A picture containing black, darkness

Description automatically generated

5/27/2023

Fabian Gobet

97885

ISCTE-IUL

A picture containing graphics, font, graphic design, logo

Description automatically generated

[10]

MongoDB Shard Cluster Setup

PISID 22/23

# Índice

## Índice de conteúdo

[Índice 1](#_Toc136192011)

[Índice de conteúdo 1](#_Toc136192012)

[Índice de imagens 2](#_Toc136192013)

[Disclaimer 4](#_Toc136192014)

[Prefácio 5](#_Toc136192015)

[Infraestrutura do cluster 5](#_Toc136192016)

[Infraestrutura dos containers 7](#_Toc136192017)

[Ficheiros de configuração mongo 10](#_Toc136192018)

[Escolha das Shard Keys 13](#_Toc136192019)

[Implementação 14](#_Toc136192020)

[Passo 1: 14](#_Toc136192021)

[Passo 2: 15](#_Toc136192022)

[Passo 3: 16](#_Toc136192023)

[Passo 4: 17](#_Toc136192024)

[Passo 4.1: 17](#_Toc136192025)

[Passo 4.2: 20](#_Toc136192026)

[Passo 4.3: 21](#_Toc136192027)

[Passo 4.4: 22](#_Toc136192028)

[Passo 5: 23](#_Toc136192029)

[Passo 6: 25](#_Toc136192030)

[Passo 7: 27](#_Toc136192031)

[Passo 8: 28](#_Toc136192032)

[Passo 9 (extra): 30](#_Toc136192033)

[Considerações finais 31](#_Toc136192034)

[Implementação em Docker + AWS EC2 34](#_Toc136192035)

[Bibliografia 38](#_Toc136192036)

## Índice de imagens

[Imagem 1. Infraestrutura do cluster 5](#_Toc136191969)

[Imagem 2. Representação dos replica sets e nome 6](#_Toc136191970)

[Imagem 3. IPS e portas de cada serviço mongodb - 1 8](#_Toc136191971)

[Imagem 4. IPS e portas de cada serviço mongodb - 2 8](#_Toc136191972)

[Imagem 5. Estrutura do diretorio 'mongo' 9](https://iscteiul365-my.sharepoint.com/personal/ffgts_iscte-iul_pt/Documents/PISID%20mongo%20cluster.docx#_Toc136191973)

[Imagem 6. run0.sh script 9](#_Toc136191974)

[Imagem 7. Ficheiro de configuração configsrv 10](#_Toc136191975)

[Imagem 8. Ficheiro de configuração shardsvr 11](#_Toc136191976)

[Imagem 9. Ficheiro de configuração de mongos 'router' 12](#_Toc136191977)

[Imagem 10. Montagem dos containers 14](#_Toc136191978)

[Imagem 11. Containers no Docker Desktop 15](#_Toc136191979)

[Imagem 12. Acesso aos containers 15](#_Toc136191980)

[Imagem 13. Execução dos scripts para lançar todas as instâncias de mongo 16](#_Toc136191981)

[Imagem 14. Ligação ao cfg0 17](#_Toc136191982)

[Imagem 15. Iniciação do replica set ‘cfg’ 17](#_Toc136191983)

[Imagem 16. cfg rs.status() output (1/3) 18](#_Toc136191984)

[Imagem 17. cfg rs.status() output (2/3) 19](#_Toc136191985)

[Imagem 18. cfg rs.status() output (3/3) 19](#_Toc136191986)

[Imagem 19. Ligação ao a0 20](#_Toc136191987)

[Imagem 20. Iniciação do replica set 'a' 21](#_Toc136191988)

[Imagem 21. Iniciação do replica set 'b' 22](#_Toc136191989)

[Imagem 22. Iniciação do replica set 'c' 23](#_Toc136191990)

[Imagem 23. Ligação ao mongos na maquina 4 23](#_Toc136191991)

[Imagem 24. Criação dos user root e admin 24](#_Toc136191992)

[Imagem 25. Adição das 3 replicas 'a','a' e 'c' como shards 25](#_Toc136191993)

[Imagem 26. sh.status output (1/2) 26](#_Toc136191994)

[Imagem 27. sh.status output (2/2) 27](#_Toc136191995)

[Imagem 28. Criação da db, collections e user javaop 27](#_Toc136191996)

[Imagem 29. Enable e sharding das coleções (1/2) 28](#_Toc136191997)

[Imagem 30. Enable e sharding das coleções (2/2) 29](#_Toc136191998)

[Imagem 31. Mudança do port para segundo mongos 'router' 30](#_Toc136191999)

[Imagem 32. Execução do segundo mongos 30](#_Toc136192000)

[Imagem 33. Exemplo de URI de ligação ao nosso cluster 31](#_Toc136192001)

[Imagem 34. Sucesso na ligação com o URI 32](#_Toc136192002)

