ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

УДК 004

Авельцов Д.О., Гайдамако В.В., Крец Н.А., Лыченко Н.М.
Институт машиноведения и автоматики НАН КР,
Кыргызско-Российский славянский университет, Бишкек, Кыргызстан
E-mail: dolpha@gmail.com, nlychenko@mail.ru

СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА ОБЛАЧНОЙ ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНОЙ СИСТЕМЫ ЭКОЛОГИЧЕСКОГО МОНИТОРИНГА

Настоящая работа связана с комплексом работ по созданию информационно-измерительных систем мониторинга параметров окружающей среды для эффективного решения экологических проблем и обеспечения доступности и оперативности получения информации. В работе представлена сервис-ориентированная архитектура облачной информационно-измерительной системы экологического мониторинга, представляющая собой набор сервисов, каждый из которых решает отдельную задачу, однако все они работают для достижения общей цели функционирования системы. Это позволяет улучшить оперативность и доступность информации, а также улучшает масштабируемость и гибкость системы мониторинга.

Ключевые слова: облачные технологии, датчики, интерфейс прикладного программирования (API), микросервисы, протокол передачи данных gRPC.

Введение. Разработка облачных информационно-измерительных систем (ОИИС) экологического мониторинга является необходимой по ряду причин. Во-первых, загрязнение воздуха, воды и почвы, изменение климата и другие современные экологические проблемы требуют систематического и точного мониторинга для эффективного управления и предотвращения негативных последствий этих явлений [1, 2]. Традиционные методы сбора данных, основанные на ручном сборе информации или использовании ограниченного числа стационарных датчиков, ограничены по объему данных и масштабу покрытия. ОИИС позволяет собирать и анализировать большие объемы данных с использованием распределенной инфраструктуры, что обеспечивает более полное и всестороннее представление экологической ситуации. Во-вторых, разработка такой системы позволяет улучшить доступность и оперативность получения информации. Облачные технологии позволяют передавать данные в режиме реального времени, а также предоставлять удаленный доступ к системе для мониторинга и анализа данных из любой точки мира. Это особенно важно для решения экологических проблем, требующих незамедлительных реакций и оперативного принятия решений. В-третьих, ОИИС экологического мониторинга может быть адаптирована под различные потребности и требования пользователей. Такой подход позволяет широко использовать систему в различных отраслях, включая промышленность, сельское хозяйство, градостроительство и другие, и эффективно решать специфические экологические проблемы каждой отрасли. В-четвертых, разработка ОИИС экологического мониторинга способствует развитию и применению инновационных технологий в экологической сфере. Она объединяет в себе преимущества сенсорных технологий, облачных вычислений, аналитики больших данных и визуализации, что позволяет получать глубокое понимание экологических процессов и разрабатывать более эффективные стратегии управления окружающей средой.

Обобщенная архитектура информационно-измерительных систем, использующих облачные технологии для организации сбора, хранения, обработки и предоставления информации, управления физическими и виртуальными датчиками, предоставления инфраструктуры сбора данных как сервиса (SensIaaS – Sensor on IaaS – инфраструктура датчиков как услуга) и измерения как сервиса (SCaaS – Sensing-as-a-Service), представлена в [3,4]. В [5] представлен вариант проектирования и разработки Web-портала экологической информации Кыргызской Республики, который включает программный инструментарий для

организации доступа к открытым источникам информации о концентрациях твердых частиц PM2.5, PM10 и индексе качества воздуха, модули для регистрации датчиков пользователей, сбора и сохранения измерительных данных, а также картографического отображения результатов мониторинга параметров окружающей среды.

