Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
ИТМО» (НИУ ИТМО)
Факультет программной инженерии и компьютерной техники

Отчёт по работе над архитектурой приложения IoT

Выполнили
Студенты группы P4114
Остапенко Ольга, Ершова Анна
Преподаватель:
Перл И. А.

Санкт-Петербург
2025

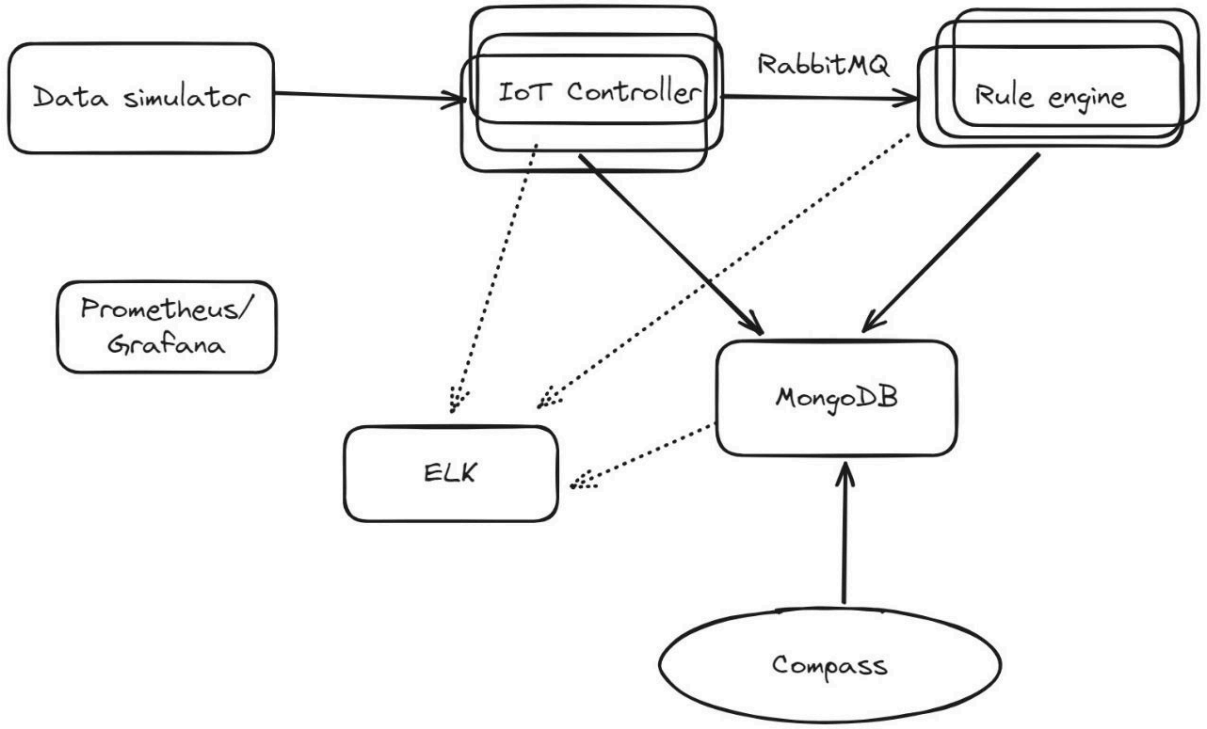
Лабораторное задание

Разработка простого IoT сервиса

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:



Компоненты системы

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.

a. Мгновенное правило - Значение поля A от устройства 42 больше 5.
b. Длящееся правило - Значение поля A 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.

Дополнительные компоненты

1. MongoDB - база данных, в которой хранятся IoT сообщения и отметки (например, алёрты) о срабатываниях правил, которые заложены в Rule Engine

Compass - приложение для просмотра содержимого базы данных MongoDB, будет использоваться вместо пользовательского интерфейса приложения.

