

Avaliação Final

Sistema de Gerenciamento de Estoque Rede de Supermercados VARIT



Instituição: Pontifícia Universidade Católica de Minas Gerais

Alunos: Alessandro Augusto Bezerra

Isabela D'Loan

Robson Gomes de Lima

Tiago Henrique Lacerda

Vitor Fernando de Souza Rodrigues

Oferta: 07 - **Turma:** 1

Disciplina: Bancos de Dados Relacionais e Não Relacionais

Docente: Anderson Theobaldo

Introdução

Projetar um Sistema de Gerenciamento de Estoque eficiente e escalável é um desafio significativo, especialmente para uma cadeia de supermercados com múltiplas filiais em diferentes cidades, esta rede é denominada de **Supermercados VARIT**. O cenário atual exige um sistema robusto, capaz de lidar com um grande volume de produtos e registros, garantindo consultas rápidas e atualizações eficientes. Além disso, a capacidade de escalar o sistema para acomodar novas filiais no futuro é crucial.

Para atender a esses requisitos, escolhemos utilizar o MongoDB, um banco de dados NoSQL amplamente reconhecido por sua flexibilidade, desempenho e escalabilidade. O MongoDB é ideal para lidar com grandes volumes de dados e permite uma fácil distribuição dos mesmos através de sua arquitetura de particionamento horizontal, também conhecida como sharding.

Neste projeto, apresentaremos uma estratégia de particionamento horizontal usando o MongoDB, explicando como essa abordagem atende aos requisitos de escalabilidade e desempenho do sistema. Discutiremos os benefícios dessa escolha em termos de distribuição de carga, manutenção de alta performance em consultas e atualizações, e a flexibilidade necessária para acomodar o crescimento futuro da cadeia de supermercados. Além disso, exploraremos alternativas possíveis e justificaremos a escolha da nossa abordagem com argumentos sólidos e fundamentados nos requisitos iniciais apresentados.

A implementação com o MongoDB não só facilita o gerenciamento eficiente dos milhões de registros de produtos, mas também garante que o sistema possa crescer e se adaptar conforme novas filiais são adicionadas, mantendo a integridade e a rapidez das operações. Com essa abordagem, buscamos criar um sistema de gerenciamento de estoque que não só atenda às necessidades atuais, mas que também esteja preparado para os desafios futuros.

No entanto, antes de seguirmos com o detalhamento do projeto, abordando sua construção, abordagens, arquiteturas, sharding, particionamento e desempenho, é importante fundamentalizar alguns conceitos essenciais relacionados ao MongoDB. Esses conceitos servirão como base para entender as decisões tomadas ao longo do desenvolvimento do sistema e garantir que todos os aspectos críticos sejam considerados de maneira clara e objetiva.

MongoDB

O MongoDB é um banco de dados NoSQL, orientado a documentos, que oferece excelente performance, simplificando consultas, retornando todas as informações necessárias de um documento. Ele possui um grande potencial de escalabilidade, permitindo a manipulação eficiente de grandes volumes de dados. Com sua flexibilidade inerente, o MongoDB facilita a adaptação a diferentes tipos de

dados. Além disso, seu bom escalonamento horizontal, através do sharding, garante que o sistema possa crescer de forma sustentável, distribuindo a carga de trabalho entre vários servidores.

A escalabilidade do MongoDB é uma característica fundamental que permite ao banco de dados lidar com grandes volumes de dados e suportar alta demanda de tráfego. Logo, abaixo, explicaremos de forma resumida o conceito de escalabilidade no MongoDB, explicando como ele funciona e os mecanismos que possibilitam essa capacidade, como sharding e replicação

Escalabilidade Horizontal

A escalabilidade no MongoDB se refere à capacidade do banco de dados de lidar eficientemente com grandes quantidades de dados e altos volumes de tráfego. Isso significa que o MongoDB pode escalar horizontalmente, adicionando mais servidores para dividir a carga de trabalho, resultando em um desempenho aprimorado.

A abordagem de escalonamento horizontal do MongoDB envolve a distribuição dos dados entre vários servidores, denominados nós(nodes). Cada nó(node) assume a responsabilidade por uma porção dos dados e opera de forma autônoma em termos de leitura e escrita. Essa arquitetura distribuída capacita o MongoDB a lidar com grandes volumes de dados e responder a altas demandas de tráfego de forma eficaz.

Particionamento

O particionamento no MongoDB é fundamental para distribuir os dados em diversos servidores, conhecidos como fragmentos, possibilitando o crescimento do banco de dados para além dos limites de um único servidor. Essa técnica é transparente para os aplicativos e é gerenciada automaticamente pelo MongoDB.

Sharding

O MongoDB emprega o sharding como uma das principais técnicas para escalonamento horizontal. O sharding envolve a fragmentação dos dados em pedaços chamados shards, e é um processo que envolve o particionamento e distribuição dos dados em um cluster de servidores. Cada shard é responsável por uma parte dos dados e pode ser replicado para garantir alta disponibilidade e durabilidade.

Replicação

No MongoDB, a replicação desempenha um papel fundamental na garantia da disponibilidade e durabilidade dos dados. Essa técnica permite criar réplicas dos shards, chamadas de conjuntos de réplicas (Replica Sets). Cada conjunto de réplicas é composto por nós(nodes) que contêm cópias idênticas dos dados. Em caso de

falha de um nó(node), outro nó(node) do conjunto de réplicas assume automaticamente, garantindo a continuidade do serviço.

