



**PUC Minas**

**Implementação de Estratégias de  
Particionamento de Dados em MongoDB para  
Otimização de Desempenho**

**Paulo Victor Sobrinho de Jesus**

**2024**

## Objetivo:

O objetivo desta atividade é permitir que se apliquem conceitos de dados em um ambiente de banco de dados distribuído. Eles devem projetar um sistema que seja escalável e eficientemente particionado para lidar com um grande volume de dados.

## Requisitos Iniciais:

Cada filial possui um grande volume de produtos em seu estoque.

O sistema precisa ser capaz de lidar com milhões de registros de produtos.

A consulta de estoque e atualizações de inventário devem ser rápidas e eficientes.

A escalabilidade do sistema é essencial, pois novas filiais podem ser adicionadas no futuro.

## Configuração do ambiente

Para configurar um cluster MongoDB que utiliza replicação e sharding, vamos precisar configurar três tipos principais de serviços ou componentes. Cada um desses serviços desempenha um papel específico na gestão de dados e na garantia de alta disponibilidade e escalabilidade. Aqui estão eles:

Repositório Git: <https://github.com/pvsobrinho/puc-minas-nosql-docker-final>

### 1. Config Servers

- **Função:** Os servidores de configuração armazenam as metadados do cluster de sharding. Estas informações incluem a estrutura e a localização dos dados dentro do cluster, mapeando quais fragmentos (shards) contêm quais dados.

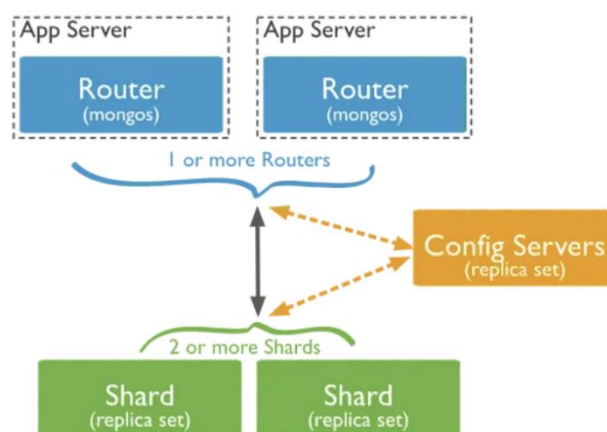
- **Importância:** Eles são cruciais para o funcionamento do cluster porque sem os dados de configuração, o sistema não saberia onde encontrar partes específicas dos dados distribuídos pelo cluster.
- **Redundância:** Geralmente, é recomendado ter um conjunto de réplicas de servidores de configuração para garantir redundância e alta disponibilidade.

## 2. Shard Servers

- **Função:** Cada servidor de shard armazena uma parte dos dados do banco de dados. Em um cluster shardado, os dados são divididos horizontalmente em fragmentos que são distribuídos entre vários servidores de shards.
- **Importância:** O sharding permite que o banco de dados distribua a carga de dados e as consultas entre vários servidores, melhorando a capacidade de leitura/escrita e a escalabilidade do sistema.
- **Configuração:** Cada shard pode ser um servidor standalone ou um conjunto de réplicas. A utilização de conjuntos de réplicas é recomendada para garantir a disponibilidade e a durabilidade dos dados.

