**Dojot Com cluster Mongodb - LAB**

**Aluno:** Marcus Artiaga Colantoni

**Orientador:** Raphael Pasquini

**Coorientador:** Rodrigo Sanches Miani

**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**

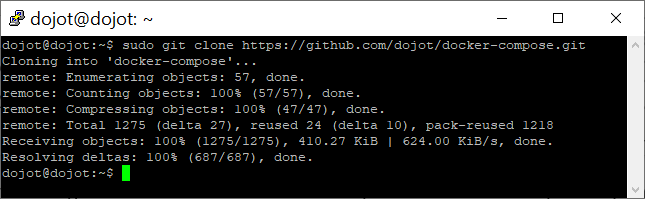
**curl -L https://github.com/docker/compose/releases/download/1.25.5/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose**

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

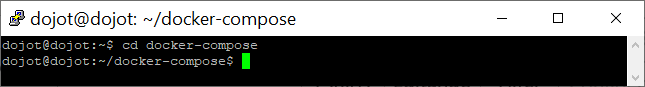
**docker-compose -v**

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

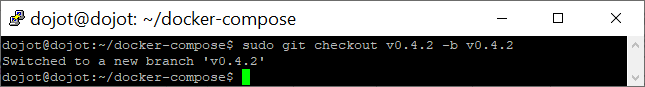
**1 - sudo git clone** [**https://github.com/dojot/docker-compose.git**](https://github.com/dojot/docker-compose.git)



**2 - cd docker-compose**

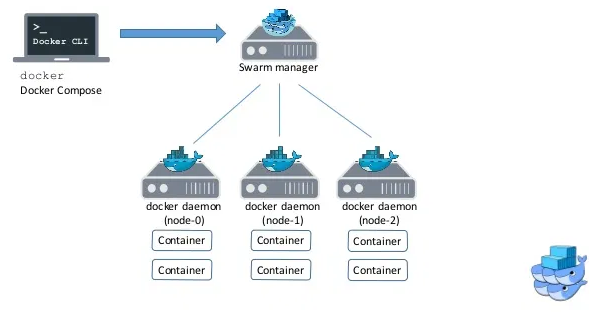


**3 - sudo git checkout v0.4.2 -b v0.4.2**



**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

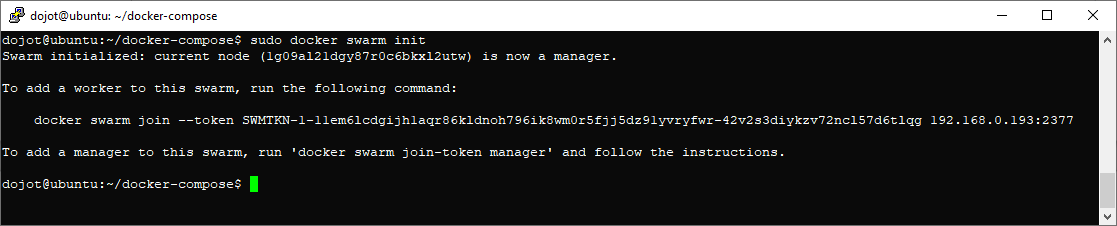


1. A rede overlay é uma rede que se sobrepõem a rede física, criando um cluster distribuído, para iniciar a rede overlay primeiramente precisamos iniciar o **swarm**, para iniciar o **swarm** utilizei o comando:

**sudo docker swarm init**

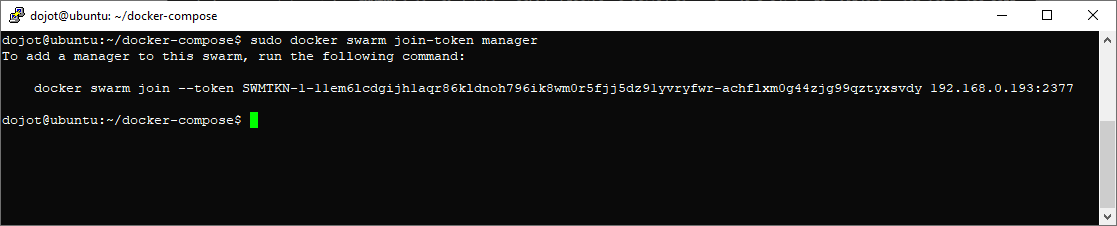
1. Em seguida é mostrado o token para unir/juntar novos nós (na forma de Works) ao swarm criado:

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

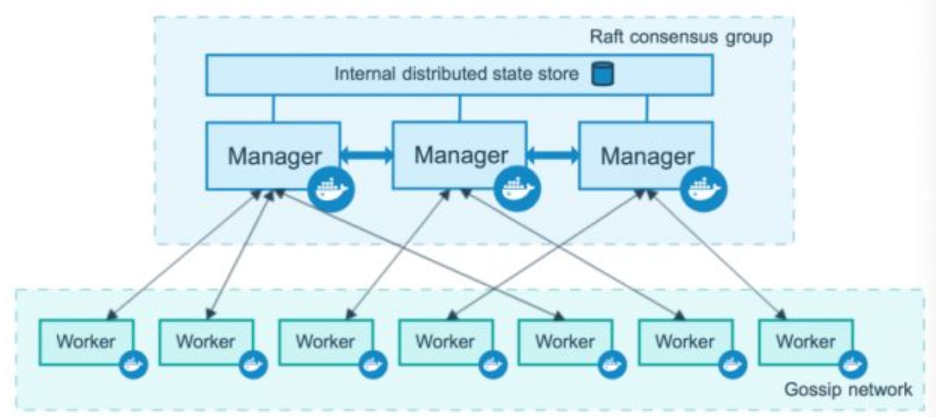


1. Para Adicionar um nó a rede swarm na forma de Leader/Manager é preciso obter um token próprio para isso:

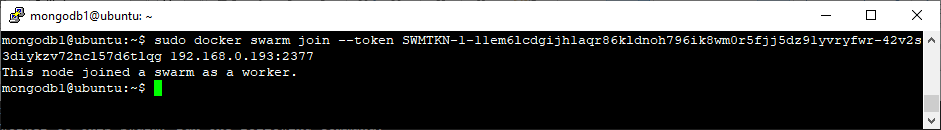
**sudo docker swarm join-token manager**



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

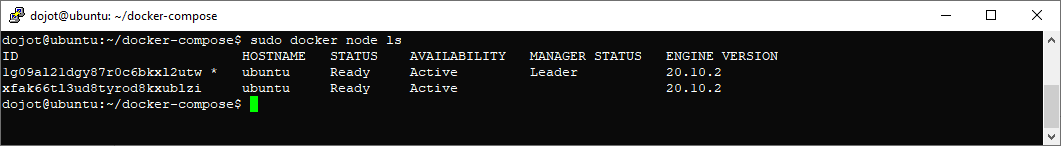


1. Agora vou adicionar outro nó ao cluster esse será adicionado como WORKDER , para isso criei uma VM chamada **Mongodb1**, com o mesmo sistema operacional Ubuntu 16.0.4.6 live server amd64, e realizei a instalação do docker, para adicionar a **VM-MONGODB1** como nó worker na rede utilizei o comando:



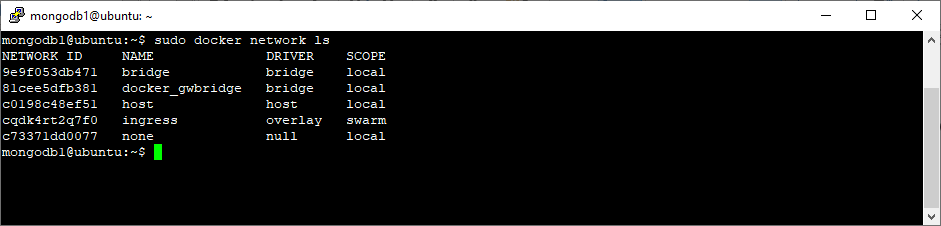
1. Agora ao consultar na VM **Dojot\_server** que é o Leader do cluster na rede swarm, é possível verificar que existem dois nós na rede:

**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**:

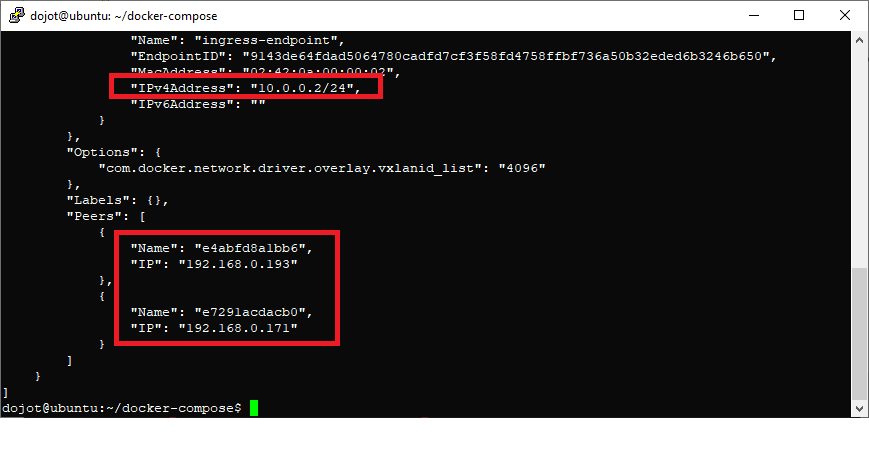
**sudo docker network ls**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 | 192.168.100.125 | Leader |
| Mongodb1 | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 512 MB | 10GB | 192.168.100.85 | worker |

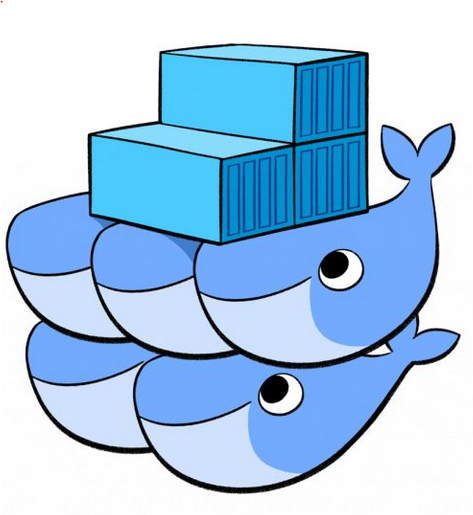
1. Agora podemos consultar qual o IP da rede swarm, inspecionando a rede ingress que que foi criada com o driver OVERLAY:

**sudo docker inspect ingress**



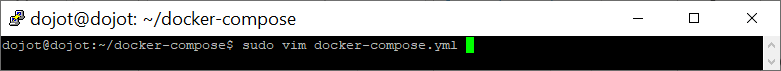
**OBS:** Podemos observar o IP de classe A 10.0.0.2/24 que é utilizado na rede swarm, com o driver overlay, e abaixo os endereços de IP dos hosts que compõem os nós da rede, sendo IP de cada uma das máquinas virtuais.