Настоящая работа представляет сервис-ориентированное архитектурное решение системы экологического мониторинга, основными функциями которой являются [5]:

- хранение, изменение и своевременное обновление информации, поступающей с датчиков,
- предоставление актуальных данных об измерениях экологических и метеорологических параметров окружающей среды, а также о датчиках на интерактивной карте КР,
- добавление датчиков или изменение/удаление уже имеющихся,
- загрузка исторических данных по выбранным датчикам,
- загрузка данных с использованием API сторонних систем,
- регистрация пользователей в системе с целью их доступа к дополнительным возможностям системы (регистрация личного датчика),
- уведомление пользователей о различных событиях.

В данной работе стоит задача спроектировать и создать эволюционный прототип системы экологического мониторинга как системы интеграции сервисов, каждый из которых решает отдельную задачу, однако все они работают для достижения общей цели функционирования системы. Это позволит улучшить оперативность и доступность информации, надежность передачи данных, а также масштабируемость и гибкость системы мониторинга.

ОИИС как решение для системы экологического мониторинга. ОИИС имеет распределенную инфраструктуру, схожую с IoT. Интернет вещей (IoT) предоставляет возможность связывать сенсорные устройства, предметы и системы, обмениваться данными и взаимодействовать друг с другом. Это создает новые возможности для сбора и передачи данных в режиме реального времени, а также для автоматизации процессов с целью оптимизации экологического мониторинга и управления. Стандартная модель инфраструктуры IoT включает в себя датчики различных типов, объединенных в сети и концентраторы (шлюзы, хабы) – устройства для связи с глобальной сетью. Далее информация передается на облачную платформу IoT, которая также может быть интегрирована с другими сервисами/платформами. Соответствующие модули обеспечивают связь с пользовательскими приложениями − как проводную со стационарными десктопами, так и беспроводную с мобильными устройствами пользователей [6,7].

Облачные информационно-измерительные системы так же, как и IoT, позволяют собирать данные с различных сенсорных устройств (включая датчики качества воздуха, воды, почвы, метеостанции и другие), равномерно размещенных на больших территориях, и передавать их в облачное хранилище для дальнейшей обработки и анализа для выявления трендов и прогнозирования, имея возможность при этом использовать мощные вычислительные ресурсы. После этого информация становится доступной для пользователей через веб-интерфейсы, мобильные приложения и другие средства визуализации данных. Это позволяет оперативно реагировать на экологические изменения, принимать эффективные меры по охране окружающей среды и обеспечивать информированное принятие решений.

Сервис-ориентированная архитектура ОИИС мониторинга. Разрабатываемая система основывается на микросервисной архитектуре программного обеспечения. Микросервисная архитектура, как вариант сервис-ориентированной архитектуры, это способ представления программного обеспечения в виде набора сервисов – небольших, слабо связанных и легко изменяемых модулей. Каждый сервис решает отдельную задачу, представляет собой самостоятельный процесс, может быть написан на наиболее подходящем для задачи языке программирования и развернут в независимой среде, то есть в контейнере. Набор таких сервисов представляет собой единую систему, все сервисы работают на достижение общей

цели, и, скорее всего, им будет необходимо обмениваться информацией, как минимум, для синхронизации [8,9]. Архитектура ОИИС мониторинга представлена на рисунке 1 в виде UML-диаграммы компонентов.