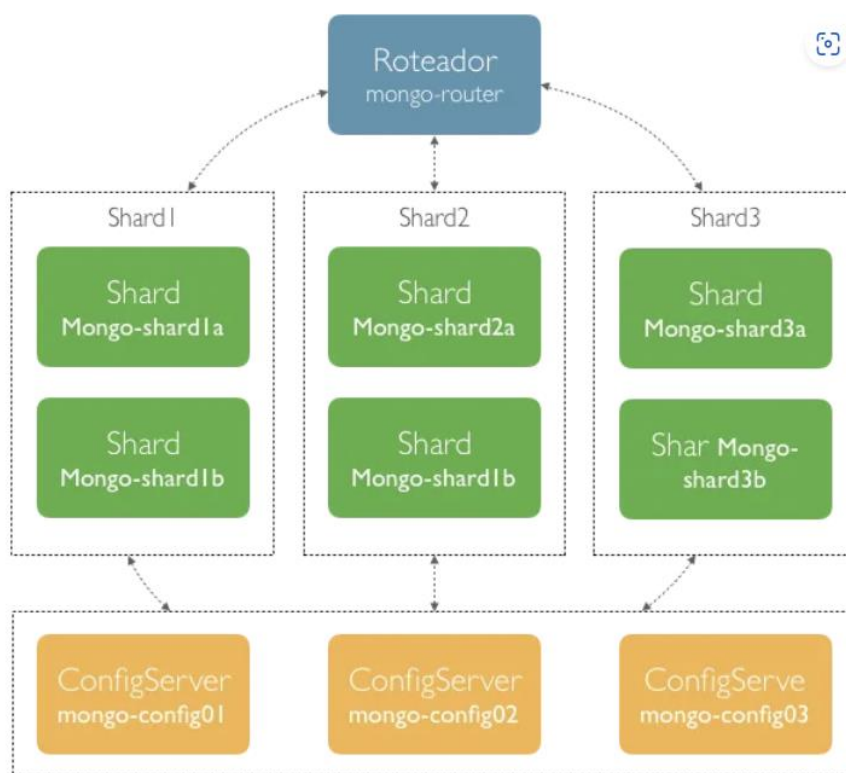
## 3. Query Routers (mongos)

- **Função:** O mongos atua como uma interface de consulta entre os clientes do banco de dados e o cluster MongoDB. Ele direciona as operações de consulta para os shards apropriados e agrega os resultados.
- **Importância:** O router de consultas simplifica a complexidade do cliente ao interagir com um cluster distribuído, fazendo parecer que o cliente está lidando com uma única instância de banco de dados.
- **Configuração:** Em um ambiente de produção, pode haver múltiplos routers de consultas para distribuir e balancear a carga de consultas entre eles.



Tipos de serviços do MongoDB

Vamos criar o cluster com 3 nós replicantes a representação gráfica seria similar a isso:



Arquitetura do cluster a ser criado

## Configurando o Docker e MongoDB

1º passo vamos criar as instancias e Shared. Os comandos do passo a passo estão contidos nas imagens.

```
Windows PowerShell
O Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\puc-teste\nosql> docker-compose up -d
time="2024-06-04T20:17:11-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 0/5
- shard2_1 Pulling                                4.8s
- configsvr1 Pulling                               4.8s
- shard1_1 Pulling                                 4.8s
- mongos Pulling                                   4.8s
- shard1_2 Pulling                                 4.8s
```

```
Windows PowerShell
0 Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\puc-teste\nosql> docker-compose up -d
time="2024-06-04T20:17:11-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 7/13
- shard2_1 Pulling 26.4s
- configsvr1 Pulling 26.4s
- shard1_1 Pulling 26.4s
- mongos [#####] 208.2MB / 265.5MB Pulling 26.4s
  ✓a8b1c5f80c2d Pull complete 3.2s
  ✓4e1f192fe085 Pull complete 0.7s
  ✓329c0ae46358 Pull complete 0.9s
  ✓a41d483e5ca3 Pull complete 1.5s
  ✓e9ce4a7bbe77 Pull complete 1.5s
  ✓57884d3aea66 Pull complete 2.2s
  - 5117a1be0e20 Downloading [=====]... 20.7s
  ✓4a739f30e1d2 Download complete 3.0s
- shard1_2 Pulling 26.4s
```

```
Windows PowerShell
0 Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

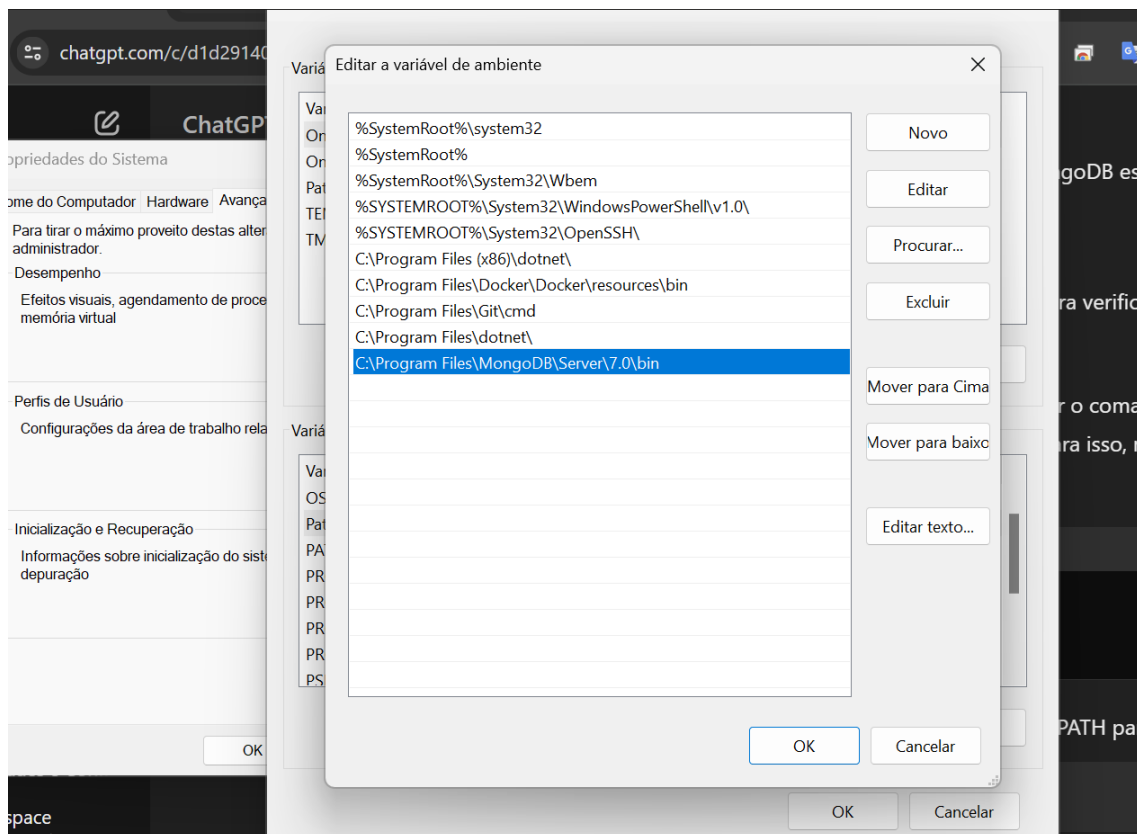
Instale o PowerShell mais recente para obter novos recursos e aprimoramentos! https://aka.ms/PSWindows

PS C:\puc-teste\nosql> docker-compose up -d
time="2024-06-04T20:17:11-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 13/13
  ✓shard2_1 Pulled 62.2s
  ✓configsvr1 Pulled 62.2s
  ✓shard1_1 Pulled 62.2s
  ✓mongos Pulled 62.2s
  ✓a8b1c5f80c2d Pull complete 3.2s
  ✓4e1f192fe085 Pull complete 0.7s
  ✓329c0ae46358 Pull complete 0.9s
  ✓a41d483e5ca3 Pull complete 1.5s
  ✓e9ce4a7bbe77 Pull complete 1.5s
  ✓57884d3aea66 Pull complete 2.2s
  ✓5117a1be0e20 Pull complete 30.2s
  ✓4a739f30e1d2 Pull complete 3.0s
  ✓shard1_2 Pulled 62.2s
[+] Running 6/6
  ✓Network nosql_default Created 0.6s
  ✓Container nosql-shard2_1-1 Started 4.9s
  ✓Container nosql-mongos-1 Started 4.9s
  ✓Container nosql-shard1_1-1 Started 4.9s
  ✓Container nosql-shard1_2-1 Started 4.9s
  ✓Container nosql-configsvr1-1 Started 4.9s
PS C:\puc-teste\nosql>
```

