УДК 004.75

Авельцов Д.О., Гайдамако В.В
Институт машиноведения и автоматики НАН КР
E-mail: dolpha@gmail.com

ВЗАИМОДЕЙСТВИЕ МИКРОСЕРВИСОВ В СИСТЕМЕ МОНИТОРИНГА ГЕОЭКОЛОГИЧЕСКИХ ПАРАМЕТРОВ ОКРУЖАЮЩЕЙ СРЕДЫ

В статье описывается взаимодействие между микросервисами облачной информационно-измерительной системы, предназначенной для мониторинга параметров окружающей среды. Внутреннее взаимодействие между сервисами осуществляется с использованием протоколов MQTT (Message Queuing TelemetryTransport) и AMQP, брокеров EMQX и RabbitMQ.

Ключевые слова: микросервисы, протокол передачи данных, gRPC, MQTT, AMQP, брокер сообщений, EMQX, RabbitMQ.

Введение. В современном мире становится все более очевидным, что проблемы, связанные с экологией, требуют серьезного внимания и незамедлительных действий. Изменение климата, загрязнение окружающей среды, убыль биоразнообразия – все это тревожные сигналы, указывающие на необходимость эффективного мониторинга и управления экологическими ресурсами. В этом контексте разрабатываемый программный проект облачной информационно-измерительной системы (ОИИС) экологического мониторинга занимает важное место, предоставляя новаторский подход к сбору, анализу и визуализации данных, необходимых для оценки и контроля состояния окружающей среды. Разрабатываемая система представляет собой сложную микросервисную архитектуру, предназначенную для масштабируемого управления датчиками, данными измерений и работы с пользователями. Модули в микросервисной архитектуре слабо связаны, могут быть реализованы на разных языках программирования, выполняться в разных средах виртуализации, на разных операционных системах, следовательно, могут легко и независимо друг от друга изменяться, что обеспечивает гибкость, надежность и масштабируемость разрабатываемой системы [1]. Система способна эффективно обрабатывать большие объемы данных датчиков и запросов пользователей. Микросервисы и их взаимодействие представлены на рисунке 1.

Сервисы системы. Сервисы датчиков:

- Шлюз API датчиков (sensorset-sensor-request-handler): выступает в качестве точки входа для данных датчиков, направляя запросы к соответствующим сервисам.
- Шлюз входящих данных датчиков (sensorset-sensor-ingress-gateway): управляет входящими данными датчиков, возможно, преобразуя или нормализуя данные перед их хранением или дальнейшей обработкой.
- Нормализатор данных датчиков (sensorset-sensor-data-normalizer): стандартизирует данные датчиков в единый формат для упрощения обработки и анализа.
- Хранилище данных датчиков (sensorset-sensor-data-store): хранит данные датчиков в базе данных для извлечения и анализа.
- Обработчик HTTP-команд (sensorset-sensor-command-http-handler) датчиков и обработчик MQTT(sensorset-sensor-mqtt-handler): обрабатывают определенные типы команд или запросов датчиков, используя различные протоколы, такие как HTTP и MQTT.
- Хранилище команд датчиков (sensorset-sensor-command-store): хранит и управляет командами датчиков, обеспечивая их выполнение и отслеживание.
- Эмуляция запросов датчиков и обработчик запросов датчиков (sensorset-sensor-request-handler): имитируют запросы датчиков для тестирования и обрабатывают фактические запросы датчиков соответственно.

- Реестр шаблонов датчиков (sensorset-sensor-template-registry): управляет шаблонами датчиков, определяя форматы данных и другие метаданные для различных датчиков.