2. RabbitMQ - очередь сообщений для обмена данными между IoT Контроллером и Rule Engine
3. Postgres/Graphana - система для сбора и мониторинга метрик о работе приложения
4. ELK Stack - система для сбора и просмотра логов разрабатываемого решения.

При выполнении работы можно использовать любой удобный язык программирования. Желательно выполнять работу на чистом Docker окружении, чтобы максимально разобраться в том, как работают компоненты на низком уровне.

Содержание отчёта

1. Отчёт должен содержать полное описание архитектуры разрабатываемого приложения со всеми допущениями и дополнениями технического задания этой работы.
2. В отчёте необходимо рассмотреть применение дизайн принципов, рассмотренных на лекции, а также гипотетическую применимость. Т.е. если тот или иной принцип не может быть применён в силу простоты разрабатываемого решения, как он мог бы быть применён, если бы решение было бы больше или сложнее в том или ином отношении.

Архитектура решения

1. IoT Controller

- **Получение данных** от симулируемых устройств.
- **Валидация входных пакетов**.
- **Сохранение валидированных данных** в MongoDB.
- **Передача данных** в RabbitMQ для последующей обработки.

2. Rule Engine

- **Чтение пакетов** из RabbitMQ.
- **Обработка мгновенных правил**: "значение поля A > 5".
- **Обработка длящихся правил**: "значение поля A > 5 в 10 последовательных пакетах".
- **Логирование результатов** в ELK Stack. Возврат пользователю ошибок.
- **Запись алертов** в MongoDB.

3. Data Simulator

- **Генерация данных** от заданного числа устройств с указанной частотой.
- **Отправка данных** в IoT Controller.

Инфраструктурные компоненты

1. **MongoDB** - хранилище данных.
2. **RabbitMQ** - система асинхронной обработки сообщений.
3. **ELK Stack** - для логирования.
4. **Prometheus + Grafana** - для сбора и мониторинга метрик.

Организация взаимодействия модулей

- Каждое приложение (кроме симулятора данных) запускается в контейнере**.**
- Компоненты взаимодействуют с помощью сети, поднятой в docker.
- Симулятор данных отправляет данные на iot controller с помощью потоков (библиотека Thread).

Дизайн-принципы и их применение

Принципы, применённые в проекте

1. Detect Failures

- Определены точки отказа системы через обработку исключений при подключении к MongoDB, RabbitMQ и Logstash.
- Нагрузочное тестирование проводится для выявления слабых мест.

2. Логирование и мониторинг ошибок

- Используется ELK Stack для хранения и анализа логов.
- Все исключения, такие как NullPointerException, логируются.
- Метрики мониторинга включают:
    - `http_requests_total`
    - `mongodb_operations_total`
    - `rabbitmq_operations_total`
    - Автоматически генерируемые метрики, такие как `python_gc_objects_collected_total`.

Гипотетическое применение других принципов

1. Decoupling system, async communication

Добавление в RabbitMQ очередей с приоритетами при увеличении набора сообщений.

На текущий момент проект уже использует RabbitMQ для асинхронной коммуникации, что позволяет минимизировать взаимозависимости между компонентами. Однако более сложная архитектура с очередями разных приоритетов не была внедрена из-за ограниченного объёма задач, которые решает система.

2. Retry failed operations

Добавление повторных попыток осуществления операций.

На данный момент система не использует сложную логику повторных попыток, так как существующие операции, такие как запись в MongoDB или отправка данных в RabbitMQ, выполняются стабильно. Добавление механизма повторных попыток могло бы быть актуально в случае частых сетевых сбоев или временной недоступности сервисов. Однако в текущем проекте такие ситуации редки, и введение ретраев добавило бы ненужную сложность.

3. Transaction checkpointing

Добавление контрольных точек транзакций.