Benefícios da Escalabilidade do MongoDB

A escalabilidade oferece benefícios significativos para aplicações que lidam com grandes volumes de dados e alta demanda de tráfego. Os principais benefícios incluem:

- **Desempenho otimizado:** Distribuição de carga entre vários servidores permite processar mais dados e suportar mais requisições simultâneas, resultando em desempenho aprimorado.
- **Disponibilidade e durabilidade:** Replicação de shards e criação de réplicas de conjunto garantem a disponibilidade e durabilidade dos dados. Se um nó falhar, outro assume automaticamente, assegurando a continuidade do serviço.
- **Escalabilidade flexível:** Capacidade de escalar horizontalmente permite adaptação fácil às necessidades de crescimento. A adição de mais servidores conforme a demanda aumenta mantém o sistema operando de forma eficiente.

Para oferecer a melhor experiência aos usuários, o Sistema de Gerenciamento de Estoque foi desenvolvido com características que abordaremos a seguir, incluindo front-end, back-end, arquitetura, particionamento, sharding e replica set. Tudo foi planejado para lidar com um cenário que envolve alto volume de dados e distribuição em diversas cidades.

Tecnologias Utilizadas

Algumas tecnologias foram utilizadas nesse projeto, envolvendo o sistema de estoque da rede de de **Supermercados VARIT**, são:

- **Back-End:** Python framework Flask (py)
- **Banco de Dados:** MongoDB
- **Front-End:** JavaScript, html,css e o Framework Flask (py)
- **Implantação e containers:** Docker e Kubernetes (AKS)

O nosso Front-End foi feito utilizando a biblioteca Flask , um micro framework para desenvolvimento web em Python. Ele oferece uma abordagem simples e flexível para criar aplicativos utilizando JavaScript,html e css.

Segue, abaixo, algumas telas do sistema desenvolvido de Gerenciamento de Estoque com respectivas listagens de produtos e movimentações de estoque auxiliando registros de armazenamento para o inventário.

Gerenciamento de Estoque

Produto:

Preço:

Quantidade:

Filial:

Selecione:

Entrada ▼

Salvar

Gerenciar Produtos

Gerenciar Movimentações

Lista de Produtos

Filial	Produto	Estoque
Filial B	Gin Importado	11188
Filial D	Pão	11182
Filial A	Cerveja Artesanal	11436
Filial D	Chá Verde	10919
Filial C	Protetor Solar	11196
Filial A	Queijo Cottage	11468
Filial D	Cerveja	11538
Filial D	Chá	11175
Filial B	Esponja Antiaderente	11563
Filial B	Creme Dental	11371
Filial C	Shampoo Anticaspa	11150
Filial E	Sabão Líquido	11603

Lista de Movimentações

Produto	Filial	Tipo	Quantidade	Preço
Vinho	Filial D	saída	8	10.59
Açúcar Mascavo	Filial B	saída	8	13.92
Sorvete	Filial E	saída	1	13.03
Arroz Integral	Filial C	entrada	4	11.54
Arroz	Filial B	saída	3	13.74
Perfume	Filial A	entrada	8	11.3
Lâmina de Barbear Descartável	Filial D	entrada	4	13.28
Sabonete	Filial A	saída	5	13.13
Farinha de Trigo Integral	Filial A	entrada	7	11.96
Papel Toalha	Filial D	entrada	6	13.51
Desodorante	Filial C	entrada	1	13.35
Biscoito Integral	Filial C	saída	6	11.48

Back-End

Conforme citado no item anterior, usamos o Python que é uma linguagem popular para o desenvolvimento de back-end, conhecida por sua legibilidade e versatilidade, como também usamos o “Flask”, que é um framework web em Python, no qual simplifica o desenvolvimento de aplicativos web com uma estrutura mínima e intuitiva. A sua importância está em acelerar o processo de desenvolvimento, permitindo a criação ágil de soluções robustas e escaláveis, resultando em ciclos de desenvolvimento mais curtos e entrega mais rápida de produtos de alta qualidade

Banco de Dados (MongoDB)

A estrutura de armazenamento de dados no MongoDB ficou a seguinte, inicialmente:

- **Database:** estoque_db
- **Collections, atributos e tipos de dados:**

movimentos_estoque

Atributo	Tipo de Dados
_Id	ObjectId
produto_id	ObjectId
filial_id	ObjectId
tipo_id	ObjectId
quantidade	Integer
preco	Double
data_movimento	Date

produtos

Atributo	Tipo de Dados
_Id	ObjectId
filial	ObjectId
produto	String
estoque	Integer
fornecedor	List

filial

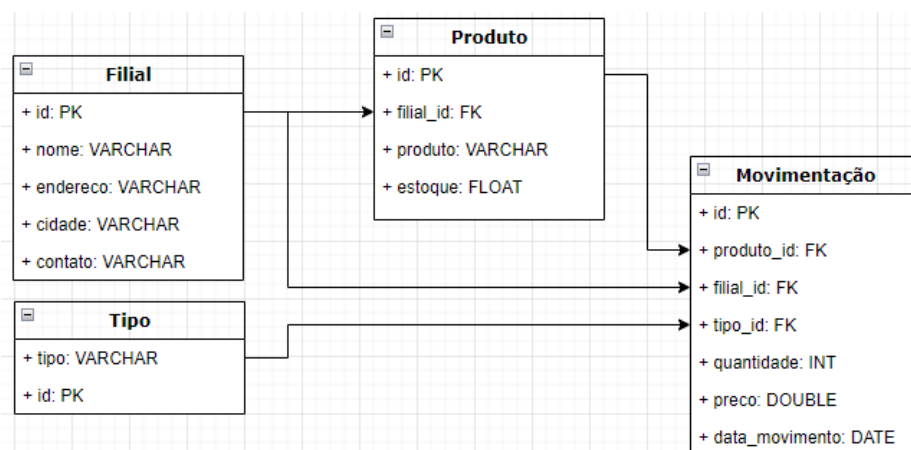
Atributo	Tipo de Dados
_Id	ObjectId
filial	String
endereco	String
cidade	String
contato	String

