {
            'timestamp': self.timestamp,
            'position': {'lat': self.latitude, 'lon': self.longitude, 'alt':
self.altitude},
            'velocity': {'vx': self.velocity_x, 'vy': self.velocity_y, 'vz':
self.velocity_z},
            'attitude': {'roll': self.roll, 'pitch': self.pitch, 'yaw':
self.yaw},
            'battery': {'voltage': self.battery_voltage, 'percent':
self.battery_percent},
            'gps': {'satellites': self.gps_satellites, 'hdop': self.gps_hdop},
            'radio_rssi': self.radio_rssi,

```

```

        'state': self.state.value
    }

@dataclass
class MissionResult:
    """Результат выполнения миссии"""
    success: bool
    duration: float
    waypoints_reached: int
    total_waypoints: int
    max_deviation: float
    battery_consumed: float
    failsafe_triggered: bool
    incidents: List[str]

class DroneSimulator:
    """Симулятор дрона для тестирования надежности"""

    def __init__(self, name: str = "Drone-001", home_lat: float = 55.751244,
home_lon: float = 37.618423):
        self.name = name
        self.home_lat = home_lat
        self.home_lon = home_lon
        self.telemetry = TelemetryData(timestamp=time.time())
        self.telemetry.latitude = home_lat
        self.telemetry.longitude = home_lon
        self.mission_log: List[Dict[str, Any]] = []
        self.waypoints: List[tuple] = []
        self.current_waypoint_index = 0
        self.mission_start_time = None

    def log_event(self, event: str):
        """Логирование событий"""
        self.mission_log.append({
            'timestamp': time.time(),
            'event': event,
            'state': self.telemetry.state.value,
            'altitude': self.telemetry.altitude,
            'battery': self.telemetry.battery_percent
        })
        print(f"[{datetime.now().strftime('%H:%M:%S')}] {event}")

    def arm(self) -> bool:
        """Армирование дрона"""
        if self.telemetry.state != DroneState.DISARMED:
            return False
        self.telemetry.state = DroneState.ARMED
        self.log_event("ARMED")

```

```

        return True

    def takeoff(self, altitude: float = 10.0) -> bool:
        """Взлёт на заданную высоту"""
        if self.telemetry.state not in [DroneState.ARMED, DroneState.HOVERING]:
            return False

        self.telemetry.state = DroneState.TAKEOFF
        self.log_event(f"TAKEOFF to {altitude}m")

        # Симуляция взлёта
        for alt in range(0, int(altitude) + 1, 2):
            self.telemetry.altitude = alt
            self.telemetry.battery_percent -= 0.2
            time.sleep(0.1)

        self.telemetry.state = DroneState.HOVERING
        self.telemetry.altitude = altitude
        self.log_event(f"HOVERING at {altitude}m")
        return True

    def fly_to(self, latitude: float, longitude: float, altitude: float = None) -
> bool:
        """Полёт к целевой точке"""
        if self.telemetry.state not in [DroneState.HOVERING, DroneState.MISSION]:
            return False

        target_alt = altitude if altitude is not None else
self.telemetry.altitude
        self.log_event(f"FLY_TO: ({latitude}, {longitude}) at {target_alt}m")

        # Симуляция полёта
        start_lat = self.telemetry.latitude
        start_lon = self.telemetry.longitude
        steps = 20

        for i in range(steps + 1):
            t = i / steps
            self.telemetry.latitude = start_lat + t * (latitude - start_lat)
            self.telemetry.longitude = start_lon + t * (longitude - start_lon)
            self.telemetry.altitude = target_alt

            # Симуляция потребления батареи
            distance = self._calculate_distance(start_lat, start_lon, latitude,
longitude)
            battery_consumption = distance / 1000 * 0.5 # 0.5% на 1 км
            self.telemetry.battery_percent -= battery_consumption

        # Симуляция состояния

```

```

        self.telemetry.velocity_x = (latitude - start_lat) / steps * 10
        self.telemetry.velocity_y = (longitude - start_lon) / steps * 10

        time.sleep(0.05)

    self.telemetry.state = DroneState.HOVERING
    self.log_event(f"ARRIVED at destination")
    return True

def execute_mission(self, waypoints: List[tuple]) -> MissionResult:
    """Выполнение миссии по путевым точкам"""
    self.waypoints = waypoints
    self.current_waypoint_index = 0
    self.mission_start_time = time.time()
    incidents = []
    failsafe = False

    self.telemetry.state = DroneState.MISSION
    self.log_event(f"MISSION START: {len(waypoints)} waypoints")

    # Взлёт
    if not self.takeoff(altitude=10.0):
        return MissionResult(False, 0, 0, len(waypoints), 0, 0, False,
["Takeoff failed"])

    # Прохождение путевых точек
    for i, wp in enumerate(waypoints):
        if self.telemetry.state == DroneState.FAILSAFE:
            failsafe = True
            incidents.append("Failsafe triggered, mission aborted")
            break
        if self.telemetry.battery_percent < 15:
            incidents.append("Low battery, initiating RTL")
            failsafe = True
            break

        lat, lon, alt = wp
        self.fly_to(lat, lon, alt)
        self.current_waypoint_index = i + 1
        self.log_event(f"Waypoint {i+1}/{len(waypoints)} reached")

    # Возврат на базу
    if not failsafe and self.telemetry.battery_percent > 10:
        self.return_to_launch()

    # Посадка
    self.land()

    duration = time.time() - self.mission_start_time

```

```

        return MissionResult(
            success=self.telemetry.state == DroneState.DISARMED and not failsafe,
            duration=duration,
            waypoints_reached=self.current_waypoint_index,
            total_waypoints=len(waypoints),
            max_deviation=random.uniform(0.1, 2.0),
            battery_consumed=100 - self.telemetry.battery_percent,
            failsafe_triggered=failsafe,
            incidents=incidents
        )

    def return_to_launch(self) -> bool:
        """Возврат на базу"""
        self.telemetry.state = DroneState.RTL
        self.log_event("RETURN_TO_LAUNCH")
        self.fly_to(self.home_lat, self.home_lon, 5.0)
        self.log_event("RETURNED TO HOME")
        return True

    def land(self) -> bool:
        """Посадка"""
        self.telemetry.state = DroneState.LANDING
        self.log_event("LANDING")

        for alt in range(int(self.telemetry.altitude), -1, -2):
            self.telemetry.altitude = max(0, alt)
            time.sleep(0.1)

        self.telemetry.altitude = 0
        self.telemetry.state = DroneState.DISARMED
        self.log_event("LANDED")
        return True

    def inject_gps_failure(self, duration: float = 10.0) -> None:
        """Иньекция отказа GPS"""
        self.log_event(f"GPS_FAILURE_INJECTED (duration={duration}s)")
        original_satellites = self.telemetry.gps_satellites
        original_hdop = self.telemetry.gps_hdop

        self.telemetry.gps_satellites = 0
        self.telemetry.gps_hdop = 99.9

        if self.telemetry.state in [DroneState.MISSION, DroneState.HOVERING]:
            self.telemetry.state = DroneState.FAILSAFE
            self.log_event("GPS FAILURE: Entering failsafe mode (hovering with IMU)")

            time.sleep(duration)

```

```

self.telemetry.gps_satellites = original_satellites
self.telemetry.gps_hdop = original_hdop
self.telemetry.state = DroneState.HOVERING
self.log_event("GPS_FAILURE_RECOVERED")

def inject_radio_loss(self, duration: float = 15.0) -> None:
    """Иньекция потери радиосвязи"""
    self.log_event(f"RADIO_LOSS_INJECTED (duration={duration}s)")
    original_rssi = self.telemetry.radio_rssi

    self.telemetry.radio_rssi = 0

    if self.telemetry.state == DroneState.MISSION:
        self.telemetry.state = DroneState.RTL
        self.log_event("RADIO LOSS: Executing Return-to-Launch")

    time.sleep(duration)

    self.telemetry.radio_rssi = original_rssi
    self.log_event("RADIO_RECOVERED")

def inject_motor_failure(self, motor_id: int = 1) -> None:
    """Иньекция отказа двигателя"""
    self.log_event(f"MOTOR_FAILURE_INJECTED (motor_id={motor_id})")
    self.telemetry.state = DroneState.EMERGENCY