[Imagem 35. Sumário das shards em cluster 32](#_Toc136192003)

[Imagem 36. Instância EC2 34](#_Toc136192004)

[Imagem 37. Propriedades da instância EC2 34](#_Toc136192005)

[Imagem 38. Novos IPs e portos para esta solução 35](#_Toc136192006)

[Imagem 39. Regras inbound na firewall 35](#_Toc136192007)

[Imagem 40. Docker Containers 36](#_Toc136192008)

[Imagem 41. mongos da machine4-AWS 36](#_Toc136192009)

[Imagem 42. Instancia EC2 37](#_Toc136192010)

# 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.

Imagens, scripts, keyfile e ficheiros de configuração podem ser encontrados em <https://github.com/FabianGobet/PISID_Mongo> [1], no diretório ‘Parte 2’.

# 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* [2] 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 publico. Como todos os containers irão estar a correr no mesmo dispositivo, o seu IP publico 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:

A picture containing text, screenshot, rectangle, diagram

Description automatically generated

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*

A diagram of different colored rectangles

Description automatically generated with low confidence

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 maquina 0 executando o comando
* ‘docker build –build-arg INDX=0 machine 0 .” no mesmo diretorio que o Dockerfile

Em alternativa podemos utilizar diretamente as imagens que estão no link docker hub previamente discriminado e construir os containers.A titulo de exemplo para a maquina 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.

*A screenshot of a computer

Description automatically generated with low confidence*

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

*A picture containing text, font, screenshot, number

Description automatically generated*

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):

A screen shot of a computer screen

Description automatically generated with low confidence

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’.

Imagem 5. Estrutura do diretorio 'mongo'

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.

*A screen shot of a computer code

Description automatically generated with low confidence*

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:

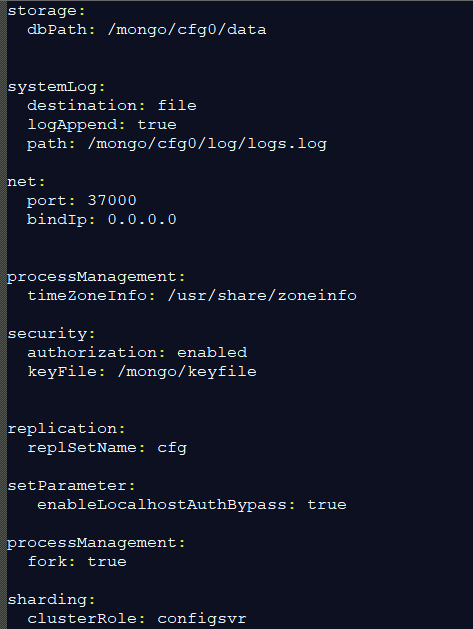


Imagem 7. Ficheiro de configuração configsrv

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:

A screen shot of a computer

Description automatically generated with medium confidence

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:

*A screenshot of a computer

Description automatically generated with medium confidence*

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 [3]. 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 [4] 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 passiveis 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 monoticidade 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

A screenshot of a computer program

Description automatically generated with medium confidence

Imagem 10. Montagem dos containers

A screenshot of a computer

Description automatically generated with low confidence

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

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer screen

Description automatically generated with medium confidence

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

A screen shot of a computer

Description automatically generated with low confidence

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 }

]

}

)

A screen shot of a computer

Description automatically generated with low confidence

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.

A picture containing text, screenshot, font

Description automatically generated

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

A screenshot of a computer program

Description automatically generated with medium confidence

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

A picture containing text, screenshot, font

Description automatically generated

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 replica 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

A picture containing text, screenshot, font

Description automatically generated

Imagem 19. Ligação ao a0

De forma análoga à replica 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

A picture containing text, screenshot, font

Description automatically generated

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 screenshot of a computer program

Description automatically generated with medium confidence

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 }

]

}

)

A screenshot of a computer program

Description automatically generated with medium confidence

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

A screenshot of a computer program

Description automatically generated with medium confidence

Imagem 23. Ligação ao mongos na maquina 4

Vamos criar dois utilizadores [5] 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"]})

A screenshot of a computer program

Description automatically generated with medium confidence

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")

A screenshot of a computer program

Description automatically generated with medium confidence

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()

A screenshot of a computer program

Description automatically generated with medium confidence

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

A screen shot of a computer program

Description automatically generated with low confidence

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"]})

A screen shot of a computer

Description automatically generated with medium confidence

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.mazemanage14”,{“numExp:”hashed”})

A picture containing text, screenshot, font

Description automatically generated

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

A picture containing text, screenshot, software, font

Description automatically generated

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 [6]:

* db.mazemanage14.getShardDistribution()

Nota:

Caso existam dados na coleção é necessário primeiro indexar a coleção [7] e só aplicar o sharding, executando:

* db.mazemanage14.createInex({"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 maquina3 também a funcionar.