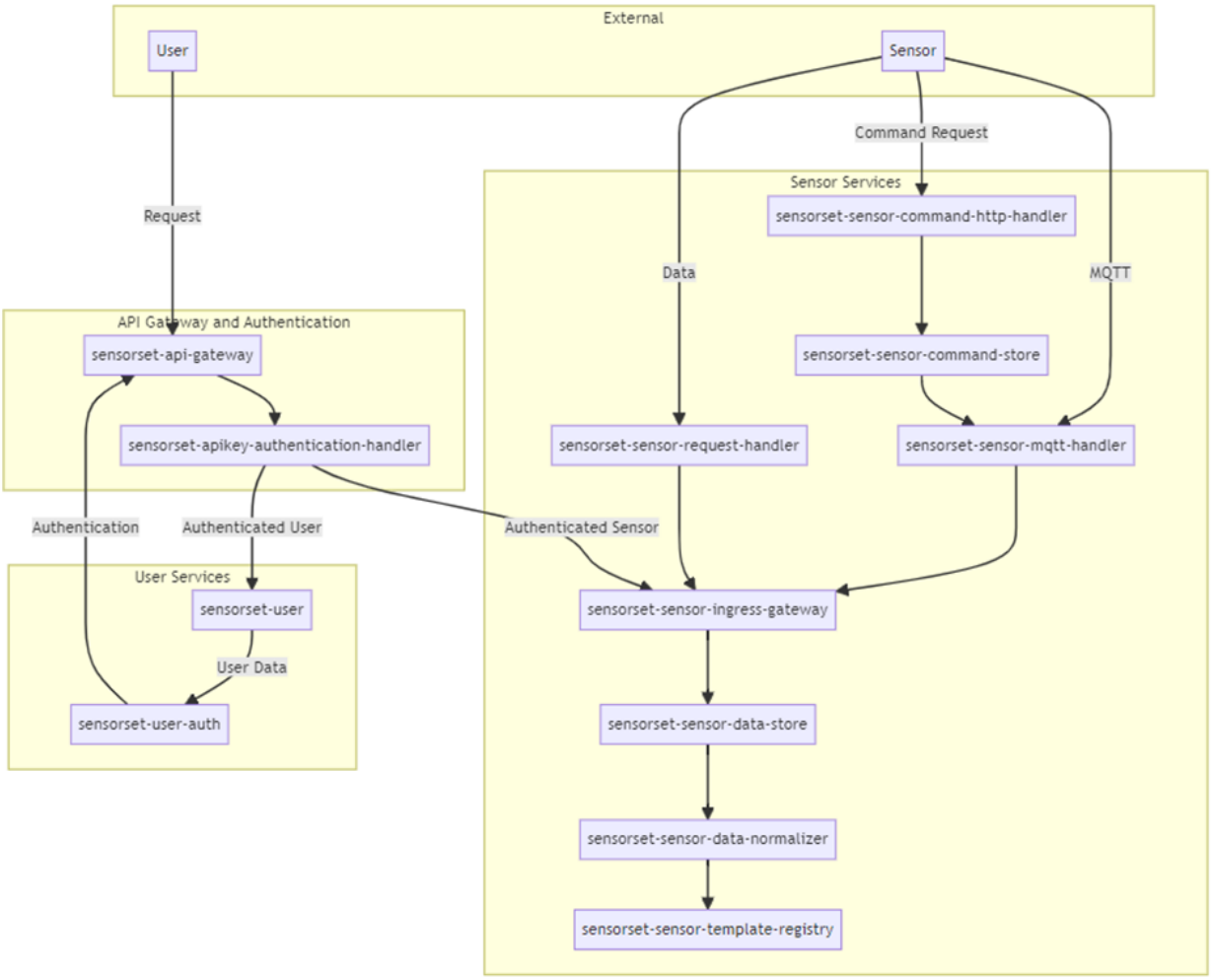

Рисунок 1 – Взаимодействие между микросервисами ОИИС

Сервисы пользователей:

- Сервис пользователей (sensorset-user): управляет данными пользователей, включая создание, извлечение и обновление информации о пользователях.
- Сервис аутентификации пользователей (sensorset-user-auth): обрабатывает аутентификацию пользователей, генерацию токенов и их проверку.
- Обработчик аутентификации Sensorset APIkey (sensorset-apikey-authentication-handler): используется для аутентификации пользователей, может использоваться также для аутентификации датчиков.

Взаимодействие с базами данных: сервисы “Хранилище данных датчиков” (sensorset-sensor-data-store) и “Сервис пользователей” (sensorset-user) взаимодействуют с базами данных (PostgreSQL или аналогичными) для выполнения операций создания, чтения, обновления и удаления данных датчиков и информации о пользователях.

Внешние API:

- Шлюз API (sensorset-API-gateway). Точка входа в систему, комплексный сервис, предназначенный для взаимодействия с различными компонентами в экосистеме управления данными сенсоров. Он выполняет функцию API Gateway и предоставляет конечные точки для аутентификации пользователя, управления устройствами, извлечения данных и выполнения команд на устройствах. Система

может взаимодействовать с внешними API для выполнения дополнительных функций, например, для отправки уведомлений или интеграции с другими платформами.

Аутентификация и авторизация:

- Сервис аутентификации пользователей интегрируется с JWT (JSON Web Tokens [2]) для генерации токенов и их проверки, обеспечивая безопасный доступ к ресурсам системы.

Назначение и детальное описание некоторых сервисов приведено в [3]. В системе можно выделить разные виды взаимодействия – взаимодействие с пользователем, взаимодействие с внешними приложениями, внутреннее взаимодействие или взаимодействие между сервисами. Для взаимодействия с внешним миром используются протоколы HTTP (Hyper Text Transfer Protocol), REST (REpresentational State Transfer) [4], а также формат JSON (Java Script Object Notation) [5]. Рассмотрим принципы взаимодействия между сервисами.

