

**Министерство науки высшего образования Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**  
**ОБРАЗОВАНИЯ**  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**  
**(Университет ИТМО)**  
**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА**  
**«Разработка простого IoT-сервиса»**  
По курсу «Многоуровневая организация программных систем»

Выполнил студент группы Р4116:

Симовин Кирилл Константинович

Преподаватель:

Перл Иван Андреевич

Санкт-Петербург

2025

Задание .....	3
Ход выполнения работы .....	5
1.1 Описание архитектуры проекта.....	5
1.1.1 Data Simulator .....	5
1.1.2 IoT-контроллер .....	6
1.1.3 Rule Engine .....	6
1.1.4 База данных.....	6
1.1.5 Брокер сообщений.....	6
1.1.6 Сбор метрик.....	7
1.1.7 Агрегация логов .....	7
1.1.8 Визуализация собранных данных .....	7
1.2 Описание собираемых метрик .....	7
1.3 Демонстрация графических интерфейсов .....	7
1.3.1 Mongo Express.....	7
1.3.2 Kafka .....	10
1.3.3 Prometheus .....	11
1.3.4 Grafana.....	11
1.4 Описание используемых Design Principles .....	12
Заключение .....	14
Список использованных источников .....	15

## Задание

Цель: отработка принципов и подходов к разработке современных многоуровневых сервисов при решении практической задачи.

Задача: разработать простое IoT-решение и показать применение основных принципов разработки, которые обсуждались на лекции.

Примерная структура решения, которое необходимо разработать:

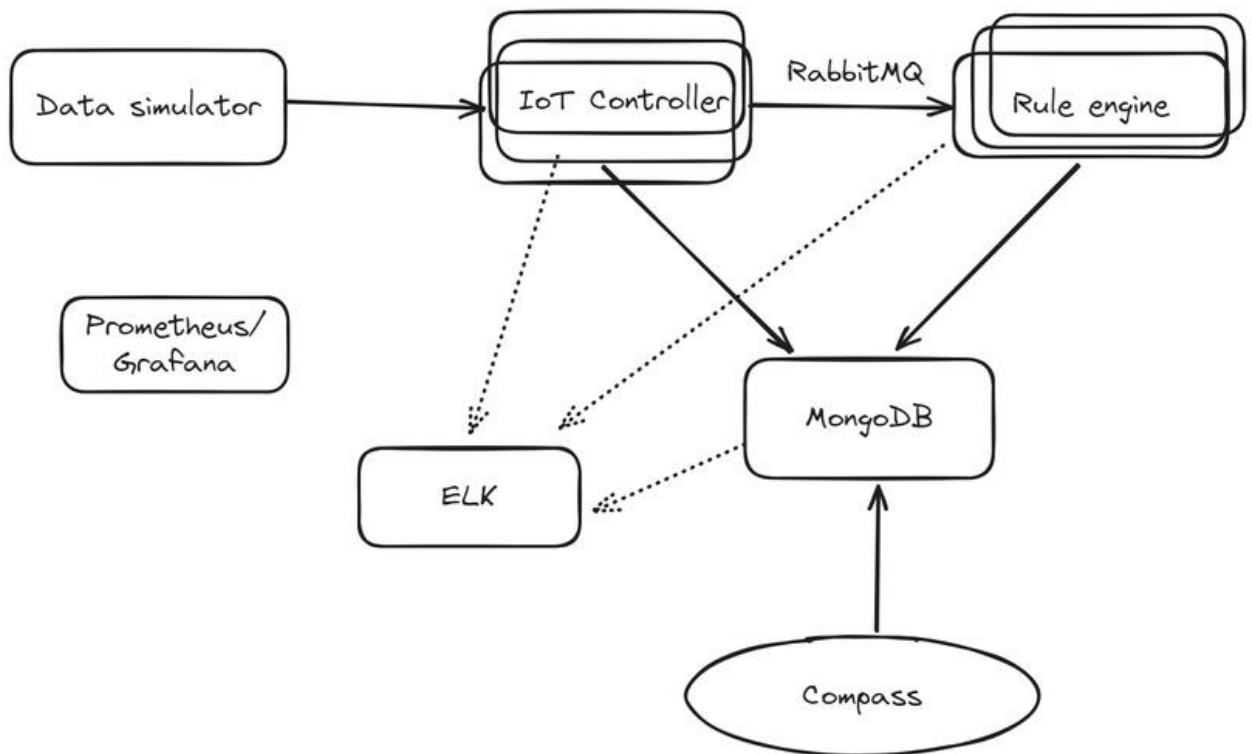


Рисунок 1 – Примерная структура решения

На рисунке выше приведены следующие компоненты:

1. IoT-контроллер – сервис, который принимает входные пакеты с данными от «устройств», подключенных к системе. Принимаемые пакеты валидируются и сохраняются в базу данных MongoDB.