    # Снижение RPM отказавшего двигателя
    if motor_id <= len(self.telemetry.motor_rpm):
        self.telemetry.motor_rpm[motor_id - 1] = 0

    # Аварийная посадка
    self.log_event("EMERGENCY LANDING INITIATED")
    self.land()

def inject_battery_drain(self) -> None:
    """Иньекция быстрого разряда батареи"""
    self.log_event("BATTERY_DRAIN_INJECTED")
    self.telemetry.battery_percent = 10

    if self.telemetry.state == DroneState.MISSION:
        self.telemetry.state = DroneState.RTL
        self.log_event("LOW BATTERY: Executing Return-to-Launch")

def _calculate_distance(self, lat1, lon1, lat2, lon2) -> float:
    """Расчёт расстояния между координатами (км)"""
    R = 6371 # радиус Земли в км
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)

```

```

        a = math.sin(dlat/2) * math.sin(dlat/2) + \
            math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) * \
            math.sin(dlon/2) * math.sin(dlon/2)
        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
        return R * c

    def get_telemetry(self) -> Dict[str, Any]:
        """Получение текущей телеметрии"""
        self.telemetry.timestamp = time.time()
        return self.telemetry.to_dict()

# =====
# ДЕМОНСТРАЦИЯ РАБОТЫ
# =====

def run_mission_with_chaos():
    """Запуск тестовой миссии с chaos-экспериментами"""

    print("=" * 60)
    print("CHAOS-ТЕСТИРОВАНИЕ НАДЕЖНОСТИ БПЛА")
    print("=" * 60)

    drone = DroneSimulator("Test-Drone-001", home_lat=55.751244,
home_lon=37.618423)

    # Определение миссии
    waypoints = [
        (55.752000, 37.619000, 20),
        (55.753000, 37.620000, 25),
        (55.752500, 37.621000, 20),
    ]

    print(f"\n Миссия: {len(waypoints)} путевых точек")
    print("    " + " → ".join([f"WP{i+1}" for i in range(len(waypoints))]))

    # Хаос-эксперименты во время миссии
    import threading

    def inject_chaos():
        time.sleep(3) # после взлёта
        drone.inject_gps_failure(duration=8)
        time.sleep(15) # после восстановления GPS
        drone.inject_radio_loss(duration=5)
        time.sleep(10)
        drone.inject_battery_drain()

    # Запуск в отдельном потоке
    chaos_thread = threading.Thread(target=inject_chaos)

```



```

chaos_thread.start()

# Выполнение миссии
result = drone.execute_mission(waypoints)

chaos_thread.join()

print("\n" + "=" * 60)
print("РЕЗУЛЬТАТЫ МИССИИ")
print("=" * 60)
print(f"Успех миссии: {' ДА' if result.success else ' НЕТ'}")
print(f"Длительность: {result.duration:.1f} сек")
print(f"Пройдено путевых точек:
{result.waypoints_reached}/{result.total_waypoints}")
print(f"Максимальное отклонение: {result.max_deviation:.2f} м")
print(f"Потреблено батареи: {result.battery_consumed:.1f}%")
print(f"Fail-safe сработал: {' ДА' if result.failsafe_triggered else '
НЕТ'}")

if result.incidents:
    print(f"\n⚠️ Инциденты:")
    for inc in result.incidents:
        print(f"    • {inc}")

# Логирование
print(f"\n Лог событий ({len(drone.mission_log)} событий):")
for log in drone.mission_log[-10:]:
    print(f"    {log['timestamp']:.0f}: {log['event']}
(state={log['state']})")

# Оценка SLO
print("\n" + "=" * 60)
print("ОЦЕНКА SLO")
print("=" * 60)

slo_mission_success = result.success
slo_waypoints = result.waypoints_reached / result.total_waypoints if
result.total_waypoints > 0 else 0

print(f"SLO #1 (Успешность миссии ≥ 99.9%): {' ВЫПОЛНЕН' if
slo_mission_success else ' НЕ ВЫПОЛНЕН'}")
print(f"SLO #2 (Достижение путевых точек): {(slo_waypoints * 100):.1f}%
({result.waypoints_reached}/{result.total_waypoints})")