Взаимодействие между сервисами. Взаимодействие между микросервисами осуществляется с помощью протокола gRPC (gRPCRemote Procedure Calls, усовершенствованный RPC от Google[6]), для обмена данными используется формат Protocol Buffers [7,8]. Это обеспечивает надежность, эффективность и гибкость в обмене данными, использование их для организации взаимодействия между микросервисами описано в [8]. Для работы с датчиками используются протоколы HTTP, MQTT (Message Queuing Telemetry Transport) [9] и AMQP (Advanced Message Queuing Protocol) – открытый стандарт передачи бизнес-сообщений между приложениями [10]. Например, шлюз API датчиков (sensorset-sensor-request-handler) может перенаправлять запросы в шлюз входящих данных датчиков (sensorset-sensor-ingress-gateway) или нормализатор данных (sensorset-sensor-data-normalizer), а обработчик MQTT (sensorset-sensor-mqtt-handler) использует MQTT брокер EMQX[11] для получения сообщений от устройств и RabbitMQ [12] для отправки сообщений в AMQP очередь. Клиент EMQX API используется для взаимодействия с EMQX брокером, а публикатор AMQP используется для отправки сообщений в RabbitMQ.

Общая картина взаимодействия сервисов представлена на рисунке 1. Для описания полного потока взаимодействия между сервисами в системе представим типичный сценарий, когда датчик отправляет данные, они обрабатываются, и пользователь взаимодействует с системой. Этот пример продемонстрирует, как сервисы работают вместе слаженно и координированно.

1. Получение данных датчика:

- Датчик отправляет данные в систему. Если данные передаются по протоколу HTTP, они попадают в “шлюз API датчиков” (sensorset-sensor-request-handler).
- “Шлюз API датчиков” направляет данные датчика в “шлюз входящих данных датчиков”(sensorset-sensor-ingress-gateway) (рис.2, 3). Если используется протокол MQTT, датчик передает данные “обработчику MQTT” (sensorset-sensor-mqtt-handler), который опубликует их – передаст в “шлюз входящих данных датчиков” (рис.4).
- “Шлюз входящих данных датчиков” может выполнить первичные проверки или трансформации данных (рис.2, 3, 4).