2. Rule engine – простой обработчик правил. Должен уметь обрабатывать мгновенные правила, т.е. основанные на конкретном пакете, и длящиеся, основанные на нескольких пакетах. Пакеты для обработки компонент получает от IoT-контроллера через очередь сообщений.

Типы правил:

а) мгновенное правило – значение поля А от устройства 42 больше 5.

б) длящееся правило – значение поля А от устройства 42 больше 5 на протяжении 10 пакетов от этого устройства.

3. Data simulator – простой генератор данных для разрабатываемого IoT-решения. Позволяет указать количество симулируемых устройств и частоту сообщений, которые генерируются каждым из них. Например, 100 устройств и 1 сообщение в секунду с устройства.

## Ход выполнения работы

### 1.1 Описание архитектуры проекта

Архитектура проекта состоит из следующих компонентов:

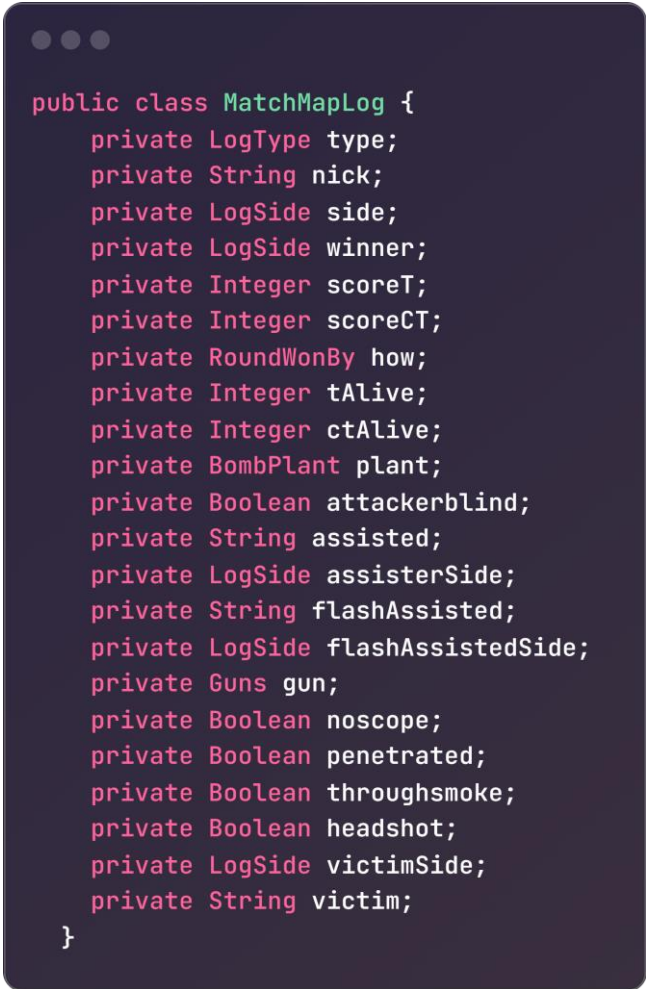
#### 1.1.1 Data Simulator

Реализован на языке программирования Java с использованием фреймворка Micronaut.

Данный сервис генерирует каждые пять секунд случайный лог матча Counter-Strike с использованием библиотеки Instancio.

Затем сгенерированный лог отправляется IoT-контроллеру на эндпоинт `http://localhost:8090/match/{id}`. Отправка осуществляется посредством использования библиотеки `micronaut-http-client`.

На рисунке 2 представлено описание структуры отправляемого лога.



```
public class MatchMapLog {
    private LogType type;
    private String nick;
    private LogSide side;
    private LogSide winner;
    private Integer scoreT;
    private Integer scoreCT;
    private RoundWonBy how;
    private Integer tAlive;
    private Integer ctAlive;
    private BombPlant plant;
    private Boolean attackerblind;
    private String assisted;
    private LogSide assisterSide;
    private String flashAssisted;
    private LogSide flashAssistedSide;
    private Guns gun;
    private Boolean noscope;
    private Boolean penetrated;
    private Boolean throughsmoke;
    private Boolean headshot;
    private LogSide victimSide;
    private String victim;
}
```

Рисунок 2 – Описание структуры отправляемого лога

### **1.1.2 IoT-контроллер**

Также реализован с использованием языка программирования Java и фреймворка Micronaut.