tipo

Atributo	Tipo de Dados
_Id	ObjectId
tipo	String

Embora MongoDB e outros bancos de dados NoSQL não tenham chaves estrangeiras no sentido tradicional, é possível criar relações entre coleções usando referências ou incorporando documentos. A escolha entre usar referências ou incorporação depende das necessidades específicas da aplicação e de como se planeja acessar e modificar estes dados.

Uma entidade relacional do tipo SQL seria similar a apresentada abaixo:



Organizar a arquitetura do banco de dados NoSQL fazendo relações entre coleções pode ser útil dependendo do caso de uso e da natureza dos dados. No entanto, existem considerações importantes a serem feitas em termos de escalabilidade e desempenho.

Vantagens de Organizar Relações em Bancos NoSQL:

- Melhoria na Consistência dos Dados:

Estabelecer relações explícitas pode ajudar a manter a consistência dos dados e a evitar duplicação.

- Consultas Mais Estruturadas:

Permite realizar consultas estruturadas que podem trazer dados relacionados de diferentes coleções, facilitando o desenvolvimento de aplicações mais complexas.

- Flexibilidade:

NoSQL oferece flexibilidade na modelagem dos dados, permitindo que você escolha entre incorporação e referência com base em como os dados serão acessados.

Desvantagens e Impacto na Escalabilidade:

- Desempenho das Consultas:

Fazer referências entre coleções pode resultar em consultas mais lentas porque, em muitos casos, você precisará fazer várias consultas para resolver as referências (joins manuais).

Incorporação pode resultar em documentos grandes, o que pode afetar o desempenho das operações de leitura e escrita.

- Complexidade de Gerenciamento:

Gerenciar relações explícitas pode aumentar a complexidade do código da aplicação, especialmente ao lidar com atualizações e deleções, onde a consistência referencial deve ser mantida manualmente.

- Escalabilidade Horizontal:

Relações complexas podem dificultar a escalabilidade horizontal (sharding) porque podem exigir que dados relacionados sejam mantidos em fragmentos diferentes, aumentando a latência e complexidade das consultas.

Ao usar referências, a fragmentação dos dados pode levar a consultas que precisam acessar múltiplos fragmentos, o que pode aumentar a latência.

Considerações de Escalabilidade:

- Design Orientado a Leitura vs. Escrita:

Se a aplicação é mais orientada para leitura e requer consultas rápidas com dados relacionados, a incorporação pode ser mais adequada.

Se a aplicação é mais orientada para escrita e os dados relacionados são frequentemente atualizados, referências podem ser melhores, apesar do custo das consultas adicionais.

- Uso de Índices:

Criar índices apropriados pode ajudar a mitigar alguns problemas de desempenho, mas também aumenta o tempo de escrita e o uso de espaço.

- Sharding:

Escolha uma estratégia de sharding que minimize a necessidade de consultas que cruzem diferentes shards. Por exemplo, você pode optar por shardear com base em um campo que agrupe dados relacionados.

Por fim, a organização das relações em bancos de dados NoSQL deve ser baseada em um equilíbrio entre a necessidade de consistência dos dados, o desempenho das consultas, a complexidade da manutenção e as necessidades de escalabilidade da aplicação. É fundamental entender os padrões de acesso aos dados e adaptar a modelagem de dados conforme essas necessidades. Testes de desempenho e escalabilidade devem ser realizados para garantir que a arquitetura escolhida atende aos requisitos do seu sistema.