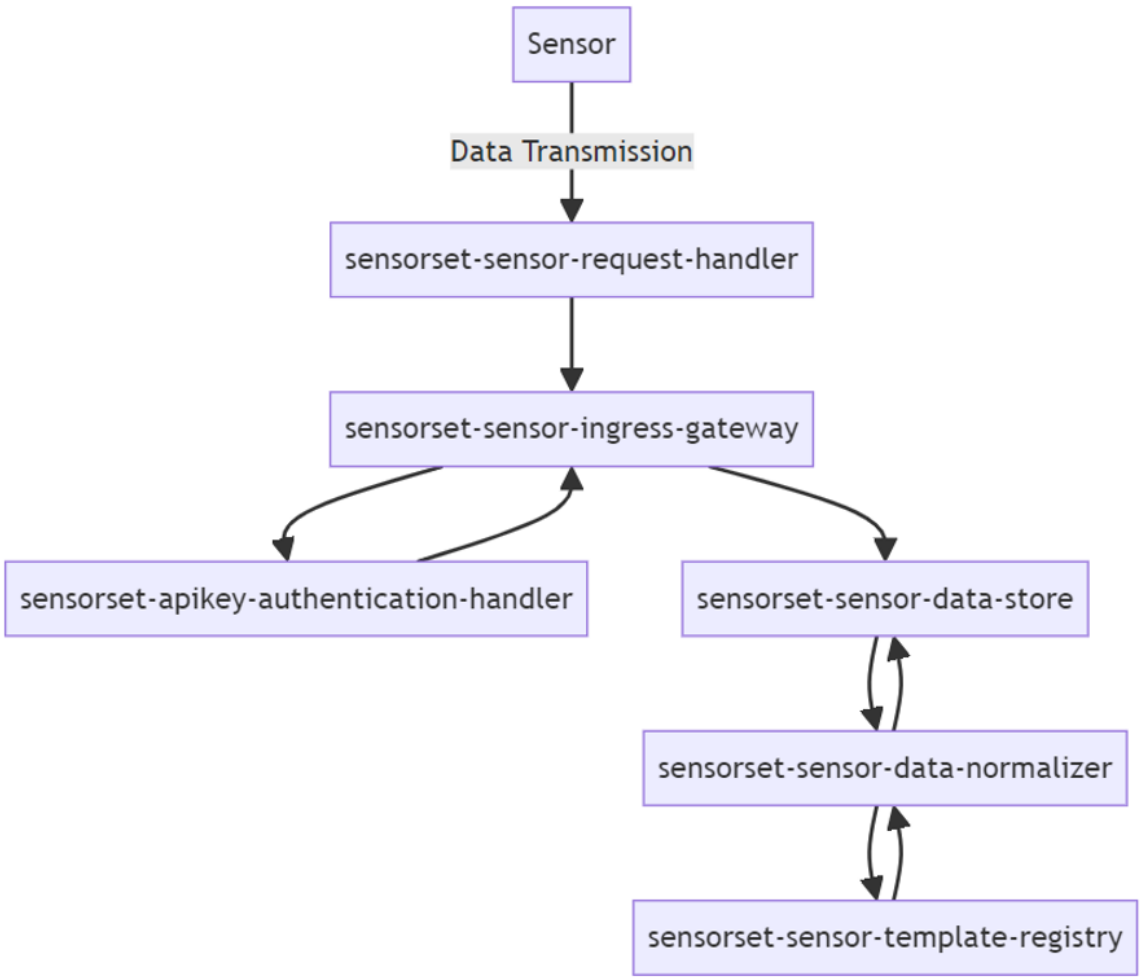

Рисунок 2 – Передача данных измерений от датчика с использованием протокола HTTP

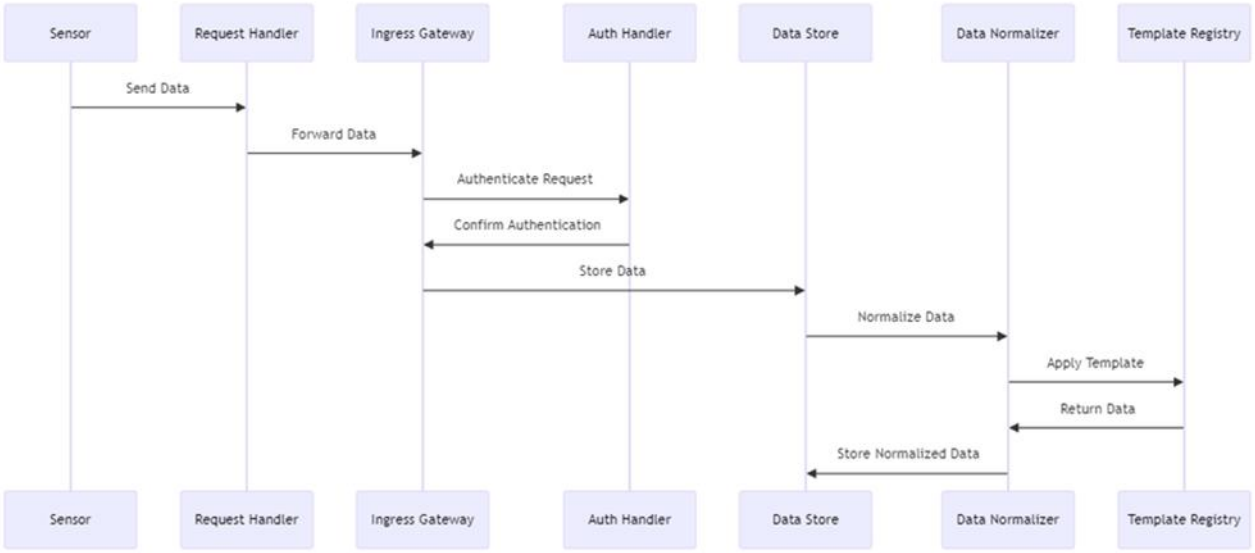

Рисунок 3 – Передача данных измерений от датчика с использованием протокола HTTP, диаграмма последовательности

2. Нормализация данных и хранение:

- Обработанные данные передаются в «нормализатор данных датчиков» (sensorset-sensor-data-normalizer), где они приводятся к единому формату.

- После нормализации данные сохраняются в «хранилище данных датчиков» (sensorset-sensor-data-store).