Контрольные точки транзакций полезны для восстановления данных при сбоях. В нашем случае, данные, которые передаются между компонентами, небольшие, и их потеря не оказывает критического влияния. Если бы система обрабатывала большие объемы данных или транзакции с высокой стоимостью, внедрение checkpointing помогло бы избежать потерь.

Исходный код

```
import requests
import time
import random
import argparse
from threading import Thread

def generate_data(device_id, frequency, endpoint):
    """
    Генерирует данные для устройства и отправляет их на заданный эндпоинт.

    Args:
        device_id (int): Идентификатор устройства.
        frequency (float): Частота сообщений (в сообщениях в секунду).
        endpoint (str): URL эндпоинта IoT Controller.
    """
    interval = 1 / frequency
    while True:
        data = {
            "device_id": device_id,
            "field_a": random.randint(0, 10),
        }
        try:
            response = requests.post(endpoint, json=data)
            if response.status_code == 200:
                print(f"[Device {device_id}] Data sent: {data}")
            else:
                print(f"[Device {device_id}] Error: {response.status_code} - {response.text}")
        except Exception as e:
            print(f"[Device {device_id}] Failed to send data: {e}")
        time.sleep(interval)

def main():
    parser = argparse.ArgumentParser(description="Data Simulator for IoT devices.")
    parser.add_argument("--devices", type=int, default=10, help="Количество устройств")
    parser.add_argument("--frequency", type=float, default=1.0, help="Частота сообщений
```

```
(в сообщениях в секунду)")
    parser.add_argument("--endpoint", type=str, required=True, help="URL эндпоинта IoT
Controller")
    args = parser.parse_args()

    threads = []
    for device_id in range(1, args.devices + 1):
        thread = Thread(target=generate_data, args=(device_id, args.frequency,
args.endpoint))
        thread.daemon = True
        threads.append(thread)
        thread.start()

    # Ожидание завершения всех потоков
    try:
        for thread in threads:
            thread.join()
    except KeyboardInterrupt:
        print("Data Simulator stopped.")

if __name__ == "__main__":
    main()
```