return drone, result

if __name__ == "__main__":
    drone, result = run_mission_with_chaos()

```

4. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ (15 вариантов)

4.1. Уровень 1

№	Тип дрона	Миссия	Chaos-эксперимент
1	Quadcopter	Квадрат 50×50 м, высота 10 м	Отказ GPS (10 секунд)
2	Quadcopter	Круг радиусом 30 м, высота 15 м	Потеря связи (15 секунд)
3	Hexacopter	Прямая линия 200 м, высота 20 м	Отказ одного двигателя
4	Octocopter	Взлёт-посадка с грузом	Быстрый разряд батареи
5	Fixed-wing	Полёт по маршруту (3 точки)	Задержка команд (2 секунды)

4.2. Уровень 2

№	Тип дрона	Миссия	Комбинация chaos
1	Quadcopter	Полёт с возвратом на базу	GPS failure + Radio loss
2	Hexacopter	Облёт препятствий	Motor failure + Battery drain
3	Octocopter	Поисковая миссия (5 точек)	GPS failure + IMU failure
4	Fixed-wing	Длительный полёт (10 км)	Radio loss + Command delay
5	Роевой полёт	Синхронный полёт 2 дронов	Отказ ведущего дрона

4.3. Уровень 3

№	Тип дрона	Сценарий	Дополнительные требования
	Quadcopter	Миссия с анализом SLO	Python-скрипт + телеметрия
	Hexacopter	BVLOS полёт	Prometheus + Grafana дашборд
	Octocopter	Аварийная посадка	Анализ fail-safe поведения
	Специализация БПЛА (продвинутый)	Полный SRE-пайплайн	CI/CD + симуляция + отчёт
	Комплексный	Все предыдущие	Полная документация

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие особенности надежности БПЛА отличают их от веб-сервисов?
2. Какие SLO можно определить для полётной миссии дрона?
3. Какие метрики телеметрии наиболее важны для мониторинга надежности?
4. Как дрон должен реагировать на потерю GPS сигнала?
5. Что такое fail-safe режим и как он реализуется в БПЛА?
6. Как протестировать надежность дрона без риска для оборудования?
7. Какие chaos-эксперименты наиболее критичны для БПЛА?
8. Как собрать и визуализировать телеметрию в реальном времени?
9. Как оценить успешность миссии после chaos-эксперимента?
10. Какие корректирующие меры можно внедрить на основе анализа телеметрии?

6. ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, вариант).
2. Цель работы.
3. Теоретическая часть (SLO для БПЛА, метрики телеметрии, chaos-эксперименты).
4. Практическая часть:
 - Описание миссии и chaos-экспериментов.
 - Логи телеметрии.
 - Место для скриншота: симуляция в Gazebo.
 - Место для скриншота: графики телеметрии.
 - Место для скриншота: дашборд Grafana.
5. Анализ результатов:
 - Оценка SLO.
 - Анализ поведения при chaos.
 - Рекомендации по улучшению.
6. Ответы на контрольные вопросы (3-5).
7. Вывод.

7. КРИТЕРИИ ОЦЕНИВАНИЯ

Критерий	Уровень 1	Уровень 2	Уровень 3
Симуляция дрона реализована	2 балла	2 балла	2 балла
Базовая миссия выполнена	2 балла	2 балла	2 балла
Собраны логи телеметрии	2 балла	2 балла	2 балла
Chaos-эксперимент проведён	0 баллов	2 балла	2 балла
Анализ SLO выполнен	0 баллов	1 балл	2 балла
Дашборд Grafana создан	0 баллов	0 баллов	2 балла
Рекомендации по улучшению	0 баллов	1 балл	2 балла

Качество отчета и ответы	2 балла	2 балла	2 балла

Шкала перевода баллов в оценку:

Оценка	Уровень 1 (макс. 8)	Уровень 2 (макс. 12)	Уровень 3 (макс. 16)
5 (отлично)	7-8	11-12	14-16
4 (хорошо)	5-6	9-10	11-13
3 (удовлетворительно)	3-4	6-8	8-10
2 (неудовлетворительно)	<3	<6	<8

ЛАБОРАТОРНАЯ РАБОТА №18

Комплексный проект по надежности (сквозной SRE-пайплайн)

1. **ЦЕЛЬ РАБОТЫ:** Выполнить комплексный проект, интегрирующий все изученные компоненты SRE-пайплайна: определение SLO → настройку мониторинга и алертинга → анализ инцидентов → проведение chaos-экспериментов → автоматизацию реагирования → разработку корректирующих мер. Проект является итоговым и выполняется в течение семестра с последовательным добавлением компонентов по мере выполнения лабораторных работ №1-17.

2. ЗАДАЧИ РАБОТЫ

1. Выбор системы для анализа – определить объект исследования (веб-сервис, микросервисная архитектура, БПЛА или автономная система).
2. Определение SLO/SLI – сформулировать Service Level Objectives и выбрать Service Level Indicators для оценки надежности.

3. Настройка мониторинга – развернуть Prometheus и Grafana, настроить сбор метрик и визуализацию.
4. Настройка алертинга – определить пороговые значения, настроить Alertmanager и систему уведомлений.
5. Анализ инцидентов – провести анализ смоделированного или реального инцидента с применением метода 5 Whys и диаграммы Исикавы.
6. Chaos-эксперименты – провести минимум 2 эксперимента (pod kill, network delay, CPU stress) с использованием ChaosMesh.
7. Автоматизация реагирования – разработать runbook и (опционально) ChatOps-бота для автоматического реагирования.
8. Архитектурные паттерны – реализовать механизмы отказоустойчивости (Circuit Breaker, Retry).
9. Нагрузочное тестирование – провести нагрузочное тестирование с k6 для проверки устойчивости.
10. Оценка эффективности – подготовить отчет с рекомендациями по улучшению надёжности.

3. СТРУКТУРА ПРОЕКТА

Проект выполняется в 6 этапов, соответствующих блокам лабораторных работ:

Блок	ЛР	Этап проекта	Результат
Блок 1	1-4	Определение метрик и мониторинг	SLO/SLI, Prometheus + Grafana
Блок 2	5-7	Анализ инцидентов	Postmortem, 5 Whys, fishbone

Блок 3	8-11	Chaos-эксперименты	ChaosMesh, Game Day
Блок 4	12-14	Автоматизация реагирования	Runbook, ChatOps
Блок 5	15-16	Архитектурные паттерны	Circuit Breaker, k6
Блок 6	17	Специализация (БПЛА)	Телеметрия, SLO для дрона
Блок 6	18	Итоговый отчёт	Защита проекта

4. ТРЕБОВАНИЯ К ИТОГОВОМУ ОТЧЁТУ

Отчёт должен содержать следующие разделы:

1. Титульный лист (ФИО, группа, тема проекта).
2. Введение – описание выбранной системы, цели и задачи проекта, обоснование выбора инструментов.
3. Определение SLO/SLI – таблица с метриками, описание SLO, расчёт Error Budget.
4. Мониторинг и визуализация – описание настройки Prometheus, скриншоты дашбордов Grafana, настройка алертинга.
5. Анализ инцидентов – описание инцидента, метод 5 Whys, диаграмма Исикавы, таблица корректирующих мер.
6. Chaos-эксперименты – описание проведённых экспериментов, результаты, анализ blast radius, скриншоты.
7. Автоматизация реагирования – runbook документ, результаты выполнения, скриншоты уведомлений.
8. Архитектурные паттерны – описание реализованных механизмов, код/конфигурации, результаты тестирования.
9. Нагрузочное тестирование – описание тестов с k6, результаты, анализ узких мест.

10. Специализация (БПЛА) – телеметрия, SLO для полётной миссии, особенности (при наличии).
11. Заключение – оценка достижения SLO, рекомендации по улучшению, итоговые метрики (MTTR, MTBF, Availability).
12. Приложение – листинги кода, конфигураций, скриншоты.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ (ШАБЛОН)

```
"""
Лабораторная работа №18
Сквозной SRE-пайплайн – итоговый проект
"""

import json
import time
from datetime import datetime
from typing import Dict, Any, List
from dataclasses import dataclass, asdict

# =====
# ЧАСТЬ 1. ОПРЕДЕЛЕНИЕ SLO/SLI
# =====

@dataclass
class SLI:
    name: str
    description: str
    measurement: str

@dataclass
class SLO:
    name: str
    sli: SLI
    target_percent: float # 99.9 = 99.9%
    window_days: int = 30

@dataclass
class ErrorBudgetAnalysis:
    slo: SLO
    allowed_downtime_minutes: float
    actual_downtime_minutes: float
    consumed_percent: float
    remaining_percent: float
    status: str
    recommendations: List[str]
```



```

# =====
# ЧАСТЬ 2. МОНИТОРИНГ (ИМИТАЦИЯ)
# =====

@dataclass
class MetricPoint:
    timestamp: float
    value: float

class MetricsCollector:
    def __init__(self):
        self.metrics: Dict[str, List[MetricPoint]] = {}

    def record(self, name: str, value: float):
        if name not in self.metrics:
            self.metrics[name] = []
        self.metrics[name].append(MetricPoint(time.time(), value))

    def get_availability(self, window_seconds: int = 300) -> float:
        # Симуляция расчёта доступности
        return 99.95

# =====
# ЧАСТЬ 3. АНАЛИЗ ИНЦИДЕНТОВ
# =====

class PostmortemAnalyzer:
    @staticmethod
    def five_whys(incident: str) -> List[str]:
        return [
            f"❓ 1. Почему {incident}? → Из-за сбоя БД",
            f"❓ 2. Почему сбой БД? → Из-за переполнения диска",
            f"❓ 3. Почему переполнился диск? → Из-за отсутствия ротации логов",
            f"❓ 4. Почему нет ротации логов? → Неправильная конфигурация",
            f"❓ 5. Почему конфигурация неправильная? → Отсутствие мониторинга",
            f"✅ Коренная причина: Отсутствие мониторинга дискового пространства"
        ]