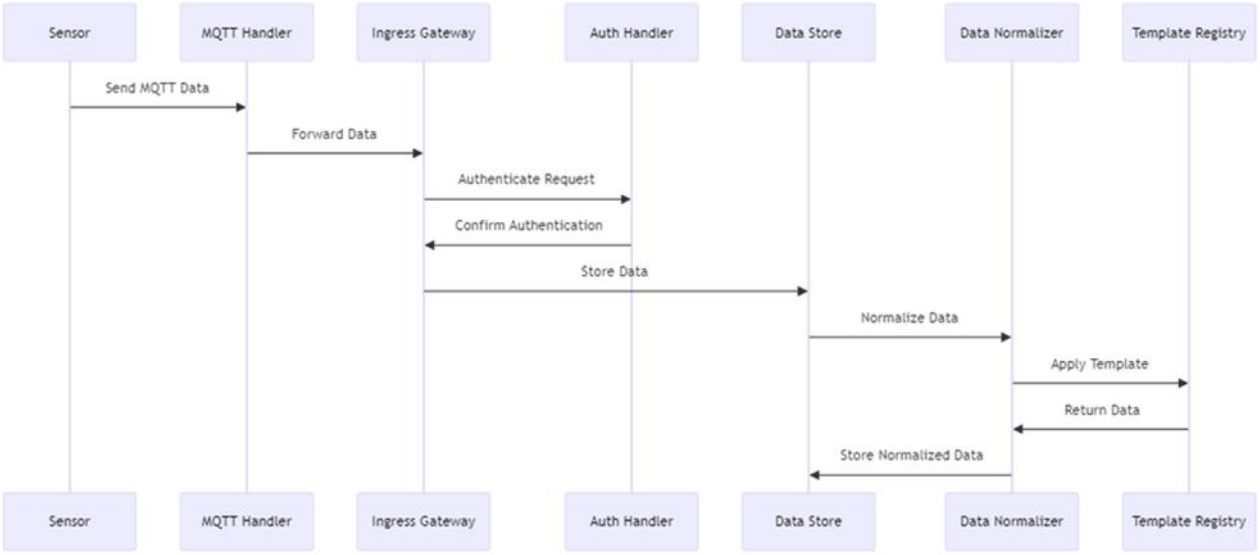

Рисунок 4 – Передача данных измерений от датчика с использованием протокола MQTT, диаграмма последовательности

3. Обработка команд датчика:

- Если необходимо отправить команды датчику (например, изменения конфигурации), они обрабатываются «HTTP-обработчиком команд датчиков» (sensorset-sensor-command-http-handler) или «MQTT-обработчиком датчиков» (sensorset-sensor-mqtt-handler) в зависимости от применяемого протокола связи.
- «Хранилище команд датчиков» (sensorset-sensor-command-store) отслеживает эти команды, обеспечивая их выполнение и регистрацию (рис.5).

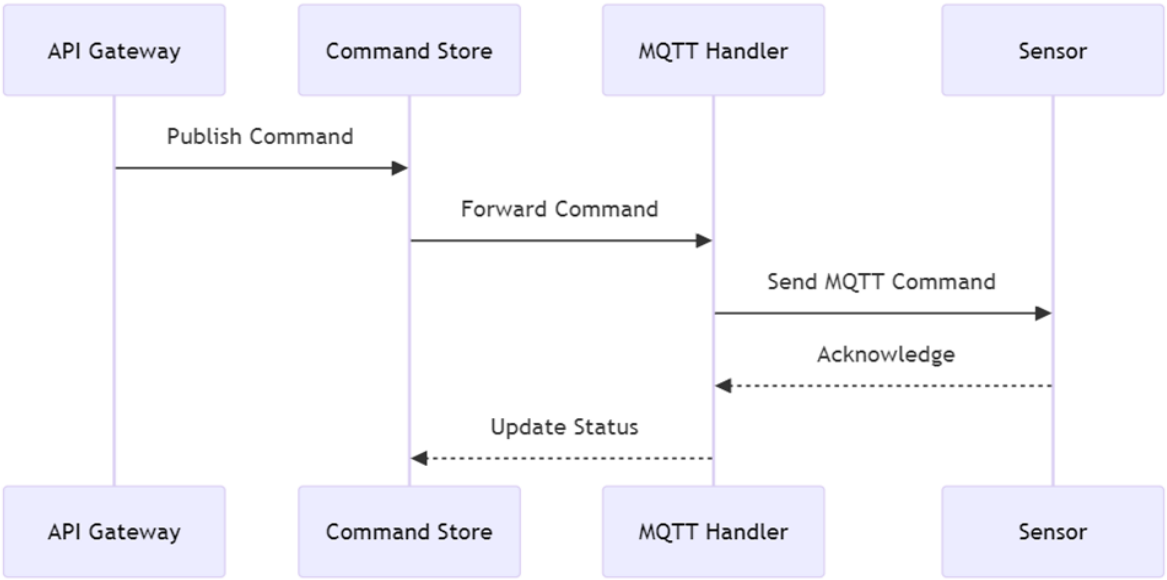

Рисунок 5 – Доставка команды пользователя к датчику с использованием протокола MQTT (публикация команды)

Обработчик MQTT публикует ровно одну команду и ждет, пока ее прочитают, и только после этого можно публиковать другую. Если же используется протокол HTTP и сервис «HTTP-обработчик команд датчиков», получение команды инициируется датчиком, который

периодически опрашивает сервис «HTTP-обработчик команд датчиков». Сервис проверяет наличие новых команд в хранилище команд и, если они есть, передает их на датчик.

4. Обработка шаблонов:

- Для новых типов датчиков шаблоны управляются «Реестром шаблонов датчиков» (sensorset-sensor-template-registry), который помогает стандартизировать форматы данных.

5. Взаимодействие с пользователями:

- Пользователи могут взаимодействовать с системой через веб- или мобильный интерфейс. Их запросы сначала попадают в «шлюз APISensorset» (sensorset-api-gateway).
- «Шлюз API Sensorset» может направить эти запросы в сервис «Sensorset User» (sensorset-user) для действий, связанных с информацией о пользователях.
- Аутентификация пользователя обрабатывается сервисом «Sensorset User Auth» (sensorset-user) с участием «обработчика аутентификации SensorsetoAPIkey» (sensorset-apikey-authentication-handler) (рис.6).