Após este processo precisamos obter a ID do container

```
PS C:\puc-teste\nosql> docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
19e603ac8bf3   mongo     "docker-entrypoint.s..." 6 minutes ago Up 6 minutes  0.0.0.0:27017->27017/tcp            nosql-mongos-1
9ab2e44c2a3f   mongo     "docker-entrypoint.s..." 6 minutes ago Up 6 minutes  27017/tcp, 0.0.0.0:27020->27019/tcp  nosql-shard1_2-1
b0dd4ed61ab7   mongo     "docker-entrypoint.s..." 6 minutes ago Up 6 minutes  27017/tcp, 0.0.0.0:27019->27019/tcp  nosql-configsvr1-1
44812eee2902   mongo     "docker-entrypoint.s..." 6 minutes ago Up 6 minutes  27017/tcp, 0.0.0.0:27018->27018/tcp  nosql-shard1_1-1
dc8f7b80ae3f   mongo     "docker-entrypoint.s..." 6 minutes ago Up 6 minutes  27017/tcp, 0.0.0.0:27021->27021/tcp  nosql-shard2_1-1
PS C:\puc-teste\nosql>
```

A depender dos eu ambiente de desenvolvimento talvez seja necessário configurar a variável de ambiente para usar o shell do mongo para reconhecimento de algumas palavras reservadas e variáveis



Programa em Python usa a instancia do Docker mongos.

Precisamos também instalar o cliente na maquina Docker. Mesmo que já tenha instalado em seu computador. Instalar dentro da maquina Docker nos trará versatilidade para manipular a instancia do banco de dados dentro do próprio Docker

```
root@b0dd4ed61ab7:/# apt-get update
apt-get install -y mongodb-clients
Ign:1 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:3 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 Release [2090 B]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1085 kB]
Get:5 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 Release.gpg [866 B]
Get:6 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:7 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0/multiverse amd64 Packages [40.9 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [44.7 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1858 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2395 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
36% [13 Packages 2240 kB/17.5 MB 13%]
```

Agora podemos testar as conexões das instancias/ shards etc..

```
PS C:\puc-teste\nosql> Test-NetConnection -ComputerName localhost -Port 27019
```

```
ComputerName      : localhost
RemoteAddress     : ::1
RemotePort        : 27019
InterfaceAlias    : Loopback Pseudo-Interface 1
SourceAddress     : ::1
TcpTestSucceeded  : True
```

```
PS C:\puc-teste\nosql> Test-NetConnection -ComputerName localhost -Port 27018
```

```
ComputerName      : localhost
RemoteAddress     : ::1
RemotePort        : 27018
InterfaceAlias    : Loopback Pseudo-Interface 1
SourceAddress     : ::1
TcpTestSucceeded  : True
```

```
PS C:\puc-teste\nosql> Test-NetConnection -ComputerName localhost -Port 27021
```

```
ComputerName      : localhost
RemoteAddress     : ::1
RemotePort        : 27021
InterfaceAlias    : Loopback Pseudo-Interface 1
SourceAddress     : ::1
TcpTestSucceeded  : True
```

```
PS C:\puc-teste\nosql> Test-NetConnection -ComputerName localhost -Port 27017
```

```
ComputerName      : localhost
RemoteAddress     : ::1
RemotePort        : 27017
InterfaceAlias    : Loopback Pseudo-Interface 1
SourceAddress     : ::1
TcpTestSucceeded  : True
```

