НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

Многоуровневая организация программных систем

РАЗРАБОТКА ПРОСТОГО IOT СЕРВИСА

Работу выполнил Хуссейн Авин

Преподаватель Перл И.А.

1/4/2024

1. ВВЕДЕНИЕ

В этом отчете описывается разработка простого сервиса Интернета вещей в соответствии с целями, изложенными в лабораторном задании. Основной целью было разработать и внедрить многоуровневое решение для Интернета вещей, включающее контроллер Интернета вещей, механизм управления правилами, имитатор данных и вспомогательные компоненты, такие как RabbitMQ, MongoDB и ELK stack. В задании особое внимание уделялось применению базовых принципов проектирования для создания масштабируемого, модульного и функционального решения, а также изучению его гипотетической масштабируемости и сложности.

2. ДОПУЩЕНИЯ

iot_system моделирует сервис, который получает данные от пользователей в следующем формате

```
class IoTData(BaseModel):
    device_id: str
    name: str
    email: EmailStr
    age: int
    x_factor: float
```

который имитирует данного пользователя с именем и адресом электронной почты, возрастом и x_factor.

3. АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Решение IoT было разработано с использованием модульной архитектуры, основанной на микросервисах.

Архитектуру этого приложения можно увидеть на графике ниже.

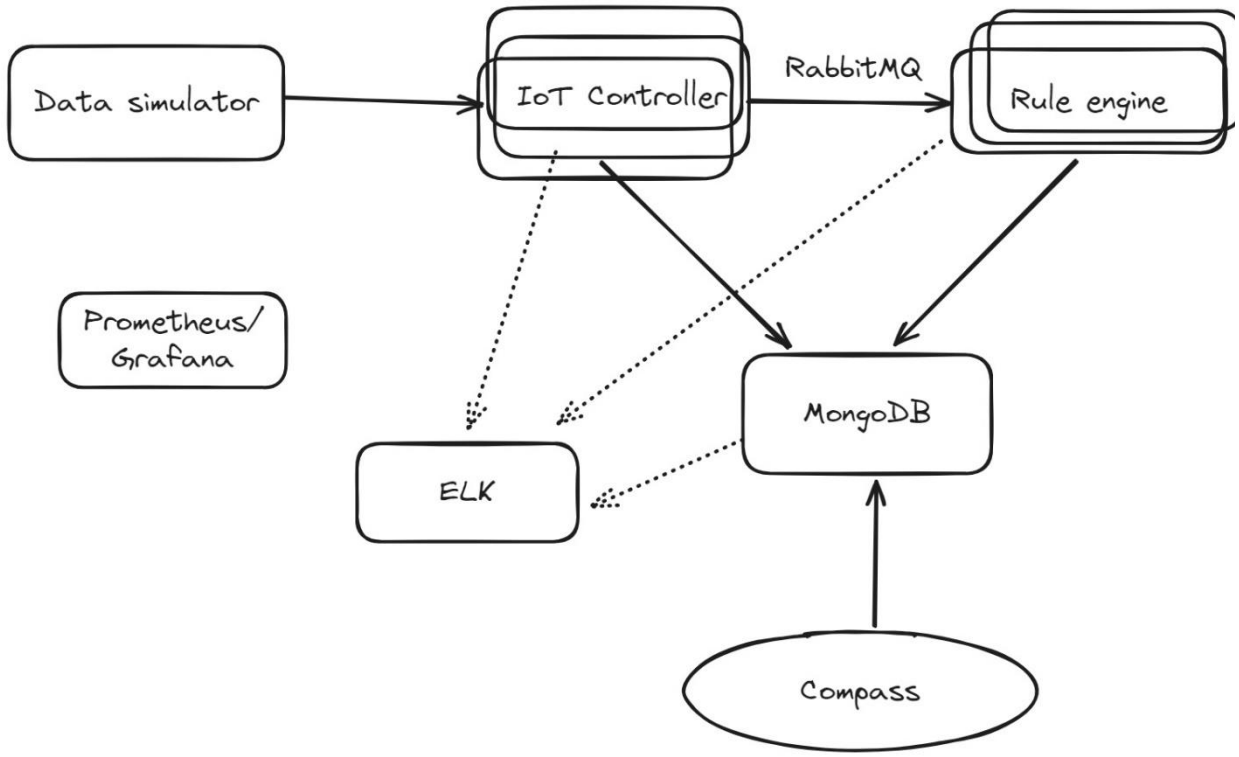

FIGURE 1 APPLICATION ARCHITECTURE

А ниже приведен обзор основных компонентов

3.1 IOT КОНТРОЛЛЕР

- **Функциональность** Принимает и проверяет пакеты входных данных от моделируемых устройств. Для проверки требуется, чтобы пользователь имел действительный адрес электронной почты и был старше 21 года. Затем принятые пакеты сохраняются в MongoDB для дальнейшей обработки и отправляются в rule engine через RabbitMQ для дальнейшей обработки.
- **Реализация** Разработан как микросервис на базе Python, использующий Fast API для обработки входящих HTTP-запросов.
- **Проверка** обеспечивает целостность входящих пакетов, проверяя обязательные поля и допустимые форматы данных.
- **Хранение** Пакеты хранятся в MongoDB для быстрого поиска и масштабируемости.

3.2 RULE ENGINE

- **Функциональность** Обрабатывает как мгновенные, так и непрерывные правила для входящих пакетов данных.
- **Реализация** Использует RabbitMQ для постановки сообщений в очередь, отделяя прием пакетов от обработки правил.
    - **Пример instant rule**: запускает оповещение, если значение поля A от устройства 42 превышает 5.
    - **Пример ongoing rule**: Запускает оповещение, если значение поля A от устройства 42 превышает 5 для 10 последовательных пакетов.

