**Dojot Com cluster Mongodb - LAB**

**Aluno:** Marcus Artiaga Colantoni

**Orientador:** Raphael Pasquini

**Mestrado UFU – Universidade Federal de Uberlândia**

Montando um LAB com Dojot em modo Cluster, a ideia é utilizar a rede swarn do Docker, para disponibilizar um cluster com redundância escalável da plataforma de Dojot, de forma que a qualquer momento possam ser adicionados novos NÓS ao cluster, trazendo alta disponibilidade e redundância em caso de falhas. Também vou integrar um cluster do Mongodb de forma separada, a fim de que os dados possam ser persistidos mesmo em caso de falhas no servidor principal.

1. Primeiramente configurei uma VM com um sistema operacional mais compatível (em meus testes) com o Dojot, que é:

**Ubuntu 16.0.4.6 Live Server amd 64.**

1. Logo após instalar o sistema operacional vou realizar a atualização do repositório:

**sudo nano /etc/apt/sources.list** (comente todos os pacotes com DEB CDROM)

**sudo apt-get update**

**sudo apt-get upgrade**

1. Agora vou realizar a instalação do docker para a orquestração dos containers:

**sudo apt update**

**sudo apt upgrade**

**sudo apt-get install curl apt-transport-https ca-certificates software-properties-common**

**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**

**sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"**

**sudo apt update**

**apt-cache policy docker-ce**

**sudo apt install docker-ce**

**sudo systemctl status docker**

1. Agora vou instalar o docker-compose

**sudo su**

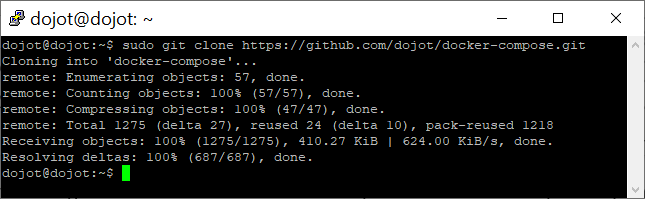
**curl -L https://github.com/docker/compose/releases/download/1.25.5/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose**

**chmod +x /usr/local/bin/docker-compose**

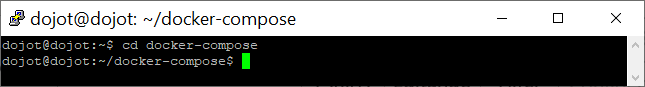
**docker-compose -v**

1. Após a instalação do docker segui com os seguintes comandos para a copiar o projeto do Dojot Linux do git :

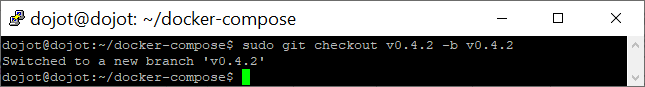
**1 - sudo git clone** [**https://github.com/dojot/docker-compose.git**](https://github.com/dojot/docker-compose.git)



**2 - cd docker-compose**

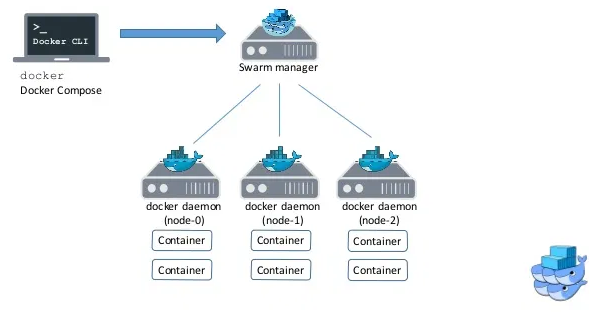


**3 - sudo git checkout v0.4.2 -b v0.4.2**



**4 - sudo docker-compose up -d (não rodei este comando ainda, pois não quero subir o dojot na forma padrão, a ideia é subi-lo na forma de serviços na rede swarn, utilizando o driver OVERLAY do Docker para criar um cluster ).**

1. Rede overlay

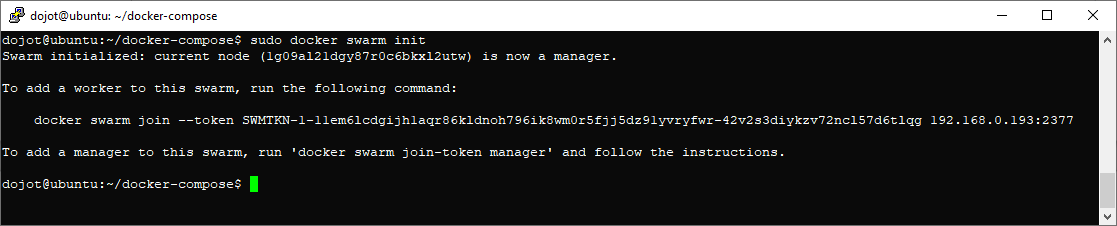


1. A rede overlay é uma rede que se sobrepõem a rede física, criando um cluster distribuído, para iniciar a rede overlay primeiramente precisamos iniciar o **swarm**, para iniciar o **swarm** utilizei o comando:

**sudo docker swarm init**

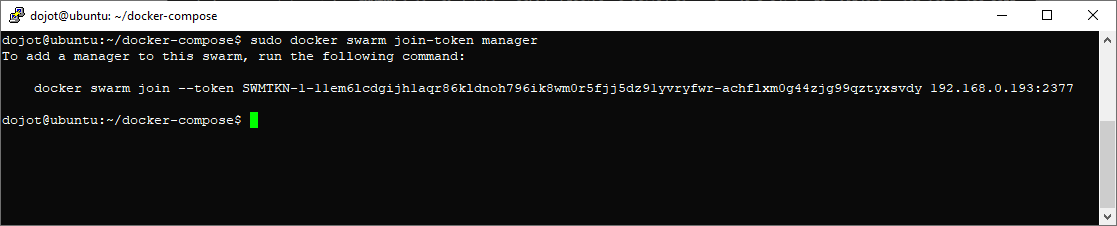
1. Em seguida é mostrado o token para unir/juntar novos nós (na forma de Works) ao swarm criado:

docker swarm join --token SWMTKN-1-11em6lcdgijh1aqr86kldnoh796ik8wm0r5fjj5dz91yvryfwr-42v2s3diykzv72ncl57d6tlqg 192.168.0.193:2377

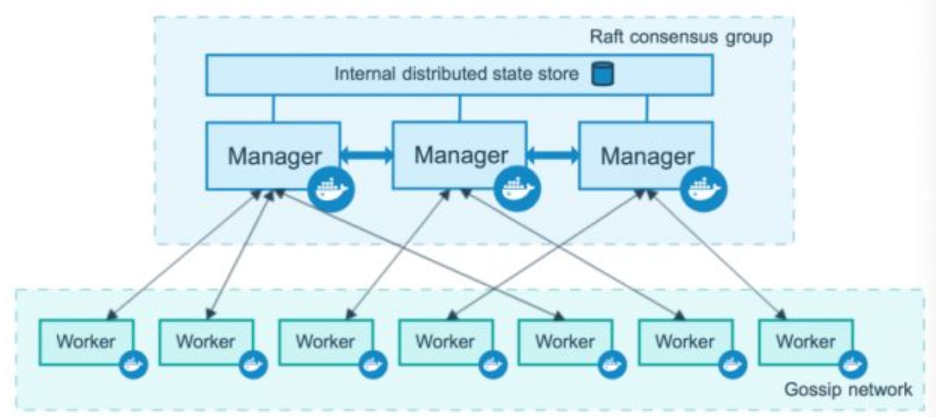


1. Para Adicionar um nó a rede swarm na forma de Leader/Manager é preciso obter um token próprio para isso:

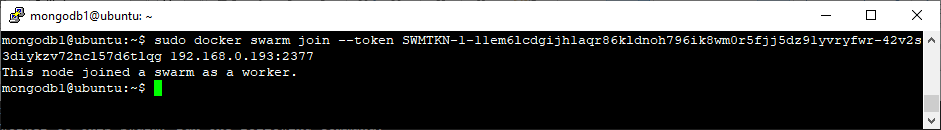
**sudo docker swarm join-token manager**



**docker swarm join --token SWMTKN-1-11em6lcdgijh1aqr86kldnoh796ik8wm0r5fjj5dz91yvryfwr-achflxm0g44zjg99qztyxsvdy 192.168.0.193:2377**

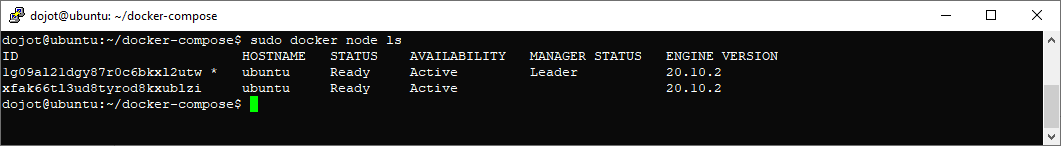


1. Agora vou adicionar outro nó ao cluster esse será adicionado como WORKDER , para isso criei uma VM chamada **Mongodb1**, com o mesmo sistema operacional Ubuntu 16.0.4.6 live server amd64, e realizei a instalação do docker, para adicionar a **VM-MONGODB1** como nó worker na rede utilizei o comando:



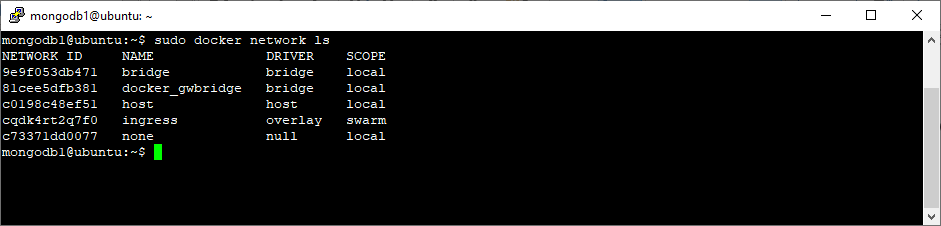
1. Agora ao consultar na VM **Dojot\_server** que é o Leader do cluster na rede swarm, é possível verificar que existem dois nós na rede:

**sudo docker node ls**

**OBS:** Um nó aparece como **LEADER** e os outros onde não há nada escrito, significa que ele é listado como worker.

1. Agora ao consultar as redes docker existentes na **VM MONGODB1**, é possível observar que já existe o driver **overlay** em uma rede chamada de **INGRESS**:

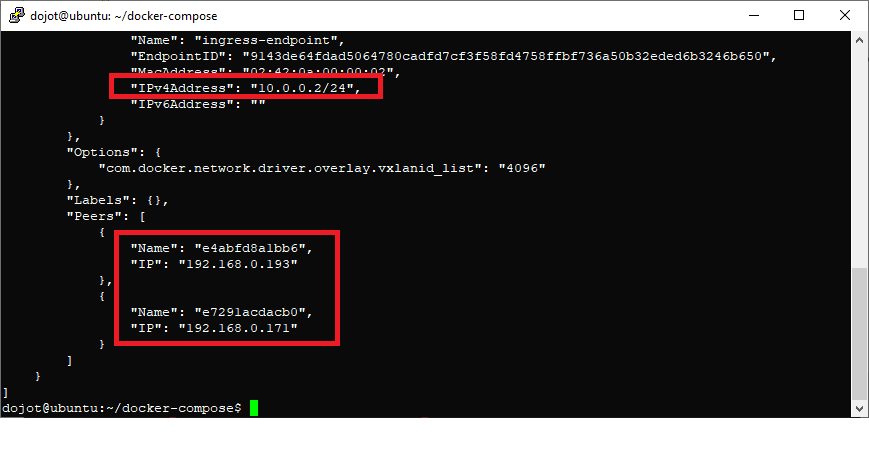
**sudo docker network ls**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CLUSTER DOJOT** | | | | | | |
| **VM** | **S.O** | **CPU** | **RAM** | **HD** | **IP** | **Status** |
| Dojot\_server | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 2448 MB | 20GB | 192.168.0.125 | Leader |
| Mongodb1 | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 1024 MB | 10GB | 192.168.0.171 | worker |

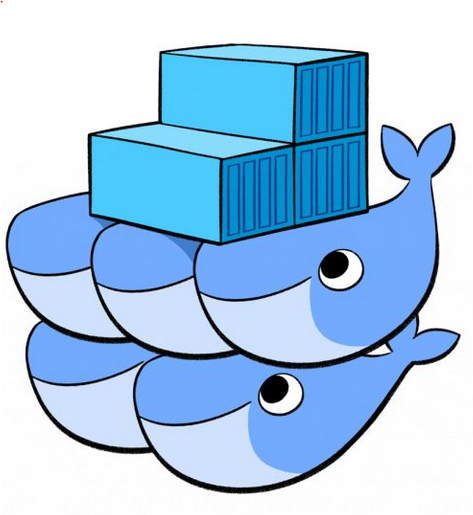
1. Agora podemos consultar qual o IP da rede swarm, inspecionando a rede ingress que que foi criada com o driver OVERLAY:

**sudo docker inspect ingress**



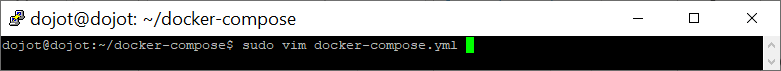
**OBS:** Podemos observar o IP de classe A 10.0.0.2/24 que é utilizado na rede swarm, com o driver overlay, e abaixo os endereços de IP dos hosts que compõem os nós da rede, sendo IP de cada uma das máquinas virtuais.

1. Agora vou subir o Dojot na forma de um serviço da rede swarm, dessa forma os container necessários a execução do Dojot serão escaláveis entre os nós que estiverem compondo a rede swarm, sendo possível aumentar a infraestrutura a qualquer momento, e ainda setar o número de réplicas que serão utilizadas



1. Primeiro preciso editar o arquivo do Docker-compose.yml que é o script de configuração do dojot, nele estão descritos todos os container necessários para subir o Dojot, porem aqui encontrei um problema de versão, o arquivo yml do Dojot foi escrito na versão 2.1 do Docker-compose, e nesta versão não é possível realizar o stack dos containers na forma de um serviço, para isso é preciso atualizar o script para a versão 3.8, para isso realizei alguns ajustes no arquivo:

**sudo vim docker-compose.yml (**para editar o arquivo de configuração**)**



1. Para facilitar o entendimento vou numerar as linhas do arquivo de configuração yml, para isso basta seguir os passo (editando o arquivo):

**ESC**

**:set number**

1. Para voltar a modo de inserção pressione a letra

i

1. Agora vou indicar as modificações de acordo com a linha (OBS: a identação no arquivo YML é critica, portando não é recomendado modificar a identação do arquivo), basicamente o comando (**depends\_on**) não é mais reconhecido na versão 3.4 em diante do docker-compose, logo comentei todas as vezes que ele aparece no arquivo de configuração YML, também realizei a modificação da primeira linha com a versão 3.8, e da última linha com o **NOME** que não é mais suportado nesta versão, também adicionei mais dois container com imagem do Mongodb a criação, de forma que um é utilizado como no do cluster do mongodb e o outro é o arbitro do cluster :

**1 version: '3.7'**

Modificações no container do mongodb padrão para utilizar uma imagem mais atual do mongodb que tenha suporte a clusterização, também adicionei uma rede overlay chamada **COLANTONI-OVERLAY**, e utilizei um comando para habilitar a opção do **REPLICASET** necessária ao cluster do mongodb:

**mongodb:**

**#image: dojot/mongo:3.2**

**image: mongo**

**restart: always**

**#restart: on-failure**

**user: "mongodb"**

**logging:**

**driver: json-file**

**options:**

**max-size: 20m**

**max-file: '5'**

**volumes:**

**- mongodb-volume:/data/db**

**- mongodb-cfg-volume:/data/configdb**

**environment:**

**MONGO\_REPLICA\_SET\_NAME: colantoni**

**command: ["--replSet", "colantoni"]**

**networks:**

**- colantoni-overlay**

**- flowbroker**

**- default**

Container adicionados no arquivo docker-compose.yml para criação de um no do cluster mongodb e o arbitro:

**mongodb2:**

**image: mongo**

**restart: always**

**#restart: on-failure**

**user: "mongodb"**

**logging:**

**driver: json-file**

**options:**

**max-size: 20m**

**max-file: '5'**

**volumes:**

**- mongodb-volume2:/data/db**

**- mongodb-cfg-volume2:/data/configdb**

**environment:**

**MONGO\_REPLICA\_SET\_NAME: colantoni**

**command: ["--replSet", "colantoni"]**

**networks:**

**- default**

**- flowbroker**

**- colantoni-overlay**

**mongodb\_arbitro:**

**image: mongo**

**restart: always**

**#restart: on-failure**

**user: "mongodb"**

**logging:**

**driver: json-file**

**options:**

**max-size: 20m**

**max-file: '5'**

**volumes:**

**- mongodb-volume3:/data/db**

**- mongodb-cfg-volume3:/data/configdb**

**environment:**

**MONGO\_REPLICA\_SET\_NAME: colantoni**

**command: ["--replSet", "colantoni"]**

**networks:**

**- default**

**- flowbroker**

**- colantoni-overlay**

**ports:**

**- "27018:27018/tcp"**

**A**dicionei na lista de volumes , os novos volumes que criei para os container **MONGODB2** e **MONGODB\_ABRITRO**:

**volumes:**

**ejbca-volume:**

**postgres-volume:**

**mongodb-volume:**

**mongodb-cfg-volume:**

**mongodb-volume2:**

**mongodb-cfg-volume2:**

**mongodb-volume3:**

**mongodb-cfg-volume3:**

**minio-volume:**

**rabbitmq-volume:**

**zookeeper-volume:**

**zookeeper-log-volume:**

**zookeeper-secrets-volume:**

**kafka-volume:**

**kafka-secrets-volume:**

**iotagent-mqtt-volume:**

**iotagent-mqtt-log-volume:**

**iotagent-mqtt-etc-volume:**

**auth-redis-volume:**

**flowbroker-volume:**

**flowbroker-redis-volume:**

**data-broker-redis-volume:**

**device-manager-redis-volume:**

**kafka-ws-redis-volume:**

**mosca-redis-volume:**

**influxdb-volume:**

Realizei algumas modificações no container do agente MQTT:

**iotagent-mqtt:**

**image: dojot/iotagent-mosca:v0.4.2**

**#depends\_on:**

**# - mosca-redis**

**# - kafka**

**# - data-broker**

**# - auth**

**# - ejbca**

**ports:**

**- 1883:1883**

**- 8883:8883**

**restart: always**

**environment:**

**DOJOT\_MANAGEMENT\_USER: 'iotagent-mqtt'**

**KAFKA\_GROUP\_ID: 'iotagent-mqtt-group'**

**ALLOW\_UNSECURED\_MODE: 'true'**

**MOSCA\_TLS\_DNS\_LIST: '192.168.100.125'**

**logging:**

**driver: json-file**

**options:**

**max-size: 100m**

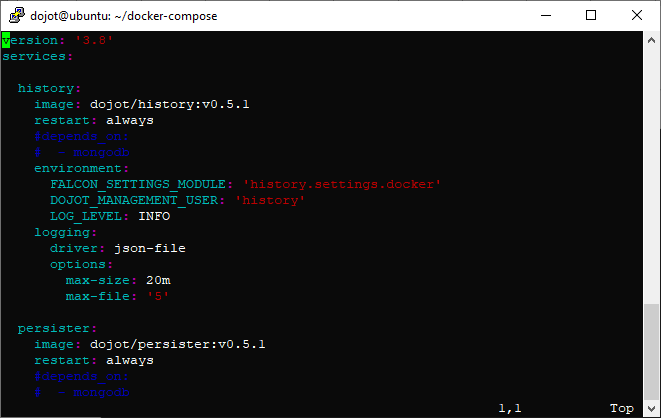
Adicionei a rede **colantoni-overlay** as redes do arquivo docker-compose.yml e especifiquei que o driver a ser utilizado é o overlay:

networks:

**flowbroker:**

**colantoni-overlay:**

**driver: overlay**

****

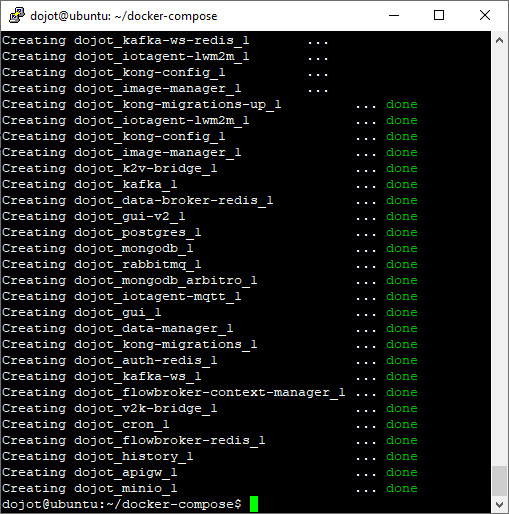
1. Para sair do arquivo e salvar:

**ESC**

**:wq!**

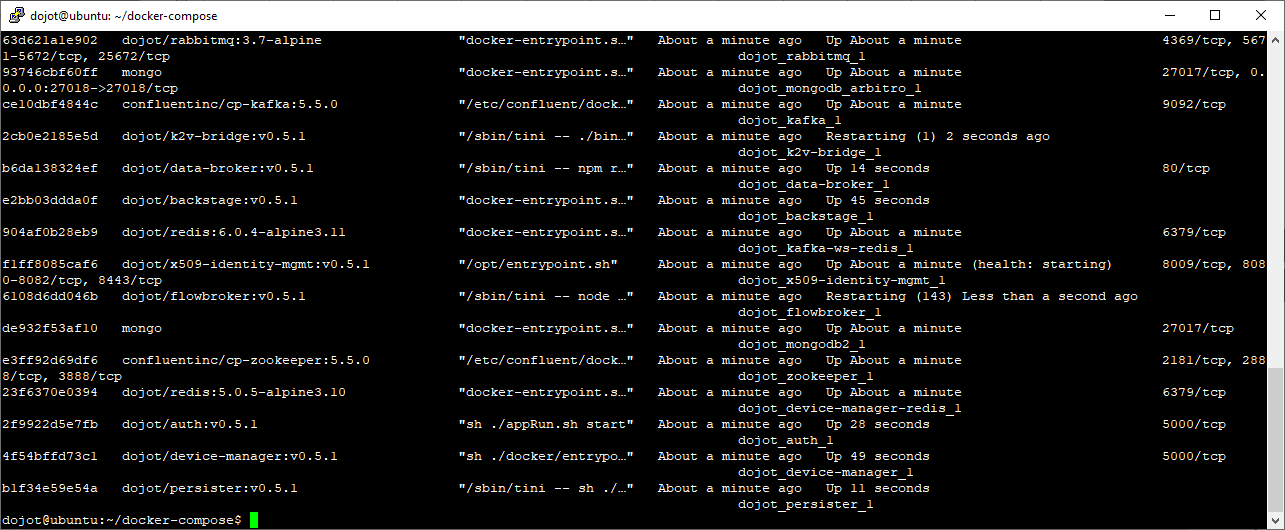
1. Agora vou rodar o arquivo de configuração YML do docker-compose com o comando:

**sudo docker docker-compose up -d**



1. Após a conclusão pode consultar os containers em execução com o comando:

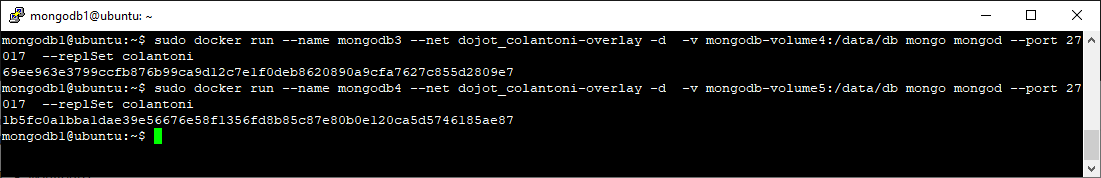
**sudo docker ps**



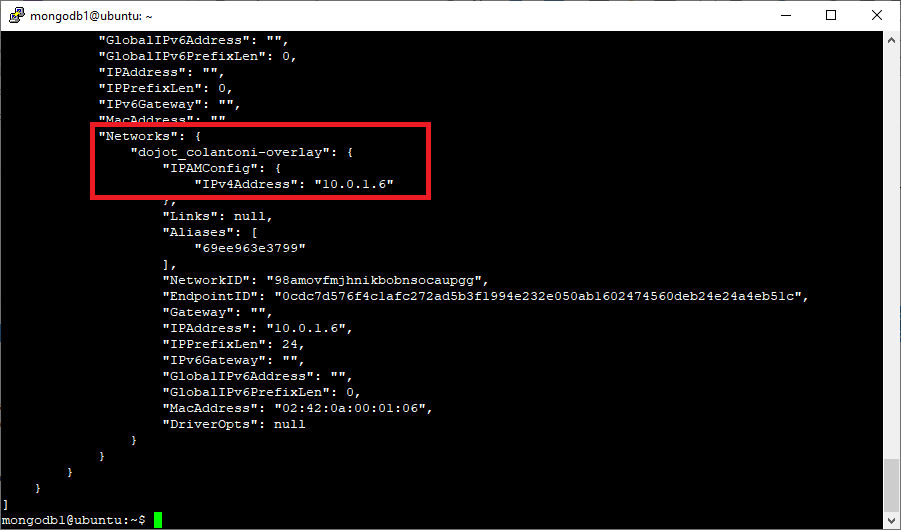
1. Agora vou adicionar mais dois containers ao cluster do Mongodb, porem agora vou subi-los de outro no, que está na VM MONGODB1:

**sudo docker run --name mongodb3 --net dojot\_colantoni-overlay -d -v mongodb-volume4:/data/db mongo mongod --port 27017 --replSet colantoni**

**sudo docker run --name mongodb4 --net dojot\_colantoni-overlay -d -v mongodb-volume5:/data/db mongo mongod --port 27017 --replSet colantoni**