Caso enfrente problemas pode remover e recriar (Esta etapa da próxima imagem é Opcional)

```

PS C:\puc-teste\nosql> docker-compose down
time="2024-06-04T21:06:26-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 6/6
  ✓ Container nosql-shard2_1-1 Removed 1.4s
  ✓ Container nosql-configsvr1-1 Removed 1.6s
  ✓ Container nosql-shard1_1-1 Removed 1.7s
  ✓ Container nosql-shard1_2-1 Removed 1.3s
  ✓ Container nosql-mongos-1 Removed 12.5s
  ✓ Network nosql_default Removed 0.4s
PS C:\puc-teste\nosql> docker-compose up -d
time="2024-06-04T21:06:40-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 6/6
  ✓ Network nosql_default Created 0.1s
  ✓ Container nosql-mongos-1 Started 0.3s
  ✓ Container nosql-shard2_1-1 Started 0.3s
  ✓ Container nosql-shard1_2-1 Started 0.3s
  ✓ Container nosql-configsvr1-1 Started 0.3s
  ✓ Container nosql-shard1_1-1 Started 0.3s
PS C:\puc-teste\nosql>

```

```

PS C:\puc-teste\nosql> docker exec -it nosql-mongos-1 /bin/bash
root@e39e3d7dd2d3:/# apt-get update
Ign:1 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 InRelease
Get:2 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 Release [2090 B]
Get:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:4 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 Release.gpg [866 B]
Get:5 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:6 http://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0/multiverse amd64 Packages [40.9 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [44.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2395 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1858 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [1085 kB]
54% [11 Packages 9496 kB/17.5 MB 54%]