  Я отслеживал пакеты для текущего правила, сохраняя список из последних 10 пакетов в базе данных, и если текущее правило не соответствовало ни одному из них, то список отбрасывается, в противном случае срабатывает предупреждение о текущем правиле.
- **Поток обработки** сообщения поступают из RabbitMQ, обрабатываются в соответствии с определенными правилами, а оповещения сохраняются обратно в MongoDB.

3.3 DATA SIMULATOR

- **Функциональность** Имитирует устройства Интернета вещей, генерируя синтетические пакеты данных. Как упоминалось выше, пакет данных состоит из имени, адреса электронной почты, возраста и x_factor.

- **Реализация** Разработана на Python с настраиваемыми пользователем параметрами, такими как количество устройств.
- **Вывод** Отправляет пакеты данных непосредственно на контроллер Интернета вещей.

3.4 ВСПОМОГАТЕЛЬНЫЕ КОМПОНЕНТЫ

- **MongoDB** Служит центральной базой данных для хранения сообщений IoT и оповещений, запускаемых правилами.
- **RabbitMQ** Выступает в качестве основы обмена сообщениями для разделения контроллера IoT и механизма управления правилами.
- **ELK Stack** собирает и визуализирует журналы приложений, помогая в отладке и мониторинге производительности.
- **Postgres/Grafana** отслеживает показатели производительности системы, такие как время обработки пакетов и скорость оценки правил.

4. ДЕТАЛИ РЕАЛИЗАЦИИ

4.1 ОБСУЖДЕНИЕ DESIGN PRINCIPLES

4.1.1 SCALABILITY

- Использование RabbitMQ гарантирует, что система сможет справляться с высокими нагрузками на данные, помещая сообщения в очередь во время пиковых нагрузок.
- Распределенная архитектура MongoDB поддерживает горизонтальное масштабирование для больших наборов данных.

4.1.2 MODULARITY

- Каждый компонент (контроллер Интернета вещей, механизм управления правилами, имитатор данных) работает независимо, что упрощает отладку и будущие усовершенствования.

4.1.3 RESILIENCE

- RabbitMQ обеспечивает надежную доставку сообщений между компонентами.
- ELK Stack помогает отслеживать журналы для упреждающего обнаружения ошибок.

4.1.4 GRACEFUL DEGRADATION

если один из сервисов недоступен, то работа продолжается в ограниченном режиме.

4.2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

- **Язык программирования:** Python для разработки сервисов и моделирования данных.
- **Docker:** Контейнерное развертывание всех компонентов, обеспечивающее согласованность в разных средах.
- **MongoDB и RabbitMQ:** для хранения и обмена сообщениями соответственно.
- **ELK Stack:** Развернут для сбора журналов и визуализации.
- **Grafana:** интегрирована с Postgres для визуализации показателей производительности.

5. ОСНОВНЫЕ ПРОБЛЕМЫ И ПУТИ ИХ РЕШЕНИЯ

5.1 DATA VALIDATION IN IOT CONTROLLER

- **Задача:** Обеспечение целостности высокочастотных пакетов данных.
- **Решение:** Реализована проверка схемы JSON с надежной обработкой ошибок и протоколированием.

5.2 НЕПРЕРЫВНАЯ ОБРАБОТКА ПРАВИЛ

- **Задача:** Эффективное управление сохранением данных во всех контейнерах системы, включая механизм управления правилами, контроллер Интернета вещей и другие службы. Это было особенно важно для отслеживания и оценки последовательности пакетов с течением времени, поддержания состояния приложения и предотвращения потери данных или несоответствий при перезапуске или обновлении контейнера. Без надлежащего сохранения данных система рисковала потерять исторические данные, необходимые для обработки непрерывных правил и сообщений в очереди.
- **Решение:** Реализованы тома Docker для обеспечения постоянного хранения всех контейнеров. Используя тома, каждая служба может поддерживать требуемое состояние или данные (например, архивные данные о пакетах механизма правил, журналы контроллера Интернета вещей и очереди сообщений RabbitMQ) при перезапуске. Такой подход обеспечивал долговечность данных, сводил к минимуму риск перегрузки памяти за счет выгрузки данных на диск и позволял легко управлять контейнерами без ущерба для функциональности.

5.3 MEMORY MANAGEMENT

- **Задача**: управление большими объемами журналов, генерируемых системой, особенно стеком ELK, что требовало значительных ресурсов памяти для индексации журналов и запросов к ним. Это привело к проблемам с памятью, что привело к нестабильности контейнера и снижению производительности.
- **Решение**: Были установлены соответствующие ограничения на объем памяти для контейнеров ELK Stack, чтобы они не потребляли чрезмерных ресурсов. Установив соответствующие ограничения на объем памяти и включив эффективную ротацию журналов, система обеспечила стабильную работу, сохранив при этом способность эффективно индексировать журналы и запрашивать их. Кроме того, настройка размера кучи Elasticsearch и оптимизация хранилища журналов помогли сбалансировать производительность и использование ресурсов

