

5/27/2023

MongoDB Shard Cluster Setup

PISID 22/23



Fabian Gobet
97885
ISCTE-IUL

Índice

Índice de conteúdo

Índice.....	1
Índice de conteúdo	1
Índice de imagens	2
Disclaimer.....	3
Prefácio	4
Infraestrutura do cluster	4
Infraestrutura dos containers	6
Ficheiros de configuração mongo.....	9
Escolha das Shard Keys	12
Implementação	13
Passo 1:	13
Passo 2:	14
Passo 3:	15
Passo 4:	16
Passo 4.1:	16
Passo 4.2:	19
Passo 4.3:	20
Passo 4.4:	21
Passo 5:	22
Passo 6:	24
Passo 7:	26
Passo 8:	27
Passo 9 (extra):.....	29
Considerações finais	30
Bibliografia	33

Índice de imagens

Imagem 1. Infraestrutura do cluster	4
Imagem 2. Representação dos replica sets e nome	5
Imagem 3. IPS e portas de cada serviço mongodb - 1	7
Imagem 4. IPS e portas de cada serviço mongodb - 2	7
Imagem 5. Estrutura do directorio 'mongo'	8
Imagem 6. run0.sh script	8
Imagem 7. Ficheiro de configuração configsvr	9
Imagem 8. Ficheiro de configuração shardsvr	10
Imagem 9. Ficheiro de configuração de mongos 'router'	11
Imagem 10. Montagem dos containers	13
Imagem 11. Containers no Docker Desktop	14
Imagem 12. Acesso aos containers.....	14
Imagem 13. Execução dos scripts para lançar todas as instâncias de mongo.....	15
Imagem 14. Ligação ao cfg0.....	16
Imagem 15. Iniciação do replica set 'cfg'	16
Imagem 16. cfg rs.status() output (1/3).....	17
Imagem 17. cfg rs.status() output (2/3).....	18
Imagem 18. cfg rs.status() output (3/3).....	18
Imagem 19. Ligação ao a0.....	19
Imagem 20. Iniciação do replica set 'a'	20
Imagem 21. Iniciação do replica set 'b'	21
Imagem 22. Iniciação do replica set 'c'	22
Imagem 23. Ligação ao mongos na maquina 4.....	23
Imagem 24. Criação dos user root e admin	23
Imagem 25. Adição das 3 replicas 'a','a' e 'c' como shards.....	24
Imagem 26. sh.status output (1/2)	25
Imagem 27. sh.status output (2/2)	26
Imagem 28. Criação da db, collections e user javaop.....	26
Imagem 29. Enable e sharding das coleções (1/2)	27
Imagem 30. Enable e sharding das coleções (2/2)	28
Imagem 31. Mudança do port para segundo mongos 'router'.....	29
Imagem 32. Execução do segundo mongos.....	29
Imagem 33. Exemplo de URI de ligação ao nosso cluster.....	30
Imagem 34. Sucesso na ligação com o URI	31
Imagem 35. Sumário das shards em cluster	31

Disclaimer

Este documento foi pensado como um guia pratico para implementar um Shard Cluster para o MongoDB. Posto isto, o prefácio discrimina o ambiente de base e configurações rudimentares que poderão ajudar a perceber a natureza de objetos e valores na implementação.

Sendo o prefácio de rigor predominantemente textual, para uma perceção rápida da implementação do cluster pode-se passar logo para a o capítulo de 'Implementação' e seguir as imagens, podendo pequenas dúvidas técnicas serem justificadas com consulta ao prefácio.

A cópia direta dos comandos deste documento está suscetível a caracteres ou formatações cujo as shells (mongosh, cmd, bash) podem interpretar como um erro.

Prefácio

No âmbito da unidade curricular de Projeto de Integração de Sistemas Distribuídos (PISID) - 2022/2023, lecionada pelo Prof. Joaquim Esmerado e Prof. Pedro Ramos, no ISCTE-IUL, este documento visa sintetizar a implementação de um *MongoDB Shard Cluster* [1] aplicado ao projeto do grupo 14 de PISID 22/23. Para este efeito é importante ter em consideração o *setup* inicial das nossas máquinas e as estruturas que suportam esta aplicação.

Para conseguirmos simular várias máquinas independentes iremos fazer uso da tecnologia dos *Docker Containers*, forçando o *routing* externo das aplicações presentes em cada um dos *containers*. Isto é, cada uma das instâncias de MongoDB fará referência aos restantes membros do cluster através de um IP público. Como todos os containers irão estar a correr no mesmo dispositivo, o seu IP público será igual, com exceção à porta disponibilizada para aceder aos respetivos serviços de cada instância de MongoDB. O reencaminhamento dos serviços e dados da porta na máquina onde estão os containers para cada um dos containers é automaticamente tratado pelo *Docker Engine*. Naturalmente, esta solução implica, num ambiente doméstico, o *port forwarding* no router local e a adição de políticas na firewall com respeito a todas as portas a ser utilizadas.

Infraestrutura do cluster

De modo a acomodar aos requisitos do projeto e a garantir a robustez do cluster, consideremos a seguinte infraestrutura:

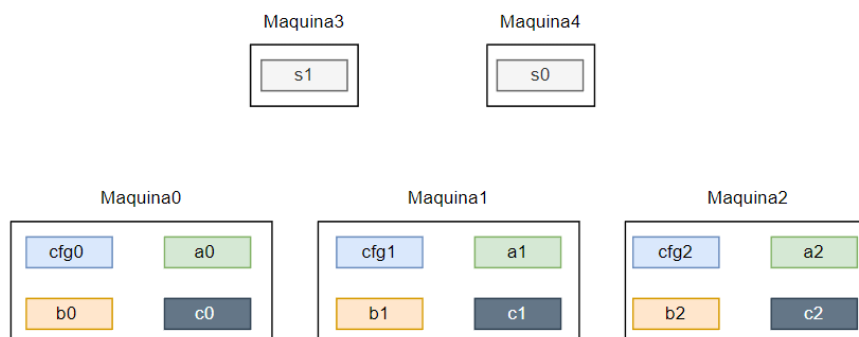


Imagem 1. Infraestrutura do cluster

As máquinas na infraestrutura da [Imagem 1](#) representam os vários containers, cuja implementação base será discriminada mais adiante.

Dentro da máquina 0 iremos ter 4 servidores a correr:

- O elemento 0 do *replica set* 'cfg'
- O elemento 0 do *replica set* 'a'
- O elemento 0 do *replica set* 'b'
- O elemento 0 do *replica set* 'c'

As máquinas 1 e 2 seguem uma lógica análoga, atendendo à diferença no índice.