```

Uma outra alternativa é recriar os containeres fora do Docker desktop (Também é uma etapa opcional ou uma alternativa para configuração do ambiente )

```

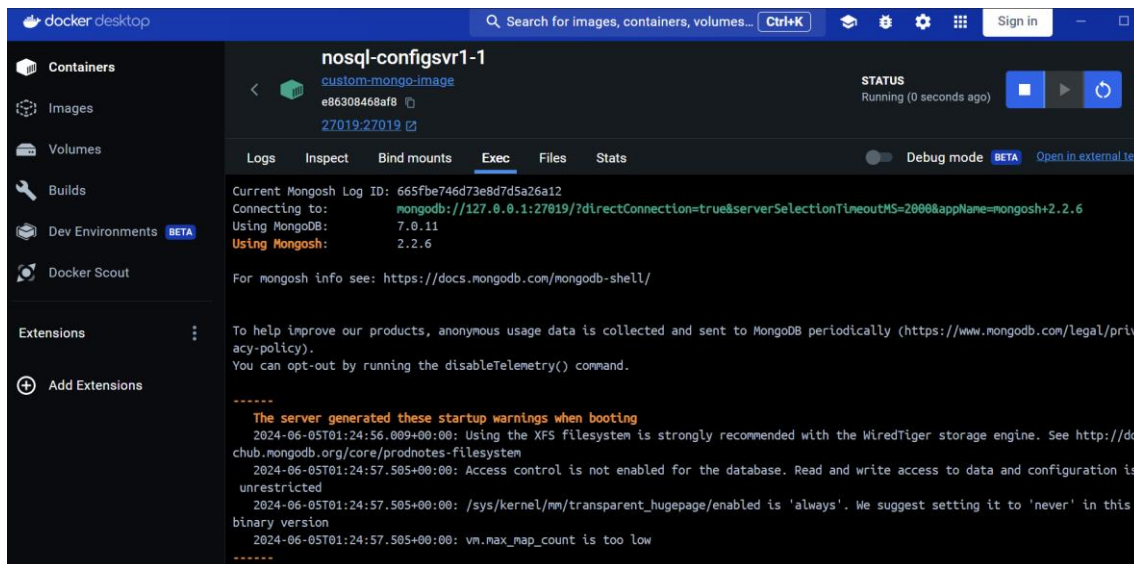
Windows PowerShell
Start a build
PS C:\puc-teste\nosql> docker build -t custom-mongo-image .
[+] Building 0.5s (9/9) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 682B 0.0s
=> [internal] load metadata for docker.io/library/mongo:latest 0.0s
=> [internal] load .dockerignore 0.0s
=> transferring context: 2B 0.0s
=> [1/5] FROM docker.io/library/mongo:latest 0.0s
=> CACHED [2/5] RUN apt-get update && apt-get install -y wget gnupg 0.0s
=> CACHED [3/5] RUN wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | apt-key add - 0.0s
=> CACHED [4/5] RUN echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 mul 0.0s
=> CACHED [5/5] RUN apt-get update && apt-get install -y mongodb-org-shell 0.0s
=> exporting to image 0.4s
=> => exporting layers 0.3s
=> writing image sha256:3e98e9a74455eed56d0c85d691d83e0abe023e206a192411e3896d434ee701dd 0.0s
=> naming to docker.io/library/custom-mongo-image 0.0s

What's Next?
View a summary of image vulnerabilities and recommendations -> docker scout quickview
PS C:\puc-teste\nosql> docker-compose up -d
time="2024-06-04T21:31:18-03:00" level=warning msg="C:\\puc-teste\\nosql\\docker-compose.yml: 'version' is obsolete"
[+] Running 2/5
  - Container nosql-mongos-1 Recreate 5.8s
  ✓ Container nosql-shard2_1-1 Recreated 5.7s
  - Container nosql-configsvr1-1 Recreate 5.8s
  ✓ Container nosql-shard1_1-1 Recreated 5.7s
  - Container nosql-shard1_2-1 Recreate 5.8s

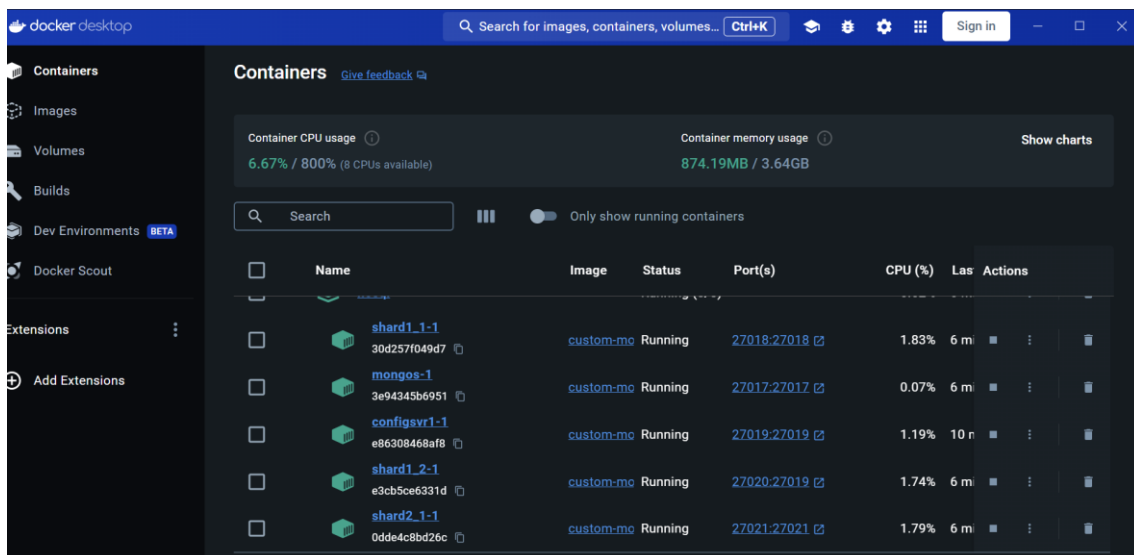
```

Instalamos o mongosh e podemos manipular os scripts do Mongo através dele diretamente na interface Docker.





Pronto. Agora temos todas as instancias rodando



No inicio deste documento explica o papel de cada instância destas da imagem acima.

O mongos atua como uma interface de consulta que interage com os clientes externos. Ele direciona as operações para os shards apropriados e agrega os resultados das consultas. Para adicionar coleções, criar índices, distribuir dados, e realizar consultas em um ambiente sharded, você interage através do mongos, que gerencia a complexidade do cluster de shards nos bastidores.