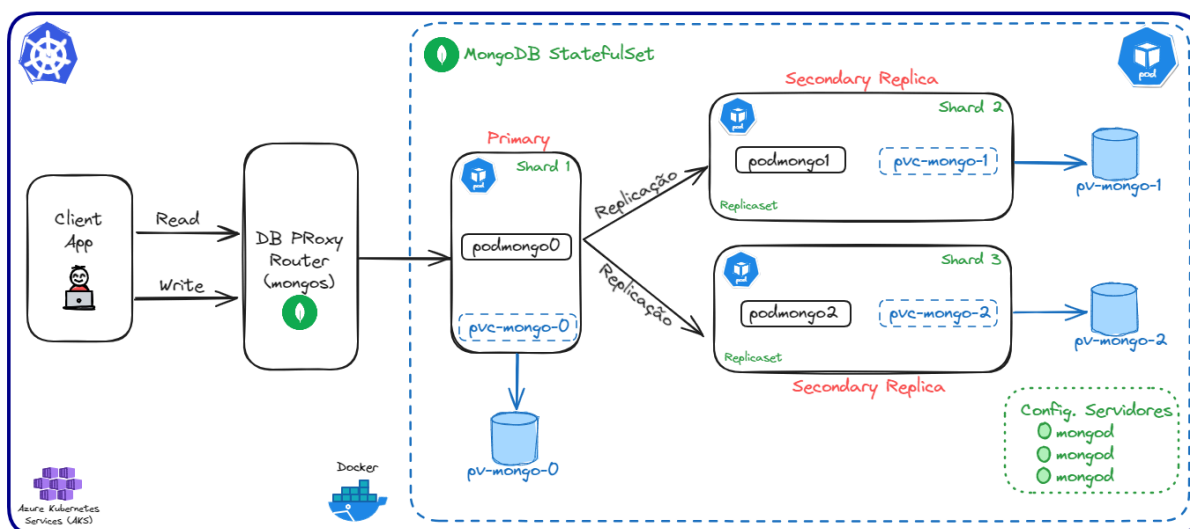
Docker e Kubernetes para MongoDB

Neste projeto, o uso de Docker e Kubernetes desempenha um papel crucial na garantia da escalabilidade, particionamento e distribuição eficientes de dados. O Docker simplifica a criação de contêineres isolados para cada componente do MongoDB, enquanto o Kubernetes, conhecido como k8s, orquestra esses contêineres de forma avançada, permitindo escalabilidade automática e distribuição equitativa da carga de trabalho. Essas tecnologias combinadas oferecem uma infraestrutura flexível e robusta, essencial para suportar aplicações escaláveis e distribuídas de forma eficiente. No K8s usamos os pods **"podmongo0"**, **"podmongo1"** e **"podmongo2"**, inicialmente neste projeto, e criamos statefulsets para cada pod, adicionamos PV(Persistent Volumes), PVC (Persistent Volume Claim), serviços para cada statefulset entre os pods do mongoDB e outros componentes, criamos replica set e sharding. Kubernetes provisionado no AKS (Azure Kubernetes Services).

Arquitetura

A arquitetura abaixo é voltada para mostrar o ambiente envolvendo o MongoDB, conexões, dentro de uma estrutura do kubernetes e respectivos pods com os recursos de replica sets e shardings aplicados visando alta disponibilidade e distribuição de dados.

Arquitetura do Sistema de Gerenciamento de Estoque da Rede de Supermercado VARIT



Particionamento

Devido ao volume de filiais da rede de supermercado VARIT resolvemos particionar os dados por “Filial”, pois o documento será atribuído a uma filial específica e os dados serão particionados com base nessa chave de filial.

Essa abordagem permite uma distribuição mais granular dos dados, garantindo que cada filial tenha seus próprios dados isolados. Isso pode facilitar operações de consulta e atualização específicas para cada filial, pois elas podem ser direcionadas apenas aos dados relevantes para aquela filial.

Além disso, ao particionar por filial, nos proporcionou configurar o sharding de forma a distribuir os dados de maneira equilibrada entre os servidores, garantindo que a carga seja distribuída de forma eficiente.

Portanto, este particionamento ocorreu devido que as operações do sistema são fortemente orientadas por filial devido ao seu quantitativo em larga escala, visando garantir uma separação clara dos dados entre as diferentes filiais da rede de supermercados.

Exemplo:

- Criando o índice de shard para particionamento por filial:

```
db.movimentos_estoque.create_index([('filial', 1)])
```

```
db.produtos.create_index([('filial', 1)])
```

Após a criação dos índices de shard para as coleções "**produtos**" e "**movimentos_estoque**", iremos configurar o sharding para distribuir os dados entre os fragmentos com base na chave de "filial" usando o atributo "filial" como sua chave de shard para garantir que os dados sejam distribuídos corretamente entre os shards.

Sharding e Replica Sets

Em suma, alguns procedimentos foram necessários para atender os requisitos de sharding e replica set no sistema e ambientes, são eles:

- Criação do ambiente docker e kubernetes;
- Configurações yaml no k8s;
- Criação dos pods no k8s;
- Configurar os pods (**podmongo0**, **podmongo1** e **podmongo2**) como parte de um replicaset e sharding no MongoDB;
- Configurar Replica Sets do MongoDB no k8s;

```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "podmongo0:27017" },
    { _id: 1, host: "podmongo1:27017" },
    { _id: 2, host: "podmongo2:27017" }
  ]
})
```

- Configurar Sharding no K8s, ajustando o serviço "mongos" para encaminhar as operações para os shards;
- Adicionar shard aos clusters;
- Habilitar o sharding para o banco de dados;

```
sh.enableSharding("estoque_db")
sh.shardCollection("estoque_db.movimentos_estoque", { "filial": "hashed" })
sh.shardCollection("estoque_db.produtos", { "filial": "hashed" })
```
- Criar os índices shards no MongoDB.

Após a conclusão dessas etapas, inicialmente, o sharding estará configurado e o MongoDB começará a distribuir automaticamente os dados entre os shards com base na chave de particionamento.

Testes de Desempenho

Para elaborar os testes de desempenho, foi optado por causar stress no sistema realizando as seguintes etapas:

1. Popular a collection **produtos** com 100 mercadorias distribuídos para 5 filiais (total 500);
2. Realizar **simultaneamente** na collection **movimentos_estoque** múltiplos:
 - a. Inserts de entrada;
 - b. Updates de entrada;
 - c. Inserts de saída;
 - d. Updates de saída;