6. РЕЗУЛЬТАТЫ

6.1 DOCKER

Были созданы следующие контейнеры

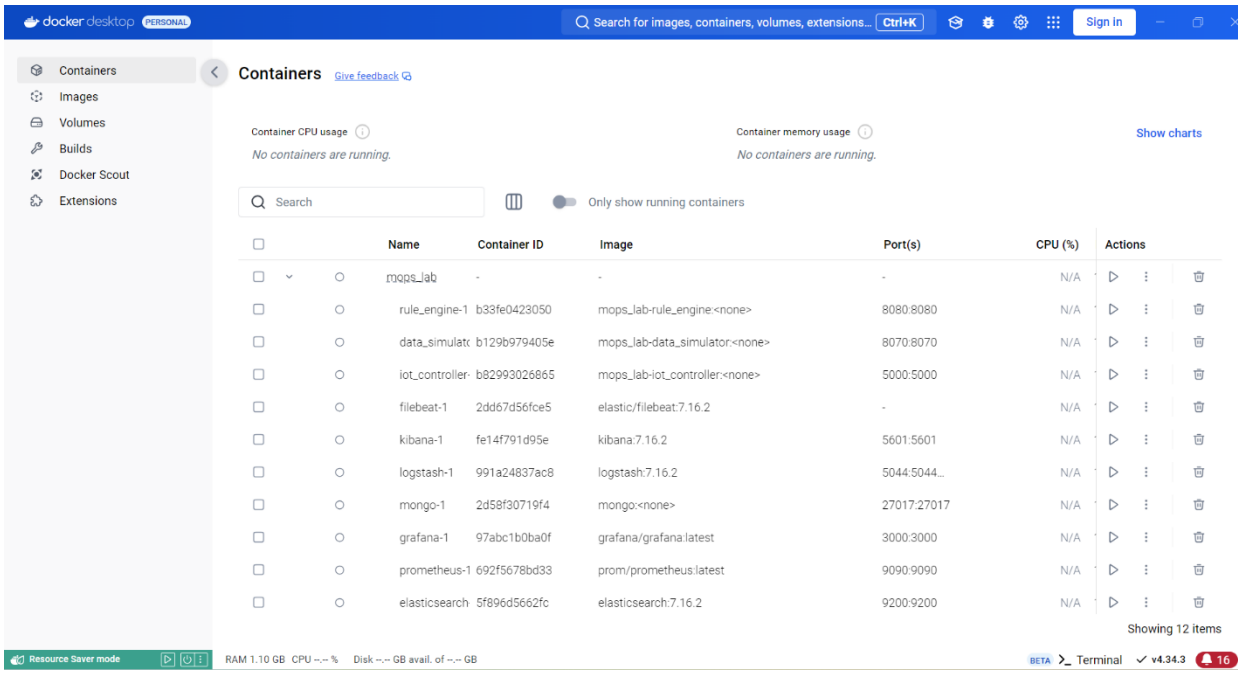

FIGURE 2 DOCKER CONTAINERS

и следующие volumes

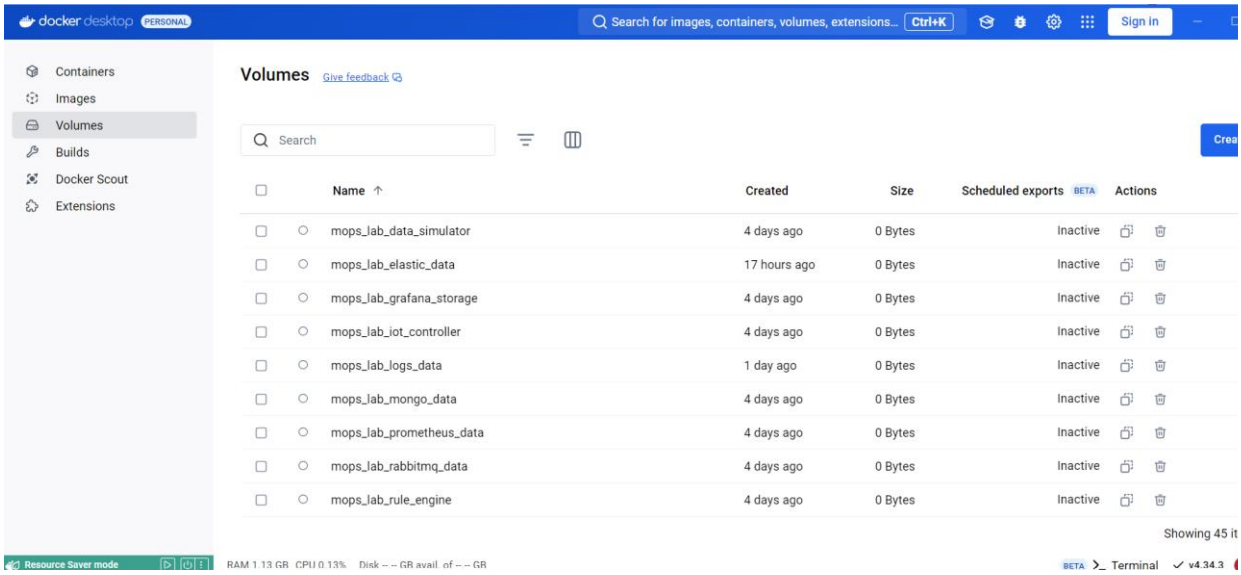

FIGURE 3 DOCKER VOLUMES

6.2 MONGO DB COMPASS

На следующем рисунке показана созданная база данных и сохраненные в ней документы, a instand/ongoing правила.