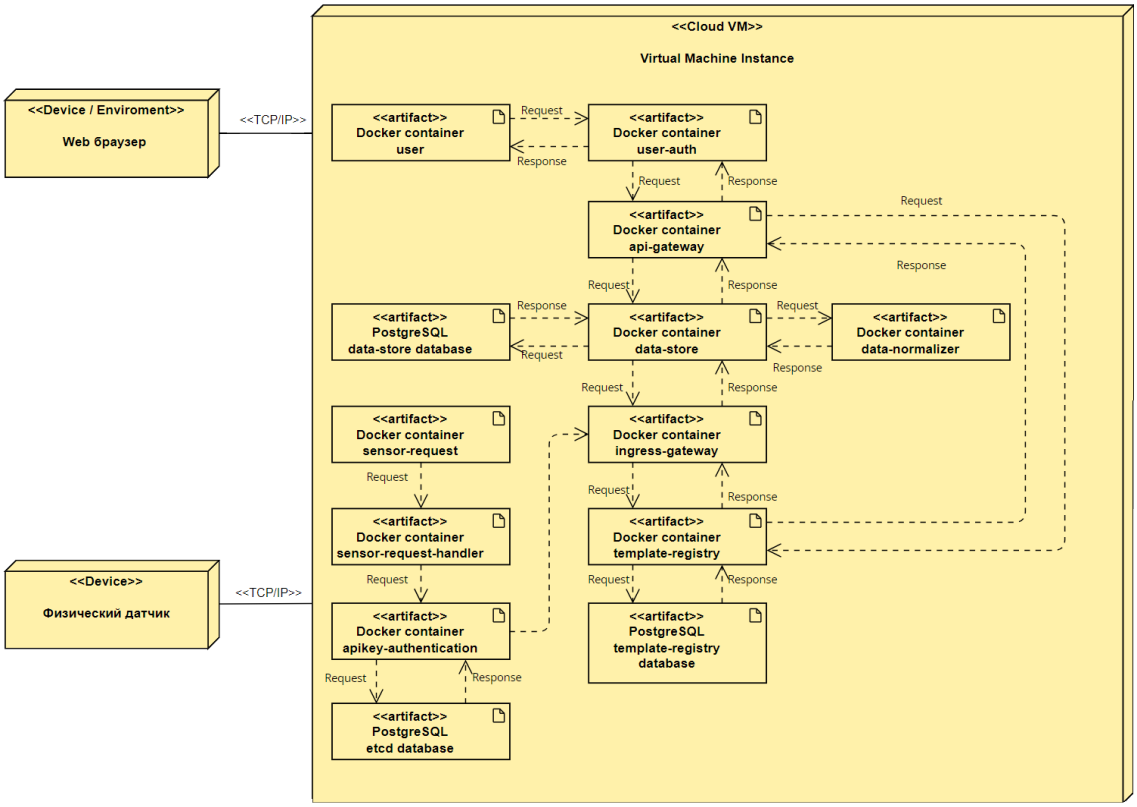

Рисунок 1 – Архитектура ОИИС мониторинга

На этой диаграмме показаны взаимосвязи между основными компонентами системы:

- Web-браузер – через него пользователь контактирует с предоставляемым системой функционалом;
- Sensors – физические датчики, зарегистрированные в системе экологического мониторинга;
- Virtual Machine Instance — экземпляр виртуальной машины, гибкая и масштабируемая облачная инфраструктура, предоставляющая различные сервисы, упакованные в контейнеры Docker [10] .

Детализация сервисов ОИИС

Sensorset-apikey-authentication-handler. Сервис предназначен для обработки аутентификации API-ключей для системы сенсоров [11]. Его основная цель – обеспечить проверку наличия ключа в распределенном хранилище данных etcd. Этот сервис важен для обеспечения безопасности и контроля доступа к системе сенсоров, что особенно важно при работе с чувствительными данными. Во время своей работы сервис принимает запросы на аутентификацию, которые передаются через gRPC-протокол (gRPC Remote Procedure Calls [12]). Запросы содержат API-ключ, который необходимо проверить. Процесс работы сервиса следующий. Получив запрос на аутентификацию, сервис осуществляет поиск ключа в базе данных etcd. В случае успешного поиска сервис возвращает ответ, указывающий на то, что ключ был найден. И возвращает ID сенсора.

Для коммуникации с клиентами и внешним миром используется протокол gRPC. Сам обмен данными осуществляется в формате Protocol Buffers, который определен в proto-файлах. Это обеспечивает надежность, эффективность и гибкость в обмене данными.

Сервис связан с двумя основными внешними компонентами: etcd (сервис PostgreSQL etcd database, база данных, где хранятся ключи), и системами, которые отправляют запросы на аутентификацию, поскольку нуждаются в проверке ключей для доступа к системе. Основной код сервиса написан на языке Go, что обеспечивает высокую производительность и параллельность в обработке запросов.