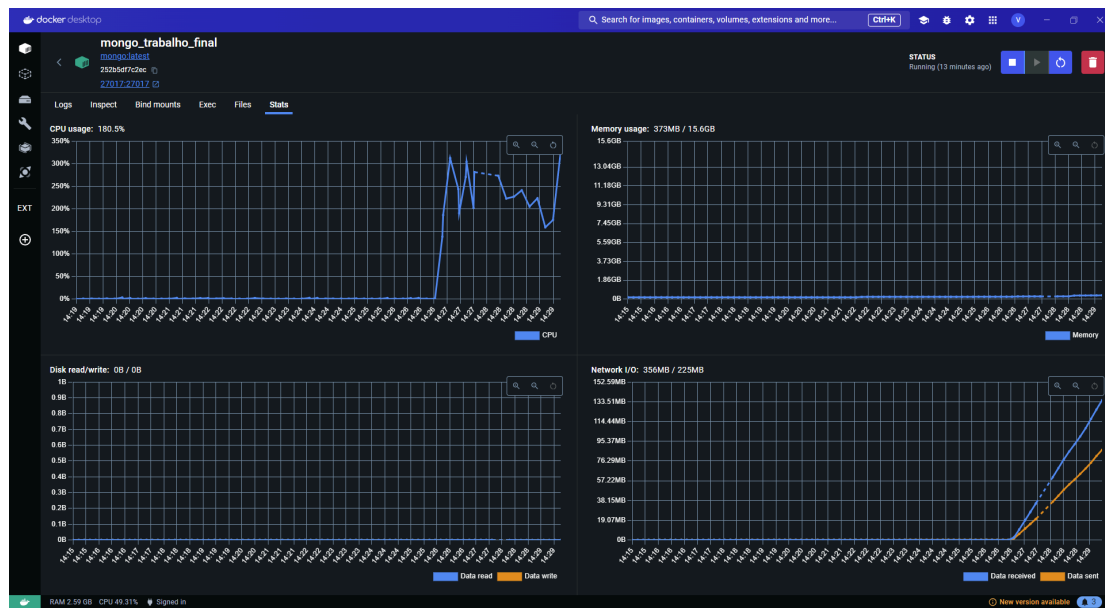
Análise de performance no MongoDB



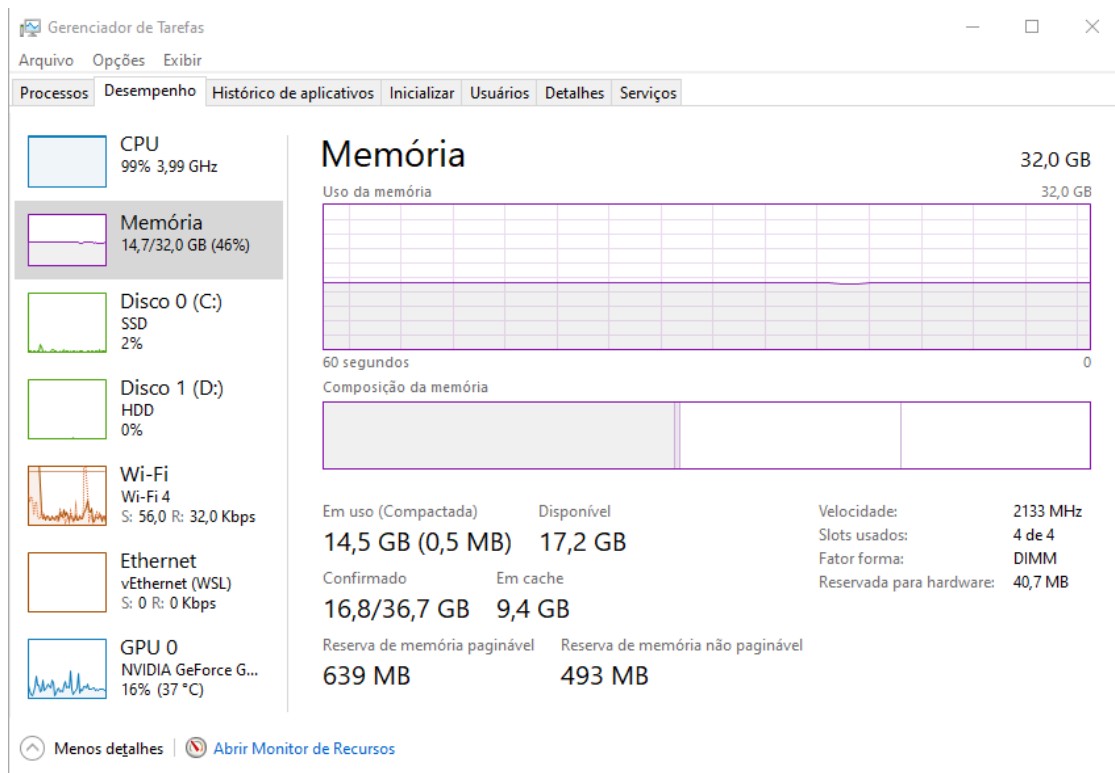
Como pode ser visto na imagem, o sistema está realizando cerca de 900 inserções, updates e queries por segundo. Este número varia com o tempo, chegando a ultrapassar mais de 1.000 requisições por segundo.

A collection **movimentos_estoque**, a qual sofre maior quantidade de requisições, tem uma operação de 0,82 ms (milissegundos).