data-simulator.py

```
from fastapi import FastAPI, HTTPException, Request
from prometheus_client import Counter, Histogram, generate_latest
from pymongo import MongoClient
import pika
import json
import logging
import logstash
import time
from fastapi.responses import PlainTextResponse
from dotenv import load_dotenv
import os

load_dotenv()

MONGO_URI = os.getenv("MONGO_URI")
MONGO_DB = os.getenv("MONGO_DB")
MONGO_COLLECTION = os.getenv("MONGO_COLLECTION")
RABBITMQ_HOST = os.getenv("RABBITMQ_HOST")
RABBITMQ_QUEUE = os.getenv("RABBITMQ_QUEUE")
LOGSTASH_HOST = os.getenv("LOGSTASH_HOST")
LOGSTASH_PORT = int(os.getenv("LOGSTASH_PORT"))
APP_PORT = int(os.getenv("APP_PORT"))

logger = logging.getLogger('python-logstash-logger')
logstash_handler = logstash.TCPLogstashHandler(LOGSTASH_HOST, LOGSTASH_PORT,
```

```
version=1)
logger.addHandler(logstash_handler)

app = FastAPI()

# Метрики Prometheus
REQUEST_COUNT = Counter(
    "http_requests_total",
    "Total number of HTTP requests",
    ["method", "endpoint", "http_status"]
)
REQUEST_DURATION = Histogram(
    "http_request_duration_seconds",
    "Histogram of request processing duration",
    ["method", "endpoint"]
)
MONGODB_OPERATIONS = Counter(
    "mongodb_operations_total",
    "Total number of MongoDB operations",
    ["operation", "status"]
)
RABBITMQ_OPERATIONS = Counter(
    "rabbitmq_operations_total",
    "Total number of RabbitMQ operations",
    ["operation", "status"]
)

try:
    mongo_client = MongoClient(MONGO_URI)
    db = mongo_client[MONGO_DB]
    collection = db[MONGO_COLLECTION]
    logger.info("Connected to MongoDB successfully.")
except Exception as e:
    logger.error(f"Failed to connect to MongoDB: {e}")
    raise

try:
    rabbitmq_connection =
pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))
    channel = rabbitmq_connection.channel()
    channel.queue_declare(queue=RABBITMQ_QUEUE)
    logger.info("Connected to RabbitMQ successfully.")
except Exception as e:
    logger.error(f"Failed to connect to RabbitMQ: {e}")
    raise

@app.post("/data")
async def receive_data(data: dict, request: Request):
    start_time = time.time()
    try:
        logger.info(f"Received data: {data}")
```

```
        # Валидация данных
        if "device_id" not in data or "field_a" not in data:
            logger.warning("Invalid data format received.")
            REQUEST_COUNT.labels(method=request.method, endpoint="/data",
http_status=400).inc()
            raise HTTPException(status_code=400, detail="Invalid data format")

        # Сохранение в MongoDB
        try:
            collection.insert_one(data)
            logger.info(f"Data saved to MongoDB: {data}")
            MONGODB_OPERATIONS.labels(operation="insert", status="success").inc()
        except Exception as e:
            MONGODB_OPERATIONS.labels(operation="insert", status="error").inc()
            raise e

        # Публикация в RabbitMQ
        try:
            data['_id'] = str(data['_id'])
            channel.basic_publish(exchange='', routing_key=RABBITMQ_QUEUE,
body=json.dumps(data))
            logger.info(f"Data published to RabbitMQ: {data}")
            RABBITMQ_OPERATIONS.labels(operation="publish", status="success").inc()
        except Exception as e:
            RABBITMQ_OPERATIONS.labels(operation="publish", status="error").inc()
            raise e

        REQUEST_COUNT.labels(method=request.method, endpoint="/data",
http_status=200).inc()
        return {"status": "success"}
    except Exception as e:
        logger.error(f"Error processing data: {e}")
        REQUEST_COUNT.labels(method=request.method, endpoint="/data",
http_status=500).inc()
        return {"status": "error", "message": str(e)}
    finally:
        duration = time.time() - start_time
        REQUEST_DURATION.labels(method=request.method,
endpoint="/data").observe(duration)

@app.get("/metrics")
async def metrics():
    # Возвращаем метрики в формате Prometheus
    return PlainTextResponse(generate_latest(), media_type="text/plain")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host=0.0.0.0, port=APP_PORT)
```