Sensorset-api-gateway. Сервис представляет собой API-шлюз для работы с устройствами и их данными, он написан на языке Go и использует библиотеки Swagger для API и gRPC для общения с другими сервисами [13].

Сервис предназначен для управления устройствами и для получения данных с этих устройств. Этот сервис обеспечивает связь между клиентами, которыми могут быть как пользователи, так и другие системы, и более низкоуровневыми сервисами, которые обрабатывают данные устройств и обеспечивают функциональность управления устройствами. Основные операции, которые этот сервис может выполнить, включают:

- Создание устройства.
- Получение информации об устройстве.
- Обновление информации об устройстве.
- Удаление устройства.

Все эти операции реализуются через REST API (Representational State Transfer – передача репрезентативного состояния – способ создания API с помощью протокола HTTP), в котором эндпоинты соответствуют операциям, и они возвращают соответствующие HTTP-ответы.

Для реализации этих операций сервис обращается к другому сервису через gRPC. Этот другой сервис, который называется Sensor Registry Service, является тем, кто в действительности создает, получает, обновляет и удаляет информацию об устройствах. Кроме того, сервис имеет еще одну функцию – Sensor Data Interactor, которая обеспечивает доступ к данным устройств. Этот интерфейс предоставляет методы для получения данных от одного или нескольких устройств, но в текущем коде эти методы не используются.

Протоколы общения включают HTTP/REST для внешних клиентов и gRPC для внутреннего общения с другими сервисами.

В общем, этот сервис – это своего рода "фасад", который предоставляет удобный API для работы с устройствами и их данными, скрывая детали реализации и внутреннюю структуру системы от клиентов [13].

Следующие точки доступа представлены в API:

- POST /api/v1/devices: Создает новое устройство. В теле запроса необходимо предоставить объект устройства;
- GET /api/v1/devices/{deviceId}: Возвращает информацию о конкретном устройстве по его идентификатору;
- PUT /api/v1/devices/{deviceId}: Обновляет информацию о конкретном устройстве по его идентификатору. Обновленный объект устройства должен быть предоставлен в теле запроса;
- DELETE /api/v1/devices/{deviceId}: Удаляет конкретное устройство по его идентификатору;
- GET /api/v1/users/{userId}/devices: Возвращает список устройств для конкретного пользователя. Можно использовать параметры page и pageSize для управления пагинацией;
- GET /api/v1/users/{userId}/devices/data: Возвращает данные от устройств конкретного пользователя. Параметры page и pageSize можно использовать для управления пагинацией;
- GET /api/v1/devices/{deviceId}/data: Возвращает данные от конкретного устройства;
- GET /api/v1/devices/data/multiple: Возвращает данные для нескольких устройств. Идентификаторы устройств должны быть предоставлены в параметре deviceIds запроса.

Объект Device содержит свойства, такие как идентификатор, ключ сенсора, имя, описание, шаблон, формат, местоположение, владелец и статус удаления. Объект Device Data содержит свойства, такие как идентификатор, идентификатор устройства, метка времени и данные.

Sensorset-sensor-template-registry. Сервис является хранилищем шаблонов сенсоров.

В центре сервиса находится объект Template, который представляет собой модель шаблона сенсора. Этот объект включает в себя различные атрибуты, такие как идентификаторы, ключ сенсора, идентификатор пользователя, формат, шаблон, описание, имя, адрес и информацию о его создании и обновлении.

Сервис предоставляет набор взаимодействий для работы с шаблонами сенсоров, а именно:

- Создание шаблона сенсора (CreateSensor).
- Получение информации о шаблоне сенсора (GetSensorInfo и GetSensorFormat).
- Обновление шаблона сенсора (UpdateSensor).
- Удаление шаблона сенсора (DeleteSensor).