По эндпоинту `/match/{id}` принимает очередной лог с указанием `id` отправителя.

Принятый лог проверяется на соответствие типа – если лог описывает начало раунда, то он пропускается и обновляется счетчик пропущенных логов. Счетчик реализован через библиотеку `micronaut-micrometer`.

Валидированный лог сохраняется в базу данных MongoDB в коллекцию `logs`, а затем отправляется в Rule Engine с использованием Kafka. Взаимодействие с Kafka происходит посредством библиотеки `micronaut-kafka`.

### **1.1.3 Rule Engine**

Подобно прошлым сервисам, реализован на Java с использованием Micronaut.

Полученный от IoT-контроллера лог из топика `log` сохраняется в MongoDB в коллекцию `instant` с указанием `id` устройства, от которого получены данные.

Если от устройства на протяжении десяти запросов подряд приходят логи и значение поля `tAlive` больше 200, то лог охрывается в коллекцию `ongoing` как длящееся правило.

### **1.1.4 База данных**

В качестве базы данных используется NoSQL БД – MongoDB. В качестве графического интерфейса для доступа к MongoDB используется Mongo Express.

### **1.1.5 Брокер сообщений**

В качестве брокера сообщений был выбран Kafka, ввиду нехватки компетенций по работе с RabbitMQ в связка с Micronaut. Взаимодействие с Kafka происходит посредством использования библиотеки `micronaut-kafka`.

В качестве графического интерфейса для доступа к Kafka используется Kafka UI.

### **1.1.6 Сбор метрик**

Для сбора метрик используется Prometheus. Данный функционал предоставляется библиотекой `micronaut-micrometer-registry-prometheus`.

### **1.1.7 Агрегация логов**

В качестве сервиса по сбору логов был выбран Loki. Отправка логов в сервис осуществляется с использованием библиотеки `com.github.loki4j.loki-logback-appender`.

### **1.1.8 Визуализация собранных данных**

В качестве сервиса визуализации собранных метрик и логов используется Grafana.

## **1.2 Описание собираемых метрик**

1. Data Simulator собирает количество сгенерированных логов.
2. IoT-контроллер собирает количество:
  - а) одобренных логов;
  - б) отклоненных логов.
3. Rule Engine собирает количество сработавших:
  - а) мгновенных правил;
  - б) длящихся правил.

## **1.3 Демонстрация графических интерфейсов**

### **1.3.1 Mongo Express**

На рисунках 3-6 представлены скриншоты коллекций в Mongo Express.

Mongo Express

Database: mops

Viewing Database: mops

Collections

Collection Name

+ Create collection

View

Export

[JSON]

Import

instant

Del

View

Export

[JSON]

Import

logs

Del

View

Export

[JSON]

Import

ongoing

Del

Database Stats

Collections (incl. system.namespaces)	3
Data Size	78.6 KB
Storage Size	168 KB
Avg Obj Size #	33.7 Bytes
Objects #	2333
Indexes #	3
Index Size	184 KB

Рисунок 3 – Список всех коллекций

Mongo Express

Database: mops

Collection: logs

Viewing Collection: logs

New Document

New Index

Simple

Advanced

Key

Value

String

Find

Delete all 16 documents retrieved

1

2

>

>>

_id	type	nick	side	winner	how	tAlive	ctAlive	plant	attackerblind	assisted	assist
<div><div></div><div></div></div> <div>6780ea1f129c635ed1fed9e1</div>	KILL	AQXHHQTVQM	T	T	TIME_IS_UP	6128	9523	A	false	ANTRH	T
<div><div></div><div></div></div> <div>6780ea24129c635ed1fed9e2</div>	KILL	KJBX	CT	T	TIME_IS_UP	4578	8	A	true	VNZJB	T
<div><div></div><div></div></div> <div>6780ea29129c635ed1fed9e3</div>	ROUND_END	LAFQ	CT	CT	BOMB_DEFUSED	6883	996	A	false	TFEMV	CT
<div><div></div><div></div></div> <div>6780ea2f129c635ed1fed9e4</div>	BOMB_PLANTED	HSQDCXJSG	T	CT	ENEMY_ELIMINATED	7028	6193	B	true	KQAVFBPSIX	CT
<div><div></div><div></div></div> <div>6780ea34129c635ed1fed9e5</div>	BOMB_DEFUSED	ZKANTKEBFZ	T	CT	TIME_IS_UP	7897	250	A	true	TAVYCM	CT