# =====
# ЧАСТЬ 4. СЧАOS-ЭКСПЕРИМЕНТЫ
# =====

@dataclass
class ChaosExperiment:
    name: str
    type: str
    duration_seconds: int

```

```

        blast_radius: str
        success: bool
        observations: List[str]

# =====
# ЧАСТЬ 5. АВТОМАТИЗАЦИЯ РЕАГИРОВАНИЯ
# =====

@dataclass
class RunbookStep:
    order: int
    action: str
    command: str
    is_auto: bool

@dataclass
class Runbook:
    id: str
    name: str
    trigger_alert: str
    steps: List[RunbookStep]

# =====
# ЧАСТЬ 6. АРХИТЕКТУРНЫЕ ПАТТЕРНЫ
# =====

class CircuitBreaker:
    def __init__(self, failure_threshold: int = 3, timeout_seconds: int = 30):
        self.failure_threshold = failure_threshold
        self.timeout_seconds = timeout_seconds
        self.failure_count = 0
        self.state = "closed"

    def call(self, func, *args, **kwargs):
        if self.state == "open":
            raise Exception("Circuit breaker is OPEN")
        try:
            result = func(*args, **kwargs)
            self.failure_count = max(0, self.failure_count - 1)
            return result
        except Exception as e:
            self.failure_count += 1
            if self.failure_count >= self.failure_threshold:
                self.state = "open"
            raise e

# =====
# ЧАСТЬ 7. НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ (ИМИТАЦИЯ)
# =====

```

```

@dataclass
class LoadTestResult:
    test_type: str
    vus: int
    duration_seconds: int
    p95_latency_ms: float
    error_rate_percent: float
    success: bool

# =====
# ЧАСТЬ 8. СПЕЦИАЛИЗАЦИЯ (БПЛА)
# =====

@dataclass
class DroneTelemetry:
    altitude: float
    battery: float
    gps_quality: int
    state: str

# =====
# ЧАСТЬ 9. ИТОГОВЫЙ ОТЧЁТ
# =====

class SREPipelineReport:
    def __init__(self, project_name: str, student_name: str):
        self.project_name = project_name
        self.student_name = student_name
        self.slos: List[SLO] = []
        self.metrics: Dict[str, Any] = {}
        self.incident_analysis = None
        self.chaos_experiments: List[ChaosExperiment] = []
        self.runbook: Runbook = None
        self.circuit_breaker: CircuitBreaker = None
        self.load_test_results: List[LoadTestResult] = []
        self.drone_telemetry: List[DroneTelemetry] = []

    def to_json(self) -> str:
        return json.dumps(asdict(self), indent=2, ensure_ascii=False,
default=str)

    def to_markdown(self) -> str:
        md = f"""# Отчёт по сквозному SRE-пайплайну

## Проект: {self.project_name}
**Студент:** {self.student_name}
**Дата:** {datetime.now().strftime('%Y-%m-%d')}

```

```

## 1. Определение SLO/SLI
| SLO | Цель | Error Budget |
| :--- | :--- | :--- |
"""
    for slo in self.slos:
        md += f"| {slo.name} | {slo.target_percent}% | {100 -
slo.target_percent}% |\n"

    md += f"""
## 2. Мониторинг
- Prometheus: ☒ настроен
- Grafana: ☒ дашборд создан
- Алертинг: ☒ настроен

## 3. Анализ инцидентов
{self.incident_analysis if self.incident_analysis else "Не выполнен"}

## 4. Chaos-эксперименты
Проведено экспериментов: {len(self.chaos_experiments)}

## 5. Автоматизация реагирования
Runbook: {self.runbook.name if self.runbook else "Не создан"}

## 6. Архитектурные паттерны
Circuit Breaker: {'☒ реализован' if self.circuit_breaker else 'X не
реализован'}

## 7. Нагрузочное тестирование
Проведено тестов: {len(self.load_test_results)}

## 8. Специализация (БПЛА)
Записей телеметрии: {len(self.drone_telemetry)}

## 9. Заключение
### Достигнутые результаты
- Определены SLO с целевой доступностью {self.slos[0].target_percent if self.slos
else 'N/A'}%
- Настроен мониторинг с визуализацией в Grafana
- Проведён анализ инцидентов с выявлением коренных причин
- Выполнены chaos-эксперименты для проверки устойчивости
- Реализована автоматизация реагирования

### Рекомендации по улучшению
1. Увеличить покрытие кода тестами до 85%
2. Внедрить автоматический откат при превышении порога ошибок
3. Расширить chaos-эксперименты на production-среду (canary)

### Итоговые метрики
- **MTTR:** < 5 минут

```

```

- **Availability:** {self.metrics.get('availability', 99.9)}%
- **Error Budget остаток:** {self.metrics.get('error_budget_remaining', 70)}%
"""

    return md

# =====
# ПРИМЕР ИСПОЛЬЗОВАНИЯ
# =====

if __name__ == "__main__":
    report = SREPipelineReport("API Gateway Reliability", "Иванов И.И.")

    # Добавление SLO
    availability_sli = SLI("api_availability", "Доступность API", "successful /
total requests")
    availability_slo = SLO("high_availability", availability_sli, 99.9)
    report.slos.append(availability_slo)

    # Добавление метрик
    report.metrics = {
        "availability": 99.95,
        "error_budget_remaining": 75,
        "mttr_minutes": 4.2,
        "mtbf_hours": 168
    }

    # Анализ инцидентов
    report.incident_analysis = "Проведён анализ инцидента с БД. Коренная причина:
отсутствие мониторинга."

    # Chaos-эксперименты
    report.chaos_experiments.append(
        ChaosExperiment("pod-kill-test", "Pod Kill", 30, "1 pod", True, ["Поды
восстановились за 25 сек"])
    )

    # Runbook
    report.runbook = Runbook(
        id="RB-001",
        name="High Error Rate Recovery",
        trigger_alert="HighErrorRate",
        steps=[
            RunbookStep(1, "Restart deployment", "kubectl rollout restart
deployment/api", True),
            RunbookStep(2, "Verify recovery", "curl -f http://api/health", True)
        ]
    )

```

```

# Circuit Breaker
report.circuit_breaker = CircuitBreaker(failure_threshold=3,
timeout_seconds=30)

# Нагрузочное тестирование
report.load_test_results.append(
    LoadTestResult("Stress", 500, 120, 450, 0.5, True)
)

# Сохранение отчёта
print(report.to_markdown())

with open("sre_pipeline_report.md", "w", encoding="utf-8") as f:
    f.write(report.to_markdown())

with open("sre_pipeline_report.json", "w", encoding="utf-8") as f:
    f.write(report.to_json())