Эти действия представлены в виде протоколов gRPC в файлах proto. Сервисы gRPC предоставляют RPC (Remote Procedure Call) интерфейс для взаимодействия с сервисом [12].

В нашем случае есть два основных сервиса: Registry Service и Sensor Metadata Service. Registry Service предоставляет метод Description By Sensor ID, который позволяет получить описание и другую информацию о шаблоне сенсора по его идентификатору. Sensor Metadata Service предоставляет более широкий функционал, позволяющий создавать, обновлять, удалять и получать информацию о шаблонах сенсоров.

Коммуникация между этим сервисом и другими сервисами осуществляется через протокол gRPC.

Сервис использует базу данных PostgreSQL для хранения данных о шаблонах сенсоров. Взаимодействие с базой данных организовано через репозиторий Template Repository.

В целом этот сервис является критически важным для обеспечения функциональности системы мониторинга. Он обеспечивает управление шаблонами сенсоров, что позволяет настроить и контролировать работу сенсоров в зависимости от потребностей пользователей или системы.

Sensorset-sensor-data-normalizer. Сервис является нормализатором данных от датчиков. Он использует gRPC для обмена данными и взаимодействия с другими сервисами.

Суть работы сервиса заключается в следующем: он принимает данные от датчиков в различных форматах, нормализует их, приводит к единому формату (в нашем случае к формату JSON [14]) и отправляет дальше. Важной особенностью сервиса является то, что он не только нормализует данные, но и использует собственные шаблоны для преобразования данных от датчиков. Эти шаблоны хранятся в отдельном сервисе "Sensor Template Registry GRPC", с которым сервис "sensorset-sensor-data-normalizer" связывается через gRPC.

Основной поток выполнения работы сервиса начинается в функции main(). Здесь сервис устанавливает gRPC соединение с сервисом шаблонов ("Sensor Template Registry GRPC"), затем создает экземпляр сервера gRPC и начинает слушать входящие запросы на определенном порту. Когда сервис получает данные от датчика через gRPC (через метод Normalize()), он передает эти данные в свою основную бизнес-логику, реализованную в методе Execute(). Здесь сервис получает информацию о датчике и использует эту информацию для нормализации входных данных и приведения их к формату JSON.

Важно отметить, что, несмотря на то, что сервис находится в стадии разработки, его основной функционал и взаимодействие с другими сервисами через gRPC уже реализованы. При этом остается добавить логику нормализации данных, которая будет основываться на шаблонах, полученных из сервиса "Sensor Template Registry GRPC".

Этот сервис играет важную роль в общей системе обработки данных от датчиков, обеспечивая их нормализацию и приведение к единому формату для дальнейшей обработки и анализа.

Sensorset-sensor-data-store. Сервис "sensorset-sensor-data-store" представляет собой систему для сбора и обработки данных от сенсоров. Этот сервис осуществляет обработку и сохранение данных, полученных от сенсоров, в базу данных PostgreSQL.

Сервис разработан на языке Go и использует следующие внешние сервисы и протоколы:

- PostgreSQL – для хранения данных.
- RabbitMQ – для приема сообщений с данными от сенсоров.
- gRPC – для вызова внешнего сервиса нормализации данных.
- Основной рабочий процесс следующий:
- Сервис начинает прослушивать RabbitMQ на наличие новых сообщений.
- При получении сообщения, содержащего данные от сенсора, сервис начинает обработку этих данных.
- Сначала данные отправляются на внешний сервис нормализации посредством gRPC вызова. Этот сервис преобразует данные в унифицированный формат. Затем нормализованные данные сохраняются в базу данных PostgreSQL. При возникновении ошибок во время обработки сообщения они логируются.

В коде представлены модели, репозитории и контроллеры для работы с данными сенсоров. Основная модель — это "Sensor Data", которая содержит информацию о данных от сенсора, такую как ID сенсора, данные от сенсора, timestamp, идентификатор пользователя, нормализованные данные.

