**Dojot 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.

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**

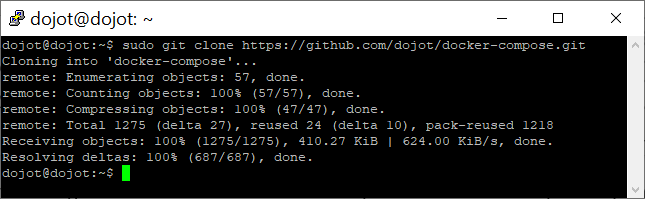
**curl -L https://github.com/docker/compose/releases/download/1.25.5/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose**

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

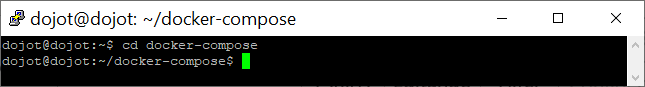
**docker-compose -v**

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

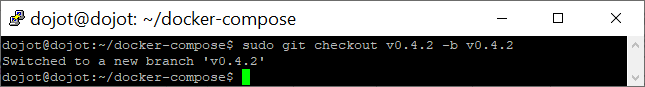
**1 - sudo git clone** [**https://github.com/dojot/docker-compose.git**](https://github.com/dojot/docker-compose.git)



**2 - cd docker-compose**

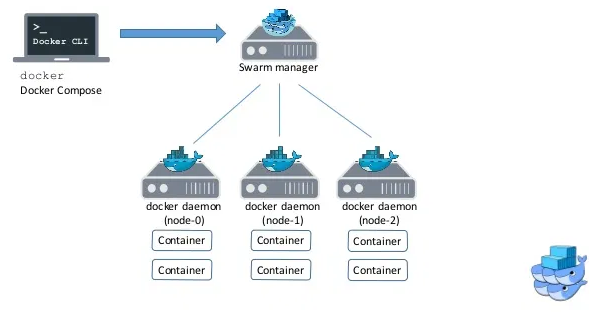


**3 - sudo git checkout v0.5.1 -b v0.5.1**



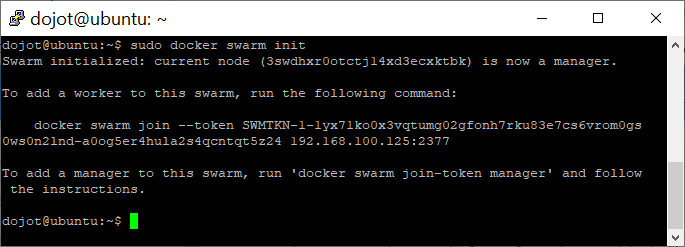
**4 - sudo docker-compose up -d (não rodei este comando ainda, pois não quero subir o dojot na forma padrão, a ideia é subi-lo na forma de serviços na rede swarn, utilizando o driver OVERLAY do Docker para criar um cluster ).**

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**

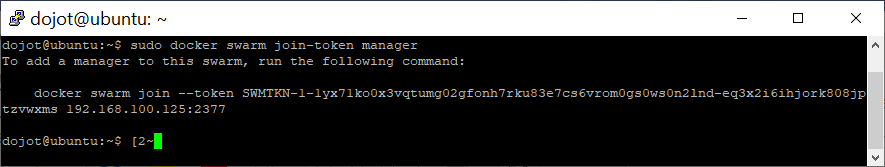


1. Em seguida é mostrado o token para unir/juntar novos nós (na forma de Works) ao swarm criado:

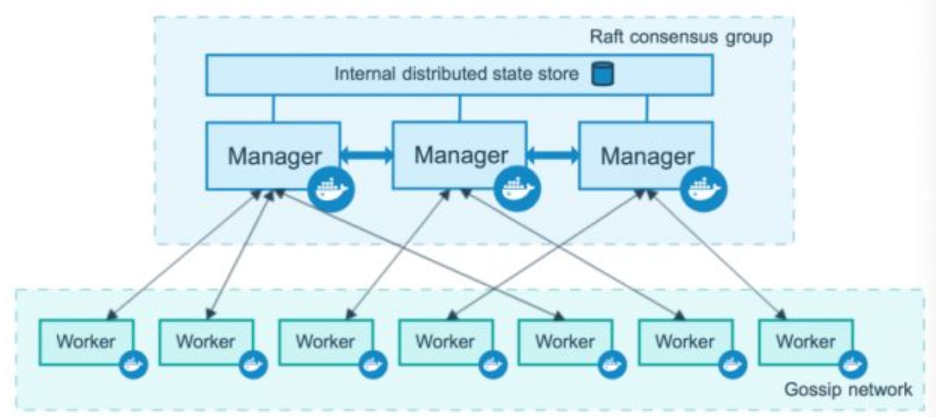
sudo docker swarm join --token SWMTKN-1-65vehcc8ahizg2tkhmlwvqno0ngnm913q0lueznc21ygnthct1-7gm90wy5m5um4bzep2h1048f1 192.168.100.125:2377

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**

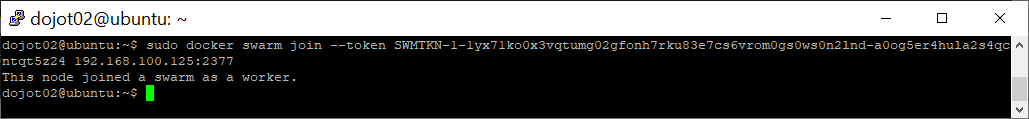


**sudo docker swarm join --token SWMTKN-1-65vehcc8ahizg2tkhmlwvqno0ngnm913q0lueznc21ygnthct1-6mduyzl06fhhqqx3l5z8uwp4q 192.168.100.125:2377**



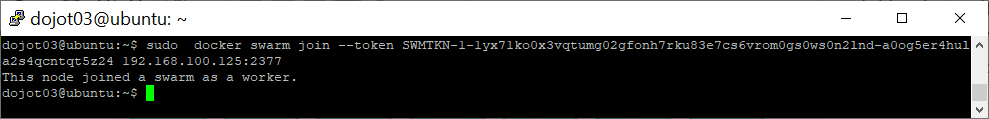
1. Agora vou adicionar outro nó ao cluster esse será adicionado como WORKDER , para isso criei uma nova VM chamada Dojot\_02, com o mesmo sistema operacional Ubuntu 16.0.4.6 live server amd64, e realizei a instalação do docker, para adicionar a VM-DOJOT\_02 como nó worker na rede utilizei o comando:

sudo docker swarm join --token SWMTKN-1-1yx71ko0x3vqtumg02gfonh7rku83e7cs6vrom0gs0ws0n2lnd-a0og5er4hula2s4qcntqt5z24 192.168.100.125:2377



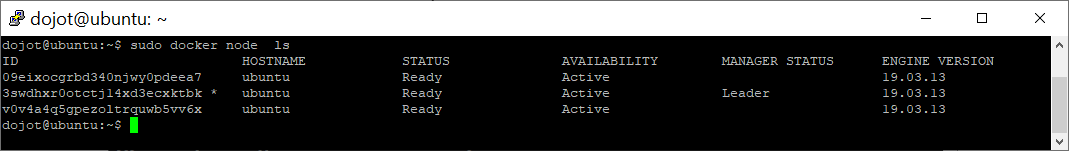
1. Agora vou adicionar outro nó ao cluster esse será adicionado também como WORKER , para isso criei uma nova VM chamada Dojot\_03, com o mesmo sistema operacional Ubuntu 16.0.4.6 live server amd64, e realizei a instalação do docker, para adicionar a VM-DOJOT\_03 como nó worker na rede utilizei

sudo docker swarm join --token SWMTKN-1-1yx71ko0x3vqtumg02gfonh7rku83e7cs6vrom0gs0ws0n2lnd-a0og5er4hula2s4qcntqt5z24 192.168.100.125:2377