Podemos também identificar 4 *replica sets* neste sistema:

- Os *replica set* com respeito aos *shards* 'a', 'b' e 'c'
- O *replica set* com respeito aos servidores de configuração do *cluster*



Imagem 2. Representação dos replica sets e nome

As máquinas 3 e 4 irão correr uma instância do *mongos* em cada, processo pelo qual se faz o acesso ao cluster para operações de configuração e operações CRUD sobre as bases de dados, ambas propagantes pelo *cluster*.

Infraestrutura dos containers

Em cada uma das máquinas iremos correr como sistema operativo o Ubuntu 22.04 (distribuição Linux). Para efeitos de ágil implementação, tomei a liberdade de criar de criar 4 imagens de Docker com as configurações e estrutura de diretórios previamente criada. Estas imagens estão disponíveis em <https://hub.docker.com/repositories/fabiangobet>. Cada imagem tem uma configuração genérica de base em Dockerfile correspondente à seguinte:

```
ARG INDX
FROM ubuntu
RUN apt-get install gnupg
RUN curl -fsSL https://pgp.mongodb.com/server-6.0.asc | gpg -o /usr/share/keyrings/mongodb-
server-6.0.gpg --dearmor
RUN echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg ]
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" | tee
/etc/apt/sources.list.d/mongodb-org-6.0.list
RUN apt-get update
RUN apt-get install -y mongodb-org
RUN mkdir mongo && cd mongo

# ----- CONFIG SERVER AND SHARD SETUP -----
RUN mkdir -p cfg${INDX}/data cfg${INDX}/log a${INDX}/data b${INDX}/data c${INDX}/data
a${INDX}/log b${INDX}/log c${INDX}/log
RUN touch cfg${INDX}/log/logs.log a${INDX}/log/logs.log b${INDX}/log/logs.log c${INDX}/log/logs.log

# APENAS GERAR KEYFILE NUMA MAQUINA E COPIAR PARA /mongo NAS OUTRAS
#RUN openssl rand -base64 756 > keyfile

#OU ENTAO COPIAR keyfile LOCAL
COPY ./keyfile /mongo/keyfile

RUN cd.. && chmod -R 700 mongo

# ----- MONGOS (ROUTER) SETUP -----
#RUN mkdir mongo && cd mongo
#RUN mkdir -p s${INDX}/data s${INDX}/log
#RUN touch s${INDX}/log/logs.log
#COPY ./keyfile /mongo/keyfile
#RUN chmod -R 700 mongo
```

Observações:

- Deve-se escolher ou server-config/shard ou mongos (router), comentando as linhas não necessárias
- Pode-se criar um keyfile numa das máquinas e depois copiar para as restantes, ou então pode utilizar-se uma já existente
- A keyfile deve estar presente no mesmo diretório que o Dockerfile
- A imagem constrói-se a partir do Dockerfile para a máquina 0 executando o comando
➤ 'docker build --build-arg INDX=0 machine 0 .' no mesmo diretório que o Dockerfile

Em alternativa podemos utilizar diretamente as imagens que estão no link docker hub previamente discriminado e construir os containers. A título de exemplo para a máquina 0 executamos o comando 'docker run -itd --name machine0 -p 37000:37000 -p 37010:37010 -p 37020:37020 -p 37030:37030 fabiangobet/mongocluster-machine0 '

Observações:

- O container é lançado em modo iterativo e detached da linha de comandos em que é executada
- As portas expostas para este exemplo estão diretamente correlacionadas com a configuração da máquina, sendo este tópico discutido na secção seguinte.
- Se as imagens não existirem localmente, o Docker Engine vai buscá-las ao repositório.

Como estamos a correr o os containers apenas numa máquina, o IP publico de todos os containers será o mesmo e os serviços estarão em portas diferentes como mencionado anteriormente. Para tal, Consideremos as seguintes tabelas síntese de IPs e portas.

Shard	CFG	A	B	C
Maquina0	ip0:37000	ip0:37010	ip0:37020	ip0:37030
Maquina1	ip1:37001	ip1:37011	ip1:37021	ip1:37031
Maquina2	ip2:37002	ip1:37012	ip2:37022	ip2:37032

Imagem 3. IPS e portas de cada serviço mongodb - 1

mongos (router)	s
Maquina3	ip3:37041
Maquina4	ip4:37040

Imagem 4. IPS e portas de cada serviço mongodb - 2

Para a nossa implementação vamos considerar que o a máquina onde estão os containers tem IP publico 46.189.143.63, isto é, ip0=ip1=ip2=ip3=ip4= 46.189.143.63.

Dentro de cada uma das imagens podemos encontrar a seguinte estrutura de diretórios (exemplo maquina1):

```
root@a713cf601487:/# tree mongo
mongo
|-- a1
|   |-- data
|   |-- log
|   '-- logs.log
|-- a1.conf
|-- b1
|   |-- data
|   |-- log
|   '-- logs.log
|-- b1.conf
|-- c1
|   |-- data
|   |-- log
|   '-- logs.log
|-- c1.conf
|-- cfg1
|   |-- data
|   |-- log
|   '-- logs.log
|-- cfg1.conf
'-- keyfile
```

Imagem 5. Estrutura do directorio 'mongo'

Neste diretório temos vários subdiretórios com respeito aos logs e dados de cada um dos servidores mongo cfg1,a1,b1,c1, e também os ficheiros de configuração de cada um dos anteriores(i.e. a1.conf).

Podemos também encontrar a keyfile, chave pelo qual os elementos dos replica set e clusters se autenticam perante os outros.

Cada um dos ficheiros de configuração terá uma estrutura ligeiramente diferente, à exceção das máquinas mongos ('routers'), cujo ficheiro de configuração difere mais.

A apresentação de um ficheiro de configuração exemplo será demonstrada na secção de 'Implementação'.

Também existe um script presente em cada uma das máquinas 0,1 e 2 com o nome runN.sh (onde N é o número da máquina), cuja função é lançar os 4 servidores mongo na máquina.

```
/run0.sh

1  mongod -f /mongo/cfg0.conf
2  mongod -f /mongo/a0.conf
3  mongod -f /mongo/b0.conf
4  mongod -f /mongo/c0.conf
```

Imagem 6. run0.sh script

Ficheiros de configuração mongo

Cada uma das instâncias de mongo terá um ficheiro de configuração diferente. Para tal, existem 3 tipo de configurações genéricas: config server, shard e mongos ('router')

Os ficheiros de configuração de um config server (i.e. cfg0.conf) têm a seguinte estrutura:

```
storage:
  dbPath: /mongo/cfg0/data

systemLog:
  destination: file
  logAppend: true
  path: /mongo/cfg0/log/logs.log

net:
  port: 37000
  bindIp: 0.0.0.0

processManagement:
  timeZoneInfo: /usr/share/zoneinfo

security:
  authorization: enabled
  keyFile: /mongo/keyfile

replication:
  replSetName: cfg

setParameter:
  enableLocalhostAuthBypass: true

processManagement:
  fork: true

sharding:
  clusterRole: configsvr
```