База данных PostgreSQL используется для хранения данных сенсоров. Соответствующий репозиторий содержит методы для добавления данных сенсора в базу данных и для получения последних данных по идентификатору сенсора.

Сервис использует RabbitMQ для прослушивания данных сенсора и вызывает соответствующую функцию обработчика для обработки полученных данных. Обработчик преобразует полученные данные в JSON-формат и сохраняет их в базу данных с помощью функции use case.

Помимо этого, существует gRPC-клиент, который взаимодействует с внешним сервисом нормализации для преобразования данных сенсора в унифицированный формат.

В целом сервис служит для сбора и обработки данных сенсоров, обеспечивая их сохранность и нормализацию для дальнейшего анализа и использования.

Sensorset-sensor-ingress-gateway. Этот сервис представляет собой шлюз приема данных от датчиков. Он служит для приема сообщений из очереди сообщений RabbitMQ, проверки идентификатора датчика и передачи данных в другую очередь в случае успешной проверки.

В основном инициируется подключение к серверу RabbitMQ телепрограмм, затем создается канал и очередь под названием "sensorset-sensor-request-handler". После установки соединения программа начинает получать сообщения из этой очереди.

Сообщение, полученное из очереди, представляет собой JSON-объект, который включает в себя идентификатор и идентификатор устройства (датчика). Данные разбираются и затем передаются на gRPC-сервис для проверки идентификатора устройства.

Сервис, к которому обращается данная программа через gRPC, называется "sensorset-apikey-authentication-handler". Этот сервис используется для аутентификации идентификаторов устройств, полученных от датчиков.

Если ответ от gRPC сервиса положительный (т.е., идентификатор устройства найден), данные, полученные от датчика, передаются в другую очередь RabbitMQ, называемую

"sensorset-sensor-data-to-db". Это гарантирует, что только данные от аутентифицированных датчиков будут переданы на следующий этап обработки.

Сервис использует протоколы AMQP для взаимодействия с RabbitMQ и gRPC для взаимодействия с сервисом проверки идентификаторов устройств.

Важно отметить, что программа выполняет бесконечное чтение сообщений из очереди, что означает, что она будет продолжать работать до тех пор, пока ее явно не остановят. Этот сервис важен для обеспечения контроля над данными, получаемыми от датчиков, так как он убеждается, что данные, которые попадают в систему, проходят проверку аутентификации.

Sensorset-sensor-request-handler. Данный сервис является обработчиком запросов с датчиков. Он слушает входящие соединения через порт, указанный в переменной окружения "SENSOR_REQUEST_HANDLER_SERVICE_PORT", по умолчанию это порт 8080.

Сервис состоит из нескольких следующих важных компонентов.

Парсер идентификаторов (idparser): его задача — находить идентификаторы в различных частях запроса. Он может искать идентификаторы в пути запроса, в параметрах запроса и в теле запроса. Идентификаторы для поиска передаются в переменной окружения "SENSOR_IDENTIFIER_PARAMETER_NAMES".

Обработчики запросов: HTTPHandler и TCPHandler. Они используют парсер идентификаторов для получения информации из запросов и формирования данных запроса для дальнейшей обработки.

Обработчик соединения: его задача — получать входящие соединения и обрабатывать их с помощью соответствующего обработчика запросов. Если во время обработки возникают ошибки, они логируются и возвращаются в ответ на запрос.

WorkerPoolServer: эта часть сервиса отвечает за распределение входящих соединений между доступными рабочими потоками.

Publisher: этот компонент используется для публикации обработанных данных запроса в очередь сообщений. В настоящий момент имя очереди жестко задано в коде как "sensorset-sensor-request-handler", но это можно исправить, перенеся его в конфигурацию.