Рисунок 4 – Коллекция валидированных логов



Mongo Express Database: mops Collection: instant

## Viewing Collection: instant

[New Document](#) [New Index](#)

[Simple](#) [Advanced](#)

Key Value String [Find](#)

Delete all 23 documents retrieved

1 2 > >>

_id	id	matchMapLog
<a href="#">Link</a> <a href="#">Delete</a> 6780ea85088b3417bf4cf2ab	75	<pre>{   "type": "LOGOUT",   "nick": "MEBE",   "side": "CT",   "winner": "CT",   "how": "BOMB_DEFUSED",   "tAlive": 1986,   "ctAlive": 6250,   "plant": "A",   "attackerblind": false,   "assisted": "LDQM",   "assisterSide": "CT",   "flashAssisted": "ZVTNTVPS",   "flashAssistedSide": "CT",   "gun": "SAWEDOFF",   "noscope": false,   "penetrated": false,   "throughsmoke": true,   "victimSide": "T",   "victim": "CKCE",   "..." }</pre>

Рисунок 5 – Коллекция мгновенных правил

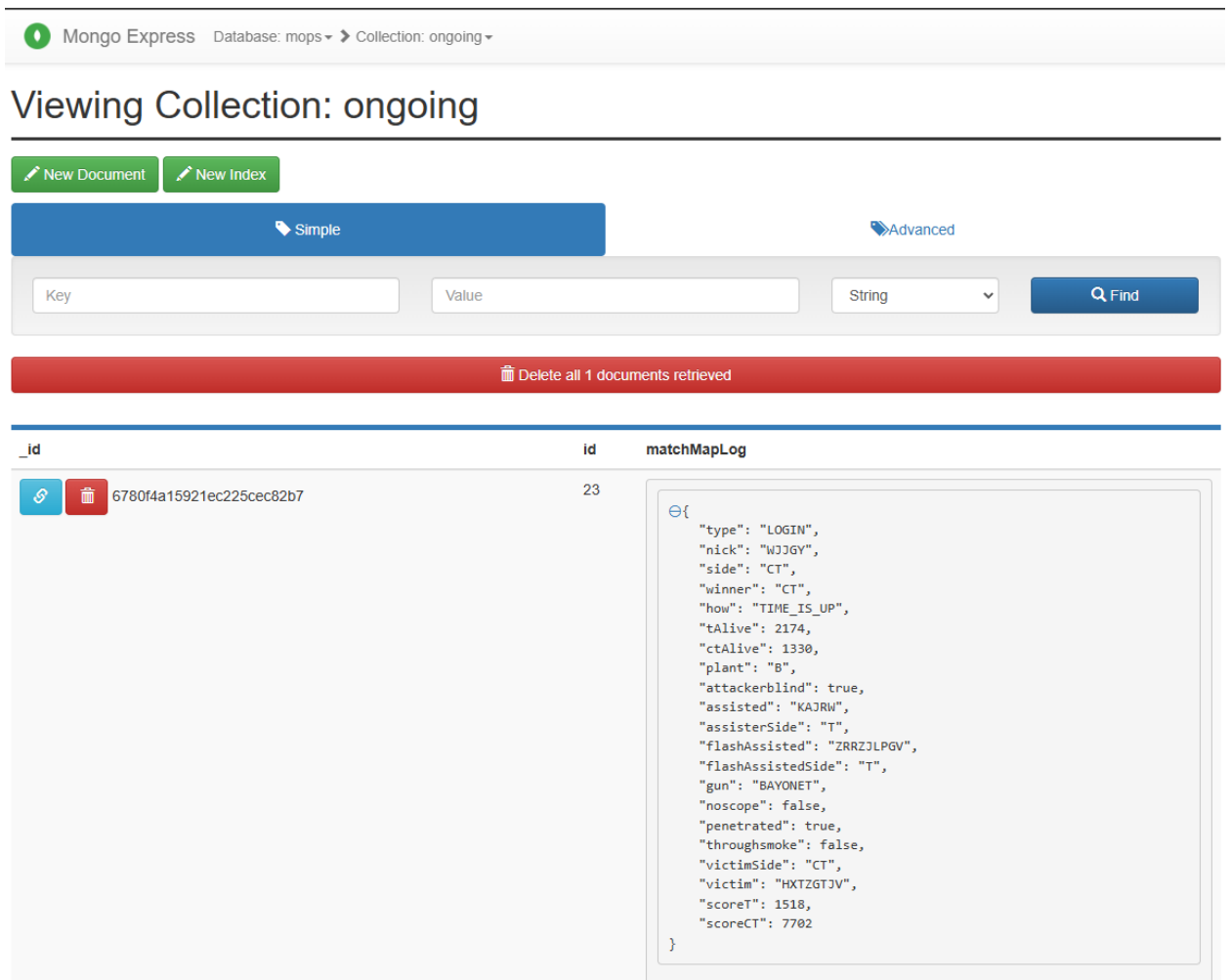