Imagem 7. Ficheiro de configuração configsvr

Aspetos importantes a notar são:

- O nome do replica set (cfg) que difere de replica para replica
- A port onde o serviço para este servidor vai ser disponibilizado (37000)
- O valor de clusterRole em 'configsvr'

Os ficheiros de configuração de um elemento de um shard (i.e. a0.conf) têm a seguinte estrutura:

```
storage:
  dbPath: /mongo/a0/data

systemLog:
  destination: file
  logAppend: true
  path: /mongo/a0/log/logs.log

net:
  port: 37010
  bindIp: 0.0.0.0

processManagement:
  timeZoneInfo: /usr/share/zoneinfo

security:
  authorization: enabled
  keyFile: /mongo/keyfile

replication:
  replSetName: a

setParameter:
  enableLocalhostAuthBypass: true

processManagement:
  fork: true

sharding:
  clusterRole: shardsvr
```

Imagem 8. Ficheiro de configuração shardsvr

Aspetos importantes a notar são:

- O nome do replica set (cfg) que difere de replica para replica
- A port onde o serviço para este servidor vai ser disponibilizado (37010)
- O valor de clusterRole em 'shardsvr'

Os ficheiros de configuração de um mongos 'router' (i.e. s0.conf) têm a seguinte estrutura:

```
systemLog:
  destination: file
  logAppend: true
  path: /mongo/s0/log/logs.log

net:
  port: 37040
  bindIp: 0.0.0.0

processManagement:
  timeZoneInfo: /usr/share/zoneinfo

security:
  keyFile: /mongo/keyfile

setParameter:
  enableLocalhostAuthBypass: true

processManagement:
  fork: true

sharding:
  configDB: cfg/46.189.143.63:37000,46.189.143.63:37001,46.189.143.63:37002
```

Imagem 9. Ficheiro de configuração de mongos 'router'

- Não tem nome de replica set
- Não tem diretório para dados
- A port onde o serviço para este servidor vai ser disponibilizado (37040)
- Em 'sharding' tem o nome da replica e os elementos do replica set dos config servers.

Existem várias opções que podemos considerar para um ficheiro de conf [2]. Para este projeto foram considerados especialmente 'enableLocalhostAuthBypass', 'authorization', e 'fork'.

É de notar que para simplificar este modelo o 'bindIP' foi posto a 0.0.0.0, podemos este valor ser modificado para corresponder aos IPs dos clientes que acedem aos serviços da máquina.

Escolha das Shard Keys

Criada a base de dados e as respetivas coleções, estamos em condições de aplicar sharding a cada uma das coleções. No entanto, devemos previamente refletir sobre importantes considerações a respeito da Shard Key [3] de cada coleção.

Os limites definitivos de um chunk e a sua localização em cada shard depende dos campos escolhidos para a indexação destes (shard key).

Existem diversos fatores a ter em consideração na escolha de um shard key de uma coleção, nomeadamente:

- Distribuição uniforme dos dados pelos shards
- Agrupamento de dados à luz de aspetos passíveis de pesquisa
- O tipo de queries que são feitos à base de dados

Uma pobre escolha de shard key pode levar a problemas como aglomeração excessiva de dados num único chunk (Jumbo Chunk) e/ou granularidade excessiva e posterior peso computacional em queries.

Desta forma, alguns dos aspetos a ter em consideração na escolha de uma shard key e na maneira como afetam o sistema, atendendo aos fatores mencionados, são o grau de aleatoriedade do campo, a sua monotonicidade e a sua cardinalidade.

Como tal, tendo em conta o projeto desenvolvido nesta UC e as características das coleções da nossa base de dados Mongo, uma escolha apropriada de shard key para cada uma das coleções é:

- mazemanage14 -> numExp:hashed, por ser monótono crescente em sentido lato e com imensas queries ao numExp
- mazelog14 -> Hora:hashed, por ser monótono crescente em sentido lato e com imensas queries à Hora
- mazetemp14 -> numExp: hashed, por ser monótono crescente em sentido lato e com imensas queries ao numExp
- mazemov14 -> numExp: hashed, por ser monótono crescente em sentido lato e com imensas queries ao numExp

Implementação

Passo 1:

Atendendo à estrutura do nosso sistema mongo ([Imagem 1](#)) e à configuração de rede pretendida ([Imagem 2](#), [Imagem 3](#) e [Imagem 4](#)) iremos montar os containers e polos a correr. Para tal executamos os seguintes comandos:

- `docker run -itd --name machine0 -p 37000:37000 -p 37010:37010 -p 37020:37020 -p 37030:37030 fabiangobet/mongocluster-machine0:1`
- `docker run -itd --name machine1 -p 37001:37001 -p 37011:37011 -p 37021:37021 -p 37031:37031 fabiangobet/mongocluster-machine1:1`
- `docker run -itd --name machine2 -p 37002:37002 -p 37012:37012 -p 37022:37022 -p 37032:37032 fabiangobet/mongocluster-machine2:1`
- `docker run -itd --name machine3 -p 37041:37041 fabiangobet/mongocluster-router0:1`
- `docker run -itd --name machine4 -p 37040:37040 fabiangobet/mongocluster-router0:1`

```
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\WorkStation>docker run -itd --name machine0 -p 37000:37000 -p 37010:37010 -p 37020:37020 -p 37030:37030 fabiangobet/mongocluster-machine0:1
2b534719bde5b35196d51e3eb9c16b038d60ae49c47966cd1be197d15ca46453

C:\Users\WorkStation>docker run -itd --name machine1 -p 37001:37001 -p 37011:37011 -p 37021:37021 -p 37031:37031 fabiangobet/mongocluster-machine1:1
8d9bb2f717a09d85613d8b7deb08fed4bbeba8b1f8e19db56da8fea8d037bdd4

C:\Users\WorkStation>docker run -itd --name machine2 -p 37002:37002 -p 37012:37012 -p 37022:37022 -p 37032:37032 fabiangobet/mongocluster-machine2:1
066af1673965e3d113d6e1251f93adc45295d431d3b8ef8a4f26d0befd91d0fc

C:\Users\WorkStation>docker run -itd --name machine3 -p 37041:37041 fabiangobet/mongocluster-router0:1
a8d2a6783edb6539022f056c17697b6e1c52752c5a5e0fa6a854d726de121fa5

C:\Users\WorkStation>docker run -itd --name machine4 -p 37040:37040 fabiangobet/mongocluster-router0:1
56b0c23c86a149aaf02e91598b04b7ba505f553f79512ff2a33ff98eb8af0255
```