Сервис имеет связь с другими сервисами системы обмена сообщениями RabbitMQ. Общая цель сервиса заключается в получении запросов от датчиков, извлечении из них полезной информации и передачи этой информации в систему обмена сообщениями для дальнейшей обработки другими сервисами.

На рисунке 2 представлен алгоритм взаимодействия сервисов, обслуживающих датчики.

Заключение. Таким образом, в работе представлена сервис-ориентированная архитектура облачной информационно-измерительной системы, предназначенной для организации сбора, хранения и представления данных мониторинга параметров окружающей среды. Применение облачных технологий в экологическом мониторинге обеспечивает возможность взаимодействия с различными устройствами и системами, что позволяет улучшить оперативность и доступность информации, а также улучшает масштабируемость и гибкость системы мониторинга. Более того, удаленный доступ к информации через облачные технологии обеспечивает оперативное реагирование на экологические ситуации и возможность принимать меры по управлению экологическими ресурсами в реальном времени. Одно из преимуществ использования облачных технологий в экологическом мониторинге заключается в возможности распределенного размещения сенсорных устройств в различных местах, включая удаленные и труднодоступные регионы, используя для этих целей в том числе сети наблюдателей-добровольцев (crowdsourcing). Это позволит охватить большую территорию и получать данные из различных экологических точек, что в свою очередь обеспечивает более полное представление о состоянии окружающей среды.