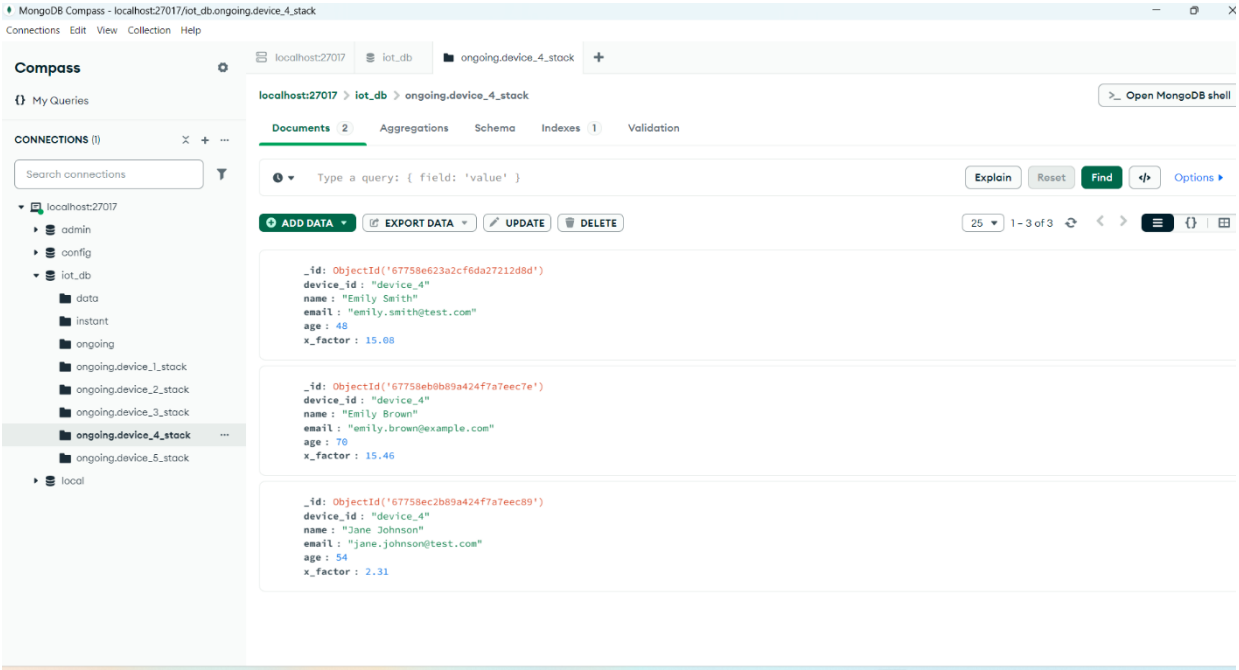

FIGURE 4 IOT_DB THE DATABASE FOR USER DATA AS WELL AS INSTANT AND ONGOING RULES IN COMPASS

6.3 RABBIT MQ

На следующем рисунке показаны каналы в rabbitmq

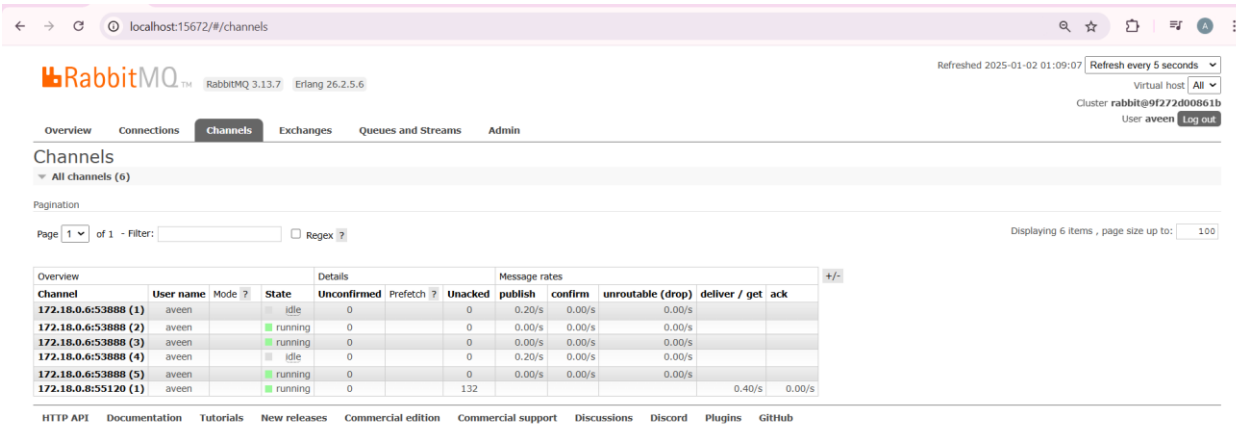

FIGURE 5 RABBIT MQ CHANNELS

6.4 PROMETHEUS

состояние Prometheus, который фиксирует показатели внутри rule_engine, data_simulator, iot_controller и RabbitMQ, показано на рисунке ниже.