Рисунок 6 – Коллекция длящихся правил

### 1.3.2 Kafka

На рисунках 7-9 представлены скриншоты из Kafka UI.

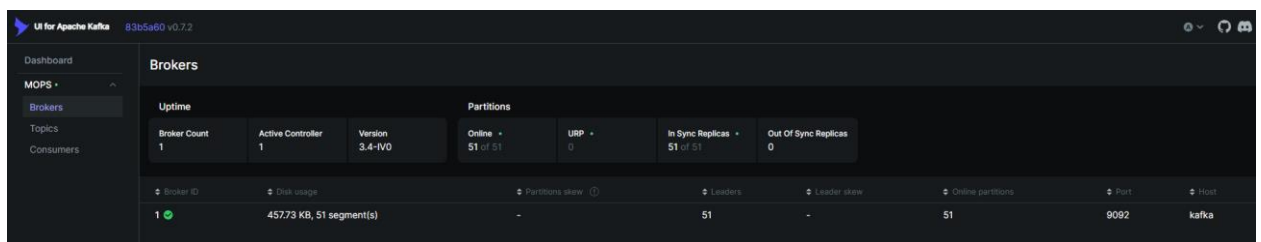


Рисунок 7 – Брокеры Kafka

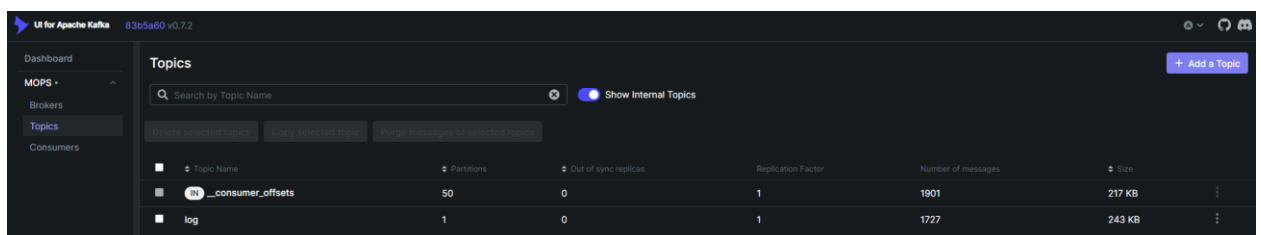


Рисунок 8 – Топики Kafka

Group ID	Num Of Members	Num Of Topics	Consumer Lag	Coordinator	State
rule-engine	1	1	N/A	1	STABLE

Рисунок 9 – Потребители Kafka

### 1.3.3 Prometheus

На рисунках 10-11 представлены скриншоты Prometheus.

Target	Labels	Last scrape	State
http://host.docker.internal:8090/prometheus	instance="host.docker.internal:8090" job="mops"	7.34s ago 41ms	UP
http://host.docker.internal:8091/prometheus	instance="host.docker.internal:8091" job="mops"	1.44s ago 63ms	UP
http://host.docker.internal:8092/prometheus	instance="host.docker.internal:8092" job="mops"	13.34s ago 65ms	UP
http://localhost:9090/metrics	instance="localhost:9090" job="prometheus"	7.34s ago 10ms	UP

