

dojōt



do IoT

Dojot v0.7.1

Guia de Instalação
Instalando a dojot no Kubernetes

Última modificação: 06/05/2021

SUMÁRIO

Introdução	2
Ambiente	2
Considerações gerais	3
Requisitos de hardware e software	4
Download do repositório	4
Instalação do ansible	5
Configuração do cluster kubernetes	5
Deploy da dojot	10
Configuração do load balancer	14

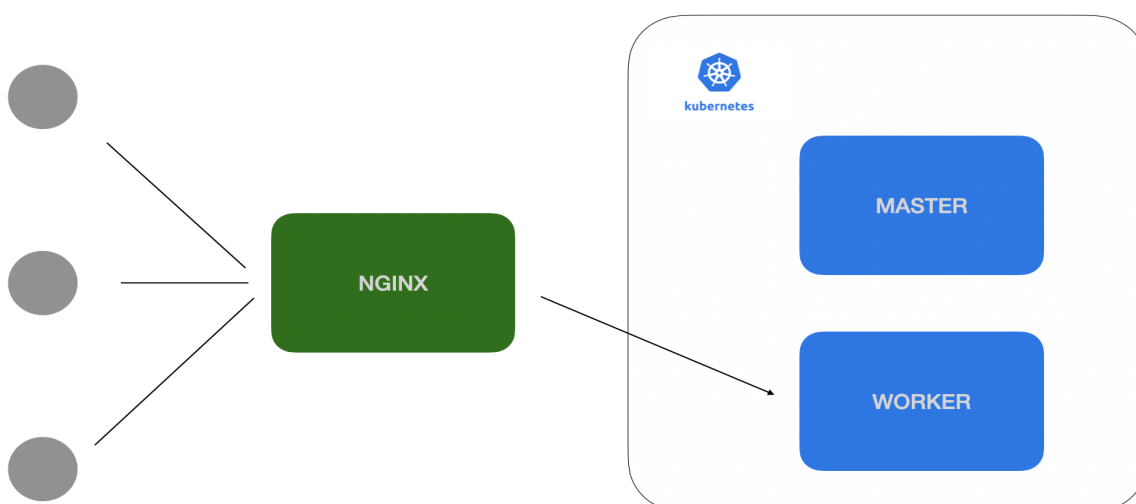
Introdução

Este documento aborda a configuração da dojot v0.7.1 no Kubernetes utilizando o NGINX como balanceador de carga Layer 4.

A partir de testes realizados pela equipe, podemos dizer que o ambiente a ser configurado com base nesse documento consegue suportar 500 dispositivos enviando mensagens de 100B para dojot a cada 15 segundos, onde todos os componentes da dojot são disponibilizados e as mensagens são processadas e armazenadas.

Ambiente

Para utilizar a dojot, podemos levar em conta o seguinte ambiente:



Na imagem acima vemos 3 nós, que podem ser máquinas virtuais ou físicas:

- **NGINX:** Load balancer layer 4. Todas as requisições TCP e UDP passam por ele e são redirecionadas para os workers do kubernetes. Nesse nó temos somente o NGINX instalado. Ele possui regras de balanceamento TCP e UDP e todos os dispositivos e clientes devem se conectar ao NGINX para acessar a dojot.
- **K8s Master:** Nó master do Kubernetes. Nele temos o *control plane* que administra o nosso cluster kubernetes. Não temos nenhum serviço da dojot sendo executado nesse nó. Sua única responsabilidade é administrar o cluster kubernetes .
- **K8s Worker:** Nó worker do kubernetes. Todos os serviços da dojot são executados nesse nó. Como veremos mais adiante, para um ambiente onde há uma carga maior de dispositivos, é necessária a utilização de mais nós no cluster k8s. Para uma maior disponibilidade, é recomendado adicionar mais workers ao cluster.