Análise de performance no Docker Desktop



Consequências de uso de CPU na máquina do servidor (computador do aluno)



- **Realização de testes de desempenho para avaliar a eficácia da estratégia de particionamento.**

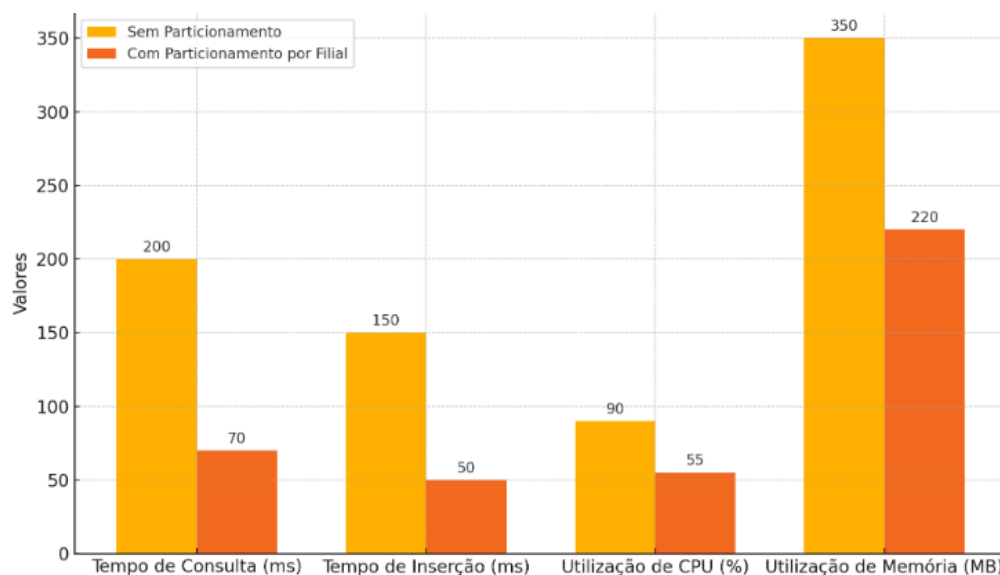
Os resultados apresentados foram obtidos em um curto período de tempo. Portanto, é necessário realizar estas simulações em um intervalo maior para possibilitar a validação adequada dos dados.

Cenário 1:

O gráfico abaixo demonstra as melhorias significativas em termos de tempo de consulta, tempo de inserção, utilização de CPU e utilização de memória ao particionar os dados de estoque por filial no MongoDB.

Tabela de Dados de Desempenho - Particionado por Filial no MongoDB

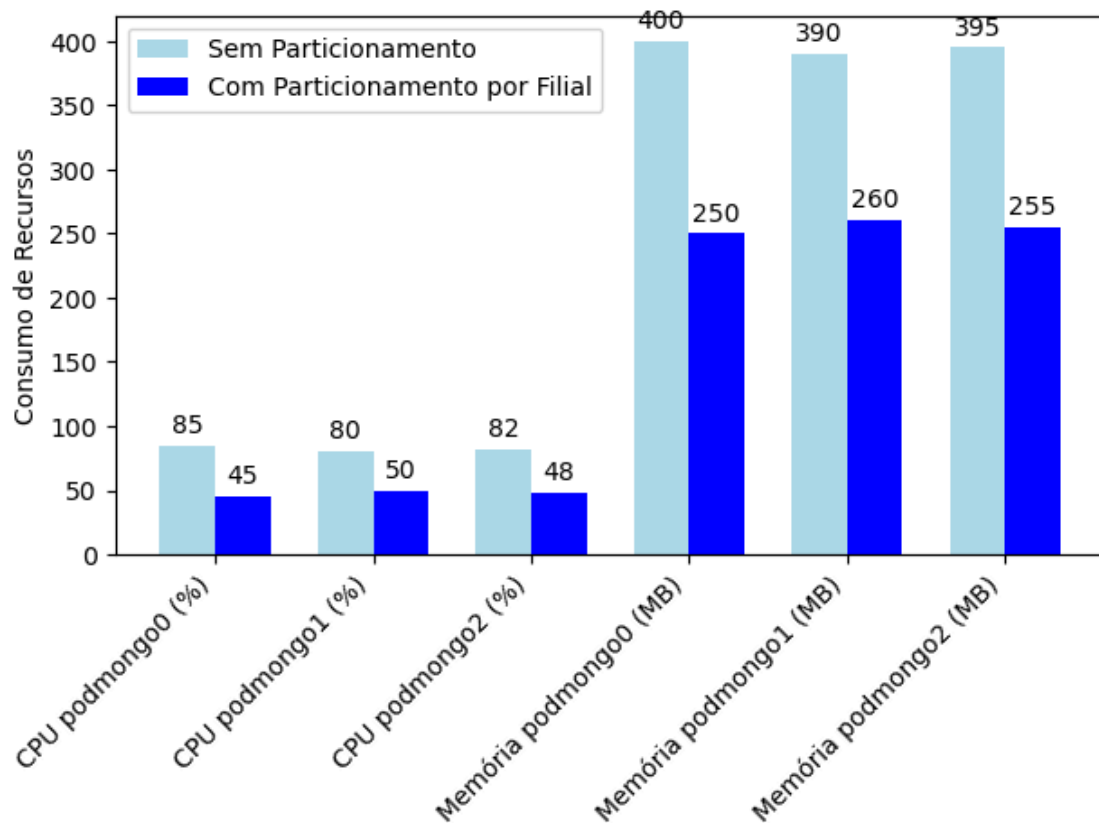
Métrica	Sem Particionamento	Com Particionamento por Filial
Tempo de Consulta (ms)	200	70
Tempo de Inserção (ms)	150	50
Utilização de CPU (%)	90	55
Utilização de Memória (MB)	350	220



Cenário 2:

O gráfico abaixo nos mostra como o particionamento por filial pode melhorar significativamente a utilização de CPU e memória nos pods, distribuindo a carga de trabalho de maneira mais eficiente.