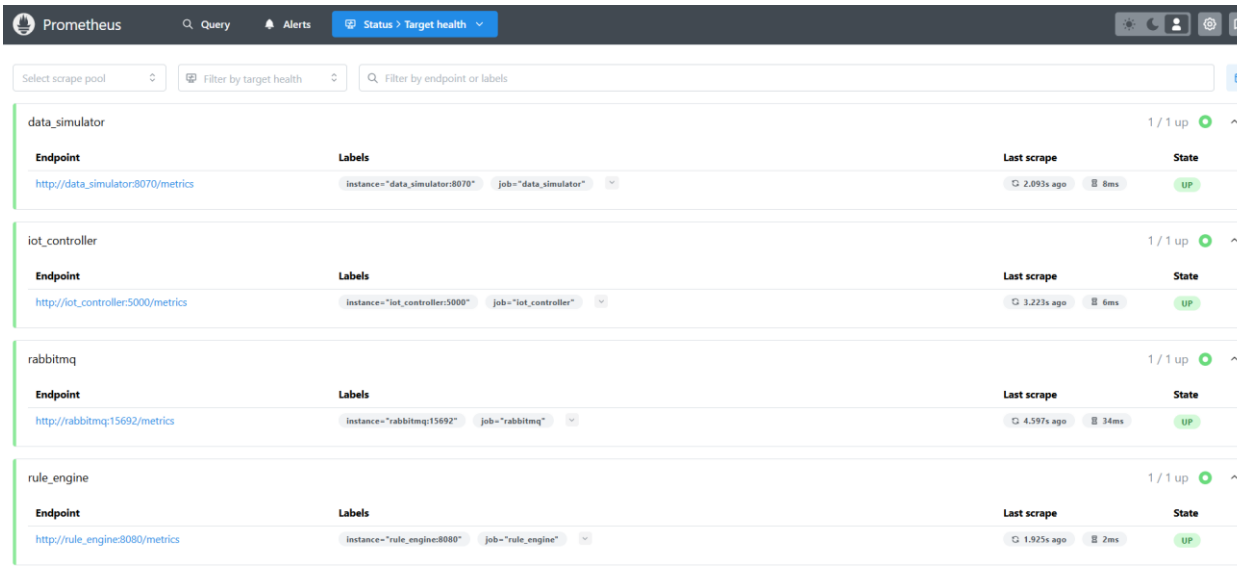

FIGURE 6 THE STATE OF PROMETHEUS

6.5 GRAFANA

Grafana используется для визуализации показателей, полученных с помощью Prometheus. В data_simulator мы фиксируем общее количество запросов, а также продолжительность каждого запроса.

