Projeto: Escalabilidade e Flexibilidade para a XPTO Clube de Compras

Por Flávia Maria Dugulin Casto

A XPTO Clube de Compras é uma plataforma que oferece aos seus usuários a oportunidade de realizar compras online com vantagens exclusivas. Através de parcerias com lojas e fornecedores, os usuários podem obter cashback, participar de sorteios de brindes e aproveitar descontos especiais. A plataforma também oferece soluções para empresas, como cartões e tickets para funcionários, incentivando o uso e ampliando o alcance da XPTO.

Um dos grandes diferenciais da XPTO é a funcionalidade de rede social, que permite a inclusão de amigos e a formação de grupos de compra. Com isso, os usuários podem alcançar descontos coletivos ainda maiores e realizar assinaturas em conjunto, potencializando a experiência de compra e promovendo a interação entre os membros.

Para garantir o sucesso e a consolidação da XPTO no competitivo mercado de e-commerce brasileiro, é crucial que a plataforma esteja atenta aos desafios e tendências do setor. O crescimento do mobile commerce, a importância da experiência do usuário e a busca por soluções inovadoras são fatores que devem ser considerados na estratégia da empresa.

A XPTO precisa se concentrar em três pilares fundamentais para alcançar seus objetivos de crescimento e oferecer um serviço de alta qualidade:

Tópico 1: Dificuldades de Escalabilidade

Problemas de escalabilidade com a atual arquitetura transacional:

1.1 Problemas de Escalabilidade com a Arquitetura Atual

A arquitetura atual da XPTO Clube de Compras, baseada em um banco de dados relacional (PostgreSQL) centralizado, apresenta os seguintes desafios:

- **Gargalos no Banco de Dados:** O banco de dados relacional pode se tornar um gargalo à medida que a plataforma cresce em número de usuários, compras, ofertas e interações sociais. Consultas complexas, junções de tabelas e o volume crescente de dados podem levar a um aumento na latência e afetar o desempenho geral da plataforma, resultando em uma experiência ruim para o usuário.

- **Escalabilidade Vertical Limitada:** A escalabilidade vertical (aumento da capacidade do servidor) possui limites e pode se tornar cara e complexa. Em algum momento, o hardware atingirá seu limite e a adição de mais recursos pode não resultar em ganhos de desempenho proporcionais.

- **Manutenção e downtime:** A realização de manutenções no banco de dados relacional pode exigir downtime, interrompendo o funcionamento da plataforma e impactando os usuários.

• **Dificuldade em Lidar com Picos de Acesso:** Em momentos de pico, como promoções ou campanhas de marketing, o banco de dados pode ficar sobrecarregado, levando a lentidão e indisponibilidade da plataforma.

• **Flexibilidade limitada:** A estrutura rígida do banco de dados relacional pode dificultar a adaptação a novas funcionalidades e mudanças nos requisitos da plataforma.

1.2 Modelos de Arquitetura Baseados em NoSQL

Para solucionar os problemas de escalabilidade, podemos considerar a adoção de bancos de dados NoSQL, que oferecem maior flexibilidade, escalabilidade e desempenho para lidar com grandes volumes de dados e acessos. Abaixo, apresento dois modelos de arquitetura com bases NoSQL:

Modelo 1: Microservices com Bancos de Dados Dedicados

Neste modelo, a plataforma é dividida em microservices, cada um com sua própria base de dados NoSQL.

- **Microservices:** "Tickets", "Ofertas", "Compras", "Recomendações", "Cadastro" e "Amigos" se tornam microservices independentes.
- **Bancos de Dados NoSQL:** Cada microservice utiliza um banco de dados NoSQL mais adequado às suas necessidades. Por exemplo:
    - **MongoDB:** para "Ofertas" e "Recomendações", devido à flexibilidade do modelo de documentos e capacidade de lidar com dados não estruturados.
    - **Cassandra:** para "Compras", por sua alta disponibilidade e escalabilidade para lidar com grande volume de transações.
    - **Neo4j:** para "Amigos", por ser um banco de dados de grafos, ideal para representar e consultar relacionamentos entre usuários.
- **API Gateway:** Um API Gateway gerencia as requisições dos usuários, direcionando-as para o microservice correspondente.