1. Ao consultar as configurações do container é possível verificar que ele possui a rede dojot\_colantoni-overlay:

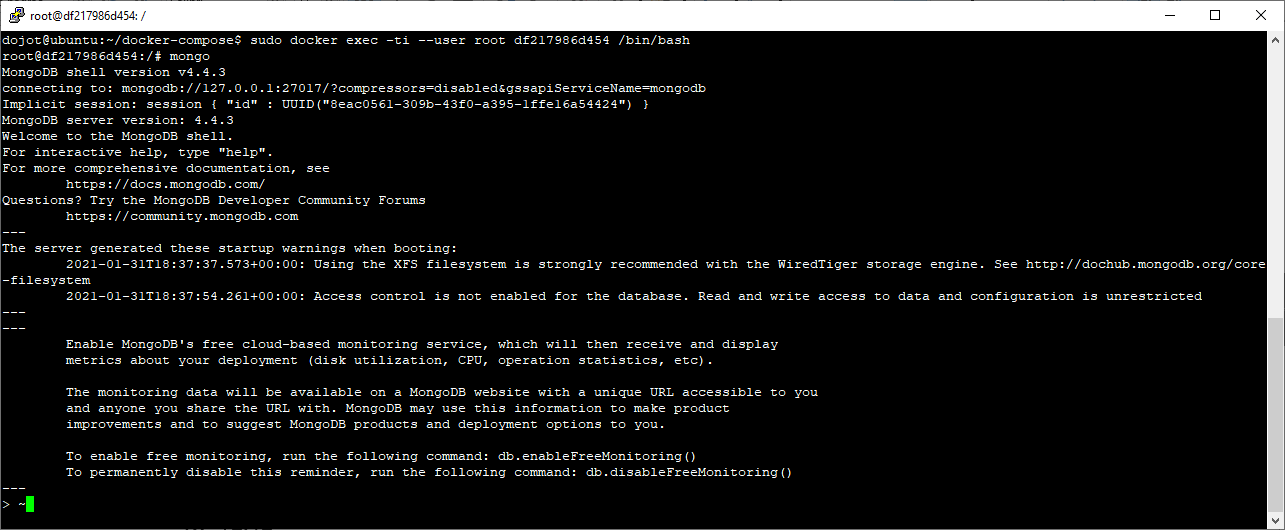
**sudo docker inspect 69ee963e3799**



1. Agora vou acessar na VM DOJOT\_SERVER o container primario do cluster mongo db:

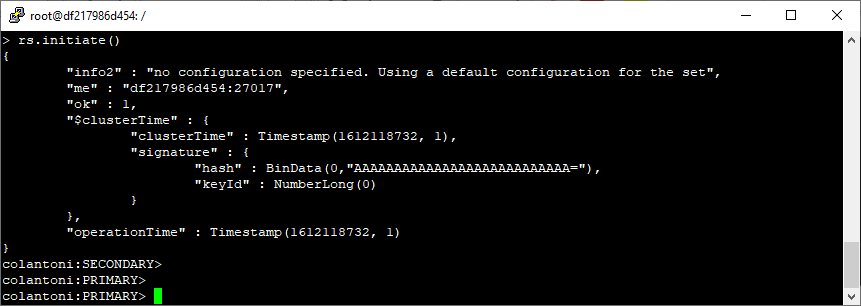
**sudo docker exec -ti --user root df217986d454 /bin/bash (id\_container)**

**mongo**

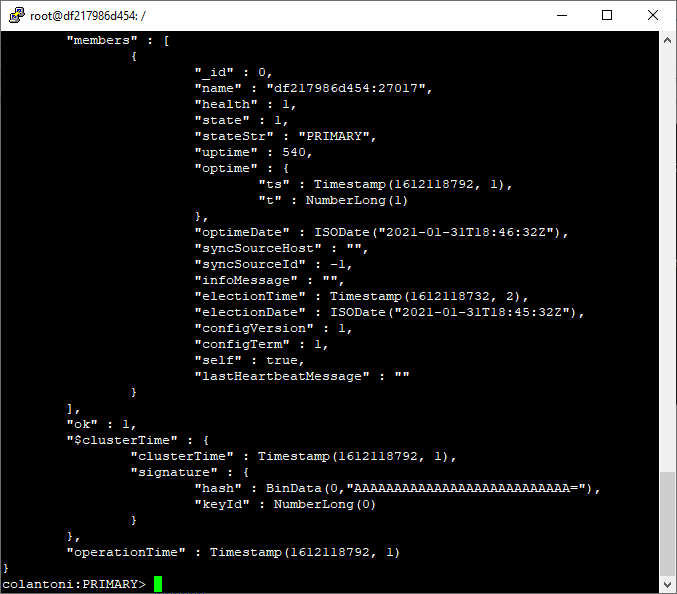


1. Agora vou iniciar o cluster :

**rs.initiate()**

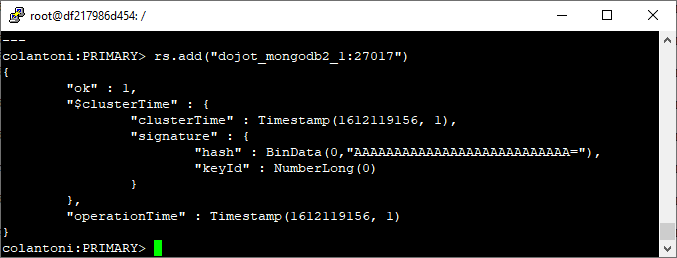


1. Cluster iniciado, agora consultando o status do cluster é possível notar que existe apenas um no cluster, de Id 0 , que é o próprio container:

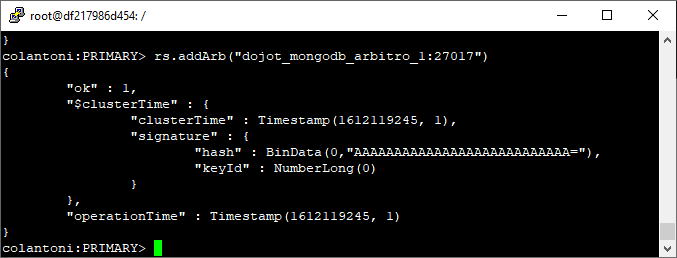


1. Agora vou adicionar os outros containers criados para o cluster, aqui é importante ressaltar que deve ser utilizado o **DNS do docker**, utilizar o IP na hora de inserir um membro no cluster não é viável, pois caso o container reinicie muito provavelmente o endereço IP sera modificado perdendo a comunicação com o cluster.