print("\n✅ Отчёт сохранён: sre_pipeline_report.md,
sre_pipeline_report.json")

```

6. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ЗАЩИТЫ ПРОЕКТА

1. Опишите выбранную систему и обоснуйте выбор SLO/SLI.
2. Какие метрики собираются в Prometheus? Как они визуализируются в Grafana?
3. Какой инцидент был проанализирован? Какая коренная причина выявлена?
4. Какие chaos-эксперименты были проведены? Каковы результаты?
5. Как организована автоматизация реагирования?
6. Какие архитектурные паттерны отказоустойчивости реализованы?
7. Каковы результаты нагрузочного тестирования?
8. Были ли достигнуты целевые SLO? Почему?
9. Какие рекомендации по улучшению надежности вы можете предложить?
10. Что бы вы изменили в проекте при наличии большего времени?

7. КРИТЕРИИ ОЦЕНИВАНИЯ ПРОЕКТА

Критерий	Вес,	5 (отлично)	4 (хорошо)	3 (удовлетворит	2 (неудовлетворит

	%			ельно)	ельно)
Полнота реализации этапов пайплайна	20	Реализовано 14+ этапов из 18	Реализовано 11-13 этапов	Реализовано 8-10 этапов	Реализовано менее 8 этапов
Качество конфигурации CI/CD	15	Конфигурация оптимальна, соблюдены best practices, код читаемый	Конфигурация работает, есть небольшие замечания	Конфигурация работает с ошибками, требуется доработка	Конфигурация отсутствует или неработоспособна
Тестирование и покрытие кода	10	Покрытие >80%, все тесты проходят, есть отчёт	Покрытие 70-80%, тесты в основном проходят	Покрытие 60-70%, есть падающие тесты	Покрытие <60%, тесты отсутствуют
Определение SLO/SLI и мониторинг	15	SLO/SLI определены корректно, Prometheus+Grafana настроены, дашборды созданы	SLO/SLI определены, мониторинг настроен с недочётами	SLO/SLI определены частично, мониторинг настроен минимально	SLO/SLI не определены, мониторинг отсутствует
Анализ инцидентов (postmortem, 5 Whys, fishbone)	10	Все методы применены, коренные причины выявлены, отчёт полный	Применены 2 из 3 методов, отчёт с недочётами	Применён 1 метод, отчёт минимальный	Анализ инцидентов не проведён
Chaos-эксперименты	10	Проведено 2+ эксперимента, результаты задокументированы, анализ выполнен	Проведён 1 эксперимент, результаты задокументированы	Проведён 1 эксперимент без анализа	Эксперименты не проводились

Архитектурные паттерны отказоустойчивости	10	Реализовано 2+ паттерна (CB, Retry, Bulkhead), код корректный	Реализован 1 паттерн, есть недочёты	Реализация присутствует, но с ошибками	Паттерны не реализованы
Автоматизация реагирования (runbook/ChatOps/Self-healing)	10	Runbook разработан, автоматизация работает	Runbook разработан, автоматизация частичная	Runbook разработан, автоматизации нет	Runbook не разработан
Качество оформления отчёта и презентации	10	Отчёт полный, структурированный, скриншоты есть, презентация готова	Отчёт с небольшими недочётами	Отчёт минимальный, презентация отсутствует	Отчёт отсутствует

ПОЯСНЕНИЯ К КРИТЕРИЯМ

1. Полнота реализации этапов пайплайна

Этапы	Баллы
14-18 этапов	5
11-13 этапов	4
8-10 этапов	3
менее 8 этапов	2

Базовый перечень этапов (18):

1. Checkout кода
2. Сборка (build)
3. Линтинг / форматирование
4. SAST (статический анализ безопасности)

5. Unit-тесты
6. Покрытие кода (coverage)
7. SCA (анализ зависимостей)
8. Сборка Docker-образа
9. Публикация Docker-образа
10. Развертывание в K8s (dev)
11. Smoke-тесты
12. Интеграционные тесты
13. SITL-тесты (для БПЛА)
14. Canary-развертывание
15. Нагрузочное тестирование (k6)
16. ОТА-подпись (Cosign)
17. Мониторинг (Prometheus+Grafana)
18. Документирование

2. Качество конфигурации CI/CD

Оценка	Описание
5	Использованы шаблоны, кэширование, параллелизация, переменные вынесены, код читаемый
4	Конфигурация работает, но есть дублирование или неоптимальные решения
3	Конфигурация работает с ошибками, требует доработки
2	Конфигурация отсутствует или неработоспособна

3. Тестирование и покрытие кода

Покрытие	Оценка
>80%	5
70-80%	4
60-70%	3
<60%	2

4. Определение SLO/SLI и мониторинг

Оценка	Описание

5	Определены 3+ SLO/SLI, Prometheus собирает метрики, Grafana дашборд содержит 5+ панелей
4	Определены 2 SLO/SLI, мониторинг настроен, дашборд содержит 3-4 панели
3	Определено 1 SLO/SLI, мониторинг настроен минимально, дашборд содержит 1-2 панели
2	SLO/SLI не определены, мониторинг отсутствует

5. Анализ инцидентов

Оценка	Описание
5	Применены все 3 метода (postmortem, 5 Whys, fishbone), коренные причины выявлены, action items сформулированы
4	Применены 2 из 3 методов, action items есть
3	Применён 1 метод, action items отсутствуют
2	Анализ не проведён

6. Chaos-эксперименты

Оценка	Описание
5	2+ эксперимента, результаты задокументированы, анализ blast radius выполнен, отчёт о восстановлении есть
4	1 эксперимент, результаты задокументированы
3	1 эксперимент, анализ отсутствует
2	Эксперименты не проводились

7. Архитектурные паттерны отказоустойчивости

Оценка	Описание
5	2+ паттерна реализованы, код корректен, есть тесты
4	1 паттерн реализован, есть недочёты
3	Реализация присутствует, но с ошибками
2	Паттерны не реализованы

8. Автоматизация реагирования

Оценка	Описание
5	Runbook разработан, автоматизация работает, есть интеграция с ChatOps или self-healing
4	Runbook разработан, автоматизация частичная
3	Runbook разработан, автоматизации нет
2	Runbook не разработан

9. Качество оформления отчёта и презентации

Оценка	Описание
5	Отчёт полный, структурированный, скриншоты есть, презентация готова, ответы на вопросы уверенные
4	Отчёт с небольшими недочётами
3	Отчёт минимальный, презентация отсутствует
2	Отчёт отсутствует

ШКАЛА ПЕРЕВОДА БАЛЛОВ В ОЦЕНКУ

Итоговая оценка вычисляется как средневзвешенная по всем критериям с учётом весов.

Диапазон баллов	Оценка
90-100%	5 (отлично)
75-89%	4 (хорошо)
60-74%	3 (удовлетворительно)
менее 60%	2 (неудовлетворительно)

ПРИМЕР РАСЧЁТА ИТОГОВОЙ ОЦЕНКИ

Критерий	Вес	Оценка студента	Взвешенный балл
Полнота этапов	20%	5	1.0
Качество конфигурации	15%	4	0.6
Тестирование и	10%	5	0.5

покрытие			
SLO/SLI и мониторинг	15%	4	0.6
Анализ инцидентов	10%	5	0.5
Chaos-эксперименты	10%	4	0.4
Архитектурные паттерны	10%	3	0.3
Автоматизация реагирования	10%	4	0.4
Качество отчёта	10%	5	0.5
Итого	100%	4.8	

Итоговая оценка: 4.8 → 5 (отлично)

ЕДИНАЯ ТАБЛИЦА ДЛЯ ВСТАВКИ В ДОКУМЕНТЫ

Ниже представлена компактная версия таблицы, которую можно разместить и в МУ для самостоятельной работы, и в ЛР №18:

Критерий	Вес, %	5 (отлично)	4 (хорошо)	3 (удовл.)	2 (неуд.)
Полнота этапов (14+/11-13/8-10/<8)	20	++	+	-	--
Качество конфигурации CI/CD	15	++	+	-	--
Тестирование и покрытие (>80/70-80/60-70/<60)	10	++	+	-	--
SLO/SLI и мониторинг (3+/2/1/0)	15	++	+	-	--
Анализ инцидентов (3/2/1/0 методов)	10	++	+	-	--
Chaos-эксперименты (2+/1/1 без ан./0)	10	++	+	-	--
Архитектурные паттерны (2+/1/1 с ошиб./0)	10	++	+	-	--