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**

**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**

**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"**

Realizei as seguinte modificações no container do Moquisto, que é o broker utilizado para receber as mensagens 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**

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

**volumes:**

**mongodb-volume:**

**mongodb-cfg-volume:**

**mongodb-volume2:**

**mongodb-cfg-volume2:**

**mongodb-volumeArb:**

**mongodb-cfg-volumeArb:**

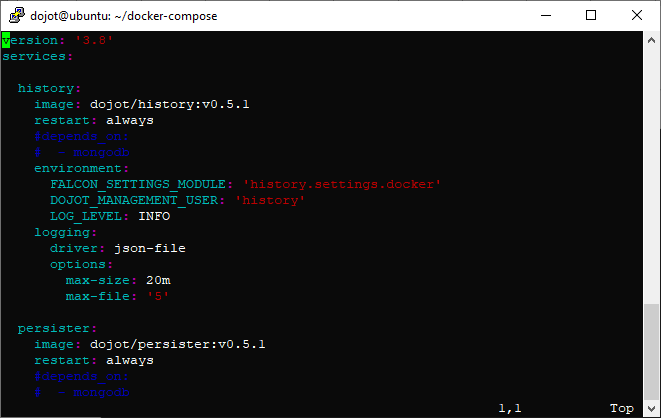
Adicionei a rede **colantoni-overlay** as redes do arquivo docker-compose.yml e especifiquei que o driver a ser utilizado é o overlay:

networks:

**flowbroker:**

**colantoni-overlay:**

**driver: overlay**

****

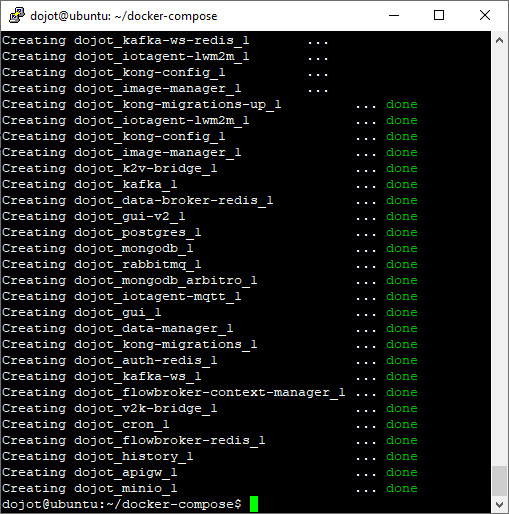
1. Para sair do arquivo e salvar:

**ESC**

**:wq!**

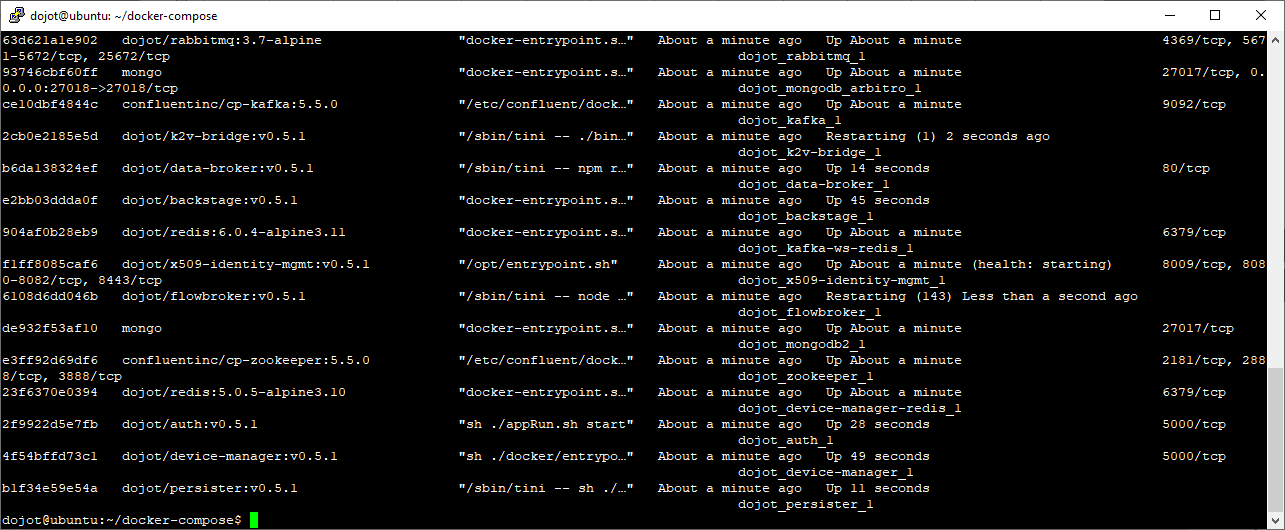
1. Agora vou rodar o arquivo de configuração YML do docker-compose com o comando:

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



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

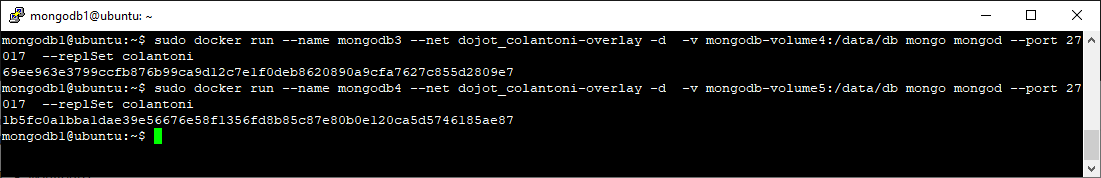
**sudo docker ps**



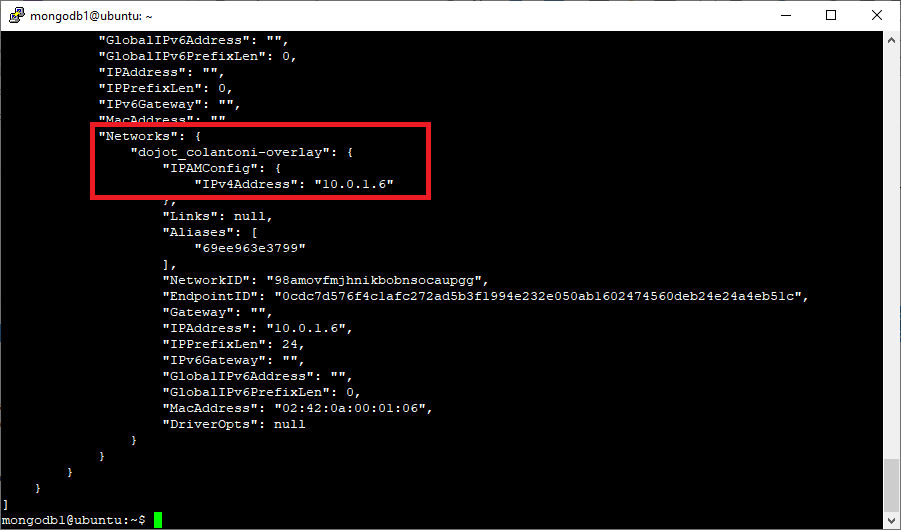
1. Agora vou adicionar mais dois containers ao cluster do Mongodb, porem agora vou subi-los de outro no, que está na VM-BRUMADINHO-MONGODB:

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

**sudo docker run --name mongodb4 --net dojot\_colantoni-overlay --restart always -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:

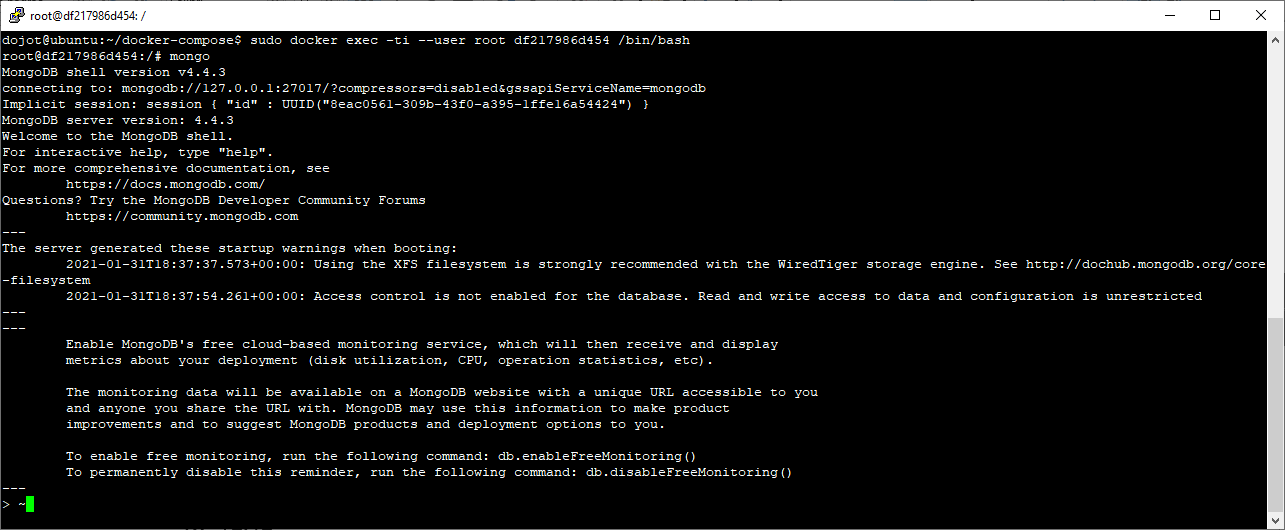
**sudo docker inspect 69ee963e3799**



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

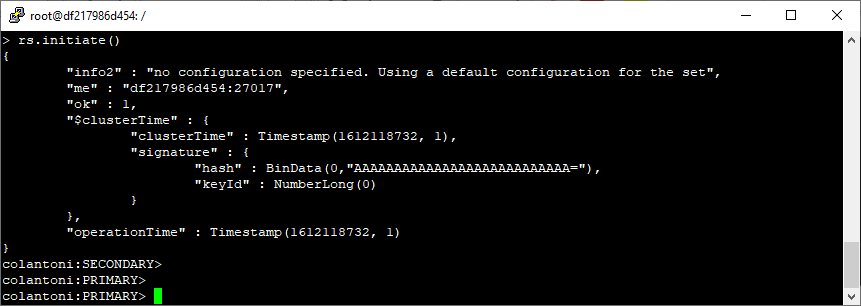
**sudo docker exec -ti --user root df217986d454 /bin/bash (id\_container)**

**mongo**

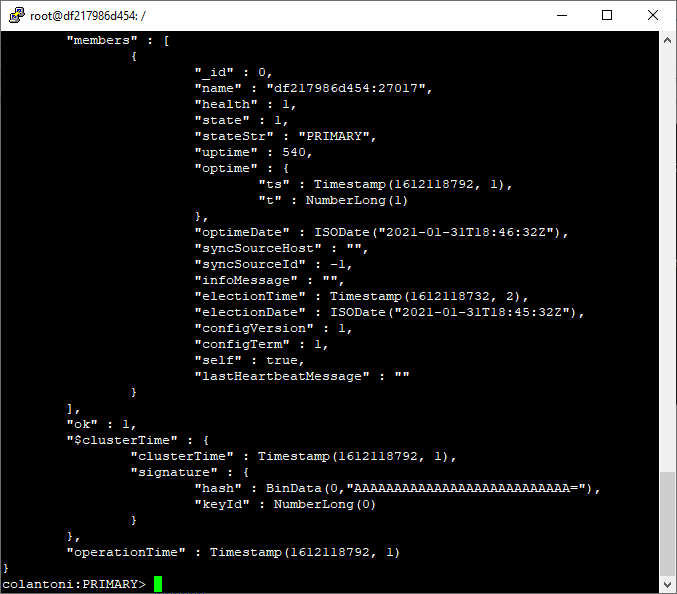


1. Agora vou iniciar o cluster :

**rs.initiate()**

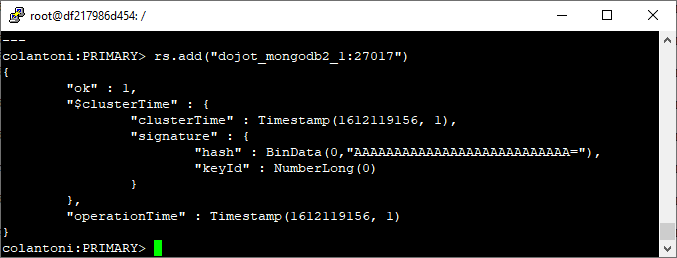


1. Cluster iniciado, agora consultando o status do cluster é possível notar que existe apenas um no cluster, de Id 0 , que é o próprio container:

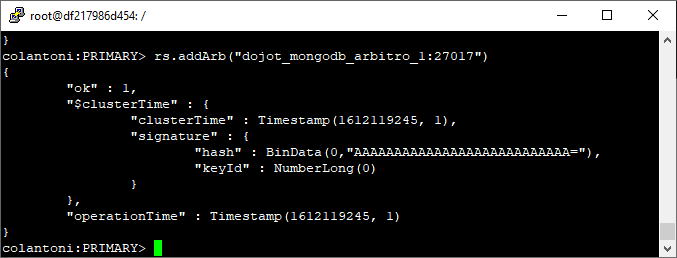


1. Agora vou adicionar os outros containers criados para o cluster, aqui é importante ressaltar que deve ser utilizado o **DNS do docker**, utilizar o IP na hora de inserir um membro no cluster não é viável, pois caso o container reinicie muito provavelmente o endereço IP sera modificado perdendo a comunicação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

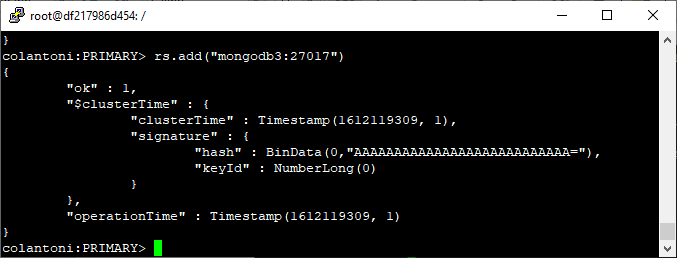
**rs.add("dojot\_mongodb2\_1:27017")**



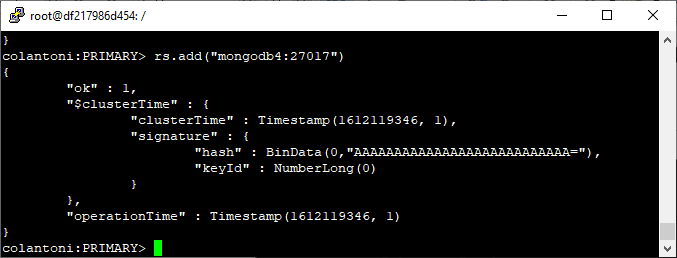
**rs.add ("dojot\_mongodb\_arbitro\_1:27017")**

