Рассмотрение вариантов проектирования

Как есть в первом пробном варианте

- **В этом проекте используется сервис generator для создания сообщений для приложений user1 и user2 с указанием адреса и поста, откуда взяты температура давление и влажность.**
- **Сообщения эти передаются по открытому каналу в очередь rabbitmq с наименованием sensor_data.**
- **Сервис collector настроен по своему каналу на эту же очередь sensor_data и слушает приход данных.**
- **Как только данные пришли, collector анализирует, для кого эти данные (user1 или user2) и, соответственно, отправляет одному из этих приложений данные по http.**
- **Приложения user1 или user2 приняв данные, с помощью фреймворка gin отрисовывают их в табличном виде в дашборде на своем сайте localhost:8082 или localhost:8083.**
- **Этот метод можно усовершенствовать так, что collector будет отправлять сообщения не самим users, а в БД REDIS, откуда users сами будут выгребать свои данные и работать с ними.**
- **Но при этом можно и им отправить сообщение о том, что вам пришли данные - заберите.**
- **Или RabbitMQ может быть как-то связан с Redis и отправлять туда данные без посредника collector?**

Рассмотрим возможные усовершенствования:

1. Использование Redis как хранилища данных

- **Если использовать Redis как промежуточное хранилище данных - Это даст несколько преимуществ:**
  - Отказоустойчивость: Если user1 или user2 временно недоступны, данные не будут потеряны
  - Масштабируемость: Можно добавлять новых пользователей без изменения логики коллектора
  - Производительность: Redis очень быстр для операций чтения/записи
- **Реализация может выглядеть так:**
  - Collector получает сообщение из RabbitMQ
  - Анализирует, для кого предназначены данные
  - Сохраняет данные в Redis с ключом, включающим идентификатор пользователя (например, user1:data:timestamp)
  - Опционально отправляет уведомление пользователю

2. Уведомления для пользователей

- **Для уведомления пользователей о новых данных есть несколько подходов:**
  - Через RabbitMQ: Создать отдельные очереди для каждого пользователя (например, user1_notifications), куда коллектор будет отправлять легкие уведомления

    - Через Redis Pub/Sub: Redis имеет механизм публикации/подписки, который можно использовать для уведомлений
    - Через WebSockets: Если у пользовательских приложений есть веб-интерфейс, можно использовать WebSockets для отправки уведомлений в реальном времени

3. Прямая интеграция RabbitMQ и Redis

- **Может ли RabbitMQ напрямую отправлять данные в Redis без посредника-коллектора.Напрямую такой интеграции нет, но есть несколько вариантов:**
    - Использование плагинов RabbitMQ: Существуют плагины, которые могут расширить функциональность RabbitMQ, но готового решения для интеграции с Redis нет
    - Использование Kafka Connect или подобных инструментов: Если вы готовы перейти на Kafka вместо RabbitMQ, то Kafka Connect имеет коннекторы для Redis
    - Использование ETL-инструментов: Такие инструменты как Apache NiFi или Logstash могут выступать в роли посредника между RabbitMQ и Redis

Рекомендуемая архитектура

- **Generator отправляет данные в RabbitMQ (как сейчас)**
- **Collector получает данные из RabbitMQ и:**
    - Сохраняет полные данные в Redis с TTL (время жизни)
    - Публикует уведомление в Redis Pub/Sub канал для соответствующего пользователя
- **User1/User2 приложения:**
    - Подписываются на соответствующий Redis Pub/Sub канал для получения уведомлений
    - При получении уведомления запрашивают данные из Redis
    - Отображают данные в своем дашборде

Эта архитектура обеспечивает:

- **отказоустойчивость,**
- **масштабируемость и**
- **производительность,**
- **сохраняя при этом относительную простоту реализации.**