Vantagens:

- **Escalabilidade independente:** Cada microservice pode escalar de forma independente, de acordo com suas necessidades.
- **Resiliência:** Falhas em um microservice não afetam os demais.
- **Flexibilidade:** Permite a utilização de diferentes tecnologias e bancos de dados NoSQL, escolhidos de acordo com as necessidades de cada microservice.
- **Desenvolvimento ágil:** Equipes podem trabalhar em microservices diferentes de forma independente, acelerando o desenvolvimento.

Modelo 2: Arquitetura Orientada a Eventos com Kafka

Neste modelo, a plataforma utiliza um sistema de mensageria (Apache Kafka) para comunicação assíncrona entre os componentes.

- **Eventos:** Eventos como "Compra realizada", "Novo usuário cadastrado" e "Amizade criada" são publicados no Apache Kafka.

- **Consumidores:** Microservices ou componentes da plataforma se inscrevem nos tópicos do Kafka para consumir os eventos relevantes.
- **Processamento assíncrono:** O processamento de eventos ocorre de forma assíncrona, desacoplando os componentes e permitindo maior escalabilidade e flexibilidade.
- **Bancos de Dados NoSQL:** Bancos de dados NoSQL, como Cassandra e MongoDB, são utilizados para armazenar dados e oferecer suporte ao processamento de eventos.

Vantagens:

- **Alta escalabilidade:** O Apache Kafka permite lidar com grande volume de eventos e tráfego.
- **Processamento em tempo real:** Eventos são processados em tempo real, permitindo ações imediatas, como atualização de saldos, envio de notificações e geração de recomendações.
- **Flexibilidade e extensibilidade:** Facilita a adição de novas funcionalidades e integrações com outros sistemas.
- **Tolerância a falhas:** O Apache Kafka garante a entrega de mensagens mesmo em caso de falhas.

Tópico 2: Limitações do Modelo de Dados

O modelo de dados relacional, apesar de sua robustez e maturidade, pode apresentar algumas limitações quando aplicado a cenários como o da XPTO Clube de Compras, que exigem alta escalabilidade, flexibilidade e performance para lidar com dados complexos e um grande volume de acessos.

2.1 Problemas Relacionados ao Modelo Relacional

1. **Rigidez do Esquema:** A estrutura rígida do modelo relacional, com tabelas predefinidas e esquemas fixos, pode dificultar a adaptação a novas funcionalidades e mudanças nos requisitos da plataforma. Adicionar novas informações ou modificar a estrutura das tabelas pode ser complexo e demandar tempo, impactando a agilidade do desenvolvimento.
2. **Dificuldade em lidar com dados não estruturados:** O modelo relacional é otimizado para dados estruturados em linhas e colunas. No entanto, a XPTO Clube de Compras pode precisar lidar com dados não estruturados, como informações de redes sociais, avaliações de produtos, imagens e vídeos. Armazenar e consultar esses dados em um modelo relacional pode ser ineficiente e complexo.
3. **Escalabilidade limitada:** A escalabilidade do modelo relacional pode ser um desafio em cenários de alto crescimento. Consultas complexas, junções de tabelas e o volume crescente de dados podem levar a gargalos e afetar o desempenho da plataforma.

2.2 Como o NoSQL Resolve as Limitações

1. **Flexibilidade do esquema:** Bancos de dados NoSQL, como MongoDB, permitem armazenar dados sem um esquema predefinido. Isso facilita a adaptação a mudanças nos requisitos e a adição de novas funcionalidades sem a necessidade de modificar a estrutura do banco de dados.
2. **Suporte a dados não estruturados:** NoSQL oferece suporte a diversos tipos de dados, incluindo dados não estruturados. Isso permite que a XPTO Clube de Compras armazene e consulte informações como avaliações de produtos, posts de redes sociais e imagens de forma eficiente.
3. **Escalabilidade Horizontal:** Bancos de dados NoSQL são projetados para escalar horizontalmente, adicionando mais nós ao cluster para lidar com o aumento do volume de dados e acessos. Isso garante maior escalabilidade e disponibilidade da plataforma.

2.3 Coexistência entre Modelo Relacional e NoSQL

Em vez de substituir completamente o banco de dados relacional, uma abordagem mais eficiente pode ser a coexistência entre o modelo relacional e o NoSQL.