****

**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 = 0 (arbitro)**

**cfg.members[2].priority = 3 (secundário qualquer mesmo no)**

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

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

**cfg.members[5].priority = 2 (secundario qualquer em outro no da rede)**

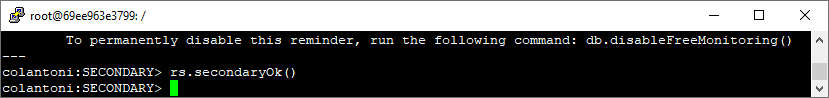
**cfg.members[6].priority = 2 (secundario qualquer em outro no da rede)**

**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()**

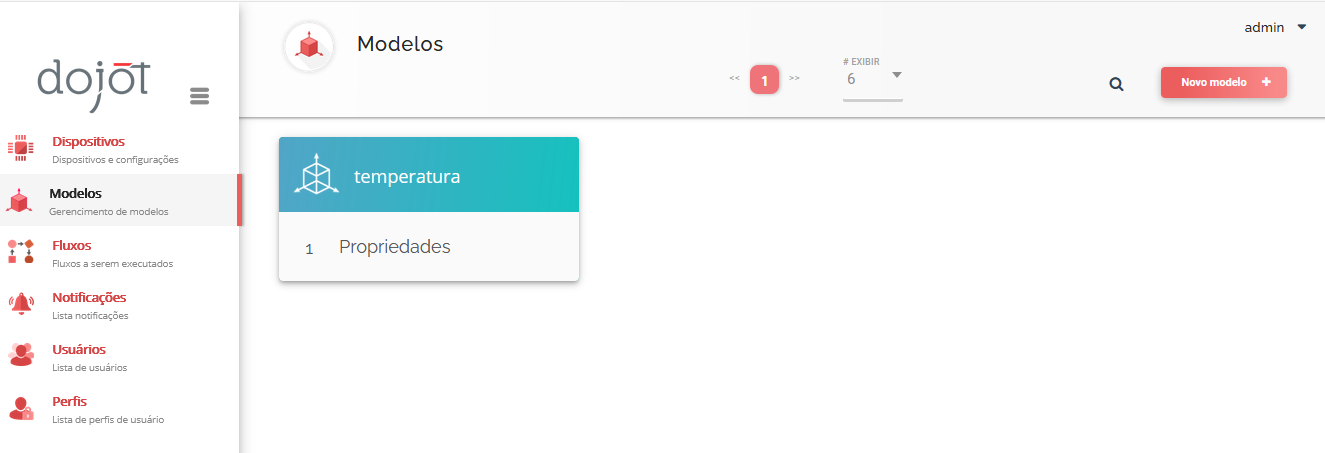
****

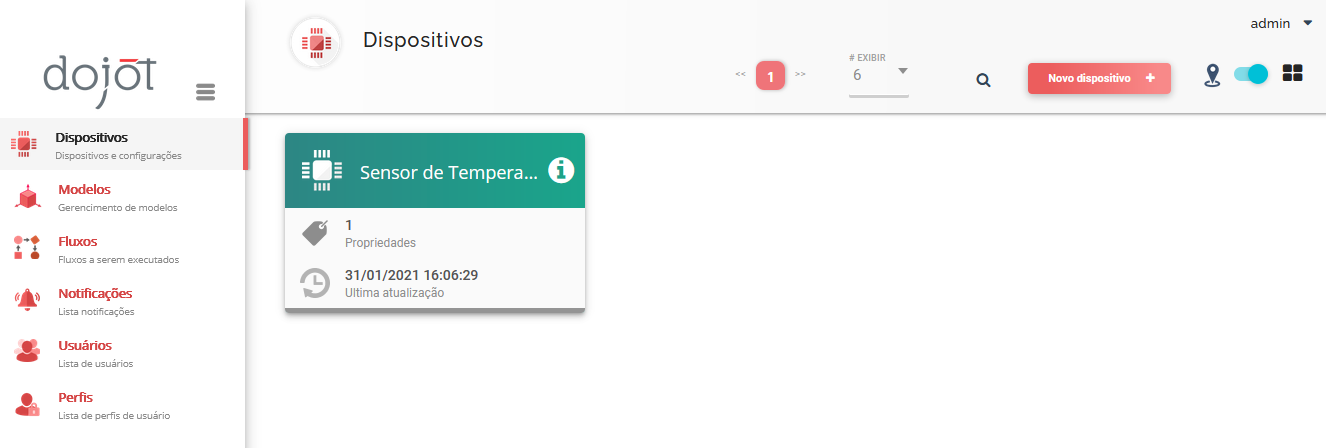
1. Agora vou testar a replicação dos dados que estão sendo salvos no mongodb, para isso vou acessar o dojot pela interface web:

URL: [http://192.168.0.125:8000/#/](http://192.168.0.193:8000/#/)



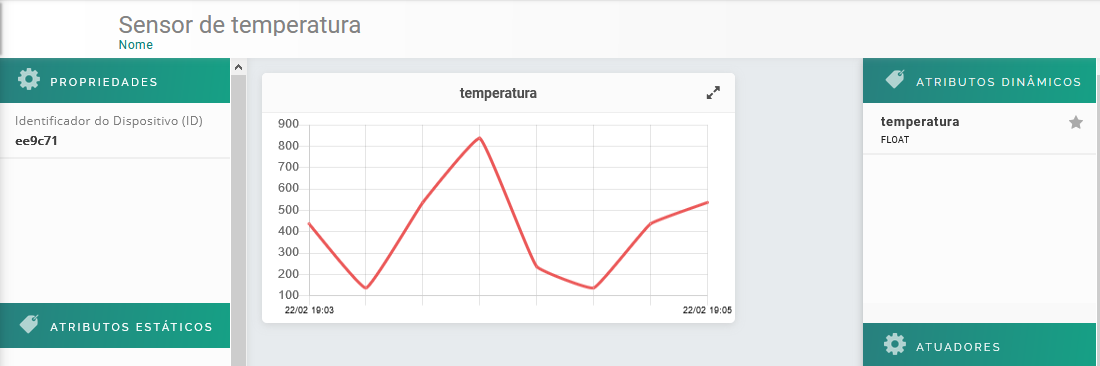
1. Vou criar um modelo chamado **TEMPERATURA** e um dispositivo chamado , “**SENSOR DE TEMPERATURA”** :





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

ID = **ee9c71**

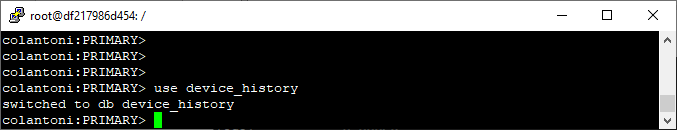


1. No container master do mongodb, vou acessar o shell do mongodb:

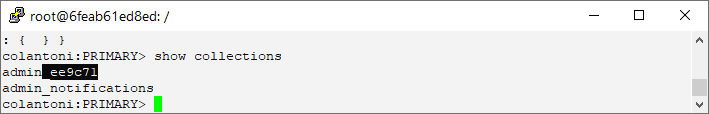
**show dbs**



**use device\_history**



**show collections**



1. Aqui já podemos ver que o dispositivo criado pelo usuário “**admin**” com ID **ee9c71.** 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**

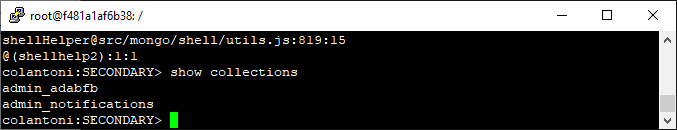
**show collections**

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:

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 436.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 136.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 536.5}'**

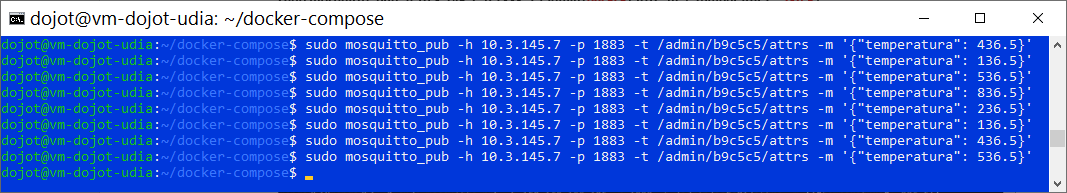
**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 836.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 236.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 136.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 436.5}'**

**sudo mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura": 536.5}'**



1. Verificando as informações salvas no cluster do mongodb, primeiro vou acessar o Master da rede, o container **dojot\_mongodb\_1**:

**Sudo docker exec -ti –user root 6feab61ed8ed /bin/bash**

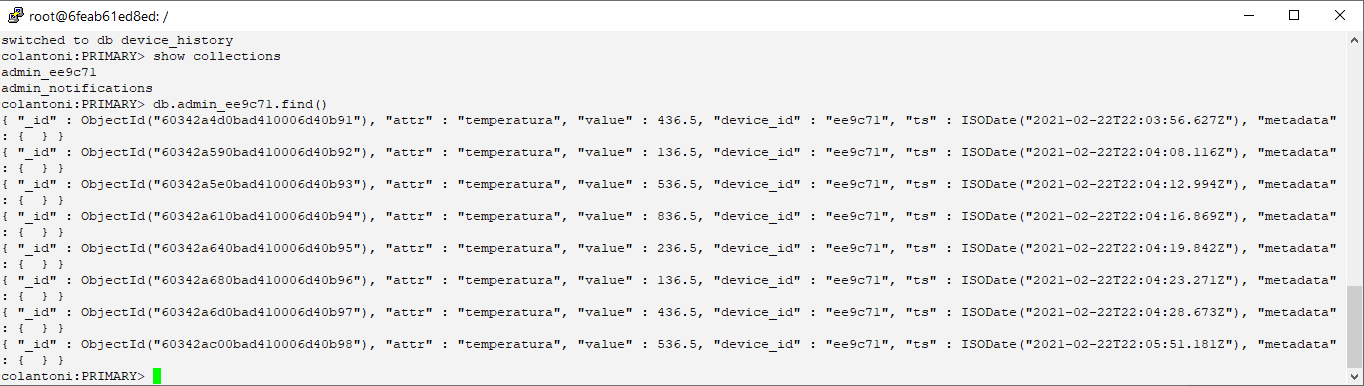
**Mongo**

**Show dbs**

**Use device\_history**

**Show collections**

**Db.** **admin\_ee9c71.find()**



1. Agora vou realizar o mesmo procedimento em outro no do cluster, o container **dojot\_mongodb2\_1**:

Teste **Sudo docker exec -ti –user root e4a6cad81e4d /bin/bash**

**Mongo**

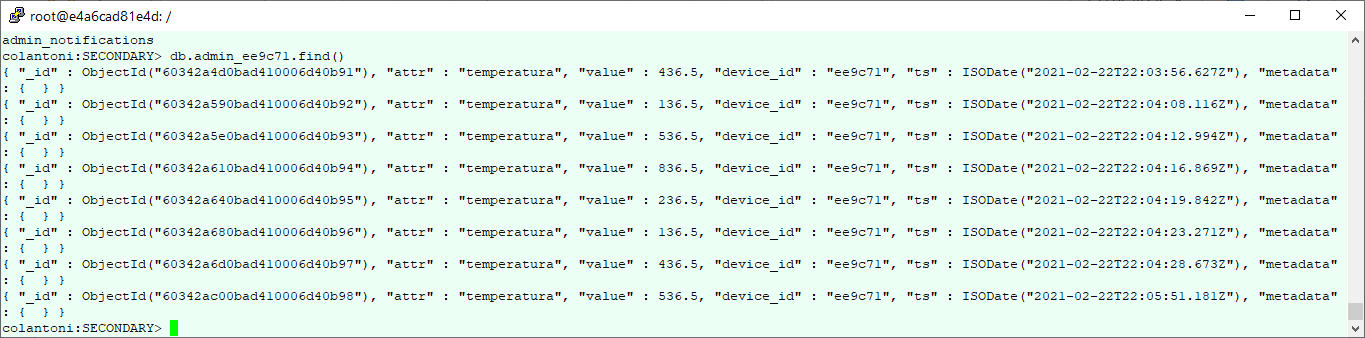
**rs.secondaryOk() --necessário por se tratar de no secundário do cluster**

**Show dbs**

**Use device\_history**

**Show collections**

**Db.** **admin\_ee9c71.find()**



1. Em caso de problemas no envio das mensagens MQTT para MOSQUITTO, é recomendável verificar o log do container para ver se tudo está rodando corretamente (**container dojot\_ejbca\_1 também deve ser observado**):

**sudo docker logs c072fb534a91teste (id container)**

1. No meu caso precisei para o container do **iotagent-mosca:v0.4.2**, depois removê-lo, e novamente rodar o arquivo de configuração **docker-compose,** para que o container fosse recriado, normalizando assim o serviço.

