Relatório Técnico Final

Projeto DataDriven Store: Uma Arquitetura de Persistência Poliglota

Autor: Vinicius Carvalho Miranda

1. Resumo

Este relatório detalha a arquitetura e implementação da **DataDriven Store**, uma aplicação de e-commerce conceitual projetada para demonstrar os benefícios de uma abordagem de **persistência poliglota**. A solução integra cinco tecnologias de banco de dados distintas — **PostgreSQL, MongoDB, Redis, ClickHouse e Neo4j** — cada uma otimizada para um domínio de dados específico. A arquitetura visa maximizar a performance, escalabilidade e flexibilidade, utilizando a ferramenta certa para cada tarefa: PostgreSQL para transações ACID, MongoDB para dados de produtos semiestruturados, Redis para cache de alta velocidade, ClickHouse para análises OLAP em tempo real e Neo4j para sistemas de recomendação baseados em grafos. O projeto foi totalmente containerizado com o Docker, garantindo um ambiente de desenvolvimento e implantação coeso e reprodutível.

2. Introdução

Aplicações de e-commerce modernas lidam com uma diversidade de dados que vai muito além do escopo de um único sistema de banco de dados relacional. Dados transacionais, catálogos de produtos com atributos dinâmicos, sessões de usuário, logs de eventos de alta volumetria e relacionamentos complexos para recomendações coexistem e demandam soluções especializadas.
A abordagem de **persistência poliglota** surge como uma resposta a essa complexidade, defendendo que diferentes tipos de dados devem ser armazenados em sistemas de banco de dados que melhor se adequam às suas características e padrões de acesso. O projeto **DataDriven Store** a ideia foi construir uma arquitetura robusta onde cada componente de dados é servido pela tecnologia mais eficiente, resultando em um sistema mais performático e escalável.

3. Arquitetura da Solução

3.1. Visão Geral e Diagrama Conceitual

A arquitetura da DataDriven Store é baseada no princípio da **separação de responsabilidades**. Cada banco de dados opera como um microsserviço de dados, focado em um domínio específico. A integração entre eles é pensada para ocorrer de forma assíncrona, através de um barramento de eventos, garantindo baixo acoplamento.

3.2. Componentes e Responsabilidades

Tecnologia	Responsabilidade Principal	Domínio de Dados	Justificativa
PostgreSQL	**Dados Transacionais (OLTP)**	Clientes, Pedidos, Estoque, Financeiras	Garante consistência, atomicidade e integridade (ACID) para operações críticas de negócio. O modelo relacional é ideal para dados estruturados com relacionamentos bem definidos.
MongoDB	**Dados Semi-estruturados**	Catálogo de Produtos, Perfis de Usuário	Oferece flexibilidade de schema para armazenar produtos com atributos variados e perfis de usuário enriquecidos, permitindo uma evolução orgânica do modelo de dados sem migrações complexas.
Redis	**Cache e Dados Voláteis**	Sessões, Carrinhos de Compra, Rankings, Cache de Produtos	Proporciona acesso a dados em memória com latência extremamente baixa, essencial para melhorar a experiência do usuário em tempo real e reduzir a carga

			sobre os bancos de dados principais.
ClickHouse	**Análise de Dados (OLAP)**	Logs de Eventos, Métricas de Interação, Telemetria	Sua arquitetura colunar é otimizada para agregações e consultas analíticas em grandes volumes de dados, permitindo a criação de dashboards e relatórios de BI com alta performance.
Neo4j	**Relacionamentos e Recomendações**	Grafo de Clientes, Produtos, Compras e Visualizações	Permite modelar e consultar relacionamentos complexos de forma eficiente, sendo a base para algoritmos de recomendação (filtragem colaborativa) e análise de padrões de comportamento.

4. Implementação

4.1. Ambiente e Tecnologias

A infraestrutura do projeto foi definida e orquestrada utilizando **Docker e Docker Compose**. Isso garante que todo o ecossistema de bancos de dados possa ser iniciado e configurado com um único comando, simplificando o desenvolvimento e a implantação. O arquivo docker-compose.yml define os serviços, volumes e redes, enquanto scripts de inicialização (.sql, .js, .sh, .cypher) são usados para criar os schemas e popular cada banco de dados com dados de exemplo.

4.2. Fluxo de Dados

Embora um barramento de eventos não tenha sido implementado, o fluxo de dados pode ser descrito da seguinte forma:

1. **Compra de Produto**:
    - Uma transação é iniciada no **PostgreSQL** para criar o pedido e atualizar o estoque.

    - Após a confirmação, um evento de compra é enviado.
    - **MongoDb -** recebe o evento e inseri os dados necessário
    - **ClickHouse** ingere o evento para análise de funil e comportamento.
    - **Neo4j** cria um relacionamento [:COMPROU] entre o cliente e o produto, fortalecendo o grafo de recomendações.
    - **Redis** pode ter o cache do produto invalidado ou atualizado.
2. **Visualização de Produto**:
    - Um evento de view é enviado diretamente para o **ClickHouse**.
    - **Redis** incrementa um contador de visualizações diárias (INCR) e atualiza o ranking de mais vistos (ZINCRBY).
    - **Neo4j** pode criar um relacionamento [:VISUALIZOU] para tracking de interesse do usuário.

5. Desafios e Soluções

- **Complexidade de Gerenciamento**:
    - **Desafio**: Gerenciar cinco sistemas de banco de dados diferentes aumenta a complexidade operacional.
    - **Solução**: A utilização do **Docker Compose** foi fundamental para abstrair essa complexidade, permitindo o gerenciamento de todo o ecossistema como uma única unidade.
- **Consistência de Dados**:
    - **Desafio**: Manter a consistência dos dados entre diferentes bancos (ex: estoque no PostgreSQL e cache no Redis).
    - **Solução**: A estratégia adotada é a de **consistência eventual**. O PostgreSQL atua como a "fonte da verdade" para dados críticos. A sincronização com os outros sistemas ocorreria através de eventos, aceitando um pequeno delay em troca de desacoplamento e resiliência. Para dados voláteis como cache, estratégias de TTL (Time To Live) garantem que os dados sejam eventualmente atualizados.

6. Conclusões

O projeto **DataDriven Store** demonstrou com sucesso a viabilidade e os benefícios de uma arquitetura de persistência poliglota para aplicações complexas de e-commerce. Ao delegar cada tipo de dado à tecnologia mais adequada, a solução alcança um nível de otimização, performance e flexibilidade que seria inatingível com uma abordagem monolítica.

7. Apêndices

- **Repositório**: https://github.com/vncs2012/fn-db-Poliglota