E finalmente neste momento vamos configurar a ligação dos nós.

nosql-shard1\_1-1

<



custom-mongo-image

30d257f049d7 

27018:27018 

Logs

Inspect

Bind mounts

Exec

Files


Stats

```
rs.initiate()
{
  info2: 'no configuration specified. Using a default configuration for the set',
  me: '30d257f049d7:27018',
  ok: 1
}
rs-shard-1 [direct: other] test>
```


Uma maneira de verificar se os shards estão funcionando corretamente é através deste comando. Também é importante que eles estejam definidos como PRIMARY


nosql-shard1\_1-1

<



custom-mongo-image

30d257f049d7 

27018:27018 

Logs

Inspect

Bind mounts

Exec

Files

Stats

STATUS

Running

```
rs.status()
{
  set: 'rs-shard-1',
  date: ISODate('2024-06-05T19:37:17.380Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 1,
  writeMajorityCount: 1,
  votingMembersCount: 1,
  writableVotingMembersCount: 1,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2024-06-05T19:37:13.713Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2024-06-05T19:37:13.713Z'),
    lastDurableWallTime: ISODate('2024-06-05T19:37:13.713Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1717616203, i: 1 }),
  electionCandidateMetrics: {
```

```
},
lastStableRecoveryTimestamp: Timestamp({ t: 1717616203, i: 1 }),
electionCandidateMetrics: {
  lastElectionReason: 'electionTimeout',
  lastElectionDate: ISODate('2024-06-05T19:28:43.580Z'),
  electionTerm: Long('1'),
  lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1717615723, i: 1 }), t: Long('-1') },
  lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1717615723, i: 1 }), t: Long('-1') },
  numVotesNeeded: 1,
  priorityAtElection: 1,
  electionTimeoutMillis: Long('10000'),
  newTermStartDate: ISODate('2024-06-05T19:28:43.651Z'),
  wMajorityWriteAvailabilityDate: ISODate('2024-06-05T19:28:43.714Z')
},
members: [
  {
    _id: 0,
    name: '30d257f049d7:27018',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 65316,
    optime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2024-06-05T19:37:13.000Z'),
```

```
optime: { ts: Timestamp({ t: 1717616233, i: 1 }), t: Long('1') },
optimeDate: ISODate('2024-06-05T19:37:13.000Z'),
lastAppliedWallTime: ISODate('2024-06-05T19:37:13.713Z'),
lastDurableWallTime: ISODate('2024-06-05T19:37:13.713Z'),
syncSourceHost: '',
syncSourceId: -1,
infoMessage: '',
electionTime: Timestamp({ t: 1717615723, i: 2 }),
electionDate: ISODate('2024-06-05T19:28:43.000Z'),
configVersion: 1,
configTerm: 1,
self: true,
lastHeartbeatMessage: ''
}
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1717616233, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
operationTime: Timestamp({ t: 1717616233, i: 1 })
```

Pronto. Finalizada esta etapa agora podemos ir para o nosso programa em python. O arquivo inicial de configuração o Docker compose e DockerFile estão no repositório GIT. Abaixo uma inicial que foi utilizada para fazer a configuração acima versão do arquivo:

### Arquivo DockerCompose:

version: '3.8'

services:

configsvr1:

image: custom-mongo-image

command: ["mongod", "--configsvr", "--replSet", "rs-config", "--port", "27019"]

ports:

- 27019:27019

volumes:

- ./data/config/configsvr1:/data/db

shard1\_1:

image: custom-mongo-image

command: ["mongod", "--shardsvr", "--replSet", "rs-shard-1", "--port", "27018"]

ports:

- 27018:27018

volumes:

- ./data/shard1\_1:/data/db

shard1\_2:

image: custom-mongo-image

command: ["mongod", "--shardsvr", "--replSet", "rs-shard-1", "--port", "27019"]

ports:

- 27020:27019

volumes:

- ./data/shard1\_2:/data/db

shard2\_1:

image: custom-mongo-image

command: ["mongod", "--shardsvr", "--replSet", "rs-shard-2", "--port", "27021"]

ports:

- 27021:27021

volumes:

- ./data/shard2\_1:/data/db

mongos:

image: custom-mongo-image

command: ["mongos", "--configdb", "rs-config/configsvr1:27019", "--port", "27017"]

ports:

- 27017:27017

### **Arquivo DockerFile:**

FROM mongo:latest

RUN apt-get update && apt-get install -y wget gnupg

# Adicionar a chave pública do MongoDB

```
RUN wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc |  
apt-key add -
```

# Adicionar o repositório do MongoDB à lista de fontes

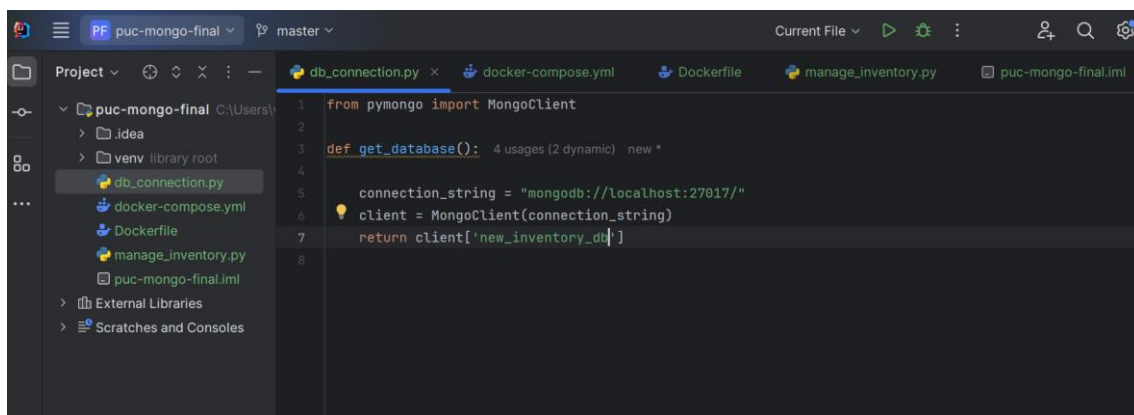
```
RUN echo "deb [ arch=amd64,arm64 ]  
https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4  
multiverse" | tee /etc/apt/sources.list.d/mongodb-org-4.4.list
```

# Atualizar a lista de pacotes e instalar o cliente mongo

```
RUN apt-get update && apt-get install -y mongodb-org-shell
```

## Aplicação em Python

Arquivo de conexão com banco de dados:



Aplicação onde tem as operações de CRUD da loja:

```
1 from db_connection import get_database
2 from pymongo.collection import Collection
3
4 db = get_database()
5
6 def create_branch(branch_name): 1 usage new *
7     db.create_collection(branch_name)
8
9 def add_product(branch_name, sku, name, price, stock): 1 usage new *
10     product = {
11         "sku": sku,
12         "name": name,
13         "price": price,
14         "stock": stock
15     }
16     collection: Collection = db[branch_name]
17     collection.insert_one(product)
18
19 def query_product(branch_name, sku): 1 usage new *
20     collection: Collection = db[branch_name]
21     return collection.find_one({"sku": sku})
22
```

```
23 def update_stock(branch_name, sku, new_stock): 1 usage new *
24     collection: Collection = db[branch_name]
25     collection.update_one( filter: {"sku": sku}, update: {"$set": {"stock": new_stock}})
26
27 # carga de dados - exemplo - para uma operação para uma quantidade de produtos em larga escala
28 # as chamadas via aplicação seriam desta forma em um ambiente onde pode executar uma grande quantidade de chamadas
29 # não haverá gargalos de desempenho por conta dos shards do mongo db
30
31 create_branch("Filial_1")
32 add_product( branch_name: "Filial_1", sku: "1234", name: "Teclado Mecânico", price: 150.00, stock: 30)
33 print(query_product( branch_name: "Filial_1", sku: "1234"))
34 update_stock( branch_name: "Filial_1", sku: "1234", new_stock: 25)
35
```