1. **Dados transacionais no relacional:** O banco de dados relacional (Postgres) pode continuar sendo utilizado para armazenar dados transacionais críticos, como informações de usuários, pedidos e pagamentos, que exigem ACID properties e integridade referencial.
2. **Dados não estruturados e de alta performance no NoSQL:** Bancos de dados NoSQL podem ser utilizados para armazenar dados não estruturados, como avaliações de produtos, informações de redes sociais e dados de sessão. Além disso, podem ser utilizados para funcionalidades que exigem alta performance e escalabilidade, como recomendações de produtos, feeds de notícias e análises em tempo real.
3. **Integração:** A comunicação entre os bancos pode ser feita por mensageria (Kafka) ou ETL para sincronizar os dados de forma consistente.

Tópico 3: Dificuldades de Manutenção de Operações

3.1 Arquitetura Proposta para Operações Otimizadas

Para facilitar a manutenção e melhorar as operações da plataforma XPTO, podemos propor uma arquitetura que combine o poder do modelo relacional com a flexibilidade e escalabilidade do NoSQL, além de adotar práticas de DevOps para garantir a eficiência e a agilidade nas operações.

Componentes da Arquitetura:

- **API Gateway:** Gerencia as requisições dos usuários, roteando-as para os microservices correspondentes e implementando políticas de segurança e controle de acesso.

- **Microservices:** A plataforma é dividida em microservices independentes, cada um com sua própria base de dados e responsabilidade específica.
    - **Microsserviço de Catálogo:** Gerencia as informações dos produtos e ofertas, utilizando um banco de dados NoSQL (MongoDB) para lidar com a flexibilidade dos dados e a necessidade de escalabilidade.
    - **Microsserviço de Compras:** Responsável por processar as compras, gerenciar o estoque e processar pagamentos, utilizando um banco de dados relacional (PostgreSQL) para garantir a consistência dos dados transacionais.
    - **Microsserviço de Recomendações:** Utiliza algoritmos de machine learning para gerar recomendações personalizadas para os usuários, com base em seus dados de navegação e histórico de compras (armazenados em MongoDB).
    - **Microsserviço de Usuários:** Gerencia o cadastro de usuários, perfis, autenticação e autorização, utilizando PostgreSQL para garantir a segurança e a integridade dos dados dos usuários.
- **Bancos de Dados:**
    - **PostgreSQL:** Utilizado para armazenar dados transacionais críticos, como informações de usuários, pedidos e pagamentos, que exigem ACID properties e integridade referencial.
    - **MongoDB:** Utilizado para armazenar dados não estruturados e de alta performance, como informações de produtos, avaliações, recomendações e dados de sessão.
    - **Redis:** Utilizado como cache para armazenar dados frequentemente acessados, como informações de produtos, ofertas e sessões de usuários, melhorando o desempenho da plataforma.
- **Apache Kafka:** Utilizado como sistema de mensageria para comunicação assíncrona entre os microservices, desacoplando os componentes e permitindo maior escalabilidade e flexibilidade.
- **Monitoramento e Logging:** Ferramentas de monitoramento, como Prometheus e Grafana, e ferramentas de logging, como ELK (Elasticsearch, Logstash e Kibana), são utilizadas para monitorar o desempenho da plataforma, identificar gargalos e solucionar problemas.
- **CI/CD:** Utilização de pipelines de CI/CD (Continuous Integration/Continuous Delivery) para automatizar o processo de desenvolvimento, teste e deploy dos microservices, garantindo a agilidade e a confiabilidade das operações.

3.2 Problemas Evitados com NoSQL desde o Início

A implementação do modelo NoSQL desde o início da plataforma poderia ter evitado os seguintes problemas:

- **Migrações complexas:** A necessidade de migrar dados de um modelo relacional para um modelo NoSQL em um estágio posterior do desenvolvimento.
- **Limitação de escalabilidade:** A dificuldade em escalar o banco de dados relacional para lidar com o crescimento da plataforma.

- **Dificuldade em lidar com dados não estruturados:** A complexidade em armazenar e consultar dados não estruturados em um modelo relacional.
- **Rigidez do esquema:** A dificuldade em adaptar o esquema do banco de dados relacional às mudanças nos requisitos da plataforma.