1. Agora ao consultar na VM Dojot\_01 que é o Leader do cluster na rede swarm, é possível verificar que existem dois nós na rede:

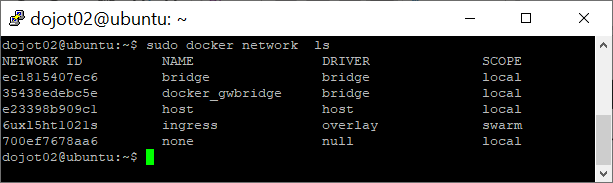
**sudo docker node ls**

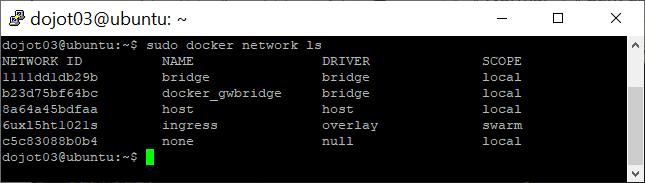


**OBS:** Um nó aparece como **LEADER** e os outros onde não há nada escrito, significa que ele é listado como worker.

1. Agora ao consultar as redes docker existentes na VM DOJOT\_02, DOJOT-03 é possível observar que já existe o driver **overlay** em uma rede chamada de **INGRESS**:

**sudo docker network ls**

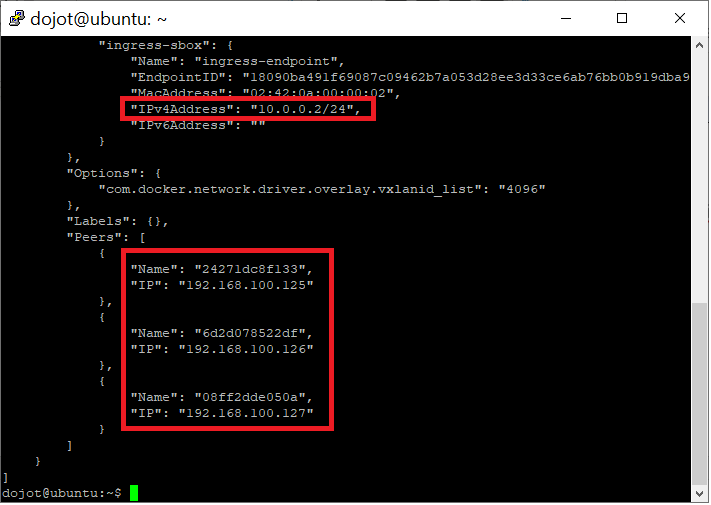




|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CLUSTER DOJOT** | | | | | | |
| **VM** | **S.O** | **CPU** | **RAM** | **HD** | **IP** | **Status** |
| Dojot\_01 | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 2048 MB | 20GB | 192.168.100.125 | Leader |
| Dojot\_02 | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 1024 MB | 20GB | 192.168.100.126 | worker |
| Dojot\_03 | Ubuntu server 16.04.6 live amd64 | 1 Core 1 Núcleo | 1024 MB | 20GB | 192.168.100.127 | worker |

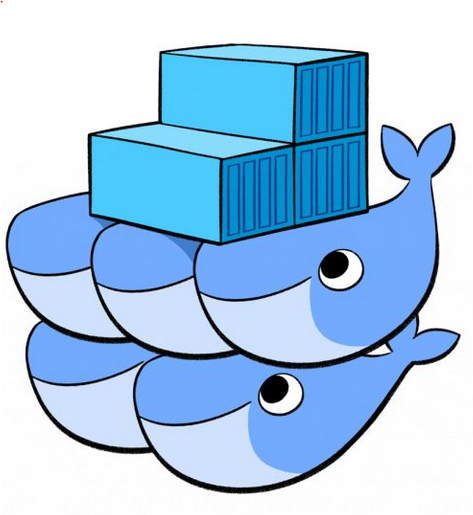
1. Agora meu cluster está com 3 nós, com as seguintes configurações:
2. Agora podemos consultar qual o IP da rede swarm, inespecionando a rede ingress que que foi criada com o driver OVERLAY:

**sudo docker inspect ingress**



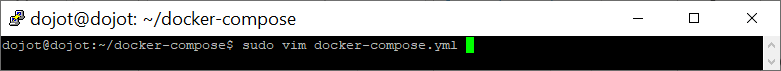
**OBS:** Podemos observar o IP de classe A 10.0.0.2/24 que é utilizado na rede swarm, com o driver overlay, e abaixo os endereços de IP dos hosts que compõem os nós da rede, sendo IP de cada uma das máquinas virtuais.

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 3.4, 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 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 3.7, e da última linha com o **NOME** que não é mais suportado nesta versão, abaixo todas as linhas que modifiquei:

**1 version: '3.7'**

**7 # depends\_on:**

**8 # - mongodb**

**20 # depends\_on:**

**21 # - mongodb**

**22 # - auth**

**23 # - kafka**

**24 # - data-broker**

**53 # depends\_on:**

**54 # - mosca-redis**

**55 # - kafka**

**56 # - data-broker**

**57 # - auth**

**58 # - ejbca**

**84 # depends\_on:**

**85 # - kafka**

**86 # - data-broker-redis**

**116 #depends\_on:**

**117 # - kafka**

**118 # - data-broker**

**119 # - auth**

**120 # - image-manager**

**140 #depends\_on:**

**141 # - postgres**

**142 # - minio**

**173 #depends\_on:**

**174 #- postgres**

**175 #- kafka**

**176 #- data-broker**

**177 #- device-manager-redis**

**194 # depends\_on:**

**195 # - apigw**

**196 # - postgres**

**197 # - auth-redis**

**198 #depends\_on:**

**199 # postgres:**

**200 # condition: service\_healthy**

**244 #depends\_on:**

**245 #postgres:**

**246 #condition: service\_healthy**

**254 # depends\_on:**

**255 # postgres:**

**256 # condition: service\_healthy**

**266 # depends\_on:**

**267 # postgres:**

**268 # condition: service\_healthy**

**269 # kong-migration:**

**270 # condition: service\_started**

**285 #depends\_on:**

**286 # - apigw**

**306 #depends\_on:**

**307 #- rabbitmq**

**308 #- kafka**

**309 #- mongodb**

**310 #- auth**

**311 #- flowbroker-context-manager**

**312 #- flowbroker-redis**

**328 #depends\_on:**

**329 #- zookeeper**

**352 #depends\_on:**

**353 # - zookeeper**

**380 # depends\_on:**

**381 # - flowbroker**

**382 # - device-manager**

**387 #depends\_on:**

**388 # - postgres**

**392 #depends\_on:**

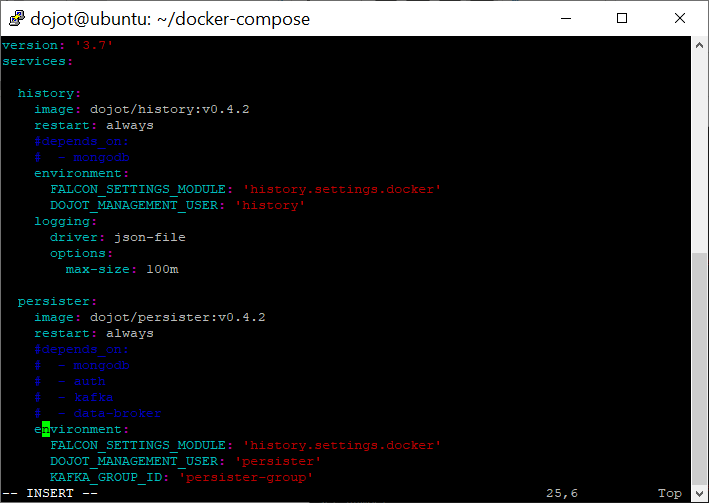
**393 # - kafka**

**394 # - data-broker**

**395 # - auth**

**396 # - mongodb**

**408 #name: ${FLOWBROKER\_NETWORK}**



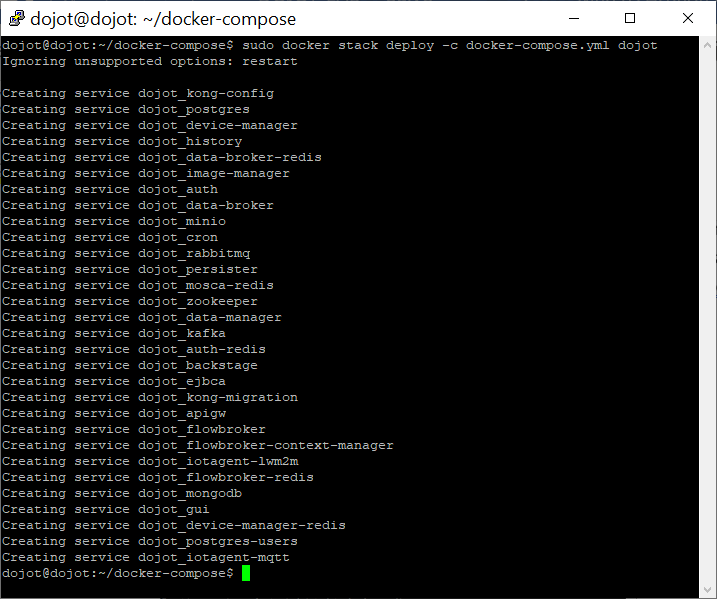
1. Para sair do arquivo e salvar:

**ESC**

**:wq!**

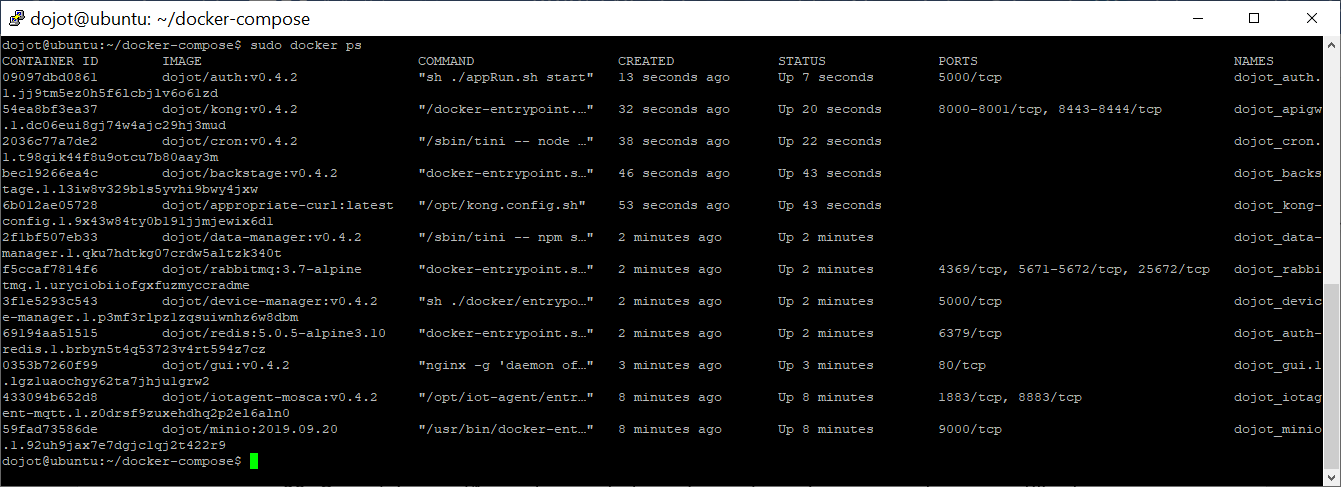
1. Agora vou rodar o arquivo de configuração YML do docker-compose em modo stack, assim os containers serão orquestrados na forma de serviços, assim será possível escala-los nos nós que compõem o cluster, e também será possível setar o numero de replicas. Para rodar o script como stack:

**sudo docker stack deploy -c docker-compose.yml dojot**



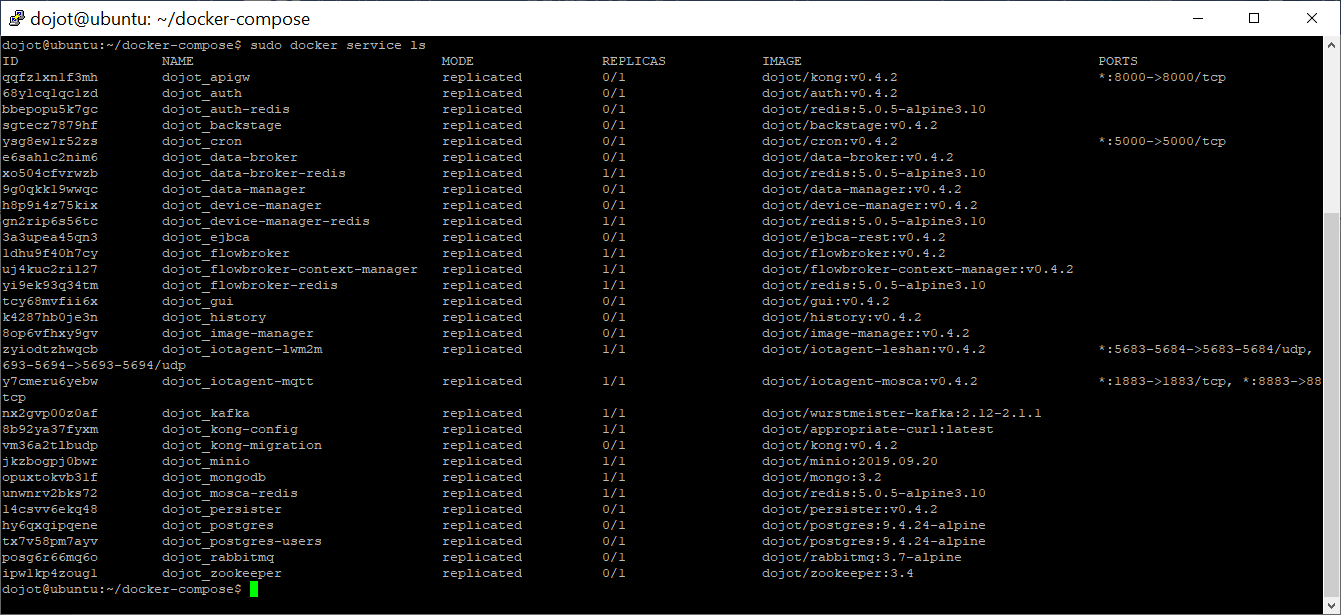
1. Da pra notar que todos os containers que compõem o DOJOT foram criados na forma de serviços com o nome que eu defino de **DOJOT\_CONTAINER**, agora ao consultar os container em execução, é possível visualizar que alguns containers inerentes ao DOJOT já estão em execução (é preciso esperar um tempo para visualização completa):