Рисунок 10 – Источники метрик

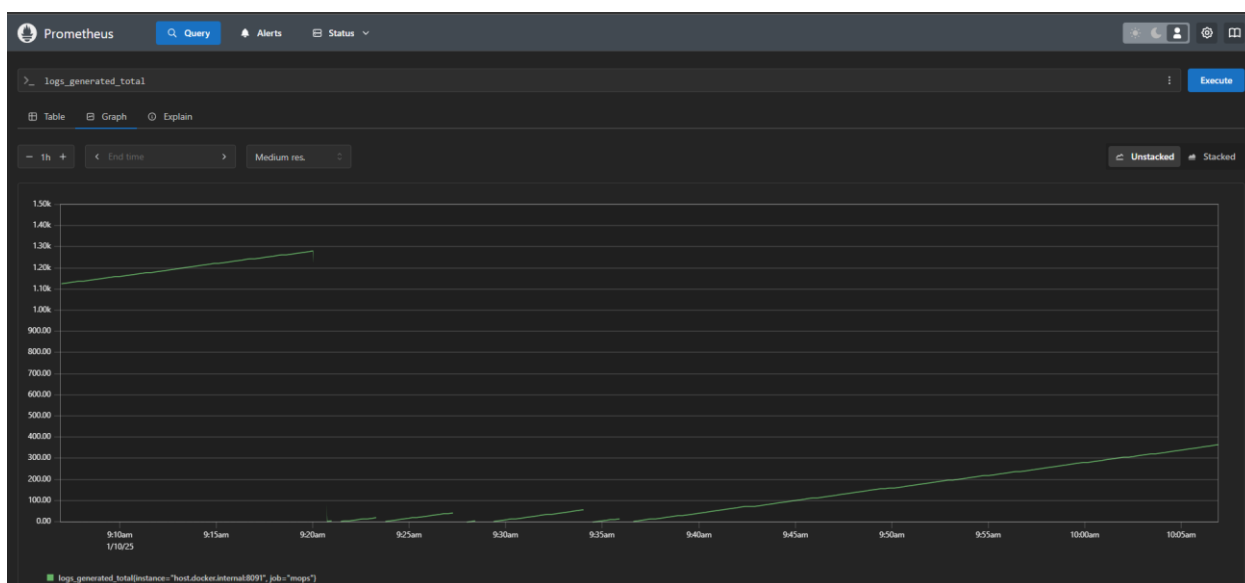


Рисунок 11 – Количество сгенерированных логов

### 1.3.4 Grafana

На рисунках 12-13 представлены скриншоты Grafana.