Imagem 10. Montagem dos containers

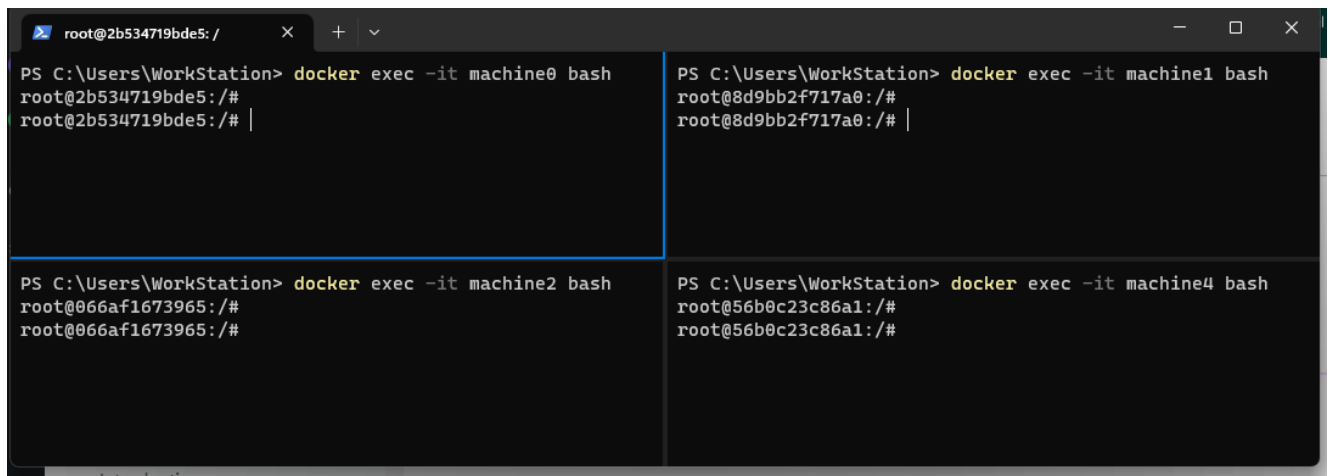
<input type="checkbox"/>	 machine0 2b534719bde5	fabiangobet/mongocluster	Running	37000:37000 Show all ports (4)	3 minutes ago			
<input type="checkbox"/>	 machine1 8d9bb2f717a0	fabiangobet/mongocluster	Running	37001:37001 Show all ports (4)	2 minutes ago			
<input type="checkbox"/>	 machine2 066af1673965	fabiangobet/mongocluster	Running	37002:37002 Show all ports (4)	2 minutes ago			
<input type="checkbox"/>	 machine3 a8d2a6783edb	fabiangobet/mongocluster	Running	37041:37041 Show all ports (4)	2 minutes ago			
<input type="checkbox"/>	 machine4 56b0c23c86a1	fabiangobet/mongocluster	Running	37040:37040 Show all ports (4)	2 minutes ago			

Imagem 11. Containers no Docker Desktop

Passo 2:

Abrir 4 terminais e executar os respetivos comandos para ligar a cada um dos containers.

- `docker exec -it machine0 bash`
- `docker exec -it machine1 bash`
- `docker exec -it machine2 bash`
- `docker exec -it machine4 bash`



```
PS C:\Users\WorkStation> docker exec -it machine0 bash
root@2b534719bde5:/#
root@2b534719bde5:/#

PS C:\Users\WorkStation> docker exec -it machine1 bash
root@8d9bb2f717a0:/#
root@8d9bb2f717a0:/#

PS C:\Users\WorkStation> docker exec -it machine2 bash
root@066af1673965:/#
root@066af1673965:/#

PS C:\Users\WorkStation> docker exec -it machine4 bash
root@56b0c23c86a1:/#
root@56b0c23c86a1:/#
```

Imagem 12. Acesso aos containers

Passo 3:

No terminal da machine 0,1 e 2 executar o respetivo script para correr as instâncias de mongo atendendo aos ficheiros de configuração (Imagem 6, Imagem 7, Imagem 8, Imagem 9)

- ./run0.sh
- ./run1.sh
- ./run2.sh

```

root@066af1673965: /
{ "t": { "$date": "2023-05-27T16:08:58.651Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 57
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:08:59.458Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 120
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:09:00.258Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 182
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:09:01.098Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 204
child process started successfully, parent exiting
root@2b534719bde5:/#

{ "t": { "$date": "2023-05-27T16:11:41.471Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 38
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:11:42.239Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 101
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:11:43.019Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 163
child process started successfully, parent exiting
{ "t": { "$date": "2023-05-27T16:11:43.811Z", "s": "I", "c": "CONTROL", "id": "5760901", "ctx": "-", "msg": "App
lied --setParameter options", "attr": { "serverParameters": { "enableLocalhostAuthBypass": { "default": true, "
value": true } } } } }
about to fork child process, waiting until server is ready for connections.
forked process: 225
child process started successfully, parent exiting
root@066af1673965:/#

PS C:\Users\WorkStation> docker exec -it machine4 bash
root@56b0c23c86a1:/#
root@56b0c23c86a1:/#
  
```

Imagem 13. Execução dos scripts para lançar todas as instâncias de mongo

Passo 4:

Passo 4.1:

A partir de agora iremos executar a iniciação dos replica sets todos a partir do terminal da máquina 0. Para tal, comecemos pelo replica set dos servidores de *config*. No Terminal da máquina 0 executar:

➤ `mongosh --port 37000`

```
root@2b534719bde5:/# mongosh --port 37000
Current Mongosh Log ID: 64722e1d7b2ba37e0ed7dab8
Connecting to:  mongodb://127.0.0.1:37000/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB:  6.0.5
Using Mongosh:  1.8.2

For mongosh info see: https://docs.mongodb.com/mongodbs-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.
```

Imagem 14. Ligação ao *cfg0*

E depois introduzir o seguinte comando:

```
rs.initiate(
{
  _id: "cfg",
  configsvr: true,
  members: [
    { _id : 0, host : "46.189.143.63:37000" },
    { _id : 1, host : "46.189.143.63:37001", priority: 0.9 },
    { _id : 2, host : "46.189.143.63:37002", priority: 0.9 }
  ]
}
)
```

```
test> rs.initiate(
... {
...   _id: "cfg",
...   configsvr: true,
...   members: [
...     { _id : 0, host : "46.189.143.63:37000" },
...     { _id : 1, host : "46.189.143.63:37001", priority: 0.9 },
...     { _id : 2, host : "46.189.143.63:37002", priority: 0.9 }
...   ]
... }
... )
{ ok: 1, lastCommittedOpTime: Timestamp({ t: 1685204518, i: 1 }) }
cfg [direct: other] test> |
```

Imagem 15. Iniciação do replica set 'cfg'

Neste momento devemos esperar um pouco para que os servidores da replica estabeleçam ligações entre si e deleguem um primary (cerca de 10s é suficiente).

