Привет. Сразу скажу, что было очень мало времени на тестовое задание. Я пока работаю на прошлой работе и она тоже забирает много сил и времени. Но я накидал некоторый MVP, чтобы ты мог оценить мой уровень, знания и опыт. Не стал делать что-то люто замороченное, не скажу, что всё протестировано и классно работает, исходил из свободных ресурсов и времени.

В любом случае, наверно, то, что я написал не продуктовый код и не планируется это выливать в прод и начинать использовать на реальных данных, поэтому где-то упрощал, где-то не сделал. Как я понимаю это задание призвано проверить навыки и умения.

При большем количестве времени и ресурсов, сделанное можно довести до отказоустойчивого «продового» состояния.

Архитектура системы

Система разделена на 3 модуля.

1. Crud, Rest - сервис, сервер на aiohttp (https://aiohttp.readthedocs.io/en/stable/)
2. Консьюмер, слушающий очередь с датчиков.
3. Крон-джоба, как отдельный процесс, чекающий diff в psql таблице с логами и если за последний час не было сообщения от датчика, то отсылающий сообщения клиентским урлам с информацией об изменении статуса.

Снаружи выставить nginx или haproxy.

В принципе использовал стандартный стек, кеширование и обработку сообщений. Каких-то необычных фреймворков или баз данных не использовал.

Основной стек:
Python3
    asyncio
    aiohttp
    и асинхронные библиотеки к питону
RabbitMQ
Redis
PostgreSQL

В будущем смотреть на ELK стек, Sentry для ошибок, nosql и колоночные базы для логов и построения отчётов.

Я разрабатывал higload-систему на подобном стеке и могу заявить, что тюнингуя систему на highload-режим и имея 4-5 серверов с 48 ядрами на каждом (включая весь стек, кеши и базы с репликами) можно достичь высокой производительности, чтобы сообщения от настоящих сенсоров без задержек разгребались консьюмерами и быстро отправлялись бы сообщения.

Тесты

Скажу честно, написать хорошие и мощные покрывающие основную логику и код тесты не удалось, по причине отсутствия достаточного времени, но если бы оно было, покрыл бы 80% самого важного кода, использовал pytest, написав патчи и фикстуры. Также, кроме юнитов. написал бы интеграционные тесты, часть из них уже написал.

Замечания

Забавно то, что они так и не исправили прикол с асинхронным вызовом callback в aiomqp, хотя и выпилили это из issue с главной, **https://github.com/Polyconseil/aioamqp/issues/150.**

Повозился, но таки вспомнил как сделать асинхронно.

Что улучшить

По ходу разработки записывал сюда то, что нужно сделать, но из-за недостатка времени не сделано ещё:

Не стал заморачиваться с механизмами авторизации, сейчас просто беру в заголовке token и проверяю.

Сделать создание таблиц через миграции, сейчас вручную вбил в консоль.

Нужна более грамотная обработка ошибок, нужно разделить на ошибки сервера и на BadRequest проверять существует уже запись или нет и подробнее обрабатывать сейчас просто выдаётся BadRequest. Куча мест, где нужна более осознанная обработка ошибок.

Использование пулла коннектов.

Более грамотное разделение на классы и адекватный роутинг (rest-server).

Консьюмер отправки сообщений на urls (возможно будет быстрее и более масштабируемо), сейчас это делается в том же консьюмере, который принимает сообщение от сенсора.

Померить скорости обработки Apache Bench, siege, яндекс-танком.

supervisord, настроить через него, определить сколько нужно процессов, измерить масштабируемость.

Не стал делать через докер-контейнеры, запускал нативные процессы в mac os.

Валидаторы на запросы, сейчас ошибки неверных данных не обрабатываются.

Добавил бы уже вне задания логгирование и переотправку, в случае, если клиент не принял сообщение.

Грамотная система логгирования из питона и в кибану (ELK).

Автозакрытие соединений после падения сервера, обработка обрывов.

Prefetch count ack=True (из разряда highload).

Логгирование и контроль за "потерянными" данными из очереди.

Написать обработчики случаев, когда данные в редисе не совпадают с данными в базах (актуализация кеша).

Добавить типизацию в код.

Sentry, Kibana, логгирование.

Логи, пишущиеся в базы делить по дням, чтобы быстрее был поиск.

Создать индексы в базах.