PRUEBA TÉCNICA DESARROLLADOR PYTHON

Objetivo:
Validar los conocimientos técnicos requeridos para el buen desempeño en el stack requerido para el mismo.

- Conocimientos a evaluar:
  - Python
  - FastApi
  - Redis
  - Redis streams
  - Diseño de arquitectura backend.

La prueba se dividirá en 2 partes:

1. Simulador y comunicación entre microservicios.
   a. Simular un dispositivo eléctrico que reporte medidas de manera aleatoria en un rango determinado. Ejemplo: (10 - 100), datos a reportar:
      - ID Dispositivo
      - Métrica 1 medida (kwh, temp, etc), Ejemplo: 55 kwh
      - Timestamp

      Criterios punto a:
      - El simulador deberá estar desarrollado en python
      - Utilizar loop de asyncio para que el simulador siempre este reportando métricas.
      - Utilizar redis streams para producir un evento cada vez que llegue una métrica

      Nota: Este servicio tendrá rol de Producer.

   b. Crear un microservicio en fastapi o django que contenga un CRUD el cual permita administrar los datos enviados por el medidor.

   c. Crear un microservicio con python que escuche los eventos reportados por el simulador y llame al microservicio de fastapi o django para almacenar las métricas.

      Nota: Este servicio tendrá rol de Consumer, tener en cuenta que cada evento consumido se deberá quitar de la lista del consumidor.

   d. Crear un microservicio conectado al mismo stream de eventos pero que reaccione de una forma diferente, en este caso se deberá simular

un alerta o notificación en caso tal de que la medida reportada supere un umbral de 50.

2. Análisis
    a. Recolección de datos de la tabla que almacena los eventos guardados por el microservicio y generar un CSV
    b. Analizar los datos del CSV y mostrar en una gráfica el número que más veces se repite del rango reportado y cuantos datos se han reportado en 1 minutos. (Se pueden agregar más comparaciones para hacer el ejercicio más completo).

Resultado esperado:

- Un productor de eventos (Métricas de dispositivo).
- Dos grupos de consumidores (1 para consumir servicio de fastapi y otro para generar alertas).
- Un microservicio en fastapi con el CRUD de métricas.
- Diagrama de diseño de arquitectura y comunicación entre microservicios.

NOTA: Cada microservicio deberá correr de forma independiente como un contenedor de Docker

Referencias:

https://fastapi.tiangolo.com/
https://redis.io/topics/streams-intro
https://aioredis.readthedocs.io/en/latest/
https://docs.docker.com/