6. «Безопасность и аутентификация:»

- Все взаимодействия защищены, «обработчик аутентификации Sensorset APIkey» (sensorset-apikey-authentication-handler) гарантирует обработку только аутентифицированных запросов.
- Аутентификация пользователя может включать проверку токенов JWT, гарантируя, что у пользователя есть необходимые разрешения для выполнения запрашиваемых действий.

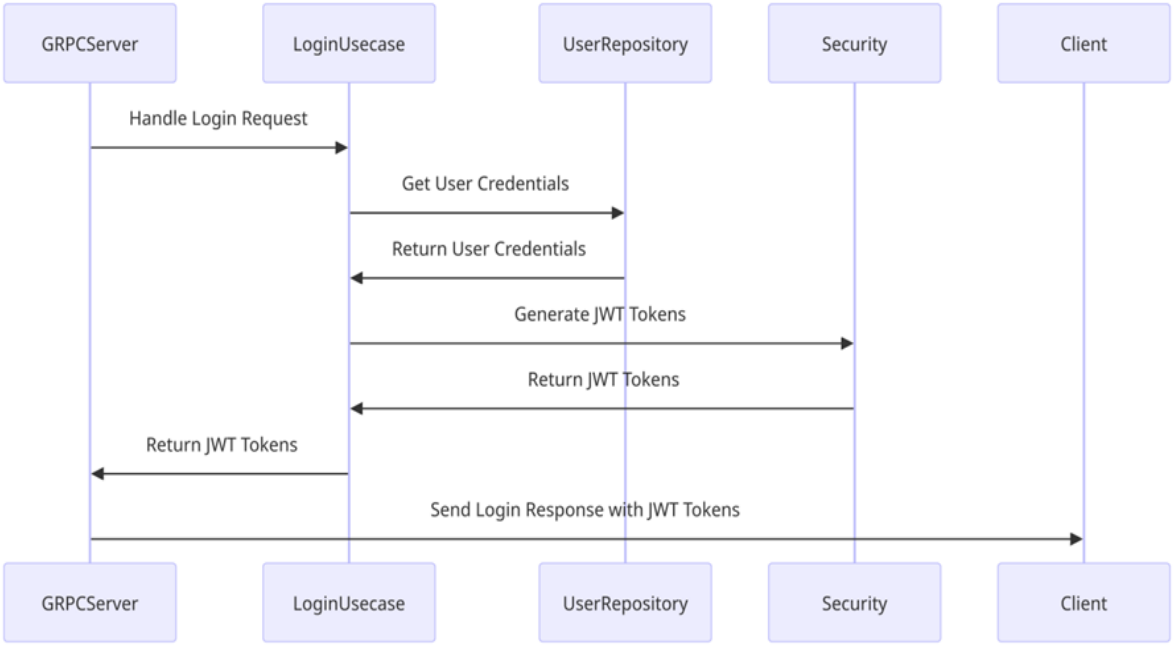

Рисунок 6 – Аутентификация пользователя. Диаграмма последовательности

7. Получение и обработка данных:

- Пользователи могут запрашивать данные датчиков, что включает запросы к «хранилищу данных датчиков» (sensorset-sensor-data-store).

- Полученные данные могут проходить процесс трансформации или агрегации, в зависимости от запроса пользователя, перед тем как быть отправленными обратно пользователю.

8. Внешняя интеграция:

- Система может взаимодействовать с внешними системами или API для дополнительных функций, таких как интеграция с другими платформами IoT или отправка уведомлений.

Этот поток демонстрирует всестороннее взаимодействие между различными сервисами в системе управления данными датчиков. От первоначального получения данных датчика до их обработки, хранения и последующего извлечения пользователями каждый сервис выполняет определенную роль, обеспечивая эффективную и безопасную работу системы. Использование шлюза API как на уровне датчика, так и на уровне пользователя упрощает маршрутизацию запросов, в то время как специализированные сервисы выполняют конкретные задачи, такие как нормализация данных, аутентификация пользователя и обработка команд. Этот подход способствует масштабируемости и надежности в дизайне системы.

Используемые протоколы. Протокол MQTT (Message Queuing Telemetry Transport**)** является стандартным протоколом межмашинного обмена в Интернете вещей (InternetofThings, IoT), используется для связи с датчиками и актуаторами, специально предназначен для надежной передачи сообщений в ненадежных сетях, сетях низкой и ограниченной пропускной способности, в условиях ограниченной мощности и времени автономной работы устройств, его применение помогает минимизировать размер сообщений и сетевой трафик [9]. Протокол работает по принципу publisher/subscriber (издатель/подписчик). Кроме сервера-издателя и клиентов, в организации связи участвуют серверы-брокеры, которых может быть множество.