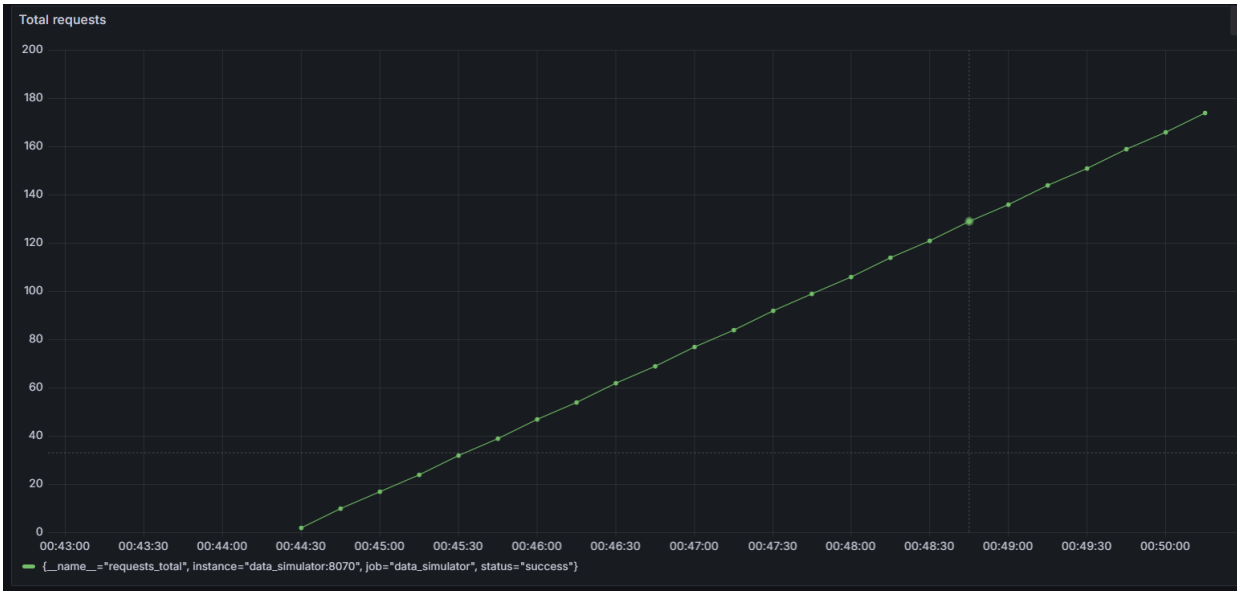

FIGURE 7 TOTAL REQUESTS METRIC AS CAPTURED IN DATA_SIMULATOR

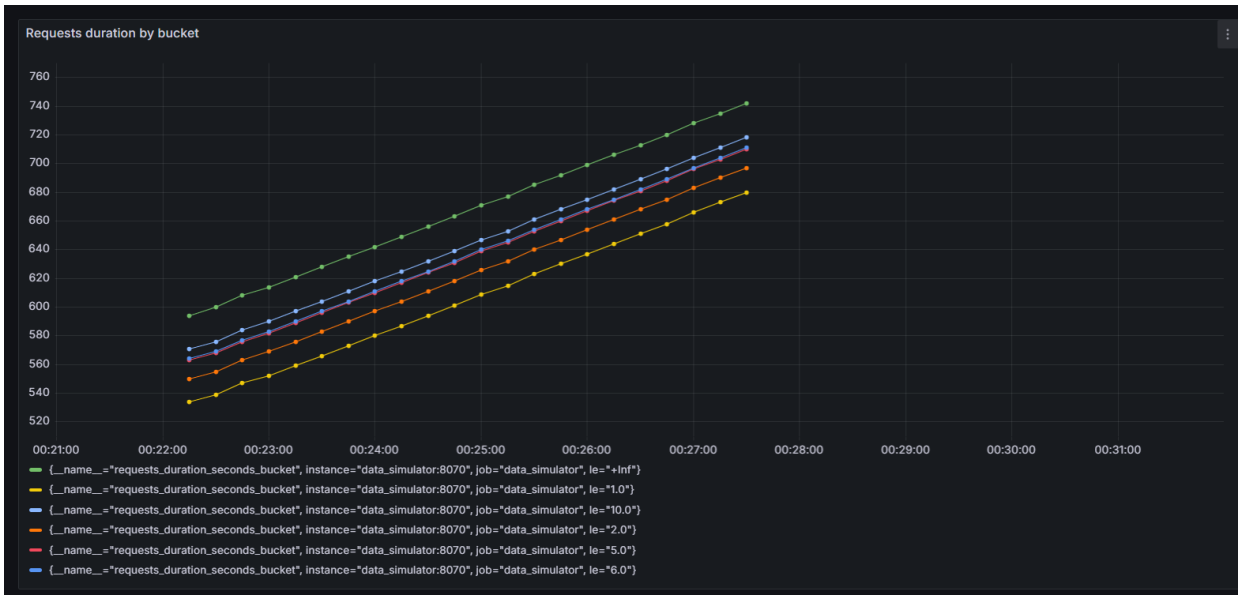

FIGURE 8 REQUESTS DURATION BY TIME BUCKET IN DATA_SIMULATOR

В IOT_controller количество принятых/отклоненных запросов было записано как показатель для Prometheus, и на следующем рисунке показан график этого показателя в Grafana.

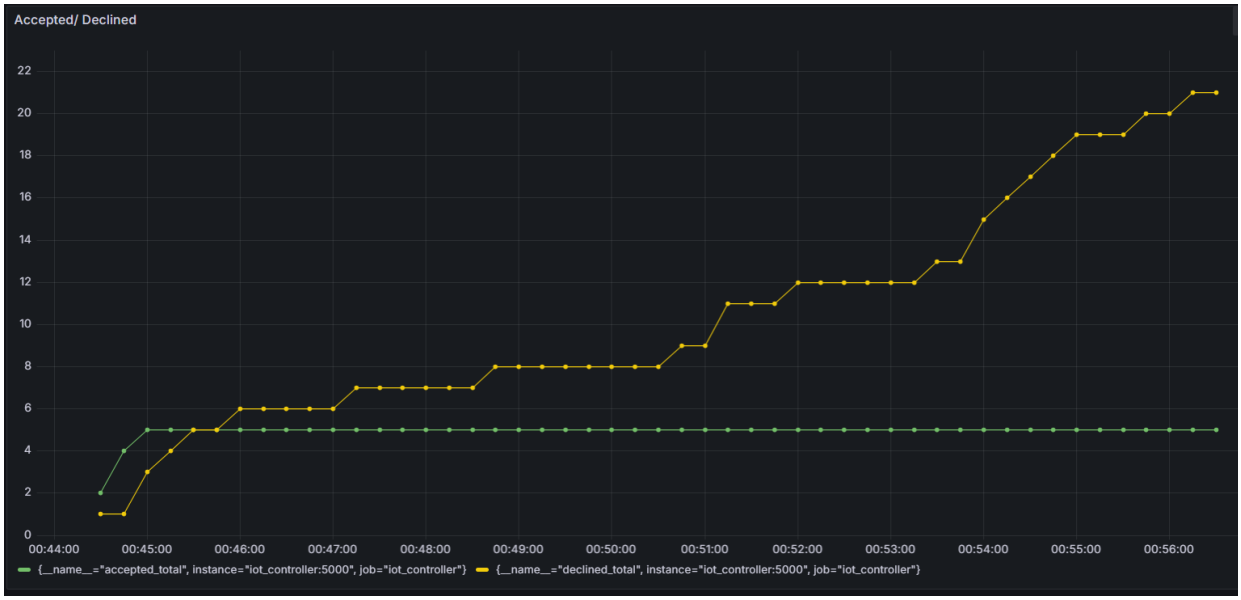

FIGURE 9 ACCEPTED/DECLINED REQUESTS IN IOT_CONTROLLER AS CAPTURED BY PROMETHUS AND VISUALIZED IN GRAFANA

А общее количество запросов, зафиксированных в iot_controller, показано на рисунке ниже.

