Prueba Técnica: Desarrollo de un Worker Java Reactivo para Procesamiento de Pedidos con Enriquecimiento de Datos

Descripción simplificada

En esta prueba, construirás un **Worker Java reactivo** que procesa mensajes de pedidos desde Kafka, los complementa con información de productos y clientes, y los almacena en MongoDB.

Escenario

Imagina que trabajas en comercio electrónico. Los pedidos se almacenan en un sistema central y se envían a Kafka para su procesamiento. Dos APIs externas proporcionan información adicional:

- **API Go:** Detalles sobre los productos de los pedidos.
- **API Nest con GraphQL:** Datos sobre los clientes que realizan los pedidos.

Tu misión

Desarrollar un **Worker Java reactivo** que:

1. **Consume** mensajes del tópico de pedidos de Kafka.
2. **Extrae** la información relevante de cada pedido.
3. **Obtiene** información adicional sobre los productos del pedido de la API Go.
4. **Consulta** la API Nest con GraphQL para obtener datos del cliente.
5. **Combina** toda la información del pedido y del cliente.
6. **Almacena** la información completa del pedido enriquecida en MongoDB.

Tecnologías requeridas

- **Java 21+:** Lenguaje de programación principal para el Worker.
- **Spring Boot 3.2.x:** Framework web Java para crear aplicaciones backend escalables y reactivas.
- **Kafka Client API:** Biblioteca para consumir y producir mensajes en Kafka.
- **Lombok:** Herramienta para generar código repetitivo.
- **Cliente HTTP reactivo (WebClient):** Para consumir la API Go.
- **Cliente GraphQL reactivo (Spring GraphQL):** Para consumir la API Nest con GraphQL.

Entregables

- **Repositorio:** Worker Java en GitHub.
- **Tópico de Kafka:** Tópico de pedidos.
- **Almacén de datos:** MongoDB.
- **URLs de las APIs:** API Go y API Nest con GraphQL.

Criterios de evaluación

- Implementación correcta del Worker Java con las tecnologías requeridas.
- Consumo exitoso de mensajes del tópico de pedidos de Kafka.
- Procesamiento correcto de la información de los pedidos.
- Integración efectiva con las APIs Go y Nest con GraphQL.
- Enriquecimiento exitoso de la información del pedido con datos del cliente.
- Almacenamiento asíncrono de la información completa del pedido en MongoDB.

Puntos clave

- El Worker debe usar Spring Boot WebFlux para un procesamiento reactivo y no bloqueante.

- Se deben usar clientes HTTP y GraphQL reactivos para las APIs.
- El almacenamiento en MongoDB debe ser asíncrono.
- La información del pedido debe enriquecerse con datos del cliente antes de guardarla en MongoDB.

Consideraciones adicionales

- Código bien documentado y legible.
- Código modular y mantenible.
- Pruebas unitarias e de integración para garantizar el correcto funcionamiento.

Esta prueba evalúa tu capacidad para desarrollar aplicaciones web reactivas que procesan y enriquecen datos de manera eficiente.