iot-controller.py

```
import pika
from pymongo import MongoClient
import json
from collections import defaultdict
import logging
import logstash
from dotenv import load_dotenv

load_dotenv()
MONGO_URI = os.getenv("MONGO_URI")
MONGO_DB = os.getenv("MONGO_DB")
MONGO_COLLECTION = os.getenv("MONGO_COLLECTION")
RABBITMQ_HOST = os.getenv("RABBITMQ_HOST")
RABBITMQ_QUEUE = os.getenv("RABBITMQ_QUEUE")
LOGSTASH_HOST = os.getenv("LOGSTASH_HOST")
LOGSTASH_PORT = int(os.getenv("LOGSTASH_PORT"))

# Настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    handlers=[
        logging.StreamHandler(),
    ]
)


logger = logging.getLogger("Rule Engine")

logstash_handler = logstash.TCPLogstashHandler(LOGSTASH_HOST, LOGSTASH_PORT,
version=1)
logger.addHandler(logstash_handler)
# Подключение к MongoDB
try:
    mongo_client = MongoClient(MONGO_URI)
    db = mongo_client[MONGO_DB]
    collection = db[MONGO_COLLECTION]
    logger.info("Connected to MongoDB successfully.")
except Exception as e:
    logger.error(f"Failed to connect to MongoDB: {e}")
    raise

# Подключение к RabbitMQ
try:
    connection =
pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))
    channel = connection.channel()
    channel.queue_declare(queue=RABBITMQ_QUEUE)
    logger.info("Connected to RabbitMQ successfully.")
```

```
except Exception as e:
    logger.error(f"Failed to connect to RabbitMQ: {e}")
    raise

# Хеш-таблица для хранения состояния устройств
device_state = defaultdict(list)

def process_instant_rule(data):
    """
    Мгновенное правило: Значение поля A от устройства 42 больше 5.
    """
    try:
        if data.get("device_id") == 42 and data.get("field_a", 0) > 5:
            alert = {
                "device_id": data["device_id"],
                "rule": "field_a > 5",
                "timestamp": data.get("timestamp")
            }
            collection.insert_one(alert)
            logger.info(f"Instant rule triggered: {alert}")
    except Exception as e:
        logger.error(f"Error processing instant rule: {e}")

def process_continuous_rule(data):
    """
    Длящееся правило: Значение поля A от устройства 42 больше 5 на протяжении 10
пакетов.
    """
    try:
        device_id = data.get("device_id")
        if device_id == 42:
            device_state[device_id].append(data)
            if len(device_state[device_id]) > 10:
                device_state[device_id].pop(0)

            if all(packet["field_a"] > 5 for packet in device_state[device_id]):
                alert = {
                    "device_id": device_id,
                    "rule": "field_a > 5 for 10 messages",
                    "timestamp": data.get("timestamp")
                }
                collection.insert_one(alert)
                logger.info(f"Continuous rule triggered: {alert}")
    except Exception as e:
        logger.error(f"Error processing continuous rule: {e}")

def callback(ch, method, properties, body):
    try:
        data = json.loads(body)
        logger.info(f"Received message: {data}")
        process_instant_rule(data)
```

```
        process_continuous_rule(data)
    except json.JSONDecodeError as e:
        logger.error(f"Failed to decode message: {e}")
    except Exception as e:
        logger.error(f"Error in callback: {e}")

channel.basic_consume(queue='iot_data', on_message_callback=callback, auto_ack=True)

logger.info("Rule Engine is running...")
try:
    channel.start_consuming()
except KeyboardInterrupt:
    logger.info("Rule Engine stopped.")
except Exception as e:
    logger.error(f"Error in main loop: {e}")
```

rule-engine.py

```
version: "3.9"

services:
  elasticsearch:
    image: elasticsearch:7.1.0
    volumes:
      - ./esdata:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      - "discovery.type=single-node"
    networks:
      - default

  logstash:
    build:
      context: .
      dockerfile: logstash/Dockerfile
    ports:
      - "9600:9600"
      - "5228:5228"
    environment:
      LOGSTASH_PORT: 5228
      LOGSTASH_INDEX: "test-index"
      ELASTIC_HOST: "elasticsearch:9200"
      ELASTIC_USERNAME: "elastic"
      ELASTIC_PASSWORD: "elastic"
    networks:
      - default
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:9600" ]
```

```
    interval: 10s
    timeout: 5s
    retries: 5

kibana:
  image: kibana:7.1.0
  hostname: kibana
  ports:
    - "5601:5601"
  networks:
    - default
  depends_on:
    - elasticsearch
  links:
    - elasticsearch
  environment:
    ELASTIC_HOST: "http://elasticsearch:9200"
    ELASTIC_USERNAME: "elastic"
    ELASTIC_PASSWORD: "elastic"

mongo:
  image: mongo:5.0
  container_name: mongo
  ports:
    - "27017:27017"
  networks:
    - default
  volumes:
    - mongodata:/data/db
  healthcheck:
    test: [ "CMD", "mongo", "--eval", "db.runCommand({ ping: 1 })" ]
    interval: 10s
    timeout: 5s
    retries: 3

rabbitmq:
  image: rabbitmq:3-management-alpine
  container_name: 'rabbitmq'
  ports:
    - 5672:5672
    - 15672:15672
  volumes:
    - ~/.docker-conf/rabbitmq/data/:/var/lib/rabbitmq/
    - ~/.docker-conf/rabbitmq/log/:/var/log/rabbitmq
  networks:
    - default

prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  volumes:
```

```
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
  ports:
    - "9090:9090"
  networks:
    - default

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=admin
  networks:
      - default

iot-controller:
  container_name: iot-controller
  build:
    context: ./iot-controller
    dockerfile: Dockerfile
  depends_on:
    logstash:
      condition: service_healthy
    mongo:
      condition: service_healthy
    rabbitmq:
      condition: service_started
  networks:
    - default
  environment:
    ELASTIC_HOST: "elasticsearch:9200"
    LOGSTASH_HOST: "logstash:5228"
    DB_URL: "mongodb://mongo:27017"
    DB_NAME: "iot"
    COLLECTION_NAME: "messages"
  ports:
    - "50051:50051"

rule-engine:
  container_name: rule-engine
  build:
    context: ./rule-engine
    dockerfile: Dockerfile
  depends_on:
    logstash:
      condition: service_healthy
    rabbitmq:
      condition: service_started
  networks:
```