Passados os 10 segundos, podemos executar o seguinte comando para verificar a integridade da réplica

➤ `rs.status()`

O output deve ser idêntico às seguintes imagens.

```
cfg [direct: primary] test> rs.status()
{
  set: 'cfg',
  date: ISODate("2023-05-27T16:26:10.251Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  configsvr: true,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1685204769, i: 1 }), t: Long("1") },
    lastCommittedWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1685204769, i: 1 }), t: Long("1") },
    appliedOpTime: { ts: Timestamp({ t: 1685204769, i: 1 }), t: Long("1") },
    durableOpTime: { ts: Timestamp({ t: 1685204769, i: 1 }), t: Long("1") },
    lastAppliedWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    lastDurableWallTime: ISODate("2023-05-27T16:26:09.803Z")
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1685204758, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate("2023-05-27T16:22:09.517Z"),
    electionTerm: Long("1"),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1685204518, i: 1 }), t: Long("-1") },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1685204518, i: 1 }), t: Long("-1") },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long("10000"),
    numCatchUpOps: Long("0"),
    newTermStartDate: ISODate("2023-05-27T16:22:09.561Z"),
    wMajorityWriteAvailabilityDate: ISODate("2023-05-27T16:22:10.527Z")
  },
  members: [
    {
      _id: 0,
      name: '46.189.143.63:37000',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 1032,
      optime: { ts: Timestamp({ t: 1685204769, i: 1 }), t: Long("1") },
      optimeDate: ISODate("2023-05-27T16:26:09.000Z"),
      lastAppliedWallTime: ISODate("2023-05-27T16:26:09.803Z"),
      lastDurableWallTime: ISODate("2023-05-27T16:26:09.803Z"),
```

Imagem 16. `cfg rs.status()` output (1/3)

```

    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '46.189.143.63:37001',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 251,
    optime: { ts: Timestamp({ t: 1685204768, i: 1 }), t: Long("1") },
    optimeDurable: { ts: Timestamp({ t: 1685204768, i: 1 }), t: Long("1") },
    optimeDate: ISODate("2023-05-27T16:26:08.000Z"),
    optimeDurableDate: ISODate("2023-05-27T16:26:08.000Z"),
    lastAppliedWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    lastDurableWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    lastHeartbeat: ISODate("2023-05-27T16:26:09.791Z"),
    lastHeartbeatRecv: ISODate("2023-05-27T16:26:08.806Z"),
    pingMs: Long("1"),
    lastHeartbeatMessage: '',
    syncSourceHost: '46.189.143.63:37000',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  },
  {
    _id: 2,
    name: '46.189.143.63:37002',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 251,
    optime: { ts: Timestamp({ t: 1685204768, i: 1 }), t: Long("1") },
    optimeDurable: { ts: Timestamp({ t: 1685204768, i: 1 }), t: Long("1") },
    optimeDate: ISODate("2023-05-27T16:26:08.000Z"),
    optimeDurableDate: ISODate("2023-05-27T16:26:08.000Z"),
    lastAppliedWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    lastDurableWallTime: ISODate("2023-05-27T16:26:09.803Z"),
    lastHeartbeat: ISODate("2023-05-27T16:26:09.791Z"),
    lastHeartbeatRecv: ISODate("2023-05-27T16:26:08.806Z"),
    pingMs: Long("1"),
    lastHeartbeatMessage: '',
    syncSourceHost: '46.189.143.63:37000',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  }
],

```

Imagem 17. `cfg rs.status()` output (2/3)

```

],
ok: 1,
lastCommittedOpTime: Timestamp({ t: 1685204769, i: 1 }),
'$clusterTime': {
  clusterTime: Timestamp({ t: 1685204769, i: 1 }),
  signature: {
    hash: Binary(Buffer.from("141ba7706d2a43ae16fbcf6ec7131631714064f2", "hex"), 0),
    keyId: Long("7237898339126083608")
  }
},
operationTime: Timestamp({ t: 1685204769, i: 1 })
}
cfg [direct: primary] test> |

```

Imagem 18. `cfg rs.status()` output (3/3)

Passo 4.2:

Uma vez estabelecida a réplica para os servidores config, iremos proceder da mesma forma para a réplica dos clusters 'a', 'b' e 'c'. Posto isto, saímos da shell do cfg0 e entramos na Shell do a0 com os seguintes comandos:

- exit
- mongosh --port 37010

```
cfg [direct: primary] test> exit
root@2b534719bde5:/# mongosh --port 37010
Current Mongosh Log ID: 64723104f2e4c28e6e63c3d8
Connecting to:  mongod://127.0.0.1:37010/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB:  6.0.5
Using Mongosh:  1.8.2

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

test>
```

Imagem 19. Ligação ao a0

De forma análoga à réplica cfg, executamos o seguinte comando:

```
rs.initiate(
{
  _id: "a",
  members: [
    { _id : 0, host : "46.189.143.63:37010" },
    { _id : 1, host : "46.189.143.63:37011", priority: 0.9 },
    { _id : 2, host : "46.189.143.63:37012", priority: 0.9 }
  ]
}
)
```

Após esperar 10 segundos podemos executar 'rs.status()' para verificar a integridade do replica set

```
test> rs.initiate(
...   {
...     _id: "a",
...     members: [
...       { _id : 0, host : "46.189.143.63:37010" },
...       { _id : 1, host : "46.189.143.63:37011", priority: 0.9 },
...       { _id : 2, host : "46.189.143.63:37012", priority: 0.9 }
...     ]
...   }
... )
{ ok: 1 }
a [direct: other] test> rs.status()
{
  set: 'a',
  date: ISODate("2023-05-27T16:36:31.791Z")
}
```

Imagem 20. Iniciação do replica set 'a'

Passo 4.3:

Iremos agora repetir o processo para a replica 'b. Posto isto, executamos

- exit
- mongosh --port 37020

Depois executamos:

```
rs.initiate(
{
  _id: "b",
  members: [
    { _id : 0, host : "46.189.143.63:37020" },
    { _id : 1, host : "46.189.143.63:37021", priority: 0.9 },
    { _id : 2, host : "46.189.143.63:37022", priority: 0.9 }
  ]
}
)
```

```
a [direct: primary] test> exit
root@2b534719bde5:/# mongosh --port 37020
Current Mongosh Log ID: 647233488c54a9bbdd9e7b45
Connecting to:  mongodb://127.0.0.1:37020/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB:  6.0.5
Using Mongosh:  1.8.2

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

test> rs.initiate(
... {
...   _id: "b",
...   members: [
...     { _id : 0, host : "46.189.143.63:37020" },
...     { _id : 1, host : "46.189.143.63:37021", priority: 0.9 },
...     { _id : 2, host : "46.189.143.63:37022", priority: 0.9 }
...   ]
... }
... )
{ ok: 1 }
b [direct: other] test> rs.status()
{
  set: 'b',
  date: ISODate("2023-05-27T16:44:24.232Z"),
  myState: 1,
  term: Long("1"),

```

Imagem 21. Iniciação do replica set 'b'

Esperamos 10 segundos e executamos:

- rs.status()

O processo e outputs deve ser idêntico aos anteriormente mostrados.