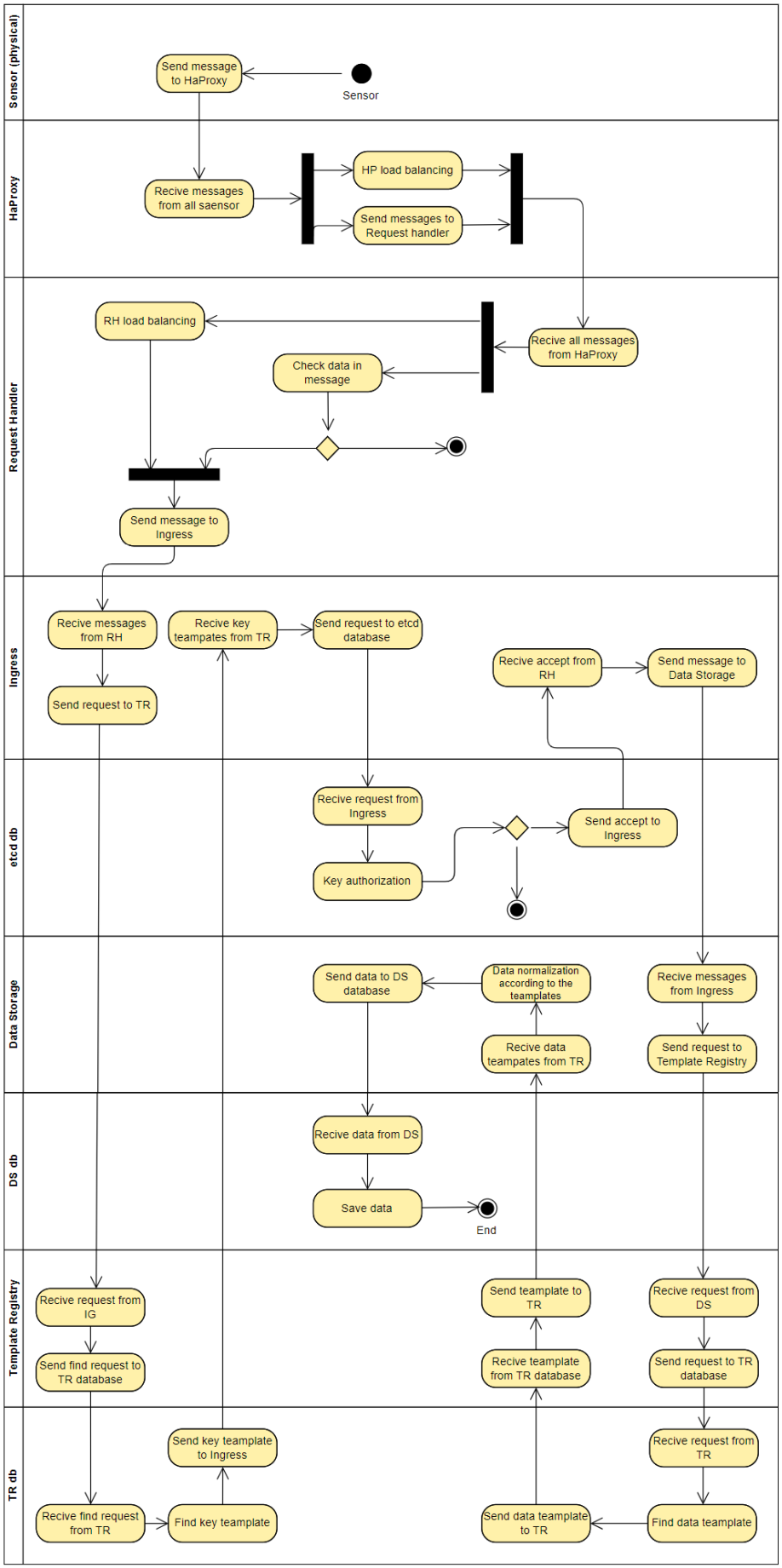

Рисунок 2 Алгоритм взаимодействия сервисов, обслуживающих датчики

Литература

1. ПРООН и ЮНЕП (2022). Качество воздуха в Бишкеке: Оценка источников выбросов и дорожная карта для содействия управлению качеством воздуха. Бишкек и Найроби.
2. Официальный сайт Госагентства охраны окружающей среды и лесного хозяйства при правительстве КР // http://ecology.gov.kg/news/view/id/269) (дата обращения: 25.09.2023).
3. Гайдамако, В. В. Инфраструктура sensor-cloud - облачные информационно-измерительные системы / В. В. Гайдамако // Проблемы автоматики и управления. – 2018. – № 2(35). – С. 109 – 118. – EDN YTDTNZ.
4. Dr. Lokesh A, Dr. Mohamed Saleem, Bhupendra Kumar Swar, Saurav Kumar,Rahul Sharma. Cloud computing: the emerging technology // International Research Journal of Modernization in Engineering Technology and Science (Peer-Reviewed, Open Access, Fully Refereed International Journal) Volume:04/Issue:10/October-2022
5. Разработка Web-портала экологической информации Кыргызской Республики / В. В. Гайдамако, Б. К. Каныбеков, Н. М. Лыченко, Д. А. Текеев // Проблемы автоматики и управления. – 2022. – № 3(45). – С. 74– 83. – EDN QYSVOQ.
6. https://oncloud.ru/blog/Ot-Interneta-veshchej-k-Internetu-vsego (дата обращения 01.11.2023)
7. Что такое Интернет вещей (Internet of Things, IoT) // https://aws.amazon.com/ru/what-is/iot/ (дата обращения 01.11.2023)
8. protobuf https://protobuf.dev/overview/ (дата обращения 05.07.2023)
9. Авельцов, Д. О. Применение протокола сериализации структурированных данных Protobuf в микросервисной архитектуре / Д. О. Авельцов // Проблемы автоматики и управления. – 2022. – № 3(45). – С. 185– 196. – EDN SJWCHJ.
10. Docker docs. URL: https://docs.docker.com (дата обращения: 05.07.2023).
11. Что такое ключ API? https://aws.amazon.com/ru/what-is/api-key/ (дата обращения 05.07.2023)
12. gRPC https://grpc.io/docs/what-is-grpc/introduction / (дата обращения 05.07.2023)
13. https://highload.today/api-gateway-endpoints / (дата обращения 05.07.2023).
14. Introdusing JSON URL: https://www.json.org/json-en.html (дата обращения 05.07.2023).