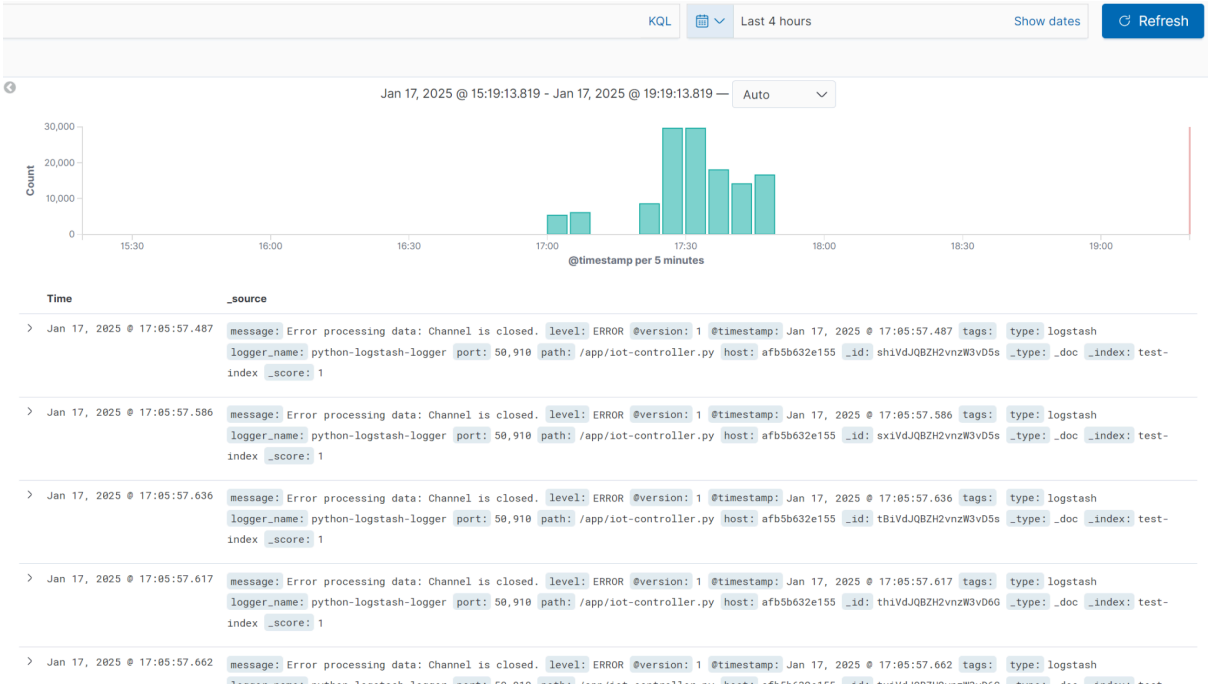
```
      - default
    environment:
      ELASTIC_HOST: "elasticsearch:9200"
      LOGSTASH_HOST: "logstash:5228"
    ports:
      - "8080:8080"

networks:
  default:
    driver: bridge

volumes:
  esdata:
    driver: local
  mongodata:
    driver: local
```