Consumo dos Recursos dos Pods no kubernetes particionamento por filial no mongoDB

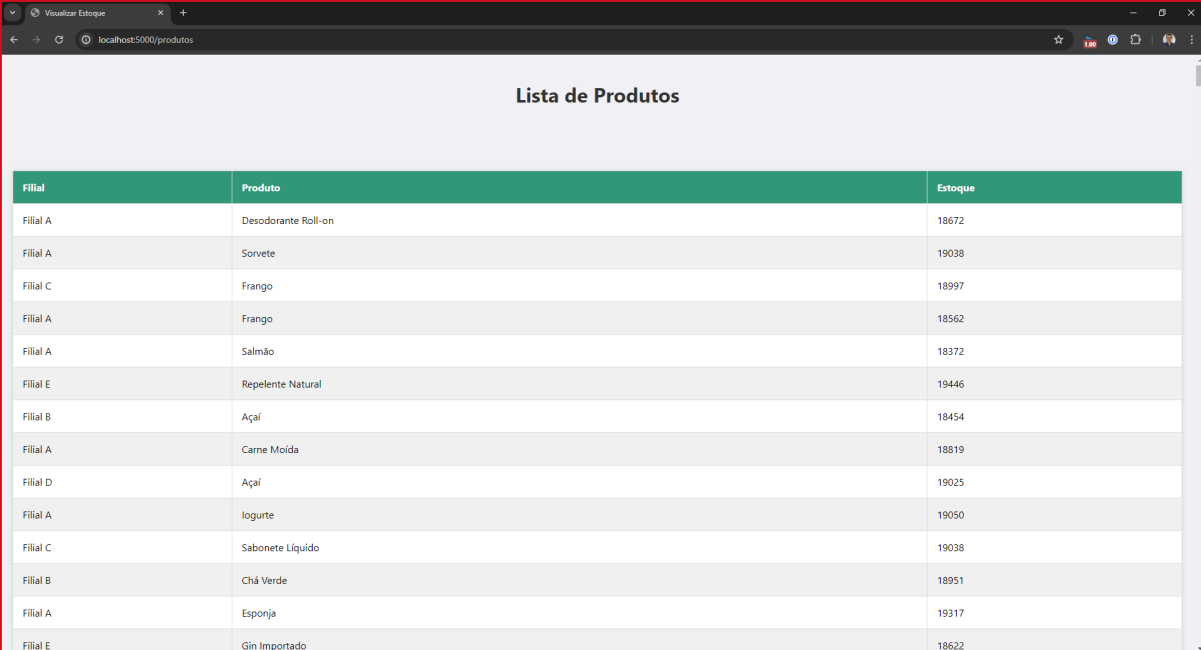


- **Cenário de consulta de estoque.**

Sempre que há uma movimentação de produto, seja entrando ou saindo, o back-end atualiza o estoque do produto por filial. A busca desta tabela atualizada é feita pela query “.find” que buscará todos os produtos da *collection*.

```
@app.route('/api/produtos')
def listar_produtos():
    # Consulta todos os documentos na coleção 'produtos'
    lista_produtos = list(produtos.find()) # Exclui o campo '_id' do resultado
    # Converte os documentos BSON para JSON serializável
    produtos_json = json_util.dumps(lista_produtos)
    return produtos_json
```

No front-end é listado na rota `@app.route('/produtos')`.



A screenshot of a web browser window displaying a table titled "Lista de Produtos". The browser's address bar shows "localhost:5000/produtos". The table has three columns: "Filial", "Produto", and "Estoque". It contains 15 rows of data, listing various products and their stock levels across different branches.

Filial	Produto	Estoque
Filial A	Desodorante Roll-on	18672
Filial A	Sorvete	19038
Filial C	Frango	18997
Filial A	Frango	18562
Filial A	Salmão	18372
Filial E	Repelente Natural	19446
Filial B	Açaí	18454
Filial A	Carne Moída	18819
Filial D	Açaí	19025
Filial A	Iogurte	19050
Filial C	Sabonete Líquido	19038
Filial B	Chá Verde	18951
Filial A	Espanja	19317
Filial E	Gin Importado	18622