Passo 4.4:

Iremos agora repetir o processo para a replica 'c'. Posto isto, executamos

- exit
- mongosh --port 37030

Depois executamos:

```
rs.initiate(
{
  _id: "c",
  members: [
    { _id : 0, host : "46.189.143.63:37030" },
    { _id : 1, host : "46.189.143.63:37031", priority: 0.9 },
    { _id : 2, host : "46.189.143.63:37032", priority: 0.9 }
  ]
}
)
```

```
root@56b0c23c86a1:/# cd mongo
root@56b0c23c86a1:/mongo# mongos -f s0.conf
{"t":{"$date":"2023-05-27T16:57:08.261Z"},"s":"I", "c":"CONTROL", "id":5760001, "ctx":"","msg":"Applied --setParameter options","attr":{"serverParameters":{"enableLocalhostAuthBypass":{"default":true,"value":true}}}}
about to fork child process, waiting until server is ready for connections.
forked process: 51
child process started successfully, parent exiting
root@56b0c23c86a1:/mongo# mongosh --port 37040
Current Mongosh Log ID: 6472366e58d98fdeef63ffec
Connecting to:      mongodb://127.0.0.1:37040/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.8.2
Using MongoDB:      6.0.5
Using Mongosh:      1.8.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

[direct: mongos] test> |
```

Imagem 22. Iniciação do replica set 'c'

Esperamos 10 segundos e executamos:

- `rs.status()`

O processo e outputs deve ser idêntico aos anteriormente mostrados.

Passo 5:

As replicas estão configuradas. Iremos agora ligar-nos aos servidores de configuração do cluster através de uma instância de mongos da máquina 4 para começar a configurar o cluster. Para tal, na linha de comandos da máquina 4 executamos:

- `cd mongo`
- `mongos -f s0.conf`
- `mongosh --port 37040`

```
root@56b0c23c86a1:/# cd mongo
root@56b0c23c86a1:/mongo# mongos -f s0.conf
{"t":{"$date":"2023-05-27T16:57:08.261Z"},"s":"I", "c":"CONTROL", "id":5760901, "ctx":"-", "msg":"Applied --setParameter options", "attr":{"serverParameters":{"enableLocalhostAuthBypass":{"default":true,"value":true}}}}
about to fork child process, waiting until server is ready for connections.
forked process: 51
child process started successfully, parent exiting
root@56b0c23c86a1:/mongo# mongosh --port 37040
Current Mongosh Log ID: 6472366e58d98fdeef63ffec
Connecting to:      mongodb://127.0.0.1:37040/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.8.2
Using MongoDB:      6.0.5
Using Mongosh:      1.8.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

[direct: mongos] test> |
```

Imagem 23. Ligação ao mongos na máquina 4

Vamos criar dois utilizadores [4] de interesse para a nossa base de dados: o root e o administrador; e de seguida autenticamo-nos como root para avançar livremente nas restantes configurações. Executamos:

- use admin
- db.createUser({user:"root",pwd:"root",roles:[{role:"root",db:"admin"}]})
- db.auth('root','root')
- db.createUser({user:"admin",pwd:"admin",roles:[{role:"clusterAdmin",db:"admin"},{role:"readAnyDatabase",db:"admin"},"readWrite"]})

```
[direct: mongos] test> use admin
switched to db admin
[direct: mongos] admin> db.createUser({user:"root",pwd:"root",roles:[{role:"root",db:"admin"}]})
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685206697, i: 5 }),
    signature: {
      hash: Binary(Buffer.from("ef4356e89801646cad0356bd87e50a2a78a97d37", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685206697, i: 5 })
}
[direct: mongos] admin> db.auth('root','root')
{ ok: 1 }
[direct: mongos] admin> db.createUser({user:"admin",pwd:"admin",roles:[{role:"clusterAdmin",db:"admin"},{role:"readAnyDatabase",db:"admin"},"readWrite"]})
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685206760, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("90ed923854b9bd7d164240026f2c8265ac8a46d5", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685206760, i: 1 })
}
[direct: mongos] admin> |
```

Imagem 24. Criação dos user root e admin

Passo 6:

Estamos em condições de adicionar as 3 shards do nosso cluster com respeito às replicas 'a', 'b' e 'c'. Para tal executamos os seguintes comandos:

- `sh.addShard("a/46.189.143.63:37010,46.189.143.63:37011,46.189.143.63:37012")`
- `sh.addShard("b/46.189.143.63:37020,46.189.143.63:37021,46.189.143.63:37021")`
- `sh.addShard("c/46.189.143.63:37030,46.189.143.63:37031,46.189.143.63:37032")`

```
[direct: mongos] admin> sh.addShard("a/46.189.143.63:37010,46.189.143.63:37011,46.189.143.63:37012")
{
  shardAdded: 'a',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685207083, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("d6ee8c4e447a081e1c4ffde17d3c21d643b99191", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685207083, i: 1 })
}
[direct: mongos] admin> sh.addShard("b/46.189.143.63:37020,46.189.143.63:37021,46.189.143.63:37021")
{
  shardAdded: 'b',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685207092, i: 5 }),
    signature: {
      hash: Binary(Buffer.from("ab633a98edcad3a8d49791354e0ab4cb28815887", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685207092, i: 5 })
}
[direct: mongos] admin> sh.addShard("c/46.189.143.63:37030,46.189.143.63:37031,46.189.143.63:37032")
{
  shardAdded: 'c',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685207099, i: 8 }),
    signature: {
      hash: Binary(Buffer.from("960074326646ccf5ab2390aa74c1bfe1d656eff2", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685207099, i: 8 })
}
[direct: mongos] admin> |
```

Imagem 25. Adição das 3 replicas 'a','a' e 'c' como shards

Podemos de seguida executar 'sh.status()' para ver o estado do nosso cluster e dos shards.