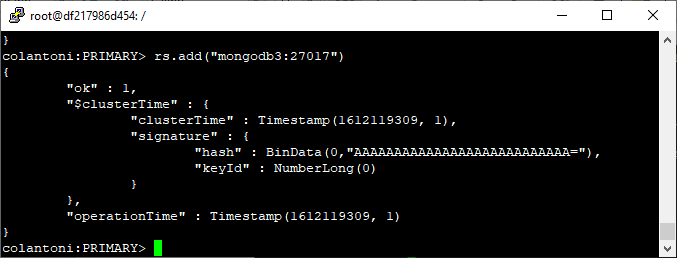
**rs.add("dojot\_mongodb2\_1:27017")**



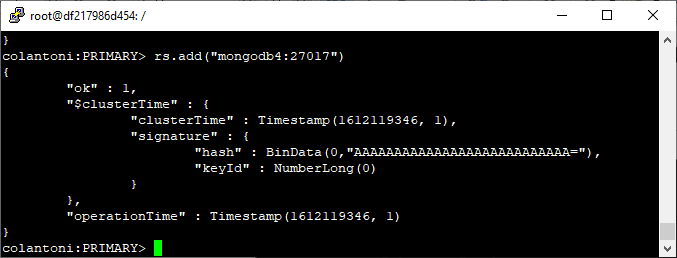
**rs.addArb("dojot\_mongodb\_arbitro\_1:27017")**

****

**rs.add("mongodb3:27017")**

****

**rs.add("mongodb4:27017")**

****

1. Agora vou definir as prioridades para que o **arbitro** escolher um novo **PRIMARY/MASTER** caso seja necessário, no container DOJOT\_MONGODB\_1:

**cfg = rs.conf()**

**cfg.members[0].priority = 5 (Container principal que quero como master)**

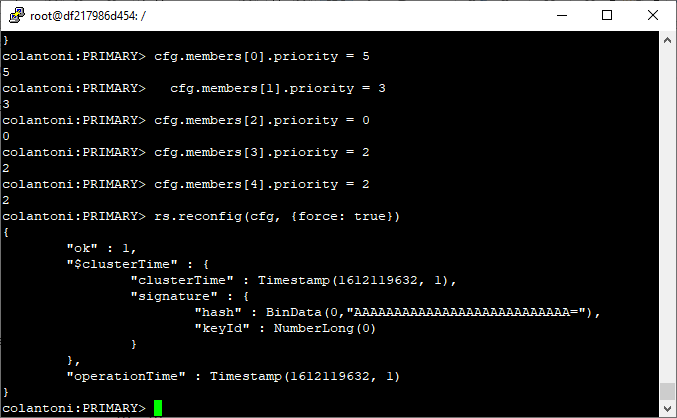
**cfg.members[1].priority = 3 (secundário qualquer mesmo no)**

**cfg.members[2].priority = 0 (arbitro)**

**cfg.members[3].priority = 2 (secundario qualquer em outro no da rede)**

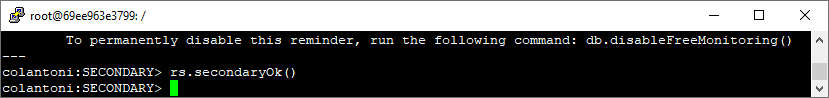
**cfg.members[4].priority = 2 (secundario qualquer em outro no da rede)**

**rs.reconfig(cfg, {force: true})**



1. Agora vou habilitar a escrita e leitura em todos os nós que compõem o cluster do mongo db, para isso, vou acessar cada container, e no **shell** do mongodb utilizar o comando:

**rs.secondaryOk()**

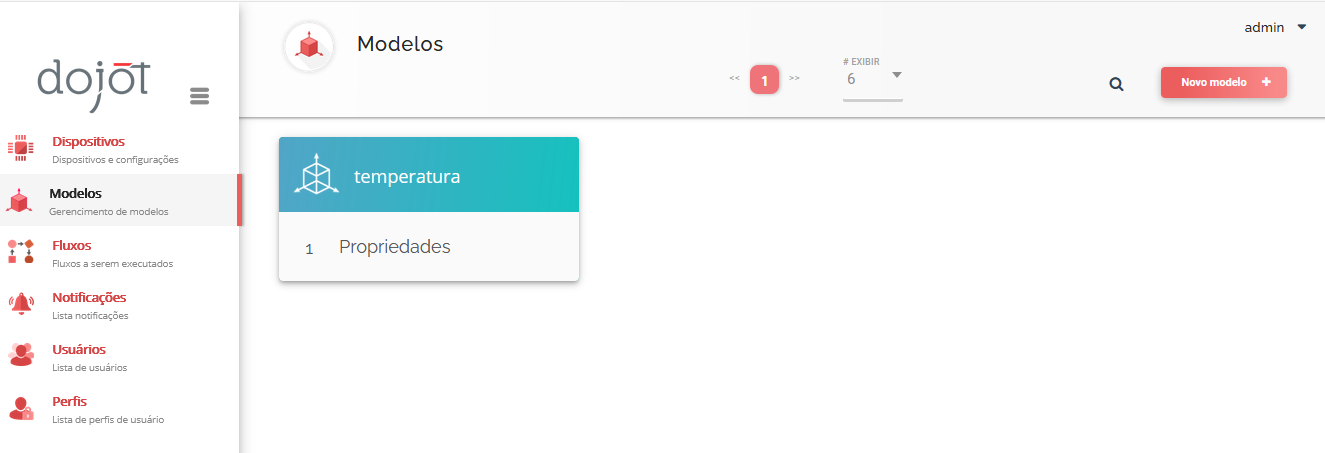
****

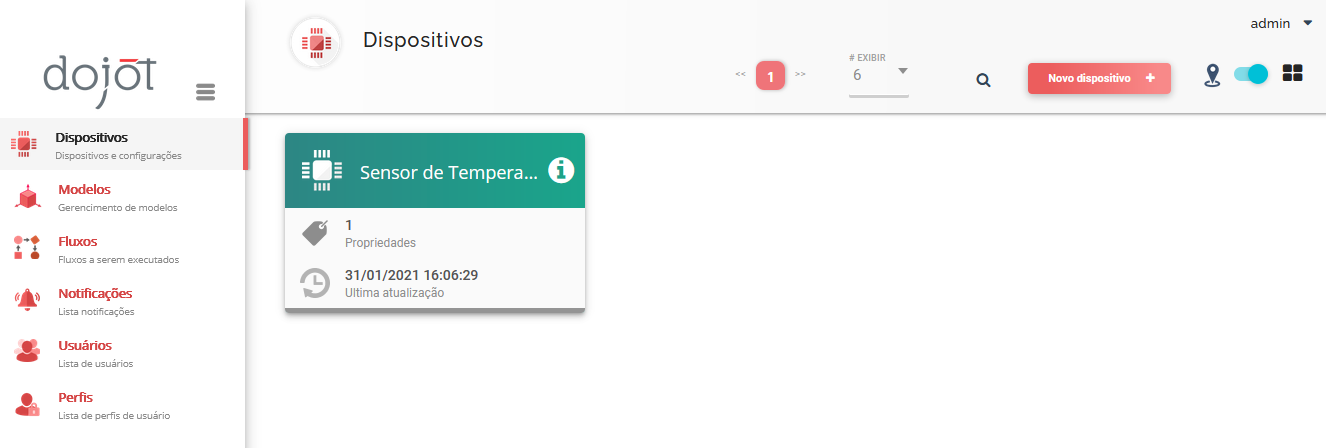
1. Agora vou testar a replicação dos dados que estão sendo salvos no mongodb, para isso vou acessar o dojot pela interface web:

URL: <http://192.168.0.193:8000/#/>



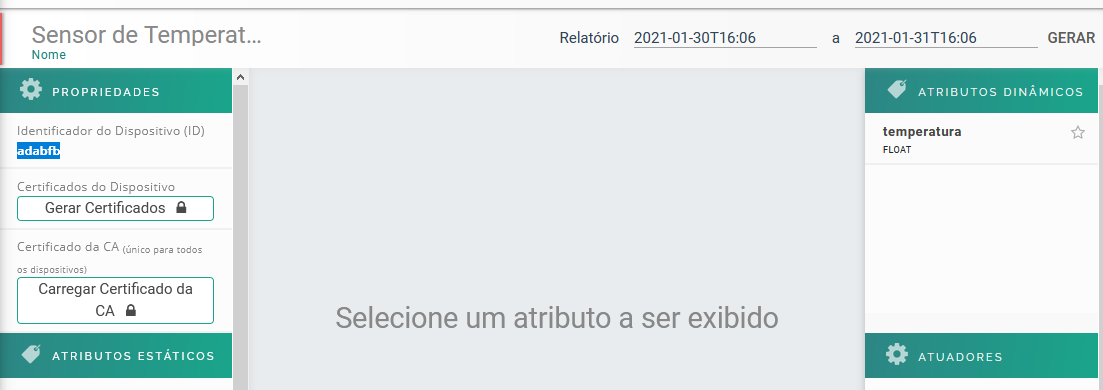
1. Vou criar um modelo chamado **TEMPERATURA** e um dispositivo chamado , “**SENSOR DE TEMPERATURA”** :





1. Agora vou consultar se o novo dispositivo está salvo no mongodb e nos demais contêineres que compõem o cluster:

ID = **adabfb**

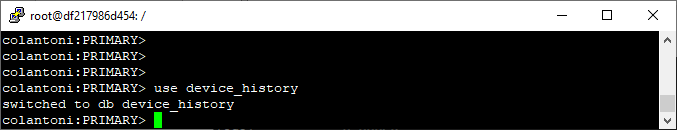


1. No container master do mongodb, vou acessar o shell do mongodb:

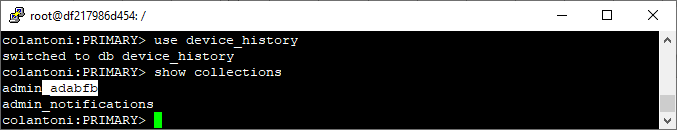
**show dbs**



**use device\_history**



**show collections**



1. Aqui já podemos ver que o dispositivo criado pelo usuário “**admin**” com ID **adabfb.** Agora vou consultar os demais nós do cluster e verificar se esse mesmo ID de dispositivos pode ser consultado:
2. Mongodb3

**mongo**

**use device\_history**

**show collections**



1. Mongodb4

**mongo**

**use device\_history**

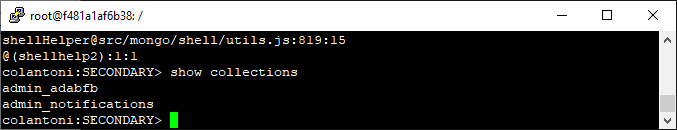
**show collections**

1. Mongodb2

**mongo**

**use device\_history**

**show collections**



1. Cluster funcionando para replicação de dados. Agora vou enviar algumas mensagens com protocolo **MQTT**, e novamente validar a replicação de dados no cluster mongodb:
2. exteste
3. teste
4. teste
5. teste
6. teste
7. teste
8. test
9. eteste
10. tests
11. etset
12. set
13. set
14. set
15. estes
16. t