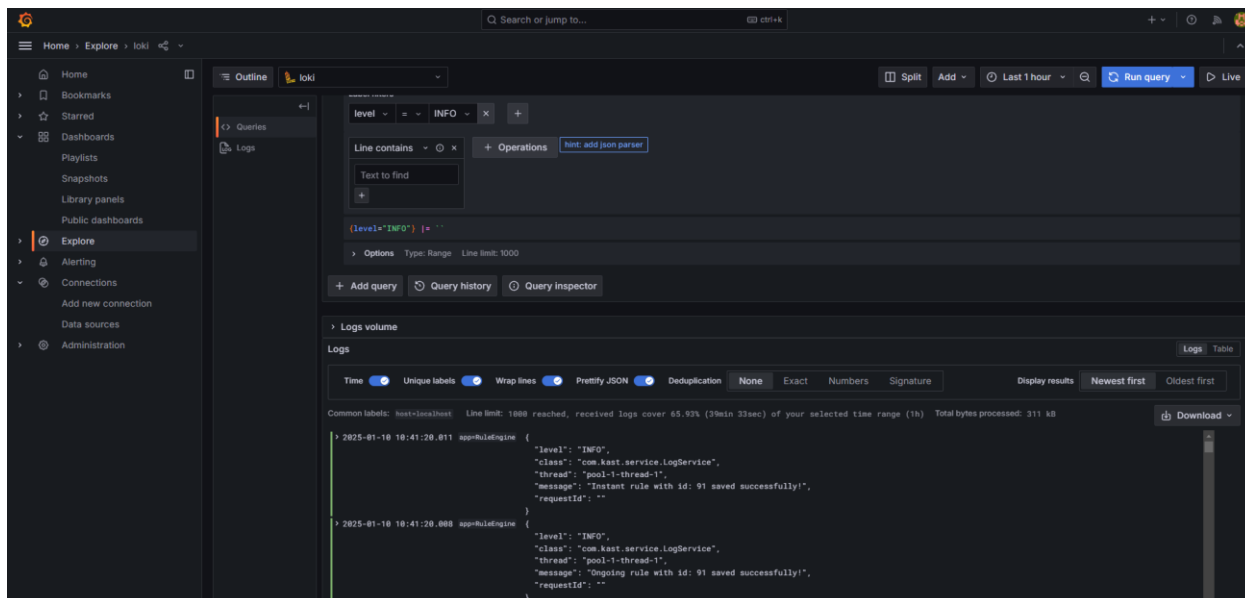


Рисунок 12 – Собранные логи с использованием Loki

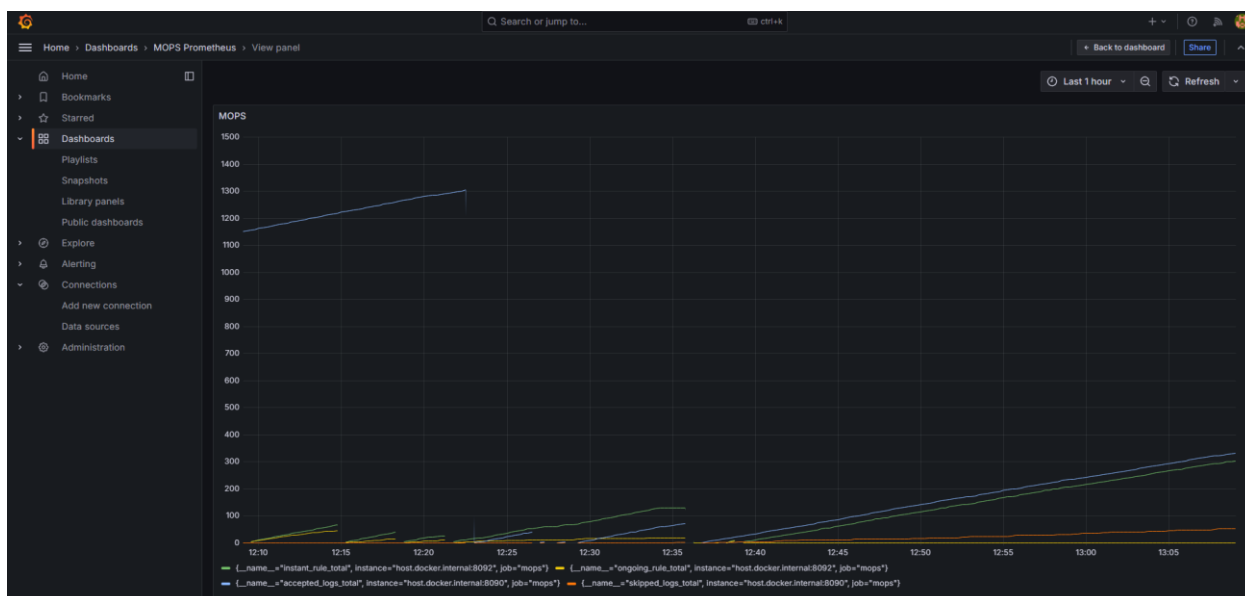


Рисунок 13 – Визуализация метрик, собранных с использованием Prometheus

## 1.4 Описание используемых Design Principles

Решение было спроектировано таким образом, чтобы удовлетворять следующим design principles:

1. Single Responsibility: Каждый микросервис отвечает только за одну задачу.
2. Independence: Каждый микросервис разрабатывается отдельно от остальных.

3. Resilience: микросервисы устойчивы к сбоям. Если Kafka брокер внезапно отключается, то работа IoT-контроллера и Rule Engine ограничивается. В IoT-контроллере и Rule Engine реализована поддержка переподключения к брокеру, а в IoT-контроллере также предусмотрено переподключение к MongoDB.

4. Graceful Degradation: если один из сервисов недоступен, то работа продолжается в ограниченном режиме.

## **Заключение**

Цель лабораторной работы – отработка принципов и подходов к разработке современных многоуровневых сервисов при решении практической задачи. В рамках работы было продемонстрировано решение по реализации простого IoT-сервиса с использованием следующих технологий:

1. Java.
2. Micronaut.
3. Kafka и Kafka UI.
4. MongoDB и Mongo Express.
5. Prometheus.
6. Loki.
7. Grafana.

Во время реализации лабораторной работы были приобретены навыки по работе с Kafka, Kafka UI, Prometheus, Loki, Grafana, Micronaut.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- 1 Tamada4a. MOPS [Electronic resource] / Tamada4a // GitHub. – 2024. – URL: <https://github.com/Tamada4a/MOPS>.