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

A picture containing text, screenshot, font

Description automatically generated

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

De seguida, no terminal da maquina3 executamos:

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

A screen shot of a computer program

Description automatically generated with low confidence

Imagem 32. Execução do segundo mongos

## Considerações finais

Os acessos à nossa base de dados ‘mqttData’ deve agora ser feito fazendo uso dos doi 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.

A screenshot of a computer

Description automatically generated with medium confidence

Imagem 33. Exemplo de URI de ligação ao nosso cluster

A screenshot of a computer

Description automatically generated with medium confidence

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

A picture containing text, screenshot, font

Description automatically generated

Imagem 35. Sumário das shards em cluster

* use config
* db.settings.updateOne({\_id:”chunksize”},{$set:{\_id:”chuncksize”, value: 1}},{upsert:true})

Este comando [9] 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:’.

# Implementação em Docker + AWS EC2

A implementação anterior, ainda que use IPs públicos para todos os servidores para acionar o routing externo e simular dispositivos independentes, pode pender de algum ceticismo relativamente ao requisito de correr os servidores em duas máquinas diferentes.

Para colmatar a questão referida, foi criada uma conta grátis na Amazon Web Services (AWS) e gerada uma instância de Elastic Compute Cloud (EC2) com uma imagem de Ubuntu 22.04 LTS, correspondente à machine0.

A screenshot of a computer

Description automatically generated with medium confidence

Imagem 36. Instância EC2

A screenshot of a computer

Description automatically generated

Imagem 37. Propriedades da instância EC2

Foram mudadas as portas e os IPs nos ficheiros de configuração de todos os ficheiros .conf de maneira a atender à seguinte tabela.

A screenshot of a computer program

Description automatically generated with low confidence

Imagem 38. Novos IPs e portos para esta solução

Foi também adicionado uma regra para permitir tráfego inboud nas nossas portas.

A screenshot of a computer

Description automatically generated with medium confidence

Imagem 39. Regras inbound na firewall

E de seguida foram seguidos exatamente os mesmos passos que em Implementação, com exceção de lançar a machine0 em Docker e de trocar os IPs e port em comandos para corresponder à tabela da imagem.

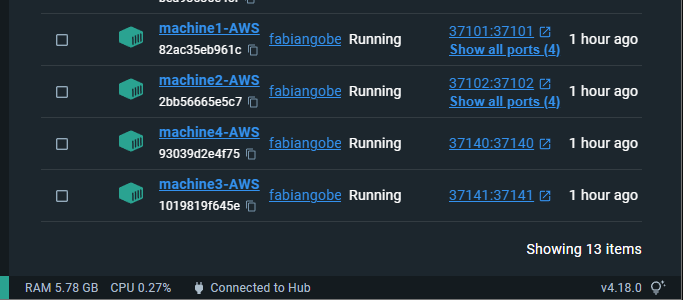


Imagem 40. Docker Containers

A screenshot of a computer program

Description automatically generated with medium confidence

Imagem 41. mongos da machine4-AWS

A screenshot of a computer program

Description automatically generated with medium confidence

Imagem 42. Instancia EC2

# Bibliografia

|  |  |
| --- | --- |
| [1] | F. Gobet, “PISID\_Mongo,” [Online]. Available: https://github.com/FabianGobet/PISID\_Mongo. [Acedido em 28 Maio 23]. |
| [2] | MongoDB, “Deploy a Sharded Cluster,” [Online]. Available: https://www.mongodb.com/docs/manual/tutorial/deploy-shard-cluster/. [Acedido em 27 Maio 23]. |
| [3] | MongoDB, “Configuration File Options,” [Online]. Available: https://www.mongodb.com/docs/manual/reference/configuration-options/. [Acedido em 27 Maio 23]. |
| [4] | MongoDB, “Shard Keys,” [Online]. Available: https://www.mongodb.com/docs/manual/core/sharding-shard-key/. [Acedido em 27 Maio 23]. |
| [5] | MongoDB, “db.createUser(),” [Online]. Available: https://www.mongodb.com/docs/manual/reference/method/db.createUser/. [Acedido em 27 Maio 23]. |
| [6] | MongoDB, “db.collection.getShardDistribution(),” [Online]. Available: https://www.mongodb.com/docs/manual/reference/method/db.collection.getShardDistribution/. [Acedido em 27 Maio 23]. |
| [7] | MongoDB, “db.collection.createIndex(),” [Online]. Available: https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/. [Acedido em 27 Maio 23]. |
| [8] | MongoDB, “List Shards,” [Online]. Available: https://www.mongodb.com/docs/manual/reference/command/listShards/#mongodb-dbcommand-dbcmd.listShards. [Acedido em 27 Maio 23]. |
| [9] | 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]. |
| [10] | Wikipedia, “MongoDB logo,” [Online]. Available: https://pt.m.wikipedia.org/wiki/Ficheiro:MongoDB\_Logo.svg. [Acedido em 27 Maio 23]. |