Автоматизация реагирования	10	++	+	-	--
Качество отчёта и презентации	10	++	+	-	--

Расшифровка знаков:

«++» - Критерий выполнен полностью, соответствует заявленным требованиям, нет ошибок и замечаний.

«+» - Критерий выполнен частично или с незначительными замечаниями. Это означает, что студент в целом справился, но есть недочеты (например, не все этапы реализованы, есть мелкие ошибки в конфигурации, оформление не идеальное). В контексте оценки «хорошо» (4) этот знак указывает на то, что работа почти идеальна, но до «отлично» немного не дотягивает.

«-» - Критерий не выполнен или выполнен на очень низком, неприемлемом уровне. Присутствуют грубые ошибки, ключевые требования не соблюдены. В контексте оценки «удовлетворительно» (3) этот знак показывает, что есть серьезные проблемы, но не настолько критичные, чтобы ставить «неуд».

«--» - Для оценки «неуд» (2), критерии не выполнены полностью.

Шкала перевода баллов в оценку для проекта

Процент баллов	Оценка
90-100%	5 (отлично)
75-89%	4 (хорошо)
60-74%	3 (удовлетворительно)
<60%	2 (неудовлетворительно)

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
«Надежность и устойчивость систем»
для студентов направления подготовки
09.04.04 Программная инженерия Направленность (профиль)
«Разработка, мониторинг и управление жизненным циклом инженерных
систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	297
1. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ	298
2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА.....	301
3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	303
4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ.....	305
5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ.....	306

ВВЕДЕНИЕ

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

Дисциплина «Надежность и устойчивость систем» является одной из ключевых дисциплин вариативной части образовательной программы и направлена на формирование у магистрантов системных знаний и практических навыков в области Site Reliability Engineering (SRE), управления надежностью распределенных систем, анализа инцидентов, Chaos Engineering и автоматизации процессов реагирования.

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 «Надежность и устойчивость систем»;

Положением об организации образовательного процесса на основе рейтинговой системы оценки знаний студентов в ФГАОУ ВО «СКФУ»;

Программой развития и воспитания Северо-Кавказского федерального университета.

Целью самостоятельной работы является закрепление и углубление теоретических знаний, полученных на лекционных занятиях, а также формирование практических навыков применения методов SRE, анализа инцидентов, Chaos Engineering и управления надежностью программных и

программно-аппаратных систем в рамках выполнения индивидуального сквозного проекта.

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.
2. Подготовиться к выполнению лабораторных работ (№1-17) и осмыслению их результатов.
3. Разработать и реализовать сквозной проект по созданию SRE-пайплайна для выбранной системы (Лабораторная работа №18).
4. Сформировать комплексный отчёт, включающий определение SLO/SLI, настройку мониторинга, анализ инцидентов, проведение chaos-экспериментов и разработку корректирующих мер.
5. Подготовиться к промежуточной аттестации (экзамену) по дисциплине.
6. Развить навыки самостоятельного поиска и анализа научно-технической информации в области управления надёжностью.

Настоящие методические указания призваны помочь студентам в организации самостоятельной работы по дисциплине «Надёжность и устойчивость систем». Систематическое выполнение всех видов самостоятельной работы, своевременная подготовка к лабораторным занятиям и выполнение сквозного 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 применения методов SRE (Site Reliability Engineering), анализа инцидентов, Chaos Engineering и управления надежностью программных и программно-аппаратных систем в рамках выполнения индивидуального сквозного проекта.

1.2. Общая характеристика сквозного проекта

Сквозной проект выполняется студентом индивидуально в течение всего семестра параллельно с изучением дисциплины и выполнением лабораторных работ (№1-17). Защита проекта проводится на 18-м лабораторном занятии.

Тема проекта: «Разработка и внедрение SRE-пайплайна для обеспечения надежности программной системы / автономного устройства»

Результат проекта: Комплексный отчет, включающий:

- Определение SLO/SLI для выбранной системы
- Настройку мониторинга и визуализации
- Анализ инцидентов (postmortem, 5 Whys, fishbone)
- Проведение chaos-экспериментов
- Автоматизацию реагирования (runbooks, ChatOps)
- Оценку эффективности и рекомендации

1.3. Формируемые компетенции

Компет енция	Индикаторы
ПК-4	ИД-1 ПК-4. Определяет ключевые показатели качества обслуживания (SLI) для различных компонентов сервисов и автономных систем (доступность, задержка, точность, целостность данных). ИД-2 ПК-4. Разрабатывает целевые уровни обслуживания

	(SLO) на основе бизнес-требований и пользовательских ожиданий, согласовывая их с заинтересованными сторонами. ИД-3 ПК-4. Рассчитывает бюджет ошибок (Error Budget) как количественную меру допустимого снижения надежности и использует его для балансировки приоритетов разработки и эксплуатации. ИД-4 ПК-4. Настраивает системы сбора и мониторинга метрик (SLI) с построением дашбордов для визуализации текущего состояния надежности и оповещений при приближении к границам SLO. ИД-5 ПК-4. Анализирует динамику соблюдения SLO и потребления бюджета ошибок для выявления узких мест и прогнозирования потенциальных сбоев.
ПК-5	ИД-1 ПК-5. Определяет показатели устойчивого состояния (steady state) системы и формулирует гипотезы о ее поведении при различных типах отказов. ИД-2 ПК-5. Проектирует эксперименты по внедрению сбоев (инфраструктурных, сетевых, нагрузочных) с контролируемой зоной поражения (blast radius). ИД-3 ПК-5. Проводит эксперименты по тестированию устойчивости с использованием инструментов chaos engineering в тестовых и production-средах. ИД-4 ПК-5. Анализирует результаты экспериментов, сопоставляя фактические метрики с ожидаемыми, и выявляет скрытые уязвимости и слабые места архитектуры. ИД-5 ПК-5. Разрабатывает и внедряет корректирующие меры по итогам экспериментов (настройка таймаутов, ретраев, circuit breaker, оптимизация архитектуры).
ПК-6	ИД-1 ПК-6. Анализирует хронику событий и технические метрики инцидента для восстановления полной и достоверной картины происшествия. ИД-2 ПК-6. Применяет методы системного анализа (5 почему, диаграмма Исикавы, анализ графов) для выявления коренных (системных) причин сбоев. ИД-3 ПК-6. Разрабатывает комплекс превентивных мер и