**sudo docker ps**



1. Os containers estão sendo executados na forma de serviços, para poderem ser utilizados pela rede swarm, e assim serem escaláveis em diversos nós que compõem o cluster, para visualizar a execução dos serviços:

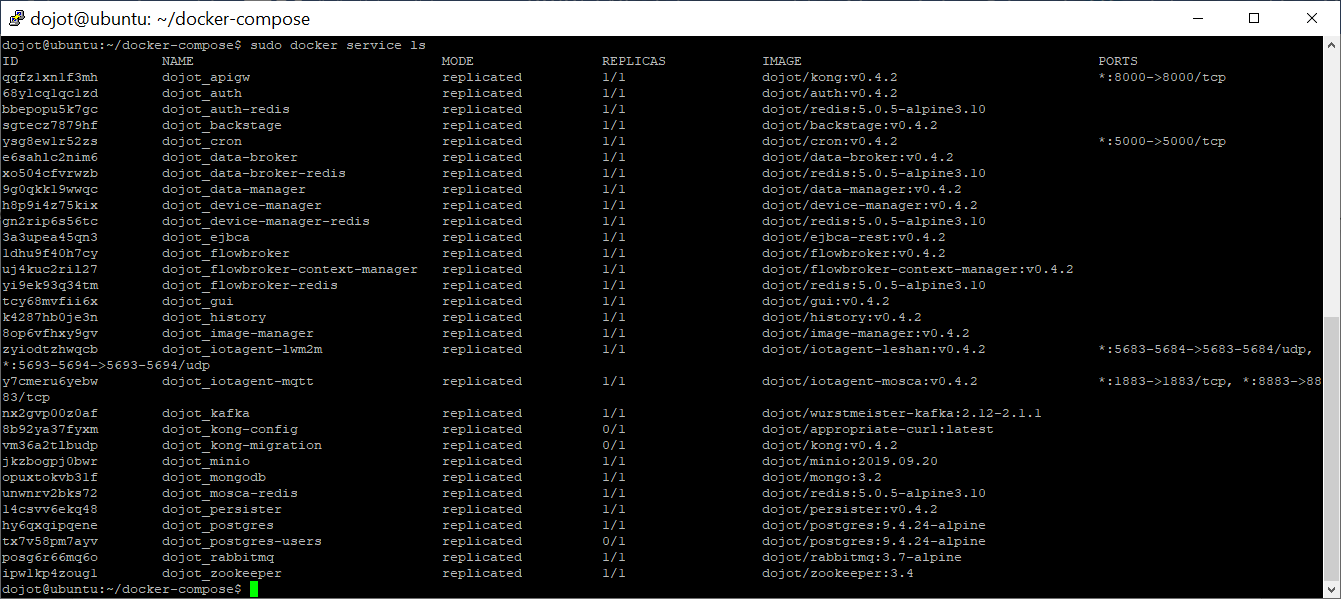
**sudo docker service ls**



**OBS:** No campo réplicas é possível notar que alguns serviços já constam como 1/1 que significa que existe um replica rodando do total de 1, e algumas como 0/1 o que significa que alguns containers ainda não subiram de um total de 1 replica, é preciso aguardar alguns minutos para que todos os containers tenham subido e constem como 1/1. Posteriormente irei aumentar o número de réplicas.

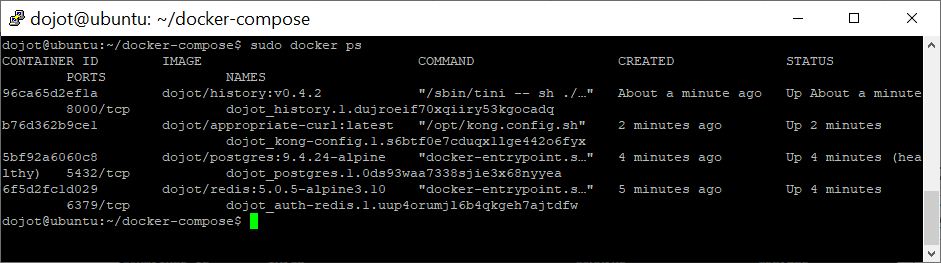
1. Após alguns minutos já possível visualizar que praticamente todos os serviços já estão em execução:

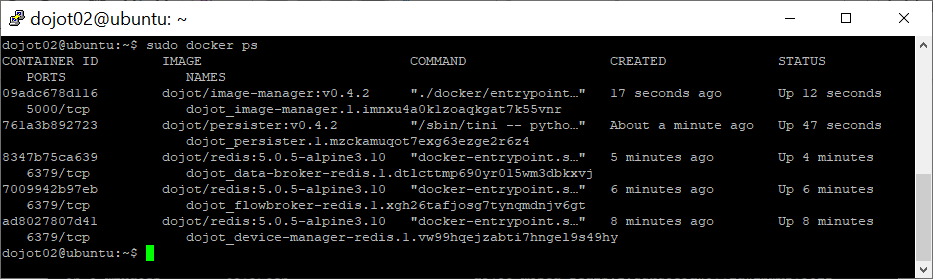
**sudo docker service ls**

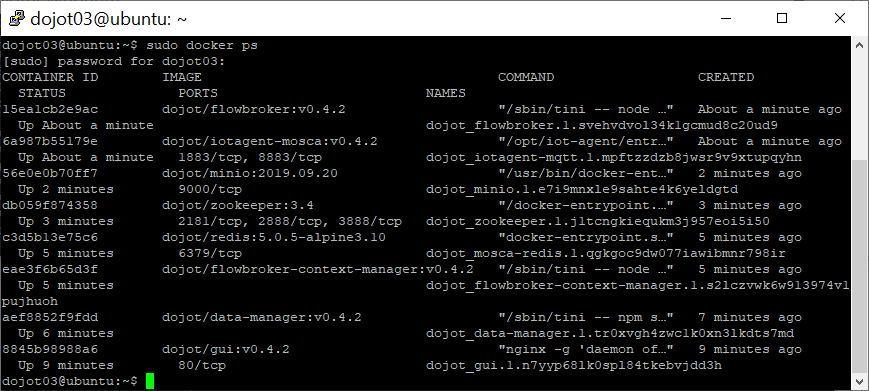


1. Agora ao consultar quais contêiner estão rodando em cada nó do cluster, podemos verificar que os container estão sendo alocados dinamicamente pela rede swarn, ficando a grande maioria no nó LEADER da rede:

**sudo docker ps**







1. Agora temos um cluster não so do banco de dados mongodb, mas de todos os serviços que compõem o DOJOT, com o swarn através da rede overlay o cluster se torna escalável a N hosts, sendo possível elevar o número de nós a qualquer momento, e ainda elevar o número de replicas dentro do cluster, neste exemplo deixei como 1/1 por uma questão de memoria RAM, pois em meus testes para suportar 2/2 precisaria dobrar a memória do nó LEADER , no caso a VM DOJOT01.
2. Agora já é possível acessar a interface Web pelo navegador na URL:

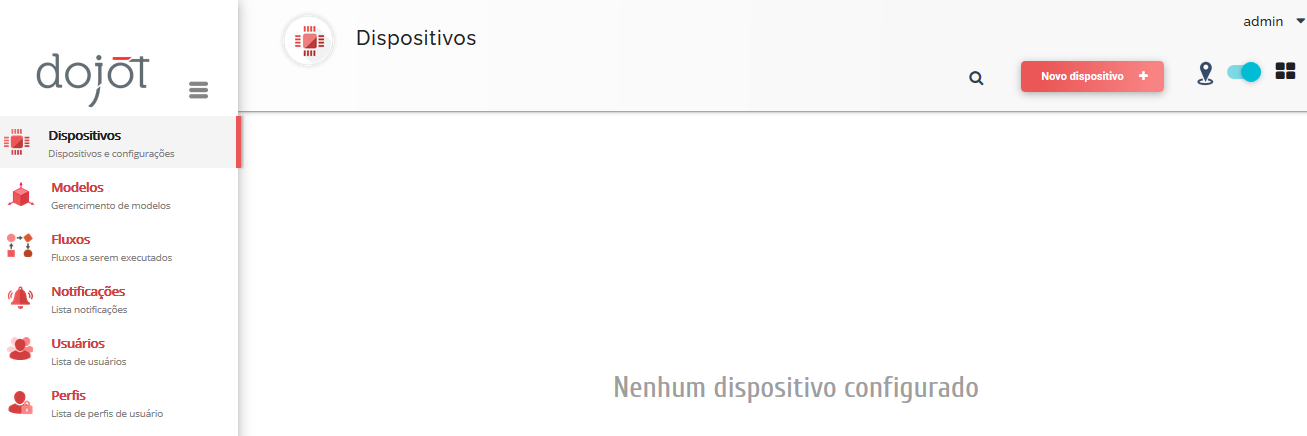
<http://192.168.100.125:8000/#/>

<Http://IP_do-servidor:8000/> (no meu caso 192.168.100.125)

1. 
2. As credenciais padrões de acesso são:

**Usuário:** admin

**Senha:** admin



1. Uma observação interessante é a redundância do cluster, permitindo que o acesso a plataforma do DOJOT seja feito por qualquer nó que componha a rede:

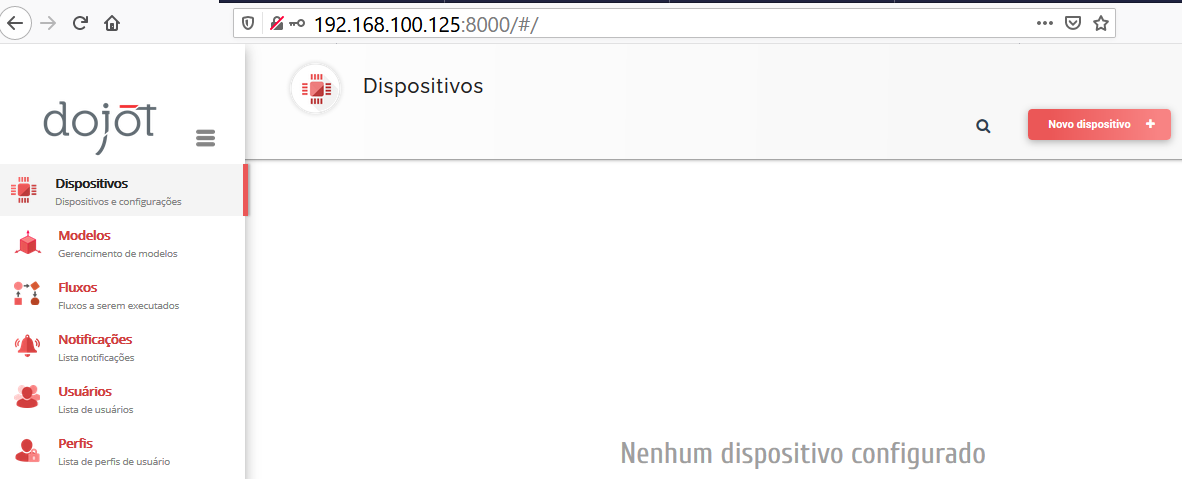
<http://192.168.100.126:8000/#/>

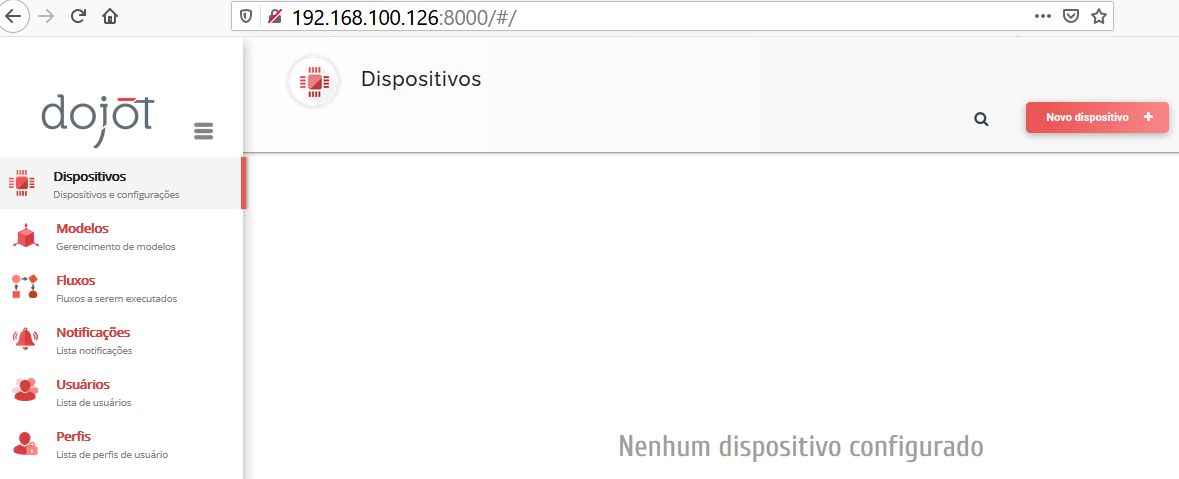


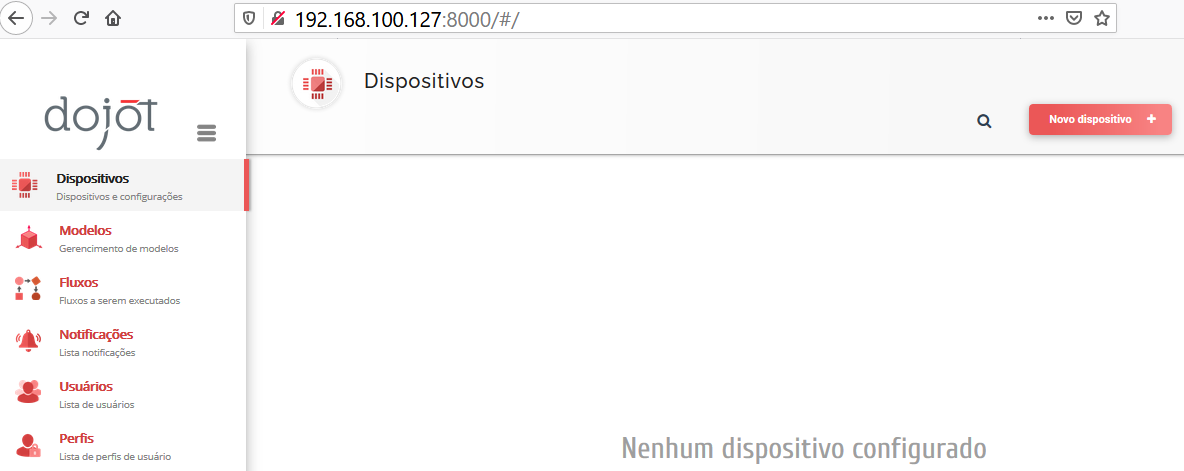
<http://192.168.100.127:8000/#/>



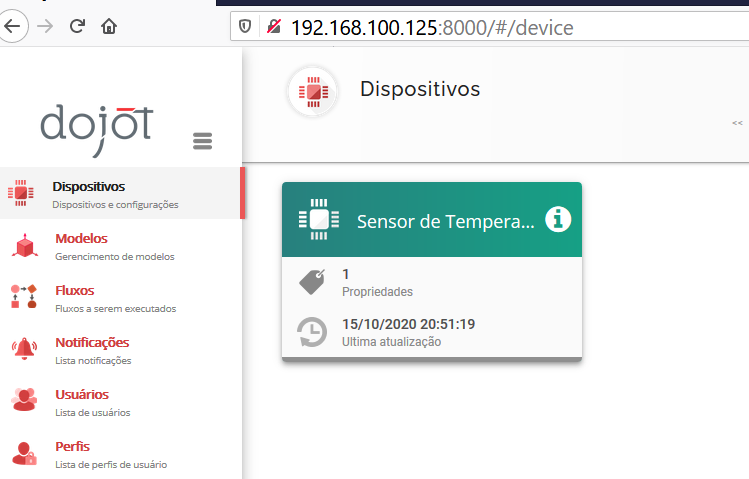
1. Faz sentido clusterizar somente o MongoDB quando posso clusterisar toda minha aplicação?
2. A continuação da pesquisa deveria seguir com foco em um DNS afim de trazer transparência do cluster ao usuário final?
3. Testes de funcionamento, primeiro vou logar no Dojot em cada um dos nós físicos:



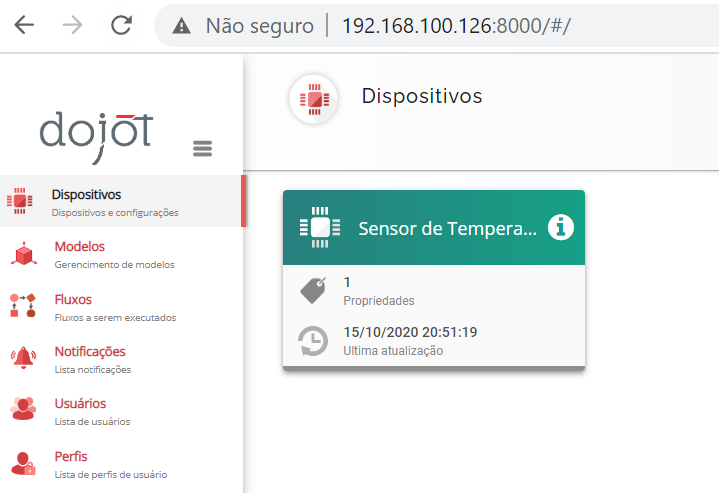




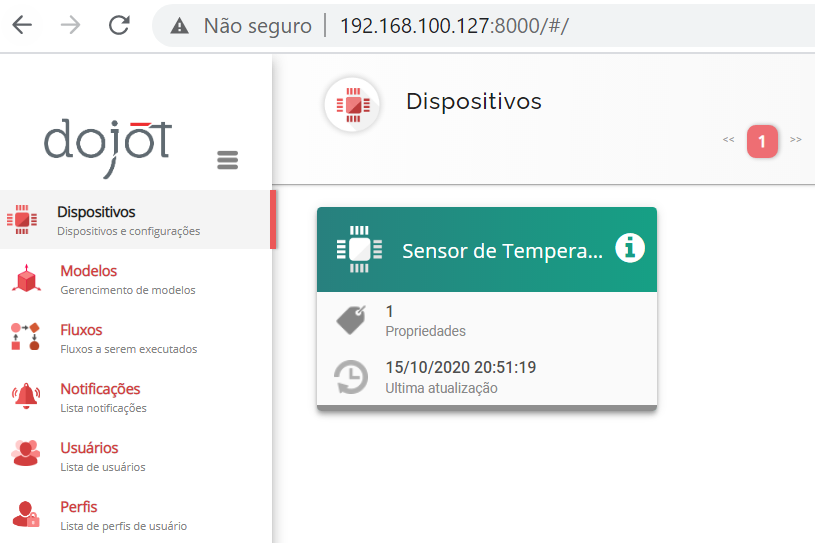
1. Agora acessando o primeiro nó do cluster no IP 192.168.100.125, através da interface web, vou criar um dispositivo chamado “Sensor de Temperatura”:



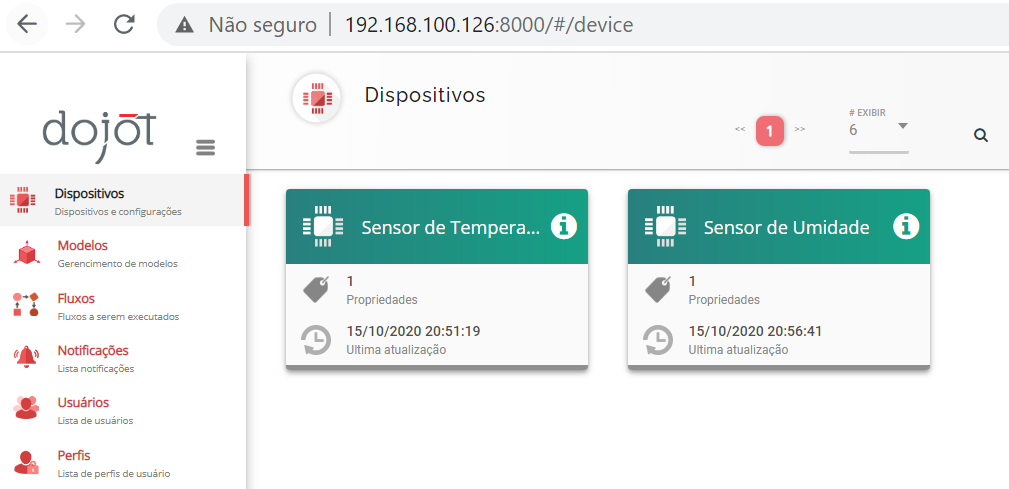
1. Agora ao acessar o cluster através do nó 2 , que esta na VM DOJOT02 , no IP 192.168.100.126 o dispositivo já se encontra criado:



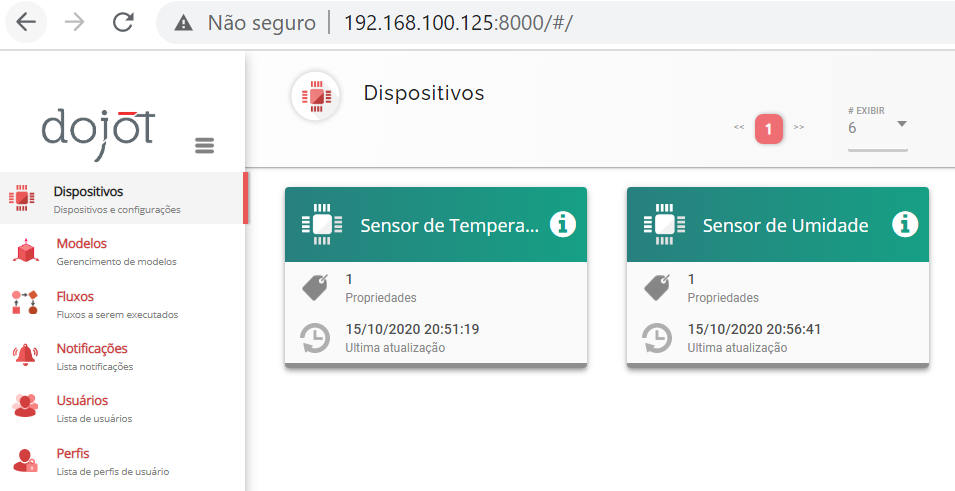
1. O mesmo pode ser observado no terceiro nó, na VM DOJOT03, IP 192.168.100.127:



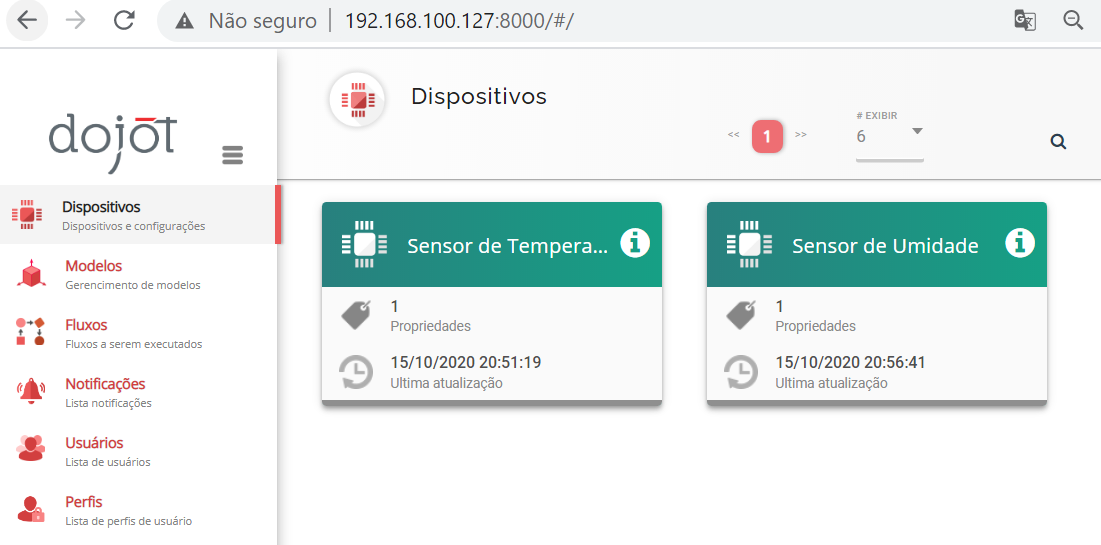
1. Agora vou criar um segundo disposto, porem agora através do nó 2, na VM DOJOT02, IP 192.168.100.126, chamado de “Sensor de Umidade”:



1. Agora vou acessar o nó 01, na VM DOJOT01 IP 192.168.100.125 e constatar que o novo dispositivo criado esta acessível:



1. O mesmo pode ser observado no terceiro nó do cluster, VM DOJOT03, IP 192.168.100.127:

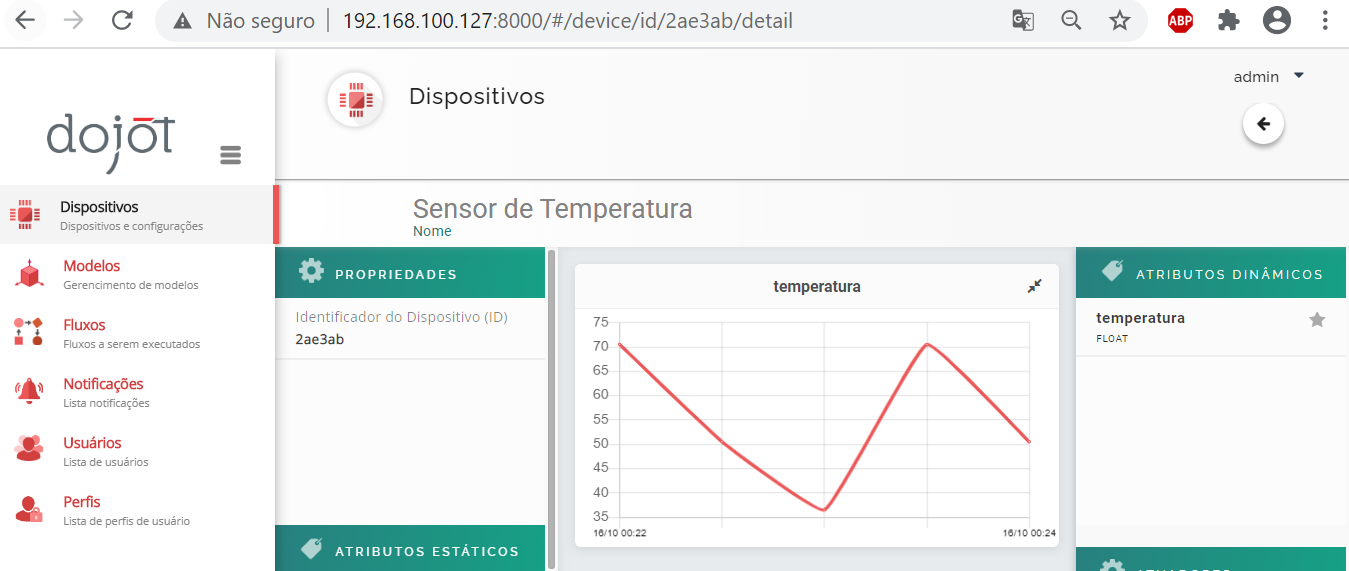


1. Como criei dispositivos virtuais, vou enviar valores de leitura através do **mosquito\_pub**, pela conexão SSH com servidor:

**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/2ae3ab/attrs -m '{"temperatura": 36.5}'**

**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/2ae3ab/attrs -m '{"temperatura": 70.5}'**

**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/2ae3ab/attrs -m '{"temperatura": 50.5}'**

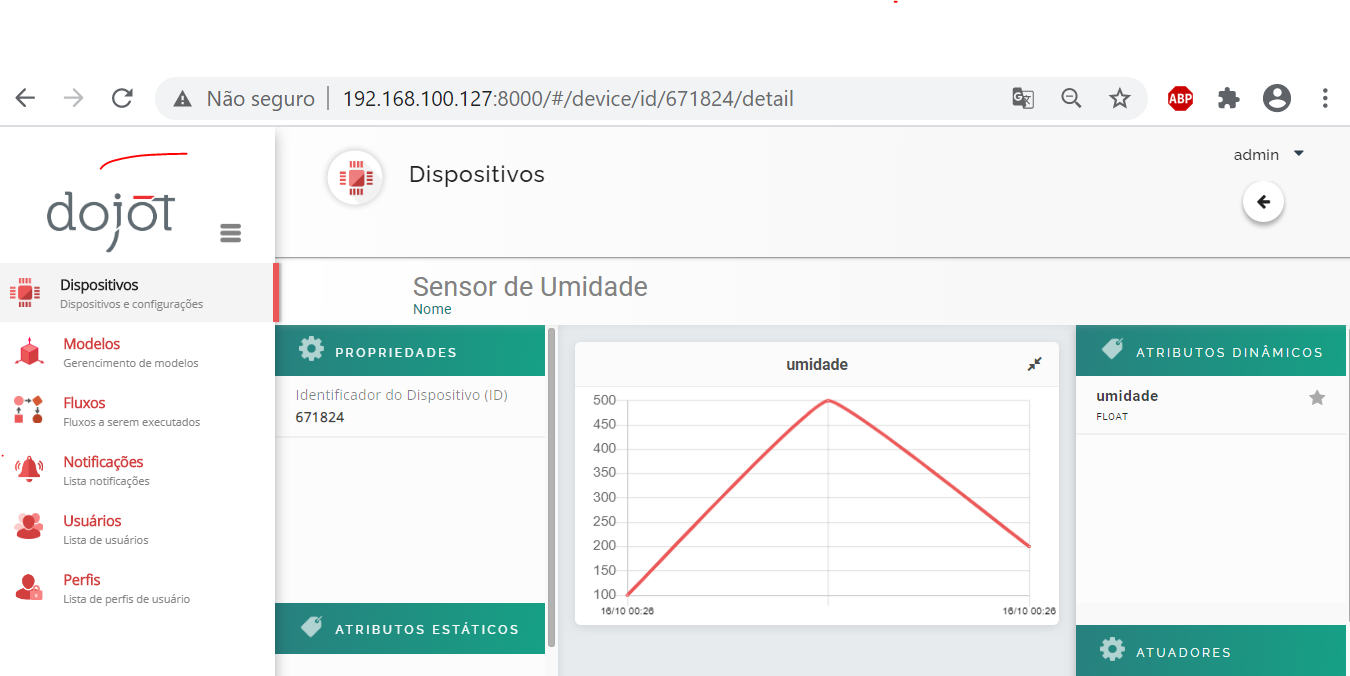


1. Agora vou enviar informações ao outro dispositivos que chamei de Sensor de Umidade:

**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/671824/attrs -m '{"umidade": 100}'**

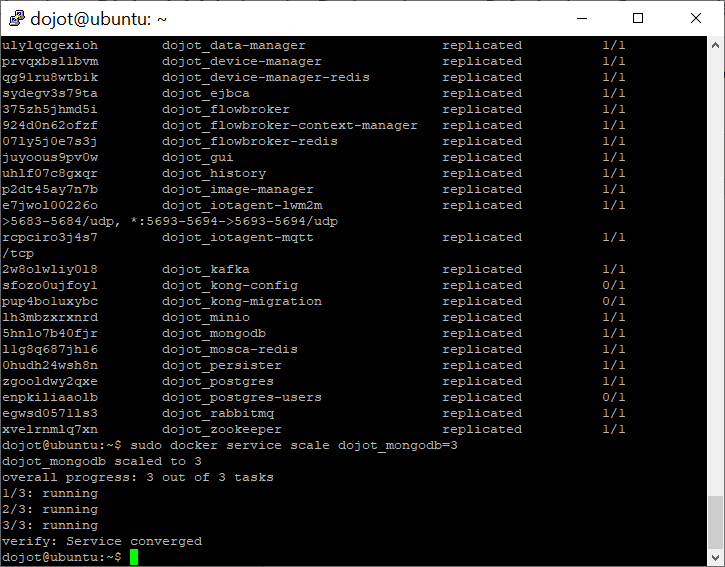
**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/671824/attrs -m '{"umidade": 500}'**

**sudo mosquitto\_pub -h 192.168.100.125 -p 1883 -t /admin/671824/attrs -m '{"umidade": 200}'**



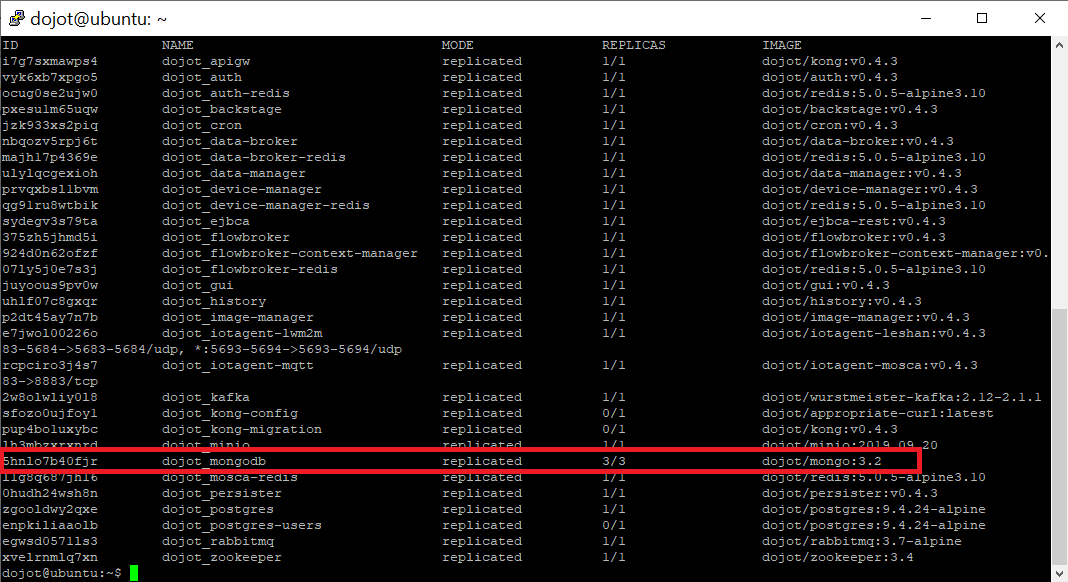
1. Aumentando numero de replicas do MongoDB

**sudo docker service scale dojot\_mongodb=3**



1. Agora ao consultar os serviços que estão rodando no docker swarm já possível verificar que existem três replicas do container do MongoDB:

**sudo docker service ls**



1. Agora
2. Teste
3. Teste
4. Teste
5. Tese
6. Teste
7. Tetes
8. Teste