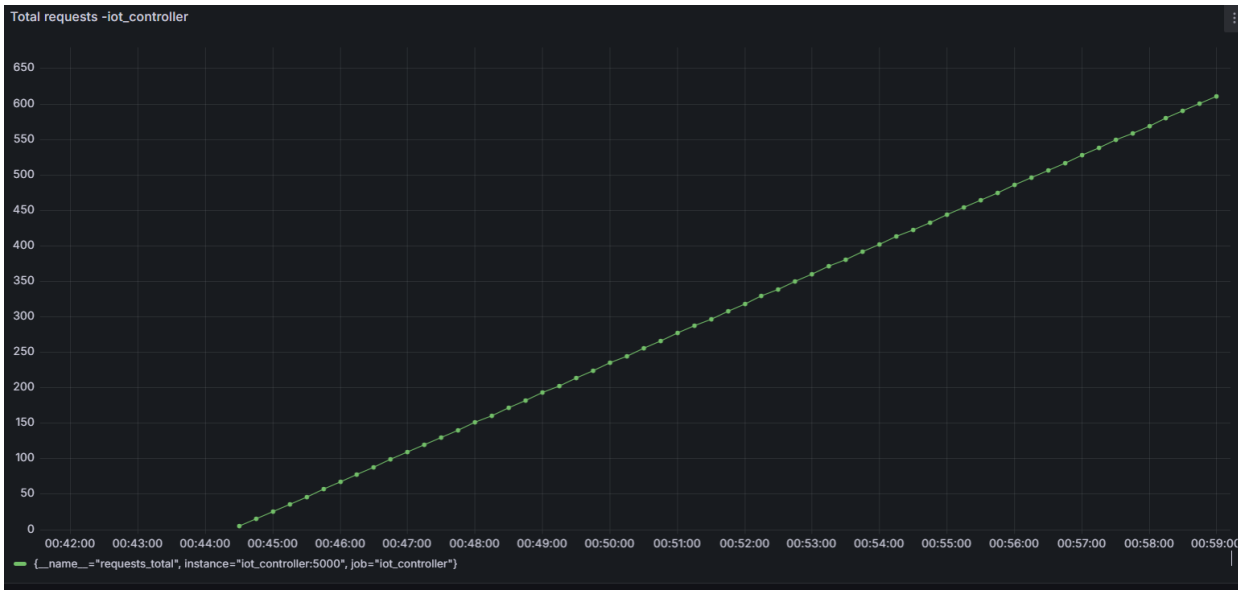

FIGURE 10 THE TOTAL NUMBER OF REQUESTS AS CAPTURED IN IOT_CONTROLLER VISUALIZED IN GRAFANA

а в rule_engine количество совпадающих мгновенных/текущих правил было зафиксировано как показатель для Prometheus, и ниже приведено изображение визуализации в Grafana.

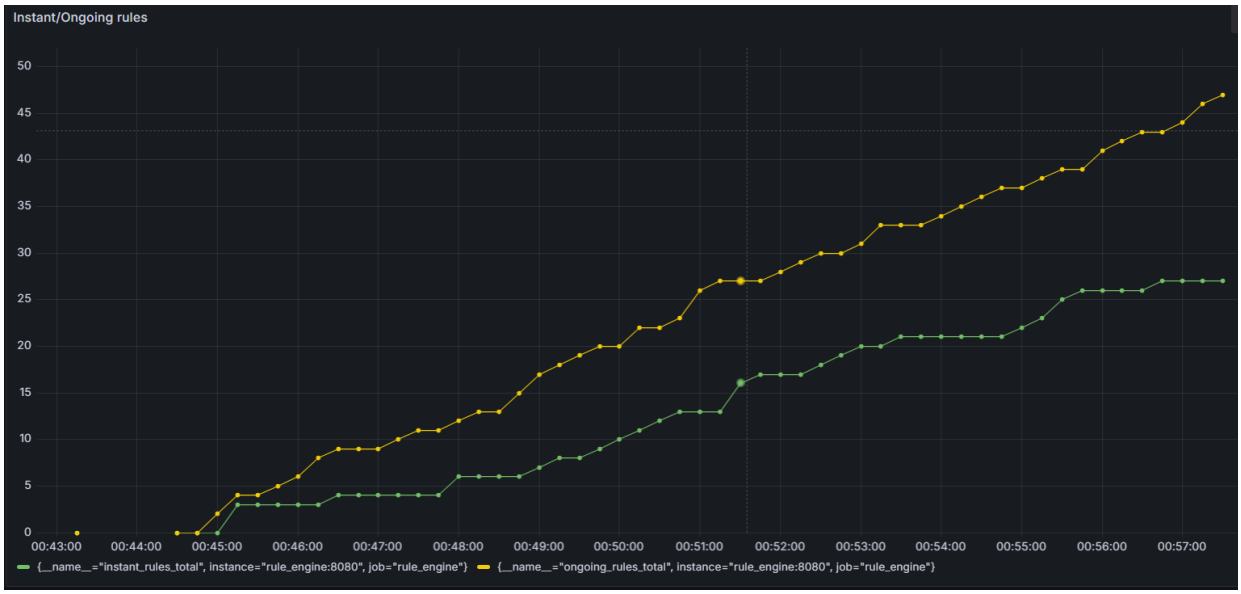

FIGURE 11 INSTANT/ONGOING RULE MATCH COUNT IN RULE_ENGINE VISUALIZED IN GRAFANA

6.6 ELK STACK

ELK Stack, состоящий из Logstash, Elasticsearch и Kibana, был развернут с использованием четырех отдельных контейнеров Docker, включая дополнительный контейнер для Filebeat. Журналы, генерируемые каждым контейнером, сохранялись в папках /var/log, где они классифицировались по типам (info и error). Интерфейс Kibana использовался для просмотра и поиска этих журналов с использованием Elasticsearch в

качестве бекенда. Визуализация логов в Kibana для данного проекта представлена на изображении ниже.