Сообщения не посылаются прямо клиентам, а публикуются у брокера по темам – топикам (topic), поэтому клиентам не нужны адреса. Брокер фильтрует сообщение по топику и доставляет его подписчикам, то есть, чтобы получить сообщение, клиент должен быть подписан на данный топик у данного брокера. Прямого соединения между издателем и подписчиком нет, но соединение между клиентом и брокером должно быть установлено. Для соединения с брокером используется протокол TCP/IP. Брокер присылает клиенту подтверждение соединения, подписки и публикации. После установки соединения клиенты могут публиковать (широковещательная публикация) и получать сообщения [9]. Клиент через определенные интервалы времени (обычно 60 с) для проверки соединения посылает брокеру сообщения, подтверждающие сохранение соединения (keepa live message) .

Формат сообщения MQTT – сообщение содержит два служебных байта, длина сообщения – до 256 байт:

- байт 1: тип сообщения (запрос клиента на подключение, подтверждение подписки, запрос ping и т.д.), флаг дублирования, инструкции для сохранения сообщений и информацию об уровне качества обслуживания (Qualityof Service, QoS);
- байт 2: оставшаяся длина сообщения.

Флаги QoS содержат значения, означающие условия доставки сообщения: 0 – сообщение доставляется не более одного раза, подтверждение доставки не требуется, то есть возможна потеря или дублирование сообщения; 1 – сообщение доставляется по крайней мере один раз: доставка подтверждается клиентом, возможна повторная отправка сообщения; 2 – сообщение будет гарантированно доставлено ровно один раз, брокер обеспечивает доставку, потеря или дублирование исключены. В приложении используется брокер EMQX [11].

Протокол AMQP (Advanced Message Queuing Protocol**) –** открытый протокол для передачи сообщений между компонентами системы с низкой задержкой и на высокой скорости**,** настраиваемый под конкретное приложение. Три главных компонента AMQP – сообщение (message), точка обмена (exchange) и очередь (queue) [10]. Сообщение – единица

обмена, содержимое безразлично серверу, может содержать заголовки. Точка обмена – место получения и выдачи сообщений, распределяет сообщения по очередям. Сообщения хранятся в очередях, пока клиент не заберет их.

Типы точек обмена: fanout – сообщение передается во все очереди; direct – сообщение передается в очередь с именем, совпадающим с ключом маршрутизации, указанным при отправке сообщения; topic – сообщение передается в очереди, для которых совпадает маска на ключ маршрутизации.

Брокеры сообщений. Брокеры сообщений – архитектурный шаблон, используемый в распределенных приложениях. Брокеры являются посредниками для общения между разными модулями системы, а особенно между микросервисами, так как микросервисы могут использовать разные форматы и протоколы сообщений. Брокер может, например, принимать сообщение от микросервиса-источника по одному протоколу в одном формате, преобразовать его в формат для макросервиса-приемника и передать его приемнику по другому протоколу. Кроме преобразования форматов, брокер может выполнять проверку на ошибки, сохранять сообщения, маршрутизировать сообщение приемникам или публиковать подписчикам. Некоторые брокеры могут балансировать нагрузку [13].

В системе используются брокеры сообщений EMQX [11] и RabbitMQ[12], но могут быть подключены и другие брокеры.

EMQX на настоящий момент является самым популярным высокопроизводительным масштабируемым MQTT-брокером, используемым для межмашинного взаимодействия. EMQX может поддерживать более чем 100 миллионов одновременных соединений на одном кластере с задержкой в 1 миллисекунду, а также принимать и обрабатывать миллионы MQTT сообщений в секунду [11].

RabbitMQ — программный брокер сообщений на основе AMQP. Включает сервер, библиотеки протоколов (HTTPи других), клиентские библиотеки AMQP для Java и .NETFramework для разных языков программированияи плагины, например, плагины мониторинга, управления через веб-интерфейс. Может поддерживать около 50 тысяч сообщений в секунду (зависит от конфигурации) [12].

Аутентификация датчиков. Чтобы обсудить интеграцию с EMQX, популярным брокером MQTT, рассмотрим, как в системе обрабатывается аутентификация датчиков. Сначала датчики должны быть зарегистрированы в системе.

1. Начальная регистрация:
   - Во время регистрации каждому датчику присваиваются уникальные учетные данные – клиентский ID и секретный ключ, которые необходимы для безопасного общения.
2. Процесс аутентификации:
   - Когда датчик пытается подключиться к системе (например, через “шлюз API датчиков”), он должен предоставить свои учетные данные.
   - Учетные данные проверяются “обработчиком аутентификации” Sensorset API key.
   - Этот обработчик может взаимодействовать с сервером аутентификации для проверки учетных данных датчика.
3. Безопасное общение: общение с системой разрешается датчику только после успешной аутентификации.

