Desafio Técnico: Processamento de Dados de Eventos de Usuário em Tempo Real (Totalmente Local com Docker e Banco SQL)

Cenário

Uma empresa de jogos online coleta dados de eventos de usuários em tempo real, como cliques, logins, compras e interações dentro do jogo. Esses eventos são enviados para um cluster Apache Kafka local, dockerizado.

Objetivo

Desenvolver uma solução completa que processe esses eventos em tempo real, utilizando uma função local (simulando AWS Lambda em Python), um cluster Kafka, um motor Spark (Scala) e um banco de dados SQL, para calcular e agregar métricas de engajamento do usuário e armazená-las no banco de dados.

Requisitos

1. **Kafka**:
    - Configurar e iniciar um cluster Kafka local usando Docker Compose.
    - Criar um tópico no Kafka para receber os eventos de usuário.
2. **Função Local (Simulando AWS Lambda)**:
    - Desenvolver uma aplicação (em Python) que simule o comportamento de uma função Lambda.
    - Esta aplicação deve:
        - Consumir mensagens de um arquivo JSON (ou simular a geração de eventos).
        - Publicar essas mensagens no tópico Kafka configurado no passo 1.
3. **Motor Spark (Scala)**:
    - Criar uma imagem Docker para a aplicação Spark em Scala.
    - Desenvolver a aplicação Spark em Scala que consuma mensagens do tópico Kafka por schedule (A cada 1 minuto).
    - A aplicação Spark deve realizar as seguintes operações:
        - Calcular o número total de eventos por tipo de evento (clique, login, compra, etc.).
        - Calcular o número de usuários únicos que realizaram cada tipo de evento.
        - Calcular o tempo médio entre eventos para cada usuário.
4. **Banco de Dados SQL**:
    - Configurar e iniciar um banco de dados SQL (PostgreSQL, MySQL, etc.)

    - Criar tabelas no banco de dados para armazenar os resultados das agregações.
5. **Armazenamento em Banco de Dados**:
    - A aplicação Spark deve armazenar os resultados das agregações nas tabelas do banco de dados SQL.
6. **Dados de Evento**:
    - Os dados de evento no Kafka seguem o seguinte formato JSON:

JSON

```python
{
"user_id": "123",
"event_type": "click",
"timestamp": "2023-10-27T10:00:00Z",
"game_id": "456",
"payload": {
"button_id": "789"
}
}
```

Critérios de Avaliação

- **Funcionalidade**: A solução deve processar os eventos corretamente e gerar as métricas de engajamento esperadas, armazenando no banco de dados.
- **Desempenho**: A aplicação Spark deve ser otimizada para minimizar o tempo de processamento.
- **Robustez**: A solução deve ser capaz de lidar com erros e falhas de forma adequada.
- **Qualidade do Código**: O código deve ser limpo, organizado e seguir as melhores práticas de desenvolvimento e ter testes unitários.
- **Arquitetura**: A arquitetura da solução deve ser bem projetada e utilizar os recursos locais de forma eficiente.
- **Armazenamento em Banco SQL**: O armazenamento no banco de dados SQL deve funcionar corretamente.

Entrega

- **Código fonte da aplicação simulando a função Lambda e da aplicação Spark.**
- **`Dockerfile` para construir a imagem Docker da aplicação Spark.**

- **Arquivo `docker-compose.yml` para configurar todos os serviços.**
- **Script SQL para criação das tabelas no banco de dados.**
- **Documentação descrevendo a arquitetura da solução e os passos para execução.**
- **Dados inseridos no banco de dados como resultado do processamento.**