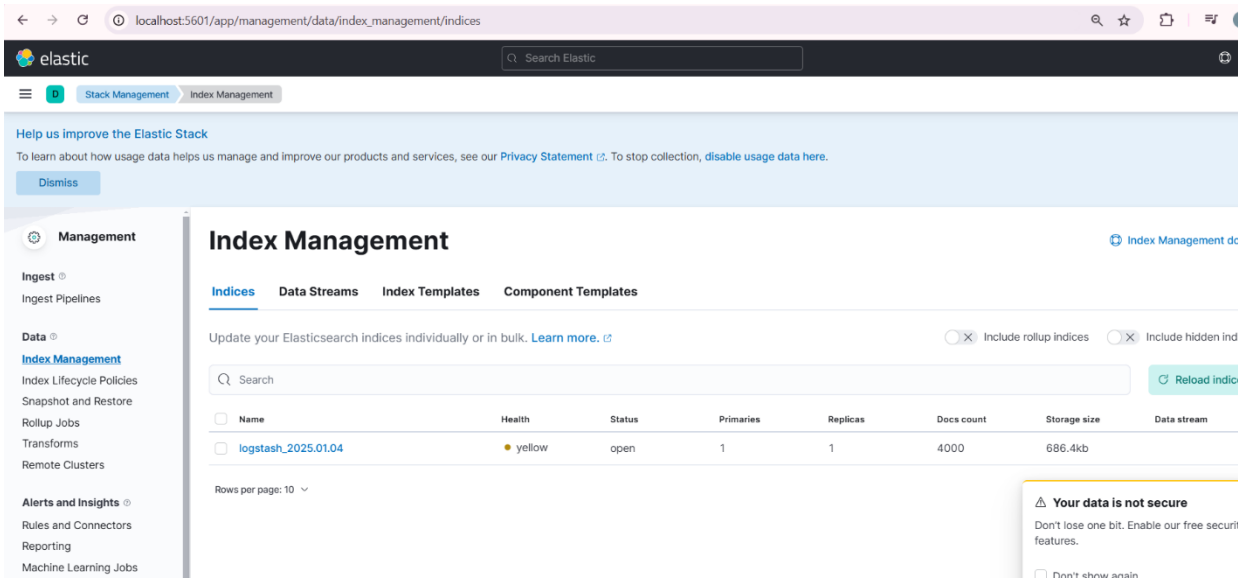

FIGURE 12 THE CONNECTION MADE BETWEEN LOGSTASH AND KIBANA TO VIEW THE LOGS

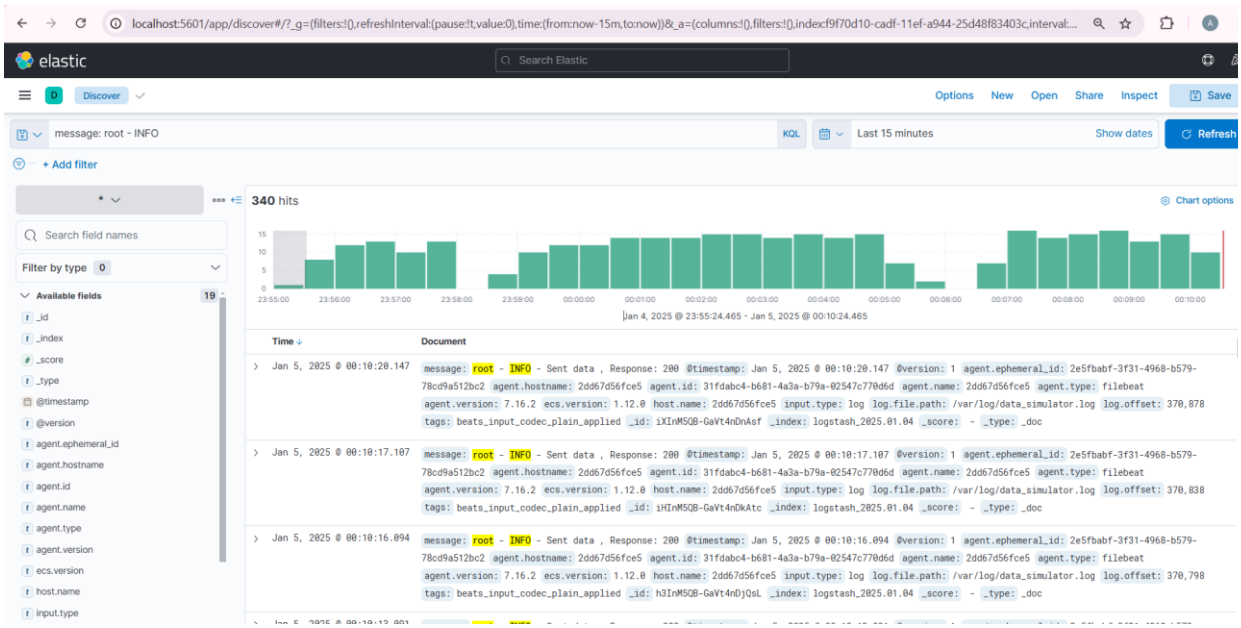

FIGURE 13 THE LOGS AS CAPTUED USING LOGSTASHA AND SEARCHED THROUGH USING ELASTICSEARCH AND VISUALIZED IN KIBANA

7. ГИПОТЕТИЧЕСКАЯ МАСШТАБИРУЕМОСТЬ И ПРИМЕНИМОСТЬ

7.1 БОЛЕЕ КРУПНАЯ СИСТЕМА

- **Применение**: Модульная архитектура может масштабироваться для обработки миллионов устройств за счет добавления дополнительных экземпляров IoT Controller и Rule Engine.
- **База данных:** Возможности шардинга MongoDB позволяют эффективно управлять большими наборами данных.
- **Обмен сообщениями:** Функция кластеризации RabbitMQ поддерживает более высокую пропускную способность сообщений.

7.2 РАСШИРЕННЫЕ ПРАВИЛА

- **Улучшение**: Интеграция моделей машинного обучения для обнаружения аномалий и предиктивного обслуживания.
- **Пример:** Модель, предсказывающая сбои устройств на основе исторических данных.

8. ЗАКЛЮЧЕНИЕ

Лабораторное задание успешно продемонстрировало разработку простого решения для Интернета вещей, основанного на современных принципах проектирования. Несмотря на то, что система является базовой, ее архитектура обеспечивает масштабируемость, устойчивость и расширяемость. Это задание позволило получить практический опыт работы с критически важными компонентами Интернета вещей, включая организацию очереди сообщений, обработку правил и ведение журналов. Принципы, применяемые здесь, могут быть применены к более сложным системам, что позволит использовать IoT-приложения в реальном мире.

9. ОГЛАВЛЕНИЕ