➤ sh.status()

```
[direct: mongos] admin> sh.status()
shardingVersion
{
  _id: 1,
  minCompatibleVersion: 5,
  currentVersion: 6,
  clusterId: ObjectId("64722e3113e6f80fac7b3941")
}
---
shards
[
  {
    _id: 'a',
    host: 'a/46.189.143.63:37010,46.189.143.63:37011,46.189.143.63:37012',
    state: 1,
    topologyTime: Timestamp({ t: 1685207082, i: 6 })
  },
  {
    _id: 'b',
    host: 'b/46.189.143.63:37020,46.189.143.63:37021,46.189.143.63:37022',
    state: 1,
    topologyTime: Timestamp({ t: 1685207092, i: 3 })
  },
  {
    _id: 'c',
    host: 'c/46.189.143.63:37030,46.189.143.63:37031,46.189.143.63:37032',
    state: 1,
    topologyTime: Timestamp({ t: 1685207099, i: 6 })
  }
]
---
active mongoses
[ { '6.0.5': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {
```

Imagem 26. sh.status output (1/2)

```
collections: {
  'config.system.sessions': {
    shardKey: { _id: 1 },
    unique: false,
    balancing: true,
    chunkMetadata: [ { shard: 'a', nChunks: 1024 } ],
    chunks: [
      'too many chunks to print, use verbose if you want to force print'
    ],
    tags: []
  }
}
}
[direct: mongos] admin> |
```

Imagem 27. sh.status output (2/2)

Passo 7:

Vamos agora criar a base de dados do nosso projeto, as coleções desta e um utilizador para o java que vai interagir com a base de dados. Para tal, executamos:

- use mqttData
- db.createCollection("mazemov14")
- db.createCollection("mazetemp14")
- db.createCollection("mazelog14")
- db.createCollection("mazemanage14")
- db.createUser({user:"javaop",pwd:"javaop",roles:["readWrite"]})

```
[direct: mongos] admin> use mqttData
switched to db mqttData
[direct: mongos] mqttData> db.createCollection("mazemov14")
{ ok: 1 }
[direct: mongos] mqttData> db.createCollection("mazetemp14")
{ ok: 1 }
[direct: mongos] mqttData> db.createCollection("mazelog14")
{ ok: 1 }
[direct: mongos] mqttData> db.createCollection("mazemanage14")
{ ok: 1 }
[direct: mongos] mqttData> db.createUser({user:"javaop",pwd:"javaop",roles:["readWrite"]})
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685207694, i: 2 }),
    signature: {
      hash: Binary(Buffer.from("b2929ff30b125b3e4dd1efa5e66b1b21177975f3", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685207694, i: 2 })
}
[direct: mongos] mqttData> |
```

Imagem 28. Criação da db, collections e user javaop

Passo 8:

De seguida temos de anunciar a nossa base de dados como sendo elegível para sharding e de seguida inicializar em cada uma das coleções o processo de sharding. A justificação da escolha das key para sharding encontra-se na secção [Escolha das Shard Keys](#). Posto isto, executamos os seguintes comandos:

- `sh.enableSharding("mqttData")`
- `sh.shardCollection("mqttData.mazemov14",{"numExp":"hashed"})`
- `sh.shardCollection("mqttData.mazetemp14",{"numExp":"hashed"})`
- `sh.shardCollection("mqttData.mazelog14",{"Hora":"hashed"})`
- `sh.shardCollection("mqttData.mazemanager14",{"numExp":"hashed"})`

```
[direct: mongos] mqttData> sh.enableSharding("mqttData")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685209887, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("fea24be5a3dd234861eb1f454628794fe9644864", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685209887, i: 1 })
}
[direct: mongos] mqttData> sh.shardCollection("mqttData.mazemov14",{"numExp":"hashed"})
{
  collectionsSharded: 'mqttData.mazemov14',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685209918, i: 52 }),
    signature: {
      hash: Binary(Buffer.from("90dd7c57f33183fe8a32d8b6f4803a08ba9951eb", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685209918, i: 48 })
}
[direct: mongos] mqttData> sh.shardCollection("mqttData.mazetemp14",{"numExp":"hashed"})
{
  collectionsSharded: 'mqttData.mazetemp14',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685209928, i: 40 }),
    signature: {
      hash: Binary(Buffer.from("694c34994c320094947f85c1598972bb3afd8d1a", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685209928, i: 36 })
}
```

Imagem 29. Enable e sharding das coleções (1/2)

```
[direct: mongos] mqttData> sh.shardCollection("mqttData.mazelog14",{"Hora":"hashed"})
{
  collectionssharded: 'mqttData.mazelog14',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685209939, i: 35 }),
    signature: {
      hash: Binary(Buffer.from("c244f2b02f980a4541dba20b23a8d01dcafc2092", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685209939, i: 31 })
}
[direct: mongos] mqttData> sh.shardCollection("mqttData.mazemanage14",{"numExp":"hashed"})
{
  collectionssharded: 'mqttData.mazemanage14',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685209948, i: 40 }),
    signature: {
      hash: Binary(Buffer.from("73f6bf3e2f5a286457158785f3f25aa68d1317b2", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685209948, i: 36 })
}
[direct: mongos] mqttData> |
```

Imagem 30. Enable e sharding das coleções (2/2)

Para cada uma das coleções podemos verificar a distribuição dos seus dados ao longo dos vários shards, executando uma variante do comando [5]:

- db.mazemanage14.getShardDistribution()

Nota:

Caso existam dados na coleção é necessário primeiro indexar a coleção [6] e só aplicar o sharding, executando:

- db.mazemanage14.createIndex({"numExp":"hashed"})
- sh.shardCollection("mqttData.mazemanage14",{"numExp":"hashed"})

Passo 9 (extra):

Neste momento temos apenas uma instância de mongos ('router') associada ao nosso cluster. Em ambiente de produção faz sentido ter mais que uma instância de mongos para não saturar uma só máquina com todo o processamento inerente às operações de interação com a base de dados e para adicionar robustez caso uma instância pare de funcionar. Esta é a razão pelo qual temos a máquina3 também a funcionar.

Para que esta funcione bem, devemos aceder o ficheiro de /mongo/s0.conf e efetuar as seguintes alterações:

```
net:
  # ALTERAR O PORTO DE 37040 PARA 37041
  port: 37041
  bindIp: 0.0.0.0
```

Imagem 31. Mudança do port para segundo mongos 'router'

De seguida, no terminal da máquina3 executamos:

- `chmod -R 700 mongo`
- `mongos -f /mongo/s0.conf`

```
root@a8d2a6783edb:/# chmod -R 700 mongo
root@a8d2a6783edb:/# mongos -f /mongo/s0.conf
{"t":{"$date":"2023-05-27T18:16:52.378Z"},"s":"I", "c":"CONTROL", "id":5760901, "ctx":"-", "msg":"Applied --setParameter options", "attr":{"serverParameters":{"enableLocalhostAuthBypass":{"default":true, "value":true}}}}
about to fork child process, waiting until server is ready for connections.
forked process: 34
child process started successfully, parent exiting
root@a8d2a6783edb:/# |
```

Imagem 32. Execução do segundo mongos

Considerações finais

Os acessos à nossa base de dados 'mqttData' deve agora ser feito fazendo uso dos dois serviços disponíveis para interagir com o cluster nos endereços 46.189.143.63:37040 e 46.189.143.63:37041.