	технических решений, направленных на устранение выявленных коренных причин и предотвращение повторения инцидентов. ИД-4 ПК-6. Автоматизирует процессы обнаружения и реагирования на инциденты с использованием скриптов и средств оркестрации для сокращения ручного труда и времени простоя. ИД-5 ПК-6. Документирует результаты анализа инцидентов в формате blameless postmortem и формирует базу знаний для повышения устойчивости операций.

1.4. Трудоёмкость самостоятельной работы

Вид работы	Часы
Изучение теоретического материала	20
Подготовка к лабораторным работам	30
Выполнение сквозного проекта	30
Подготовка к экзамену	10
Итого	90

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

2.1. Темы для самостоятельного изучения

Тема	Литература	Часы
Основы SRE и управление надежностью	[1, гл. 1-3]	2
Метрики SLO/SLI/Error Budget	[1, гл. 4-5]	2
Анализ инцидентов (5 Whys, fishbone)	[2, гл. 7]	2
Chaos Engineering	[3, гл. 1-	2

		3]	
	Observability (метрики, логи, трейсы)	[1, гл. 6]	2
	GitOps и Argo CD	[4]	2
	ОТА-обновления для встроенных систем	[5]	2
	Нагрузочное тестирование (k6)	[6]	2
	Архитектурные паттерны отказоустойчивости	[7]	2
0	Self-healing и автоматизация реагирования	[8]	2

2.2. Рекомендуемая литература

1. Beyer B., Jones C., Petoff J., Murphy N.R. Site Reliability Engineering: How Google Runs Production Systems. - O'Reilly Media, 2016. (русский перевод доступен)

2. Rosenthal C., Jones N. Chaos Engineering: Building Confidence in System Behavior through Experiments. - O'Reilly Media, 2020.

3. Majors C., Fong-Jones L., Miranda G. Observability Engineering: Achieving Production Excellence. - O'Reilly Media, 2022.

4. Argo CD Documentation - <https://argo-cd.readthedocs.io/>

5. PX4 OTA Documentation - <https://docs.px4.io/main/en/advanced/ota.html>

6. k6 Documentation - <https://k6.io/docs/>

7. Microsoft. Резильентные архитектуры в Azure. - Документация Microsoft (рус.)

8. Kubernetes Documentation - <https://kubernetes.io/ru/docs/>

2.3. Рекомендации по работе с литературой

- При изучении теоретического материала рекомендуется вести конспект в виде структурированных заметок.
- Для каждой темы выделить ключевые термины и составить глоссарий.
- Выполнять практические примеры из литературы в тестовой среде.
- Использовать официальную документацию как основной источник для выполнения лабораторных работ.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

3.1. Структура лабораторного практикума

Лабораторный практикум состоит из 18 работ, объединённых в 6 тематических блоков:

Блок	№ ЛР	Тема
Блок 1. Основы метрик и измерений	1 -4	MTBF/MTTR, SLO/SLI, Error Budget, Prometheus+Grafana
Блок 2. Анализ инцидентов	5 -7	Postmortem, 5 Whys, fishbone, документирование
Блок 3. Эксперименты и Chaos Engineering	8 -11	Проектирование, canary, ChaosMesh, Game Day
Блок 4. Автоматизация реагирования	1 2-14	Runbooks, ChatOps, Self-healing
Блок 5. Архитектурные паттерны	1 5-16	Circuit Breaker, Retry, нагрузочное тестирование
Блок 6. Специализация и проект	1 7-18	Надежность БПЛА, сквозной SRE-пайплайн

3.2. Порядок выполнения лабораторных работ

1. Ознакомиться с теоретическим обоснованием работы.
2. Выбрать вариант задания в соответствии с номером, выданным преподавателем.
3. Разработать решение (скрипты, конфигурации, отчёты).
4. Выполнить проверку в тестовой среде.
5. Оформить отчёт по установленной форме.
6. Защитить работу перед преподавателем.

3.3. Рекомендации по подготовке к лабораторным работам

- Изучить соответствующий раздел теоретического материала перед выполнением работы.
- Подготовить тестовую среду (локальный Kubernetes, Python окружение, установленные инструменты).
- Ознакомиться с примерами выполнения из методических указаний.
- При возникновении вопросов — консультироваться с преподавателем.

3.4. Рекомендации по выполнению сквозного проекта (ЛР №18)

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

Требования к проекту:

- Выбрать систему для анализа (веб-сервис, микросервисная архитектура, БПЛА).
- Определить SLO/SLI (минимум 3 метрики).
- Настроить мониторинг (Prometheus + Grafana).
- Провести анализ инцидента (5 Whys, fishbone).
- Выполнить chaos-эксперименты (минимум 2 типа).

- Автоматизировать реагирование (runbook/ChatOps).
- Представить итоговый отчёт.

4. ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

1. Что такое SRE (Site Reliability Engineering) и чем он отличается от DevOps?
2. Как рассчитываются метрики MTBF, MTTR, Availability?
3. Что такое SLO, SLI и Error Budget? Как они связаны?
4. Какие принципы blameless postmortem вы знаете?
5. Как работает метод 5 Whys? Каковы его ограничения?
6. Какие категории используются в диаграмме Исикавы для IT-инцидентов?
7. Что такое Chaos Engineering и какие принципы лежат в его основе?
8. Какие инструменты для Chaos Engineering вы знаете?
9. Что такое blast radius и как его контролировать?
10. Какие типы chaos-экспериментов существуют?
11. Что такое observability и какие три столпа она включает?
12. Как работает Prometheus? В чем отличие pull-модели от push-модели?
13. Что такое GitOps? Какие преимущества даёт подход?
14. Как работает Argo CD?
15. Что такое OTA-обновления и какие требования к их безопасности?
16. Какие архитектурные паттерны отказоустойчивости вы знаете?
17. Чем отличается Circuit Breaker от Retry?
18. Что такое canary deployment и чем он отличается от blue-green?
19. Как проводится нагрузочное тестирование с k6?

20. Какие метрики DORA используются для оценки эффективности DevOps?

5. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

5.1. Критерии оценивания лабораторных работ

Оценка	Критерии
5 (отлично)	Работа выполнена в полном объеме, пайплайн работает без ошибок, студент демонстрирует понимание всех этапов, отвечает на дополнительные вопросы, отчет оформлен по требованиям
4 (хорошо)	Работа выполнена полностью, но есть небольшие замечания (неоптимальная конфигурация, неполный отчет). Студент отвечает на большинство вопросов
3 (удовлетворительно)	Работа выполнена с помощью преподавателя, есть ошибки, которые студент исправляет с наводящими вопросами. Отчет оформлен минимально
2 (неудовлетворительно)	Работа не выполнена или выполнена неверно, студент не понимает принципов, не может объяснить свои действия.

5.2. Критерии оценивания сквозного проекта (ЛР №18)

Критерий	ес, %	5 (отлично)	4 (хорошо)	3 (удовлетворительно)	2 (неудовлетворительно)
Полнота реализации этапов пайплайна	0	Реализовано 14+ этапов из 18	Реализовано 11-13 этапов	Реализовано 8-10 этапов	 Реализовано менее 8 этапов
Качество		Конфиг	Конфиг	Конфиг	Конфигу