docker-compose.yml

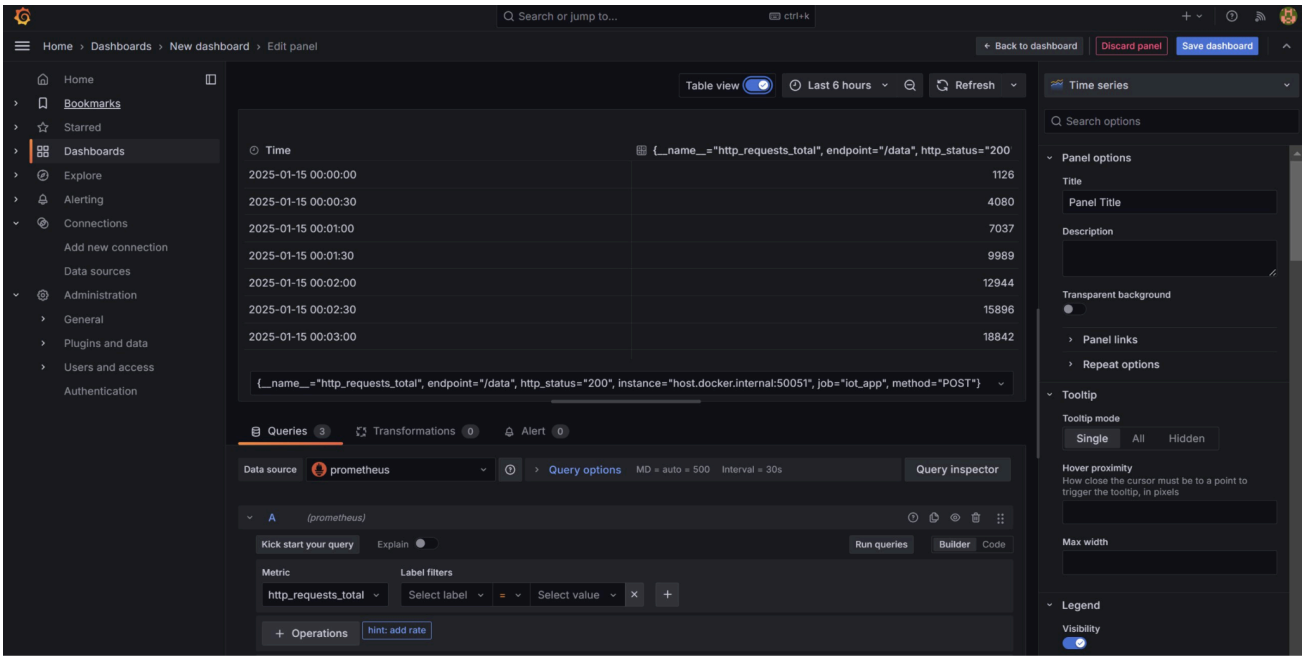
Работа приложения

В MongoDB записываются сообщения от симулятора данных и алерты:

Полученный график в Kibana отображает распределение событий (логов) по времени. Ниже графика представлен список событий, где каждая строка представляет собой отдельное лог-сообщение. Логи отсортированы по времени.



Отображение метрик в Grafana:



Заключение

В ходе лабораторной работы, мы научились использованию асинхронных коммуникаций, централизованного логирования и мониторинга, а также распределенных систем хранения данных. Была построена система которая, эффективно обрабатывает входящие данные и обеспечивает стабильность работы. Дальнейшее развитие архитектуры может быть направлено на повышение масштабируемости и адаптацию под более сложные сценарии использования.