Tanto o MongoDB Atlas, como o MongoDB Compass como os drivers utilizados em ambientes de codificação suportam a instanciação de vários IPs com respeito aos routers de um cluster, com flags como 'nearest' para ligar ao mais próximo ou simplesmente para garantir a ligação se uma das instâncias dos mongos ('router') se desligar. Este processo é análogo à ligação direta a um replica set.

The screenshot shows the 'New Connection' window in MongoDB Atlas. The 'Authentication' tab is selected. The 'URI' field contains the connection string: `mongodb://javaop:javaop@46.189.143.63:37040,46.189.143.63:37041/?authMechanism=DEFAULT&authSource=mqttData&readPreference=nearest`. Under 'Advanced Connection Options', the 'Authentication Method' is set to 'Username/Password'. The 'Username' field contains 'javaop'. A yellow warning banner states: 'TLS/SSL is disabled. If possible, enable TLS/SSL to avoid security vulnerabilities.' At the bottom, there are 'Save', 'Save & Connect', and 'Connect' buttons.

New Connection FAVORITE

Connect to a MongoDB deployment

URI ⓘ Edit Connection String

mongodb://javaop:javaop@46.189.143.63:37040,46.189.143.63:37041/?authMechanism=DEFAULT&authSource=mqttData&readPreference=nearest

▼ Advanced Connection Options

General **Authentication** TLS/SSL Proxy/SSH In-Use Encryption Advanced

Authentication Method

None **Username/Password** X.509 Kerberos LDAP AWS IAM

Username

javaop

ⓘ TLS/SSL is disabled. If possible, enable TLS/SSL to avoid security vulnerabilities.

Save Save & Connect Connect

Imagem 33. Exemplo de URI de ligação ao nosso cluster

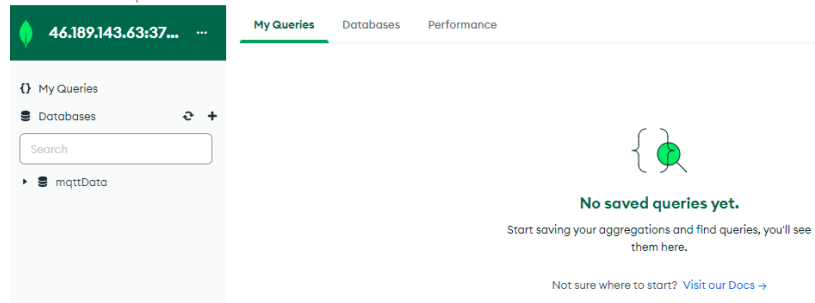


Imagem 34. Sucesso na ligação com o URI

Por fim, existem dois comandos que penso serem de interesse para esta implementação, nomeadamente:

➤ `db.adminCommand({ listShards: 1 })`

Este comando [7] permite visualizar os shards que temos no nosso cluster e algumas das suas propriedades.

```
[direct: mongos] mqtttData> db.adminCommand({ listShards: 1 })
{
  shards: [
    {
      _id: 'a',
      host: 'a/46.189.143.63:37010,46.189.143.63:37011,46.189.143.63:37012',
      state: 1,
      topologyTime: Timestamp({ t: 1685207082, i: 6 })
    },
    {
      _id: 'b',
      host: 'b/46.189.143.63:37020,46.189.143.63:37021,46.189.143.63:37022',
      state: 1,
      topologyTime: Timestamp({ t: 1685207092, i: 3 })
    },
    {
      _id: 'c',
      host: 'c/46.189.143.63:37030,46.189.143.63:37031,46.189.143.63:37032',
      state: 1,
      topologyTime: Timestamp({ t: 1685207099, i: 6 })
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1685215084, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("27b3f995d0d0bb40673238ce04240e0d9b14983d", "hex"), 0),
      keyId: Long("7237898339126083608")
    }
  },
  operationTime: Timestamp({ t: 1685215084, i: 1 })
}
[direct: mongos] mqtttData> |
```

Imagem 35. Sumário das shards em cluster

- use config
- `db.settings.updateOne({_id:"chunksize"},{$set:{_id:"chunksize", value: 1}},{upsert:true})`

Este comando [8] permite alterar o tamanho máximo de cada chunk nos shards para 1 MB (default é 64 MB). No entanto, deve ser utilizado com cuidado e devem ser feitas ponderações sobre o volume de dados em débito de escrita na base de dados. De modo a não comprometer a igualdade entre chunks e a evitar migrações manuais, deve-se usar este comando idealmente depois do [‘Passo 8:’](#).

Bibliografia

- [1] MongoDB, “Deploy a Sharded Cluster,” [Online]. Available:
<https://www.mongodb.com/docs/manual/tutorial/deploy-shard-cluster/>. [Acedido em 27 Maio 23].
- [2] MongoDB, “Configuration File Options,” [Online]. Available:
<https://www.mongodb.com/docs/manual/reference/configuration-options/>. [Acedido em 27 Maio 23].
- [3] MongoDB, “Shard Keys,” [Online]. Available:
<https://www.mongodb.com/docs/manual/core/sharding-shard-key/>. [Acedido em 27 Maio 23].
- [4] MongoDB, “db.createUser(),” [Online]. Available:
<https://www.mongodb.com/docs/manual/reference/method/db.createUser/>. [Acedido em 27 Maio 23].
- [5] MongoDB, “db.collection.getShardDistribution(),” [Online]. Available:
<https://www.mongodb.com/docs/manual/reference/method/db.collection.getShardDistribution/>. [Acedido em 27 Maio 23].
- [6] MongoDB, “db.collection.createIndex(),” [Online]. Available:
<https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/>. [Acedido em 27 Maio 23].
- [7] MongoDB, “List Shards,” [Online]. Available:
<https://www.mongodb.com/docs/manual/reference/command/listShards/#mongodb-dbcommand-dbcmd.listShards>. [Acedido em 27 Maio 23].
- [8] MognoDB, “Modify Range Size in a Sharded Cluster,” [Online]. Available:
<https://www.mongodb.com/docs/manual/tutorial/modify-chunk-size-in-sharded-cluster/>. [Acedido em 27 Maio 23].
- [9] Wikipedia, “MongoDB logo,” [Online]. Available:
https://pt.m.wikipedia.org/wiki/Ficheiro:MongoDB_Logo.svg. [Acedido em 27 Maio 23].