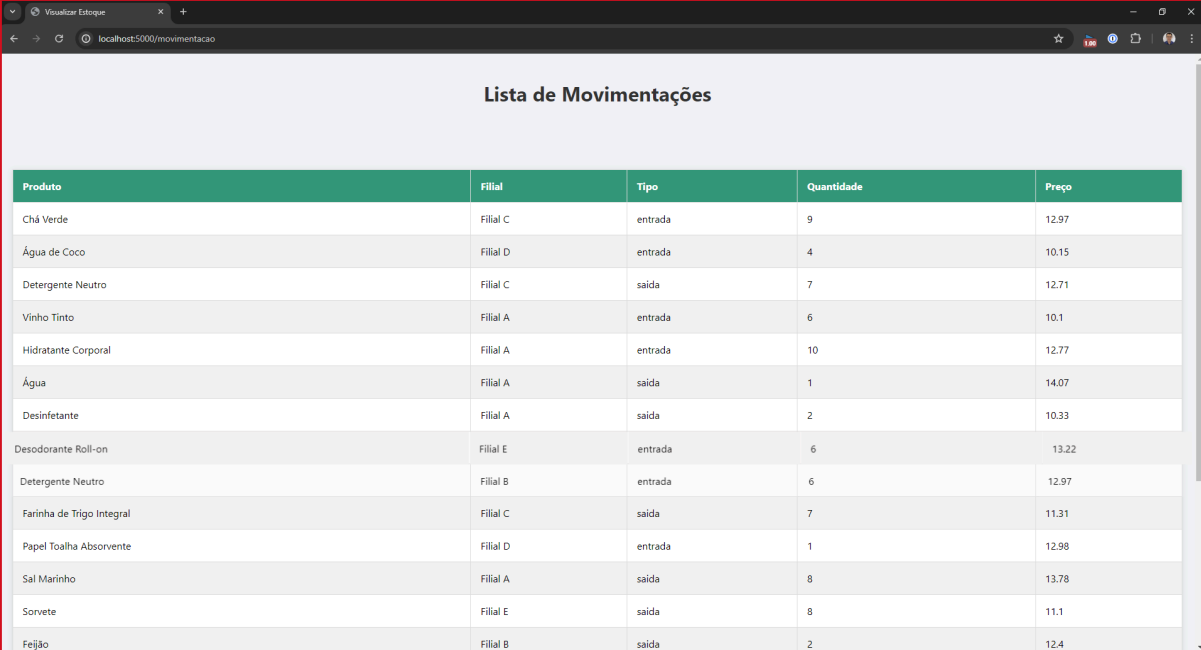
- **Atualizações de inventário.**

Para consultar as últimas atualizações de inventário, nós realizamos query na rota do flask, requisitando as últimas 20 inserções/alterações.

```
@app.route('/api/movimentacao')
def listar_movimentacao():
    # Consulta todos os documentos na coleção 'movimentacao'
    lista_movimentacao = list(movimentos_estoque.find().sort('_id', -1).limit(20)) # Exclui o campo '_id' do resultado
    # Converte os documentos BSON para JSON serializável
    movimentacao_json = json_util.dumps(lista_movimentacao)
    return movimentacao_json
```

Não é possível realizar query de todas as movimentações, pois corre o risco de derrubar o sistema dada a alta quantidade de itens.

No front-end é listado as últimas 20 movimentações na rota `@app.route('/movimentacao')`.



Produto	Filial	Tipo	Quantidade	Preço
Chá Verde	Filial C	entrada	9	12.97
Água de Coco	Filial D	entrada	4	10.15
Detergente Neutro	Filial C	saida	7	12.71
Vinho Tinto	Filial A	entrada	6	10.1
Hidratante Corporal	Filial A	entrada	10	12.77
Água	Filial A	saida	1	14.07
Desinfetante	Filial A	saida	2	10.33
Desodorante Roll-on	Filial E	entrada	6	13.22
Detergente Neutro	Filial B	entrada	6	12.97
Farinha de Trigo Integral	Filial C	saida	7	11.31
Papel Toalha Absorvente	Filial D	entrada	1	12.98
Sal Marinho	Filial A	saida	8	13.78
Sorvete	Filial E	saida	8	11.1
Feijão	Filial B	saida	2	12.4

- **Código utilizado.**

Todo o código utilizado para elaboração deste relatório pode ser acessado no seguinte link do github: <https://github.com/robsonglima/Estoque-Supermercado-PUC-Minas>. Os seguintes inputs devem ser utilizados para rodar:

- Ativar MongoDB e conectar na porta 27017
- Iniciar o flask: `python run.py`
- Acessar o site <http://localhost:5000/>
- Para criar o banco de dados, collections e popula-los, basta rodar os seguintes scripts:
 - `entradas.py`
 - `saidas_em_massa.py`
 - `saida_entrada_em_massa.py`

Conclusão

É imprescindível entender que o MongoDB oferece vantagens significativas para o sistema criado de Gerenciamento de Estoque da Rede de Supermercados VARIT. Com recursos direcionados para escalabilidade horizontal, sharding, particionamento, entre outros, ele lida eficientemente com grandes volumes de dados entre filiais geograficamente dispersas. O sharding distribui os dados uniformemente, garantindo desempenho consistente conforme o sistema cresce. O particionamento organizado dos dados facilita consultas específicas e atualizações eficientes, os processos de read/writes. Além disso, a configuração de replica set proporciona alta disponibilidade, garantindo a continuidade do serviço em caso de falha. Em conjunto, esses recursos tornam o MongoDB uma escolha poderosa para o desenvolvimento de diversos sistemas.

Este projeto não apenas permitiu a aplicação de conceitos de gerenciamento de dados distribuídos e escalabilidade, mas também destacou a importância de escolher ferramentas adequadas para atender aos requisitos específicos do sistema. A experiência adquirida proporcionou um entendimento prático de como projetar e implementar sistemas robustos para aplicações comerciais escaláveis.

O projeto de desenvolvimento do sistema de gerenciamento de estoque envolveu a superação de vários desafios relacionados à configuração de ambiente, conexão com banco de dados, desenvolvimento de funcionalidades e integração frontend-backend. A resolução desses desafios não apenas permitiu que a aplicação funcionasse corretamente, mas também proporcionou uma valiosa experiência em lidar com problemas comuns de desenvolvimento web com MongoDB.

Por fim, é recomendável a implementação de testes automatizados e a configuração de um ambiente de integração contínua. Esses passos garantirão que novas funcionalidades e correções de bugs sejam implementadas de maneira segura e eficiente, reduzindo o risco de introduzir novos erros ao sistema. A adoção de práticas ágeis no desenvolvimento permitirá ajustes rápidos e iterativos, alinhando o progresso do projeto com as necessidades dos usuários e as exigências do mercado.