конфигурации CI/CD	5	урация оптимальна, соблюдены best practices, код читаемый	урация работает, есть небольшие замечания	урация работает с ошибками, требуется доработка	рация отсутствует или неработоспособна
Тестирование и покрытие кода	0	Покрытие >80%, все тесты проходят, есть отчёт	Покрытие 70-80%, тесты в основном проходят	Покрытие 60-70%, есть падающие тесты	Покрытие <60%, тесты отсутствуют
Определение SLO/SLI и мониторинг	5	SLO/SLI определены корректно, Prometheus+Grafana настроены, дашборды созданы	SLO/SLI определены, мониторинг настроен с недочётами	SLO/SLI определены частично, мониторинг настроен минимально	SLO/SLI не определены, мониторинг отсутствует
Анализ инцидентов (postmortem, 5 Whys, fishbone)	0	Все методы применены, коренные причины выявлены, отчёт полный	Применены 2 из 3 методов, отчёт с недочётами	Применён 1 метод, отчёт минимальный	Анализ инцидентов не проведён
Chaos-эксперименты	0	Проведено 2+ эксперимента, результаты задокументированы, анализ выполнен	Проведён 1 эксперимент, результаты задокументированы	Проведён 1 эксперимент без анализа	Эксперименты не проводились
Архитектурные паттерны отказоустойчив	0	Реализовано 2+ паттерна (CB,	Реализован 1 паттерн, есть	Реализация присутствует,	Паттерны не реализованы

ости		Retry, Bulkhead), код корректный	недочёты	но с ошибками	
Автоматизация реагирования (runbook/ChatOps/Self-healing)	0	Runbook разработан, автоматизация работает	Runbook разработан, автоматизация частичная	Runbook разработан, автоматизации нет	Runbook не разработан
Качество оформления отчёта и презентации	0	Отчёт полный, структурированный, скриншоты есть, презентация готова	Отчёт с небольшими недочётами	Отчёт минимальный, презентация отсутствует	Отчёт отсутствует

ПОЯСНЕНИЯ К КРИТЕРИЯМ

1. Полнота реализации этапов пайплайна

Этапы	Баллы
14-18 этапов	5
11-13 этапов	4
8-10 этапов	3
менее 8 этапов	2

Базовый перечень этапов (18):

1. Checkout кода
2. Сборка (build)
3. Линтинг / форматирование
4. SAST (статический анализ безопасности)
5. Unit-тесты
6. Покрытие кода (coverage)

7. SCA (анализ зависимостей)
8. Сборка Docker-образа
9. Публикация Docker-образа
10. Развертывание в K8s (dev)
11. Smoke-тесты
12. Интеграционные тесты
13. SITL-тесты (для БПЛА)
14. Canary-развертывание
15. Нагрузочное тестирование (k6)
16. OTA-подпись (Cosign)
17. Мониторинг (Prometheus+Grafana)
18. Документирование

2. Качество конфигурации CI/CD

Оценка	Описание
5	Использованы шаблоны, кэширование, параллелизация, переменные вынесены, код читаемый
4	Конфигурация работает, но есть дублирование или неоптимальные решения
3	Конфигурация работает с ошибками, требует доработки
2	Конфигурация отсутствует или неработоспособна

3. Тестирование и покрытие кода

Покрытие	Оценка
>80%	5
70-80%	4
60-70%	3
<60%	2

4. Определение SLO/SLI и мониторинг

Оценк а	Описание
5	Определены 3+ SLO/SLI, Prometheus собирает метрики, Grafana дашборд содержит 5+ панелей

4	Определены 2 SLO/SLI, мониторинг настроен, дашборд содержит 3-4 панели
3	Определено 1 SLO/SLI, мониторинг настроен минимально, дашборд содержит 1-2 панели
2	SLO/SLI не определены, мониторинг отсутствует

5. Анализ инцидентов

Оценка	Описание
5	Применены все 3 метода (postmortem, 5 Whys, fishbone), коренные причины выявлены, action items сформулированы
4	Применены 2 из 3 методов, action items есть
3	Применён 1 метод, action items отсутствуют
2	Анализ не проведён

6. Chaos-эксперименты

Оценка	Описание
5	2+ эксперимента, результаты задокументированы, анализ blast radius выполнен, отчёт о восстановлении есть
4	1 эксперимент, результаты задокументированы
3	1 эксперимент, анализ отсутствует
2	Эксперименты не проводились

7. Архитектурные паттерны отказоустойчивости

Оценка	Описание
5	2+ паттерна реализованы, код корректен, есть тесты
4	1 паттерн реализован, есть недочёты
3	Реализация присутствует, но с ошибками
2	Паттерны не реализованы

8. Автоматизация реагирования

Оценка	Описание

а	
5	Runbook разработан, автоматизация работает, есть интеграция с ChatOps или self-healing
4	Runbook разработан, автоматизация частичная
3	Runbook разработан, автоматизации нет
2	Runbook не разработан

9. Качество оформления отчёта и презентации

Оценка	Описание
5	Отчёт полный, структурированный, скриншоты есть, презентация готова, ответы на вопросы уверенные
4	Отчёт с небольшими недочётами
3	Отчёт минимальный, презентация отсутствует
2	Отчёт отсутствует

ШКАЛА ПЕРЕВОДА БАЛЛОВ В ОЦЕНКУ

Итоговая оценка вычисляется как средневзвешенная по всем критериям с учётом весов.

Диапазон баллов	Оценка
90-100%	5 (отлично)
75-89%	4 (хорошо)
60-74%	3 (удовлетворительно)
менее 60%	2 (неудовлетворительно)

ПРИМЕР РАСЧЁТА ИТОГОВОЙ ОЦЕНКИ

Критерий	Вес	Оценка студента	Взвешенный балл
Полнота этапов	20%	5	1.0
Качество конфигурации	15%	4	0.6

Тестирование и покрытие	10%	5	0.5
SLO/SLI и мониторинг	15%	4	0.6
Анализ инцидентов	10%	5	0.5
Chaos-эксперименты	10%	4	0.4
Архитектурные паттерны	10%	3	0.3
Автоматизация реагирования	10%	4	0.4
Качество отчёта	10%	5	0.5
Итого	100%	4.8	

Итоговая оценка: 4.8 → 5 (отлично)

ЕДИНАЯ ТАБЛИЦА ДЛЯ ВСТАВКИ В ДОКУМЕНТЫ

Ниже представлена компактная версия таблицы, которую можно разместить и в МУ для самостоятельной работы, и в ЛР №18:

Критерий	ес, %	5 (отлично)	4 (хорошо)	3 (удовл.)	2 (неуд.)
Полнота этапов (14+/11-13/8-10/<8)	0	++	+	-	-
Качество конфигурации CI/CD	5	++	+	-	-
Тестирование и покрытие (>80/70-80/60-70/<60)	0	++	+	-	-
SLO/SLI и мониторинг (3+/2/1/0)	5	++	+	-	-
Анализ инцидентов (3/2/1/0 методов)	0	++	+	-	-

Chaos-эксперименты (2+/1/1 без ан./0)	0	++	+	-	-
Архитектурные паттерны (2+/1/1 с ошиб./0)	0	++	+	-	-
Автоматизация реагирования	0	++	+	-	-
Качество отчёта и презентации	0	++	+	-	-

Расшифровка знаков:

«++» - Критерий выполнен полностью, соответствует заявленным требованиям, нет ошибок и замечаний.

«+» - Критерий выполнен частично или с незначительными замечаниями. Это означает, что студент в целом справился, но есть недочеты (например, не все этапы реализованы, есть мелкие ошибки в конфигурации, оформление не идеальное). В контексте оценки «хорошо» (4) этот знак указывает на то, что работа почти идеальна, но до «отлично» немного не дотягивает.

«-» - Критерий не выполнен или выполнен на очень низком, неприемлемом уровне. Присутствуют грубые ошибки, ключевые требования не соблюдены. В контексте оценки «удовлетворительно» (3) этот знак показывает, что есть серьезные проблемы, но не настолько критичные, чтобы ставить «неуд».

«--» - Для оценки «неуд» (2), критерии не выполнены полностью.

Шкала перевода баллов в оценку для проекта

Процент баллов	Оценка
90-100%	5 (отлично)
75-89%	4 (хорошо)

60-74%	3 (удовлетворительно)
<60%	2 (неудовлетворительно)