Интеграция с EMQX.

1. Получение данных от датчиков через MQTT (рис.4):
   - Датчики часто отправляют данные, используя MQTT, легкий протокол обмена сообщениями, идеально подходящий для устройств.
   - EMQX, будучи масштабируемым брокером MQTT, хорошо подходит для обработки сообщений MQTT в этом сценарии.

- MQTT-обработчик датчиков в системе предназначен для взаимодействия с EMQX, получая сообщения MQTT от датчиков.

2. Маршрутизация сообщений:

- Как только MQTT-обработчик датчиков получает данные от EMQX, он может направить эти данные в другие сервисы для обработки.
- Например, данные могут быть отправлены в “нормализатор данных датчиков” (sensorset-sensor-data-normalizer) для стандартизации, а затем храниться в “хранилище данных датчиков” (sensorset-sensor-data-store).

3. Обработка команд датчиков:

- Система также может использовать EMQX для отправки команд датчикам.
- Команды, выпущенные через систему, через sensorset-sensor-mqtt-handler ,направляются в EMQX, который затем доставляет эти команды соответствующим датчикам (рис.5).

4. Масштабируемость и надежность: EMQX поддерживает кластеризацию и может обрабатывать высокий объем данных, что делает его подходящим для масштабируемых решений.

Система может использовать эти возможности EMQX для эффективной и надежной обработки данных датчиков. Интеграция с EMQX позволяет эффективно обрабатывать сообщения MQTT как для получения данных от датчиков, так и для отправки команд датчикам. Эта конфигурация обеспечивает надежную и масштабируемую инфраструктуру для управления коммуникациями датчиков.

Заключение. Микросервисная архитектура обеспечивает гибкость, надежность и масштабируемость приложения. Использование протоколов MQTT, AMQP с брокерами EMQX и RabbitMQ предоставляет возможность надежной передачи данных в системе, а использование токенов JWT для аутентификации пользователей и датчиков обеспечивает безопасное общение между сервисами. В дальнейшем необходимо провести нагрузочное тестирование системы и проработать вопросы горизонтального масштабирования сервисов, брокеров и баз данных. Возможно, понадобится распределение данных по нескольким серверам баз данных (партиционирование), добавление master-slaveузлов, добавление балансировщиков нагрузки для устранения «узких мест». Для обмена данными с другими платформами может потребоваться разработка специальных сервисов. Большинство микросервисов реализованы на языке Go [14], но могут быть использованы и другие языки, для построения диаграмм использовался онлайн-сервис Mermaid [15].

Литература

1. Авельцов Д.О. Применение микросервисной архитектуры в разработке программного обеспечения системы мониторинга параметров окружающей среды/ Проблемы автоматики и управления. – 2019. — No2 (37).
2. JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. / URL: https://jwt.io/ (датаобращения 25.10.2023)
3. Авельцов Д.О., Гайдамако В.В., Крец Н.А., Лыченко Н.М. Сервис-ориентированная архитектура облачной информационно-измерительной системы экологического мониторинга (в настоящем сборнике)
4. HTTP/RESThttps://dzone.com/refcardz/rest-foundations-restful?chapter=1(датаобращения 25.10.2023)
5. IntroducingJSON / URL:https://www.json.org/json-en.html (датаобращения 25.10.2023)
6. gRPC: A high performance, open source universal RPC framework / https://grpc.io/(датаобращения 25.10.2023)

7. ProtocolBuffers. / https://protobuf.dev/(дата обращения 25.10.2023)
8. Авельцов Д.О. Применение протокола сериализации структурированных данных protobuf в микросервисной архитектуре // Проблемы автоматики и управления. – 2022. – № 3 (45). – С. 185–196.
9. MQTT - The Standard for IoT Messaging/ URL: https://mqtt.org/(дата обращения 25.10.2023)
10. AMQP is the Internet Protocol for Business Messaging / URL:https://www.amqp.org/(датаобращения 25.10.2023)
11. EMQX: The World's #1 Open Source Distributed MQTT Broker/ https://www.emqx.io/(дата обращения 25.10.2023)
12. RabbitMQ is the most widely deployed open source message broker / URL: https://www.rabbitmq.com/(дата обращения 25.10.2023)
13. Брокеры сообщений — что это, из чего состоят, плюсы и минусы: сравниваем apachekafka, redis и rabbitmq. / URL:https://academy.mediasoft.team/article/brokery-soobshenii-chto-eto-iz-chego-sostoyat-plyusy-i-minusy-sravnivaem-apache-kafka-redis-i-rabbitmq/ (дата обращения 25.10.2023)
14. The Go Programming Language / URL: https://go.dev/URL:https://mermaid.js.org/(датаобращения 25.10.2023)
15. Mermaid Diagramming and charting tool / URL:https://mermaid.js.org/(дата обращения 25.10.2023)