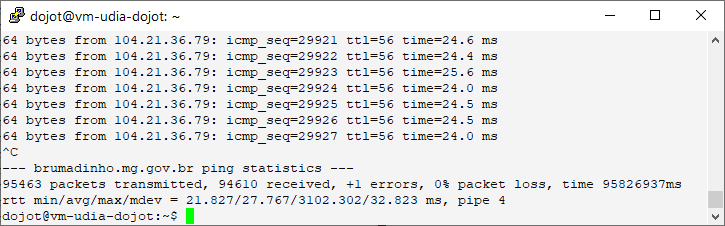
**sudo docker stop** **c072fb534a91teste (parar o container)**

**sudo docker rm c072fb534a91teste (remove o container)**

**sudo docker-compose.yml (roda novamente o arquivo de configuração)**

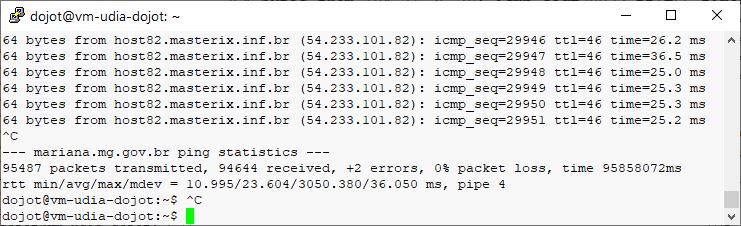
1. Testando o atraso de propagação de Uberlândia para a cidade de brumadinho em Mingas gerais, vou deixar o ping rodando por cerca de **24 horas**:

**ping brumadinho.mg.gov.br (website da prefeitura de Brumadinho)**

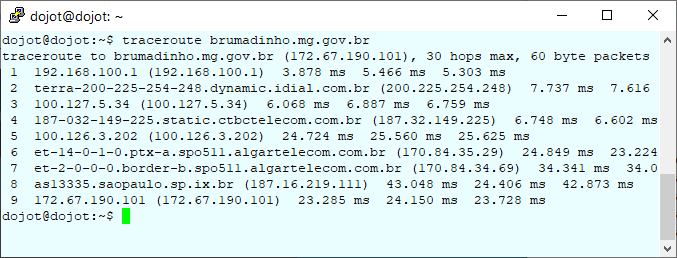


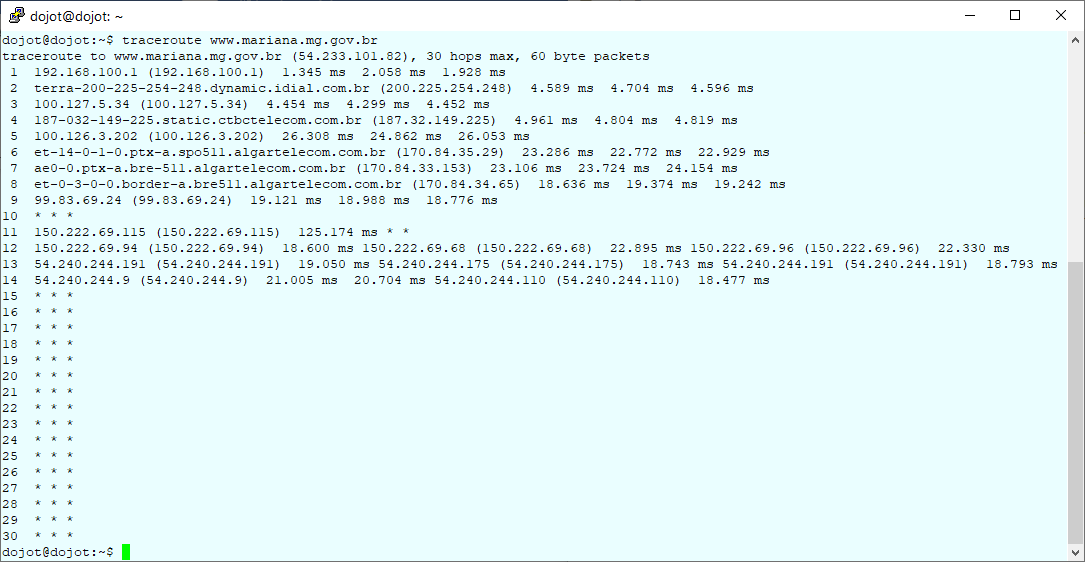
1. Testando o atraso de propagação de Uberlândia para a cidade de brumadinho em Mingas gerais, vou deixar o ping rodando por cerca de **24 horas**:

ping [www.mariana.mg.gov.br](http://www.mariana.mg.gov.br) (Website da Prefeitura de Mariana)



1. Verificando as rotas utilizadas de Uberlândia para **Brumadinho** e **Mariana,** :





1. Agora vou recriar este atraso de propagação que existe em as cidades, no cluster que crirei. Primeiro vou verificar a latência atual entre as VM´s:

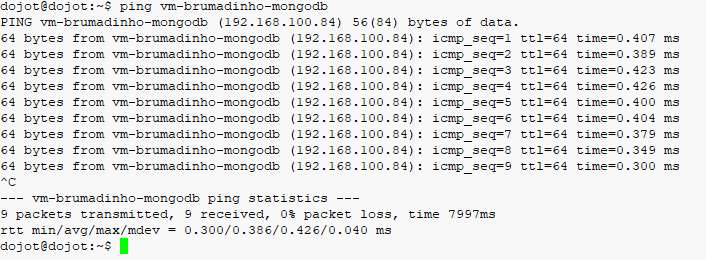
Dojot Principal = **VM-UDIA-DOJOT**

Mongodb replica 1 = **VM-BRUMADINHO-MONGODB**

Mongodb replica 2 = **VM-MARIANA-MONGODB**

1. Disparando ping da VM-UDIA-DOJOT para VM-BRUMADINHO-MONGODB:

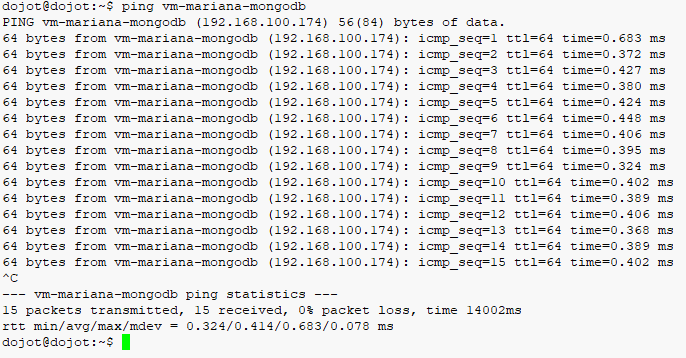
**ping vm-brumadinho-mongodb**



Média de atraso de propagação = **0.040ms**

1. Disparando ping da **VM-UDIA-DOJOT** para **VM-MARIANA-MONGODB:**

**ping vm-mariana-mongodb**



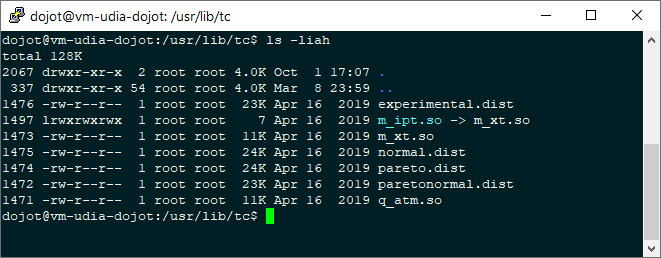
Média de atraso de propagação = **0.078ms**

|  |  |  |  |
| --- | --- | --- | --- |
| **SERVIÇO** | **NOME** | **IP** | **Atraso até Dojot** |
| Dojo Principal | VM-UDIA-DOJOT | 192.168.100.125 | 0 ms |
| Mongodb Replica 01 | VM-BRUMADINHO-MONGODB | 192.168.100.84 | 0.040 ms |
| Mongodb Replica 02 | VM-MARIANA-MONGODB | 192.158.100.174 | 0.078 ms |

1. O cenário atual de **VM** é o seguinte:
2. Agora vou configurar o TC (trafic control) para gerar um o mesmo atraso que existe fisicamente entre o servidor e as cidades de Brumadinho e Mariana, no protótipo. Para isso primeiro vou analisar como esta a distribuição dos arquivos de variação do TC:

cd /usr/lib/tc/

ls -liah



Aqui temos 4 arquivos que podem ser utilizados na distribuição de valores aleatórios durante a execução das variações de delay imposto pelo TC:

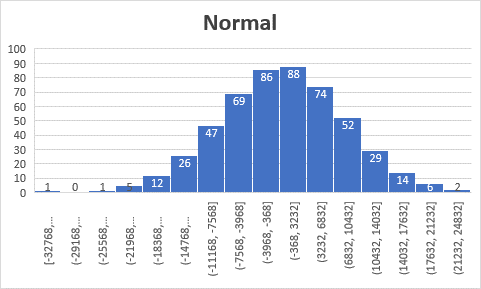
**Normal.dist**

**Pareto.dist**

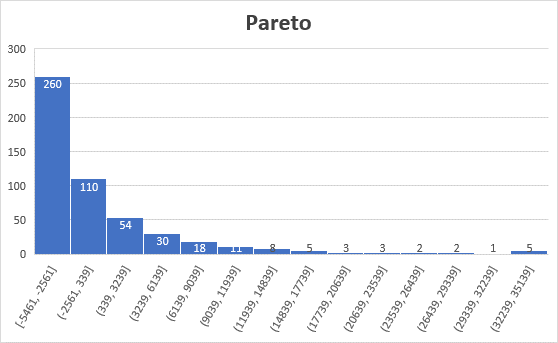
**Paretonormal.dist**

Experimental.dist

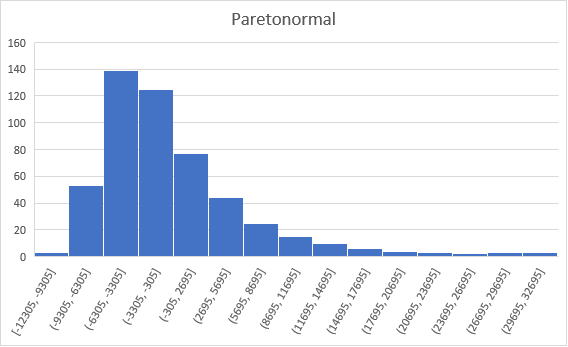
1. Agora vou analisar o histograma de cada um deles, começando pelo Normal.dist



1. Agora o Pareto:



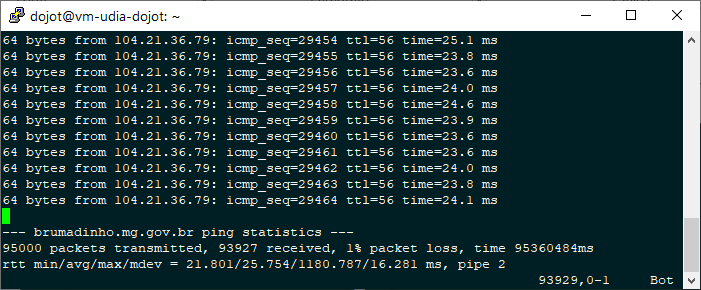
1. Paretonormal:

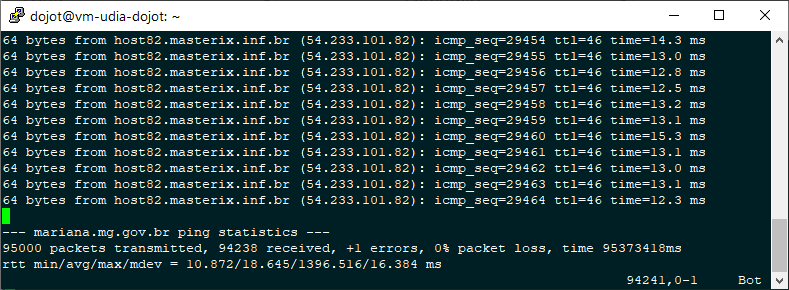


1. Agora vou coletar amostras do mundo real, na VM-UDIA-DOJOT vou disparar 95000 pacotes ICMP, o que gastou pouco mais de 24 horas para ser executado, com destino a para a prefeitura de Brumadinho e Mariana, e armazenar o resultado em arquivo .log:

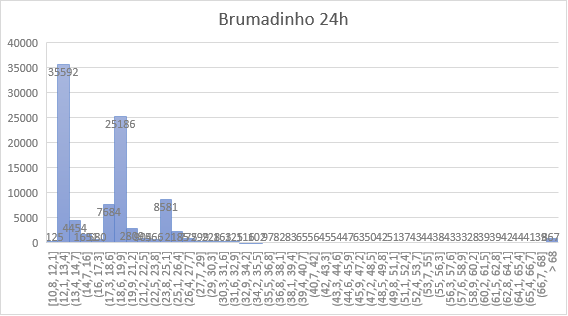
**ping -c 95000 brumadinho.mg.gov.br | tee ~/ping\_brumadinho.log**

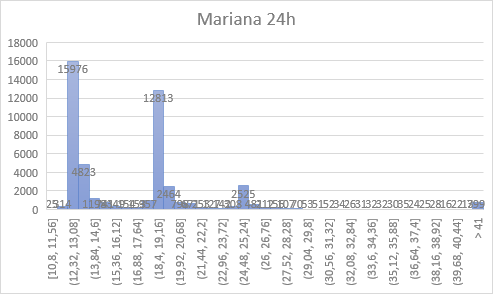
**ping -c 95000 mariana.mg.gov.br | tee ~/ping\_mariana.log**



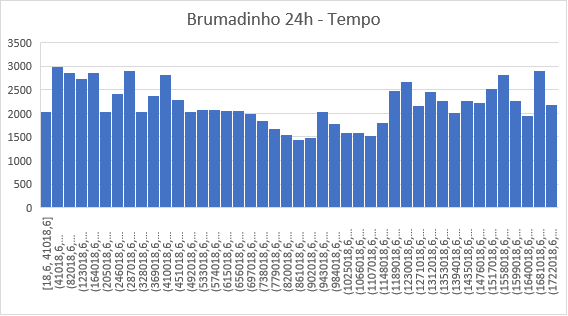


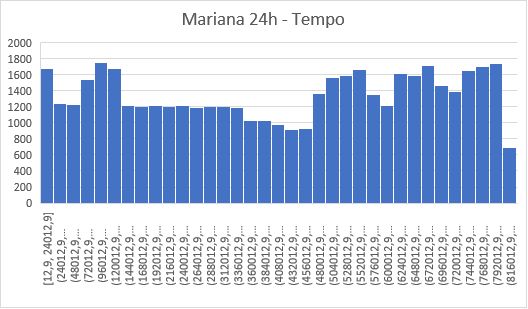
1. Primeiro vou gerar um histograma dos dois arquivos completos como todas as respostas recebidas:





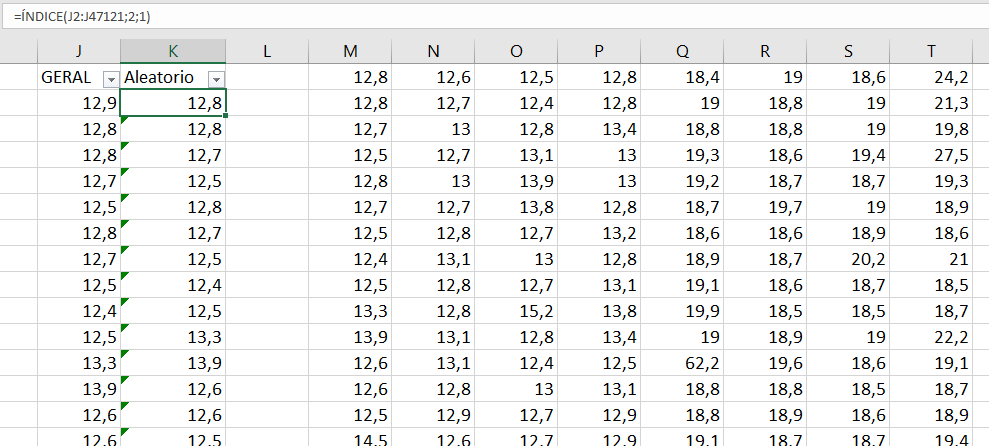
1. Também criei os histogramas dos pacotes distribuindo ao longo do tempo, acredito que a visualização ficou melhor:



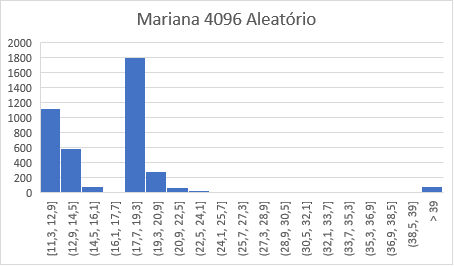


1. Agora vou criar a minha distribuição personalizada e salvar no arquivo **Experimental.dist**, utilizando os valores que capturei nos testes de 24 horas, espero conseguir uma distribuição mais próxima do que foi obtido nos testes reais, do que as distribuições, NORMAL, PARETO e PARETONORMAL:

**=ÍNDICE(J2:J47121;2;1)** (Função do Excel para capturar valor aleatório de uma listagem)

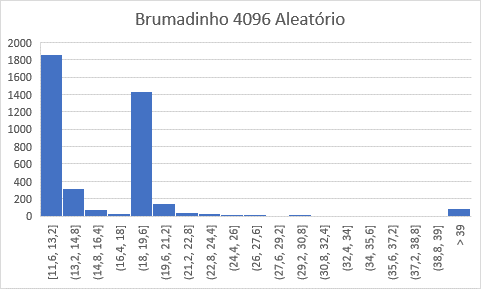


1. Montei os histogramas das distribuições aleatórias, que gerei para as duas cidades, e eles ficaram muito parecidos com os histogramas, obtidos com base nos dados obtidos a partir dos testes reais de 24 horas:



|  |  |  |  |
| --- | --- | --- | --- |
| **Distribuição Mariana 4096** | | | |
| **Máx. de Aleatório** | **Mín. de Aleatório** | **Média de Aleatório** | **DesvPadp de Aleatório** |
| 95,7 | 11,3 | 17,3357666 | 7,161139674 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Mariana 24h** | | | |
| **Máx. de GERAL** | **Mín. de GERAL** | **Média de GERAL** | **DesvPadp de GERAL** |
| 99,6 | 10,8 | 17,51846986 | 7,850452167 |



|  |  |  |  |
| --- | --- | --- | --- |
| **Brumadinho 24h** | | | |
| **Máx. de delay** | **Mín. de delay** | **Média de delay** | **DesvPadp de delay** |
| 99,6 | 10,8 | 18,09838492 | 7,820950785 |

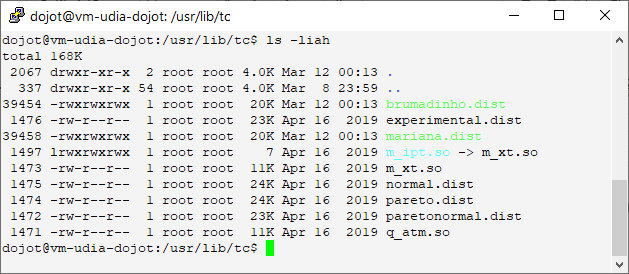
|  |  |  |  |
| --- | --- | --- | --- |
| **Distribuição Brumadinho 4096** | | | |
| **Máx. de aleatórios** | **Mín. de aleatórios** | **Média de aleatórios** | **DesvPadp de aleatórios** |
| 95,9 | 11,6 | 16,78554687 | 7,521330416 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Brumadinho 50%** | | | |
| **Máx. de completo** | **Mín. de completo** | **Média de completo** | **DesvPadp de completo** |
| 47,95 | 5,8 | 8,392773437 | 3,760665208 |

1. Extrai 4096 valores aleatórios, a partir dos dados que capturei nos testes de disparo de ICMP por 24 horas, para Brumadinho e Mariana, salvei cada um em um **arquivo.dist**:

**Brumadinho.dist**

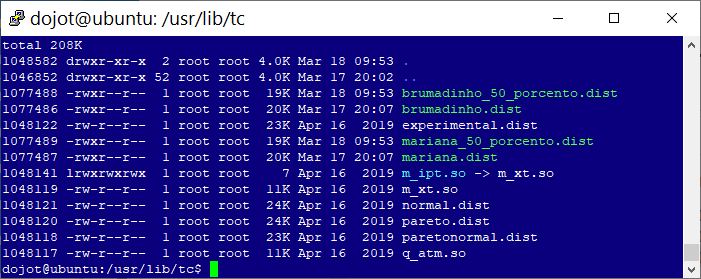
**Mariana.dist**



1. Em seguida criei outros dois arquivos de distribuição dividindo cada valor (dos 4096) por 2, de forma a ter 50% da distribuição:

**Brumadinho\_50\_procento.dist**

**Mariana\_50\_procento.dist**



1. Agora vou configurar a interface de rede do servidor principal do Dojot, **VM-UDIA-DOJOT** utilizando a distribuição que criei para brumadinho:

**sudo tc qdisc del root dev ens33 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens33 root netem** (inicia criação de regras na interface)

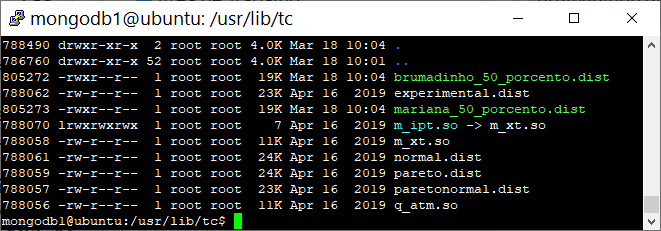
**sudo tc qdisc change dev ens33 root netem delay 12.5ms 5ms distribution brumadinho\_50\_porcento**

1. Na **VM-BRUMADINHO-MONGODB** também foi configurar o mesmo arquivo de distribuição com os outros 50 porcento:

**sudo tc qdisc del root dev ens33 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens33 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens33 root netem delay 12.5ms 5ms distribution brumadinho\_50\_porcento**



1. Na **VM-MARIANA-MONGODB** também foi configurar o mesmo arquivo de distribuição com os outros 50 porcento:

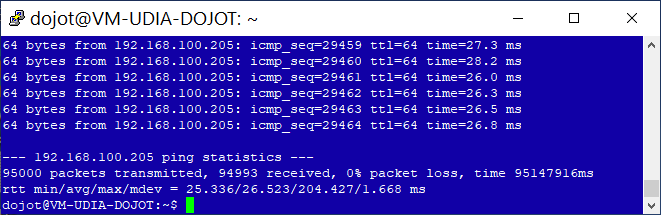
**sudo tc qdisc del root dev ens33 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens33 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens33 root netem delay 12.5ms 5ms distribution mariana\_50\_porcento**

1. Agora vou disparar mais 9500 pacotes ICMP da VM-UDIA-DOJOT, com destino a VM-BRUMADINHO, e armazenar os dados um arquivo:

**ping -c 9500 192.168.100.205 | tee ~/ping\_distribuicao\_brumadinho.log**



1. O Histograma obtido com essa captura, utilizando o TC com a distribuição personalizada para Brumadinho (disparo realizado por 24 horas):



1. O valor médio do ficou bem próximo do real:

|  |  |  |  |
| --- | --- | --- | --- |
| **Máx. de DELAY4** | **Mín. de DELAY3** | **Média de DELAY** | **DesvPadp de DELAY2** |
| 204 | 25,3 | 26,47440969 | 1,651145397 |

1. Agora vou adicionar uma nova placa de rede na VM-UDIA-DOJOT e também na VM-MARIANA-MONGODB, para que seja possível utilizar ao mesmo tempo uma comunicação entre as duvas VM´s do mongodb utilizando diferentes distribuições com o TC, de moro a emular os delays de propagação.
2. Na VM-UDIA-DOJOT adicionei um driver de rede através do gerenciamento do VM-WARE, após iniciar a VM e acessá-la por SSH:

**dmesg |grep eth** (listar interfaces de rede)

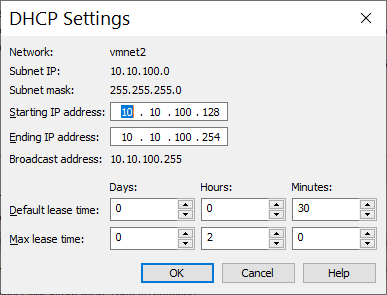
**sudo vim /etc/network/interfaces** (acessar arquivo de configuração)

**auto ens38** (linha adicionada ao final do arquivo)

**iface ens38 inet dhcp** (linha adicionada ao final do arquivo)

**sudo ifconfig ens38 up** (subindo a placa)

**sudo dhclient ens38** (forçando o dhcp na interface)

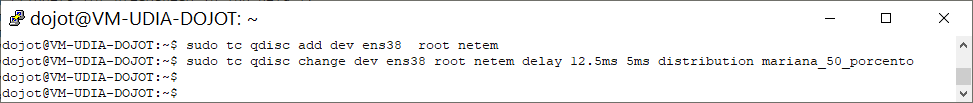


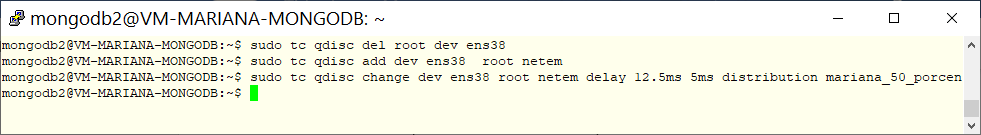
1. Agora vou realizar as mesas configurações de delay utilizando o TC, nas novas interfaces de rede que adicionei nas duas VM´s:

**sudo tc qdisc del root dev ens38 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens38 root netem** (inicia criação de regras na interface)

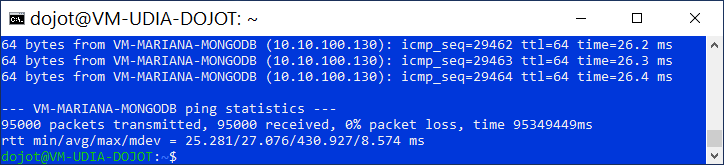
**sudo tc qdisc change dev ens38 root netem delay 12.5ms 5ms distribution mariana\_50\_porcento**



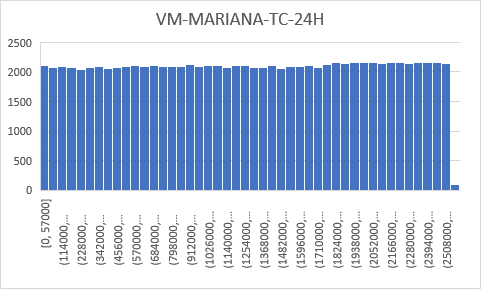


1. Iniciando disparo da VM-UDIA-DOJOT pela interface ens38 (**10.10.100.129**) para a VM-MARIANA-MONGODB na interface ens39 (**10.10.100.130**) , utilizando a distribuição personalizada que crei com base nas amostras obtidas nos teste de 24 horas ,entre a VM-UDIA-DOJOT e a prefeitura de Mariana:

**ping -c 95000 VM-MARIANA-MONGODB | tee ~/ping\_distribuicao\_mariana2.log**



1. O Histograma obtido com essa captura, utilizando o TC com a distribuição personalizada para Mariana (disparo realizado por 24 horas):



1. O valor médio do ficou bem próximo do real:

|  |  |  |  |
| --- | --- | --- | --- |
| **Máx. de delay** | **Mín. de delay** | **Média de delay** | **DesvPadp de delay** |
| 430 | 25,2 | 27,02552316 | 8,555323602 |

1. Problema atual:

1 – Como configurar o Docker swarm para utilizar duas interfaces de rede distintas?

2 – Como configurar o Docker swarm para que se comunique com **a VM-1 na interface ens33** e a **VM-2 na interface ens38** ?

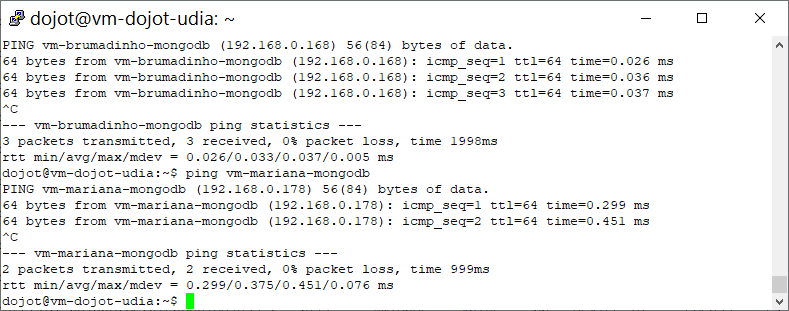
3 – Seria possível redundância entre as duas interfaces?

OU

1 - **O TC poderia configurar diferentes distribuições de delay dependendo do destino ? (hoje esta orientando a interface ens38 ).**

1. Testando como ficariam as estatísticas se caso a configuração de delay no TC estivesse configurada apenas nas VM´s do mongodb, VM-BRUMADINHO e VM-MARIANA, desta forma poderia utilizar a interface ens38 que já está configurada para o Docker swarm.
2. Na VM-UDIA-DOJOT (sem configuração de delay, modo padrão) :

**sudo tc qdisc del root dev ens38 (**remove qualquer regra que exista na interface**)**



1. Na VM-MARIANA-MONGODB :

**sudo tc qdisc del root dev ens38 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens38 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens38 root netem delay 25ms 10ms distribution mariana\_100\_porcento**

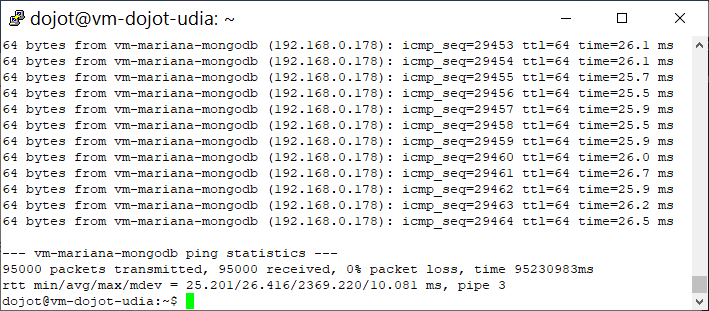
1. Na VM-BRUMADINHO-MONGOBD:

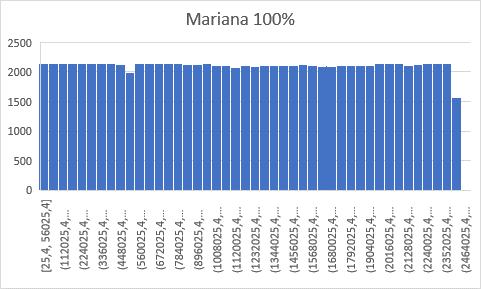
**sudo tc qdisc del root dev ens38 (**remove qualquer regra que exista na interface**)**

**sudo tc qdisc add dev ens38 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens38 root netem delay 25ms 10ms distribution brumadinho\_100\_porcento**

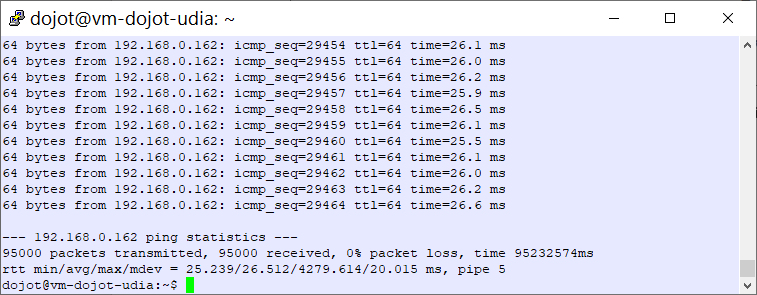
1. Agora vou simultaneamente disparar pacotes ICMP nas duas VM´s do mongodb , VM-BRUMADINHO e VM-MARIANA e armazenar as capturas de tempo de resposta em um arquivo para depois comparar com os resultados obtidos na configuração de 50%.
2. **ping -c 95000 VM-MARIANA-MONGODB | tee ~/ping\_distribuicao\_mariana\_100.log**

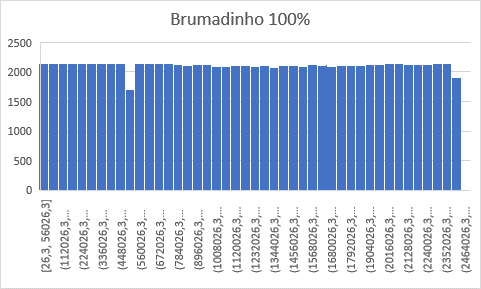




|  |  |  |  |
| --- | --- | --- | --- |
| **Máx. de delay ms** | **Mín. de delay ms4** | **Média de delay ms3** | **DesvPadp de delay ms2** |
| 2369 | 25,2 | 26,36633684 | 10,07465508 |

1. **ping -c 95000 VM-BRUMADINHO-MONGODB | tee ~/ping\_distribuicao\_brumadinho\_100.log**





|  |  |  |  |
| --- | --- | --- | --- |
| **Máx. de delay ms** | **Mín. de delay ms4** | **Média de delay ms3** | **DesvPadp de delay ms2** |
| 4279 | 25,2 | 26,46217368 | 20,00926407 |

1. teste

**sudo tc qdisc add dev docker\_gwbridge root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev docker\_gwbridge root netem delay 12.5ms 5ms distribution brumadinho\_50\_porcento**

1. Realizando testes de redundância no cluster mongodb:
2. Primeiramente vou verificar se todos os containers estão alinhados, com todos os dados existentes até o momento:

**sudo Docker exec -ti –user root nome\_containter /bin/bash**

**mongodb**

**show dbs**

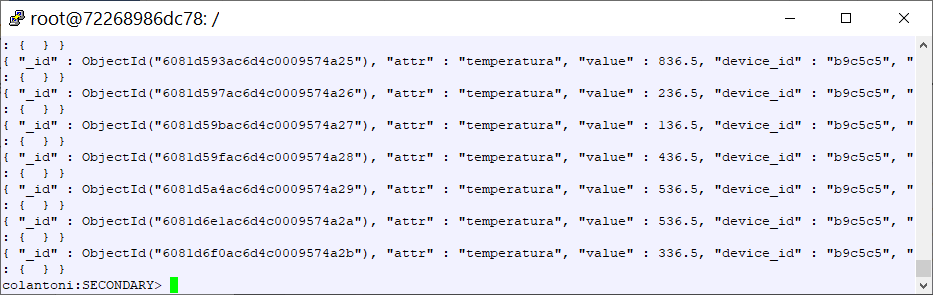
**use device\_history**

**show collections**

**db.admin\_b9c5c5.find()**

**VM-UDIA-DOJOT : dojot\_mongodb\_1**

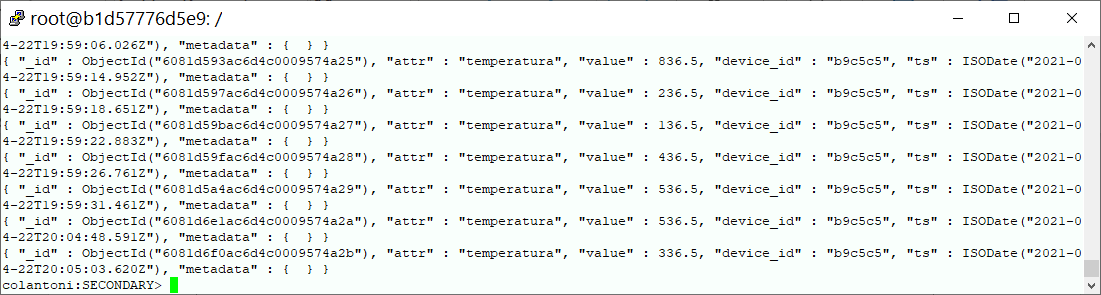


**VM-UDIA-DOJOT : dojot\_mongodb2\_1** 

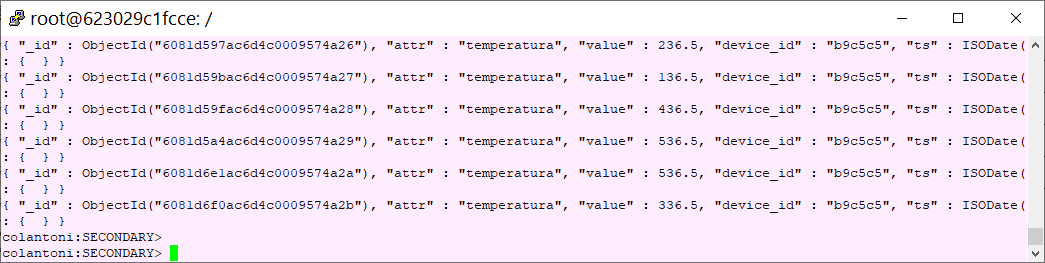
**VM-BRUMADINHO : mongodb3**



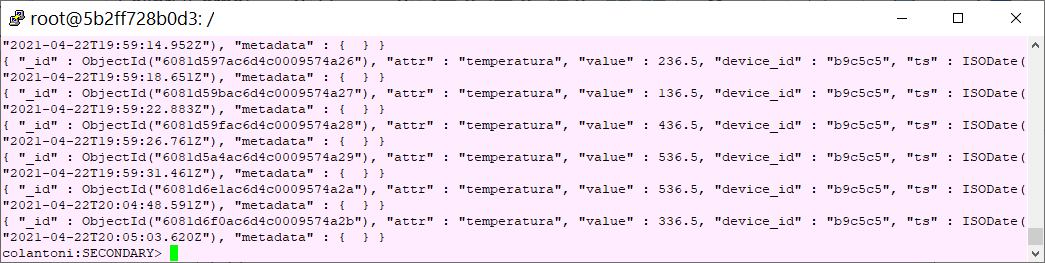
**VM-BRUMADINHO : mongodb4**



**VM-MARIANA : mongodb5**



**VM-MARIANA : mongodb6**

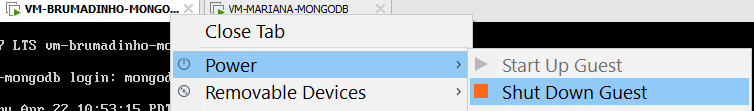




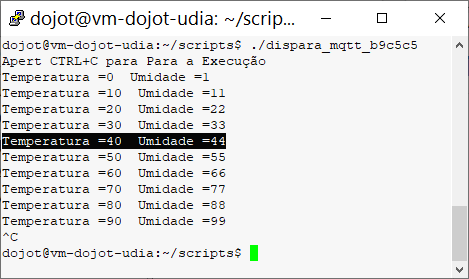
1. Agora vou iniciar o scritp em shell para disparar mensagens MQTT no dispositivo **b9c5c5**, infinitamente enquanto estiver rodando, em seguida vou desligar a VM-BRUMADINHO e após 20 segundos (4 saltos do script) vou parar o script, em seguida vou verificar qual o último registro salvo nos nós em estado (1 – UP) e qual o ultimo registro salvo na VM-BRUMADINHO que foi reiniciada durante o disparo das mensagens MQTT:
2. Iniciando o script para disparar as mensagenes MQTT:

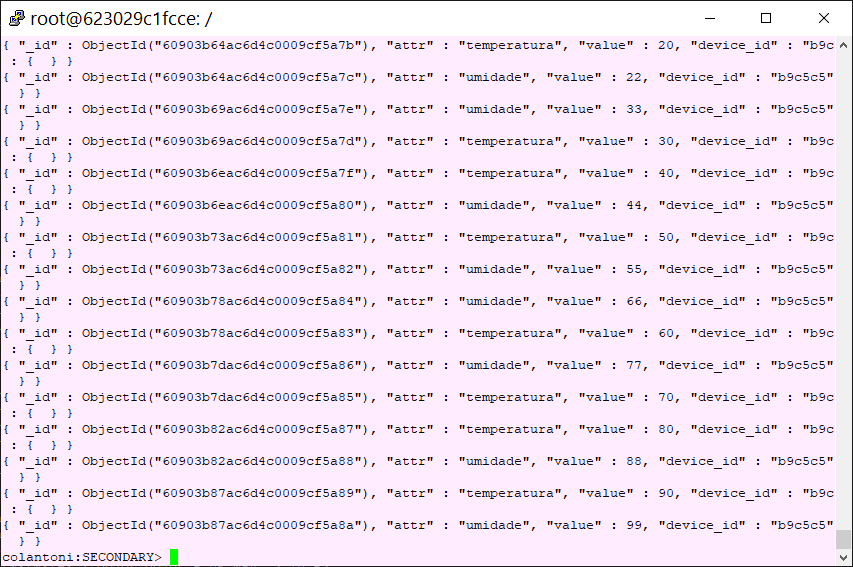
**./dispara\_mqtt\_b9c5c5**

1. Parando a VM-BRUMADINHO, e após 20 segundos parando o script:



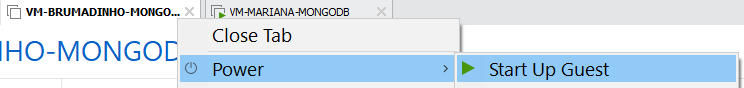
1. Ate o disparo em destaque, temperatura 40 e umidade 44 a VM-BRUMADINHO estava ligada, após o seu desligamento o script executou ate a temperatura 90 e umidade 99:



**MARIANA : mongodb5** 

**VM-MARIANA : mongodb6** 

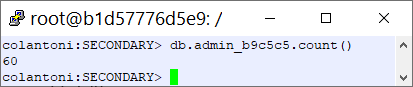
1. Agora vou iniciar novamente a VM-BRUMADINHO e verificar qual o último registro salvo nos container mongodb:



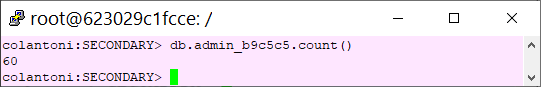
1. Para facilitar a comparação vou utilizar um comando para contagem de registros dentro da coleção, desta forma fica mais fácil averiguar se o número de registros está correto, e cobre diferenças de ordenação dos dados (no shell do mongodb):

**db.admin\_b9c5c5.count()**

1. **VM-BRUMADINHO: mongodb4**



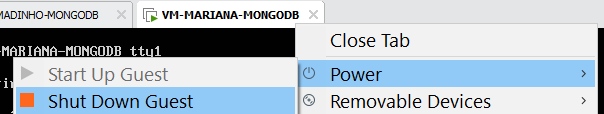
1. **VM-MARIANA: mongodb5**



1. **VM-UDIA-DOJOT: dojot\_mongodb\_1**



1. Agora vou repetir o teste porem vou desligar a VM-MARIANA e deixar as demais VM´s ligadas, sem seguida vou acessar novamente o shell do mongodb e verificar se o numero total de registros é igual:



1. Disparando 6 mensagens MQTT (que irão totalizar 12 registros salvos) enquanto a VM-MARIANA esta desligada:

**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":777,"umidade":17}'**

**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":666,"umidade":16}'**

**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":555,"umidade":15}'**

**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":444,"umidade":14}'**

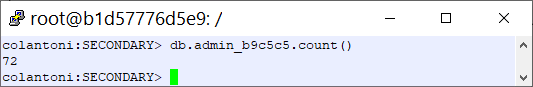
**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":333,"umidade":13}'**

**mosquitto\_pub -h 10.3.145.7 -p 1883 -t /admin/b9c5c5/attrs -m '{"temperatura":222,"umidade":12}'**

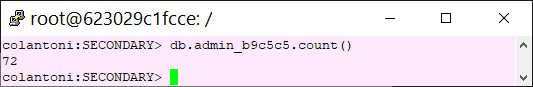
1. Agora vou verificar o total de registros salvos dentro da coleção do dispositivo **b9c5c5**:
2. **VM-UDIA-DOJOT: dojot\_mongodb\_1**



1. **VM-BRUMADINHO: mongodb4**



1. **VM-MARIANA: mongodb5**



1. Teste
2. Este
3. Tsets
4. Etes
5. Te
6. Teste
7. Teste
8. Teste
9. Test
10. tes

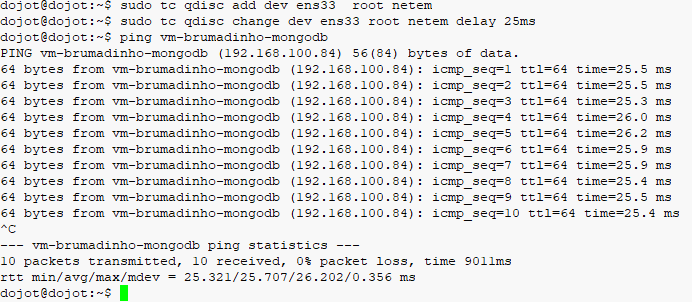
|  |  |  |  |
| --- | --- | --- | --- |
| **VM** | **CONTAINER** | **CLUSTER** | **IP rede Overlay – Swarm** |
| VM-UDIA-DOJOT | dojot\_mongodb\_1 | Master | 10.0.1.2 |
| dojot\_mongodb\_2 | Slave | 10.0.1.6 |
| dojot\_mongodb\_arbitro\_1 | Arbitro | 10.0.1.4 |
| VM-BRUMADINHO-MONGODB | mongodb3 | Slave | 10.0.1.10 |
| mongodb4 | Slave | 10.0.1.12 |
| VM-MARIANA-MONGODB | mongodb5 | Slave | 10.0.1.7 |
| mongod6 | Slave | 10.0.1.9 |

1. O cenário atual do cluster completo considerando o Dojot e as instâncias do Mongod é o seguinte:
2. Configurando atras de propagação entre as VM´s afim de replicar o atraso obtido nos testes de latência realizado entre as cidades (25ms) entre os links. Na VM-UDIA-DOJOT:

**sudo tc qdisc add dev ens33 root netem (Inicia criação de regras para interface)**

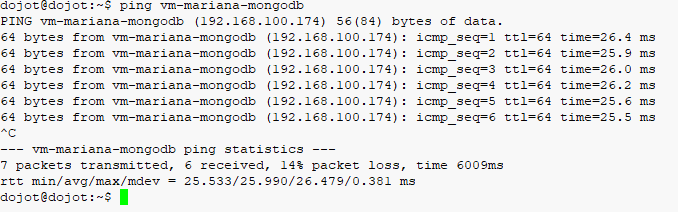
**sudo tc qdisc change dev ens33 root netem delay 25ms (Configura delay de 25ms na interface)**

**ping vm-brumadinho-mongodb (disparando ping para verificar novo atraso)**



1. Disparando ping da VM-UDIA-DOJOT para a VM-MARIANA-MONGODB:

**ping vm-udia-dojot**



1. Para validar vou disparar alguns pacotes das VM´s slaves para o Dojot:

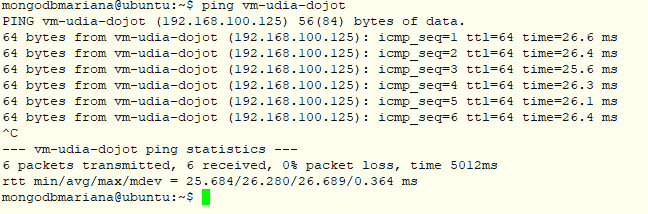
**VM-BRUMADINHO-MONGODB** -> **VM-UDIA-DOJOT**

**ping vm-udia-dojot**



**VM-MARIANA-MONGODB** -> **VM-UDIA-DOJOT**

**ping vm-udia-dojot**

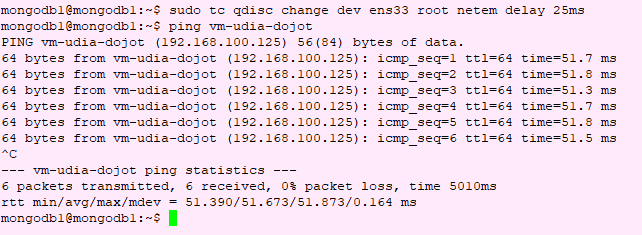


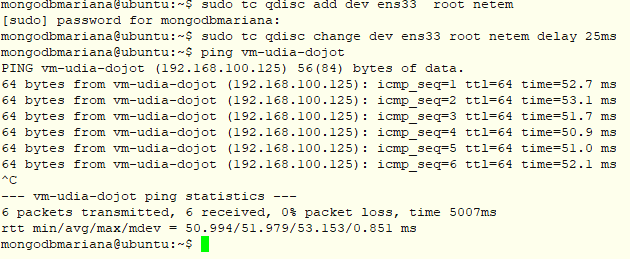
1. Agora vou configurar um atraso de 25ms também nas VM-BRUMADINHO-MONGODB, VM-MARIANA-MONGODB:

**sudo tc qdisc add dev ens33 root netem (Inicia criação de regras para interface)**

**sudo tc qdisc change dev ens33 root netem delay 25ms (Configura delay de 25ms na interface)**

**ping vm-udia-dojot (disparando ping para verificar novo atraso)**

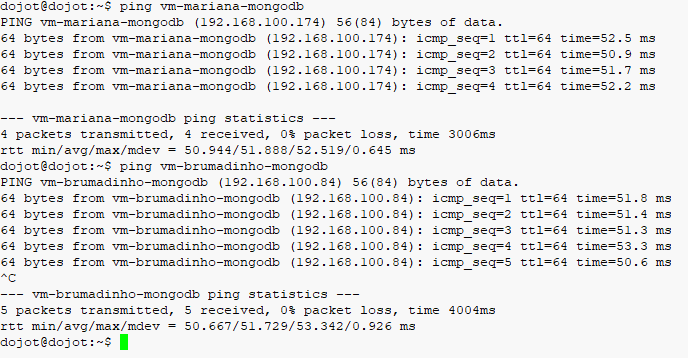




1. O atraso total ficou de 50ms no mínimo, sendo 25ms em cada direção, podemos observar também na dojot:

**ping vm-mariana-mongodb**

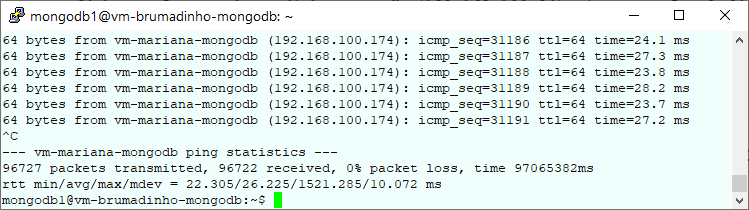
**ping vm-brumadinho-mongodb**



1. Agora utilizei uma configuração diferente no TC, onde configurei, delay, variação randômica e valor correlato:

**sudo tc qdisc add dev ens33 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens33 root netem delay 25ms 10ms 25%** (define delay =25ms, Variação randômica =10ms, Valor correlato= 25% ), ping executou por **24 horas**:



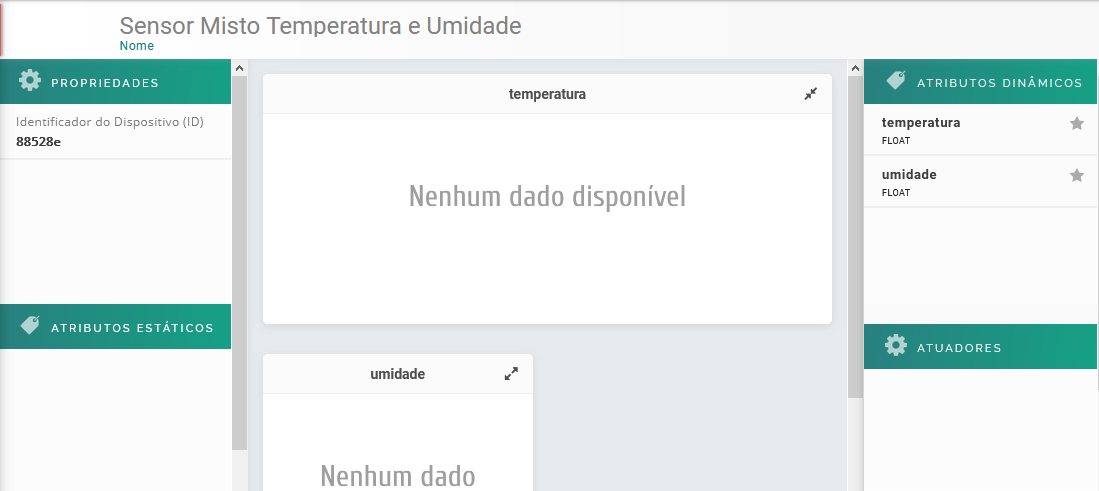
1. Novo teste trocando o valor de variação para

**sudo tc qdisc add dev ens33 root netem** (inicia criação de regras na interface)

**sudo tc qdisc change dev ens33 root netem delay 25ms 10ms distribution paretonormal** ,ping executou **24 horas**:



1. Agora vou um arquivo de distribuição uniforme, através de valores obtidos do mundo real, primeiro vou disparar 4095 pacotes ICMP para as prefeituras de **Brumadinho** e **Mariana**, depois vou calcular a média e o desvio padrão sobre essas amostras, em seguida, vou subtrair a média de todas as amostras, finalmente vou dividir cada amostra pelo desvio padrão e multiplicar por 30. Ao final vou remover todas as amostrar que estiverem fora do intervalor , menor que 10ms e maior que 36ms:
2. Amostras coletadas
3. Teset
4. teset
5. Para realização dos testes criei um novo dispositivo com 2 atributos, sendo uma temperatura e o outro umidade: id= **ebc035**



1. Criei um script em shell para realizar o envio de mensagens mqtt para os dois atributos do novo dispositivo que criei:

**mkdir scripts** (cria uma pasta chamad scripts)

**cd scripts** (entra na pasta)

**sudo vim dispara\_mqtt\_** **ebc035.sh** (cria um arquivo .sh)

1. Dentro do arquivo SH que acabei de criar inseri o seguinte código: (Ele roda infinitamente a cada **5 segundos**, pressionando **CTRL+C** a execução para):

#! /bin/bash

while :; do

echo "Aperte CTRL+C para Parar a Execução";

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":0,"umidade":1}'

echo "Temperatura =0 Umidade =1";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":10,"umidade":11}'

echo "Temperatura =10 Umidade =11";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":20,"umidade":22}'

echo "Temperatura =20 Umidade =22";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":30,"umidade":33}'

echo "Temperatura =30 Umidade =33";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":40,"umidade":44}'

echo "Temperatura =40 Umidade =44";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":50,"umidade":55}'

echo "Temperatura =50 Umidade =55";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":60,"umidade":66}'

echo "Temperatura =60 Umidade =66";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":70,"umidade":77}'

echo "Temperatura =70 Umidade =77";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":80,"umidade":88}'

echo "Temperatura =80 Umidade =88";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":90,"umidade":99}'

echo "Temperatura =90 Umidade =99";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":100,"umidade":1}'

echo "Temperatura =100 Umidade =1";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":90,"umidade":11}'

echo "Temperatura =90 Umidade =11";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":80,"umidade":22}'

echo "Temperatura =80 Umidade =22";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":70,"umidade":33}'

echo "Temperatura =70 Umidade =33";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":60,"umidade":44}'

echo "Temperatura =60 Umidade =44";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":50,"umidade":55}'

echo "Temperatura =50 Umidade =55";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":40,"umidade":66}'

echo "Temperatura =40 Umidade =66";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":30,"umidade":77}'

echo "Temperatura =30 Umidade =77";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":20,"umidade":88}'

echo "Temperatura =20 Umidade =88";sleep 5;

mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/ebc035/attrs -m '{"temperatura":10,"umidade":99}'

echo "Temperatura =10 Umidade =99";sleep 5;

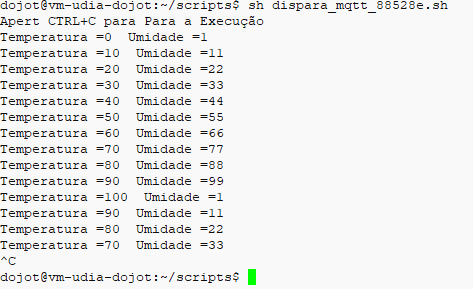
done

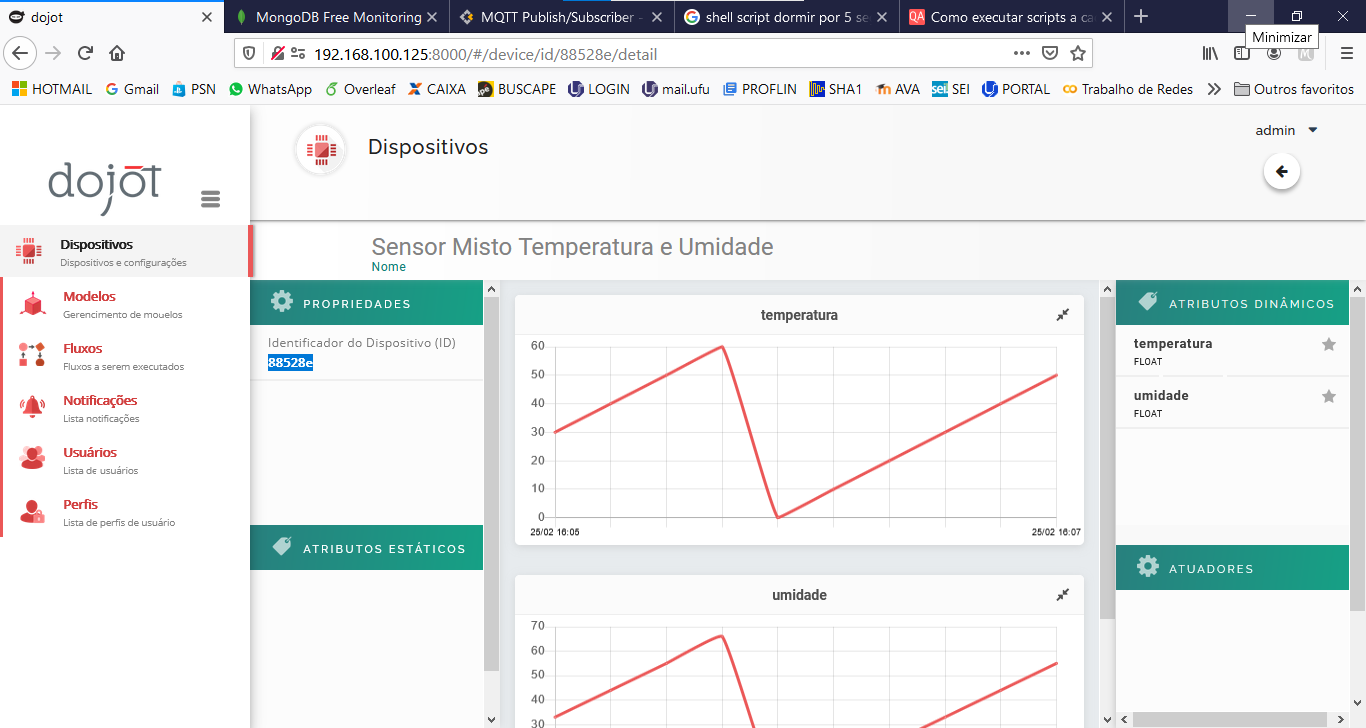
1. Concedendo permissão de execução para o script criado

**sudo chmod -x dispara\_mqtt\_88528e.sh**

1. Rodando o scritp

**sh** **dispara\_mqtt\_88528e.sh**

****



1. Verificando replicação de dados na VM-BRUMADINHO-MONGODB, no container mongodb6

**sudo docker exec -ti --user root 30601d0322e5 /bin/bash**

**mongo**

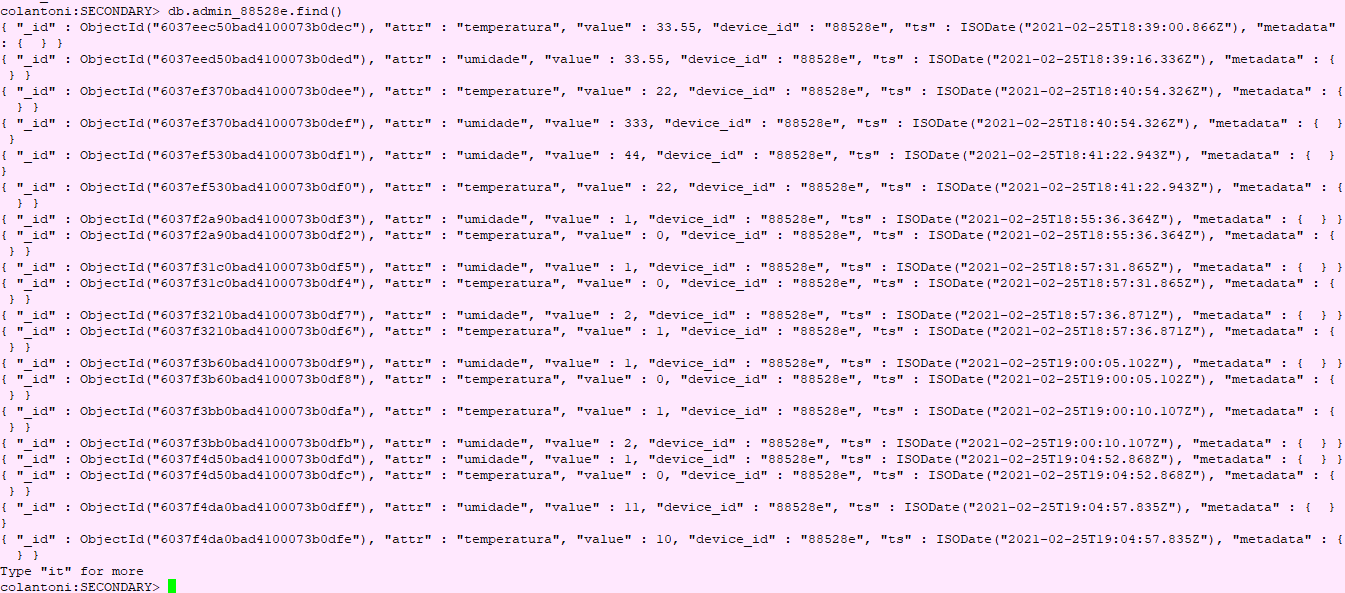
**rs.secondaryOk()**

**show dbs**

**use device\_history**

**show collections**

**db.admin\_88528e.find()**



1. Agora vou configurar uma ferramenta de monitoramento para armazenar as métricas do servidor e do cluster Mongodb, para iniciar a fase de testes.
2. Etapa
3. Etapa
4. Etapa
5. Etapa
6. Etapa
7. Etapa
8. Etapa
9. Etapa
10. Etapa
11. Etapa
12. Etapa
13. Etapa
14. Etapa
15. Etapa
16. Etapa
17. Etapa
18. Etapa
19. Etapa
20. Etapa
21. Etapa
22. Etapa
23. Etapa
24. Etapa
25. t