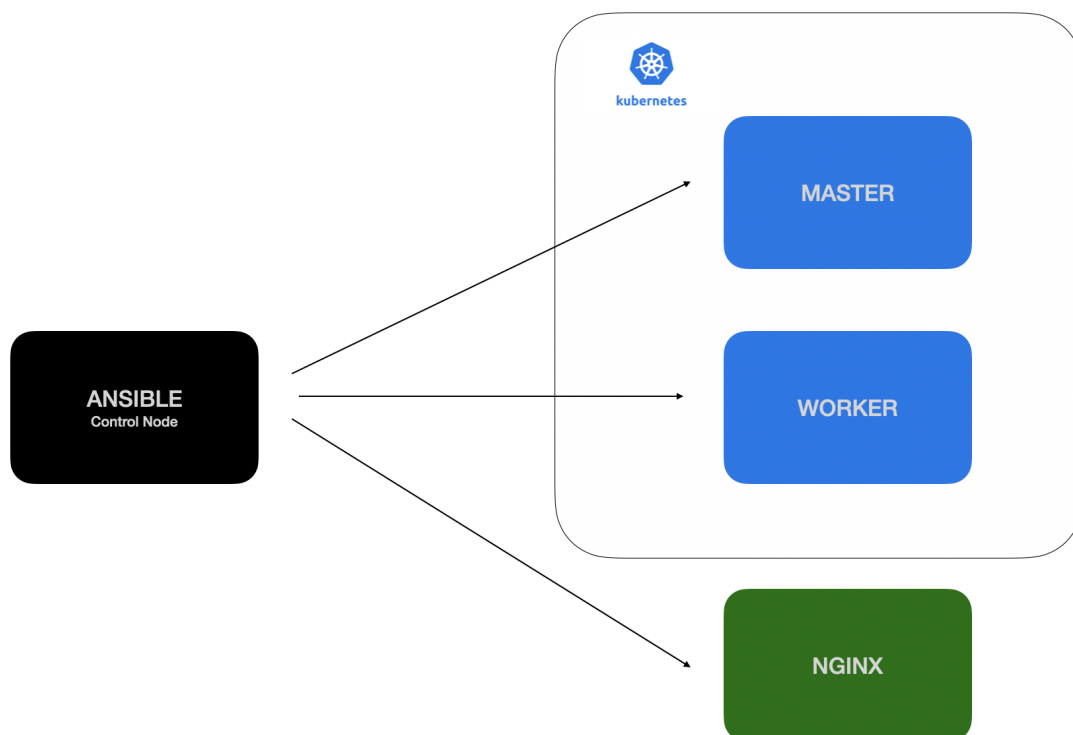
É importante lembrar que mesmo para um ambiente simples, é de extrema importância o mapeamento de volumes no cluster, para que em caso de falhas, o sistema consiga manter o estado e os dados dos contêineres.

Considerações gerais

Atualmente temos playbooks do Ansible para automatizar a configuração dos ambientes. Para isso, assumimos que os requisitos de hardware e software estão sendo atendidos. As máquinas precisam ser acessíveis por SSH (E é recomendado que todas elas possuam o mesmo usuário e senha) para que os comandos consigam ser executados pelo Ansible.

Mesmo sendo possível executar os playbooks de dentro do cluster k8s, vamos abordar nesse documento a execução de fora do cluster.

Abaixo um exemplo do ambiente com o Ansible:



Considerando o modelo acima, temos uma máquina com o Ansible instalado, chamada "control node", que podemos utilizar para realizar as instalações e configurações dos nós (*atualmente não é possível utilizar o MS Windows como control node*). O Ansible então, através dos playbooks criados pela equipe da dojot, poderá realizar a instalação e configuração do NGINX e do k8s, assim como o deploy completo da dojot.

Os playbooks para instalação se encontram no repositório [dojot/ansible-dojot](https://github.com/dojot/ansible-dojot). Esse repositório é versionado e temos os playbooks adequados para cada versão da dojot.

Requisitos de hardware e software

Todos os testes foram executados no Ubuntu 18.04 (bionic 64).

Software:

- Instalado previamente pelo usuário (Control Node):
 - Ubuntu 18.04 LTS
 - Ansible 2.9.6 (Recomendamos a instalação com pip3);
 - sshpass;
 - GIT.
- Instalado juntamente com o deploy da Dojot:
 - Kubernetes: Kubeadm 1.17;
 - Docker: Docker CE 19.03.4;
 - Balanceador de carga L4: NGINX Open Source 1.8.

Hardware:

- NGINX: 1 core de processamento (2.2GHZ), 1GB de RAM e 20GB de disco;
- Master Node: 2 cores de processamento (2.2GHZ), 2GB de RAM e 30GB de disco;
- Worker Node: 4 cores de processamento (2.2GHZ), 6GB de RAM e 40GB de disco;

Download do repositório

Para iniciar a instalação, precisamos baixar o repositório com os playbooks do Ansible. O seguinte comando é executado no control node do ansible para clonar o repositório "ansible-dojot":

```
$ git clone https://github.com/dojot/ansible-dojot.git
```

Será criado um diretório com o nome "ansible-dojot". Devemos então acessar esse diretório e mudar para a tag da versão da dojot que queremos utilizar. Lembrando que a branch master sempre aponta para a última tag (versão) da dojot:

```
$ cd ansible-dojot  
$ git checkout v0.7.1 -b v0.7.1
```

Nesse exemplo, mudamos a tag para a versão v0.7.1 da dojot. Todos os playbooks e scripts nessa tag estão adaptados para essa versão específica da dojot, ou seja, para cada versão temos uma tag específica.

Instalação do ansible

Ainda dentro do control node, precisamos agora instalar o Ansible para que possamos executar futuramente os playbooks da dojot. Para isso basta rodar o seguinte comando dentro do diretório “ansible-dojot” (lembrando que é necessário ter o pip3 instalado previamente):

```
$ pip3 install -r requirements.txt
```

Configuração do cluster kubernetes

Se já temos um cluster kubernetes na versão 1.19 podemos pular essa etapa (Para verificar a versão do k8s basta rodar o comando “*kubectl get nodes*”), do contrário, precisamos criar um. Para isso, precisamos de no mínimo 2 máquinas físicas ou virtuais com os requisitos de hardware descritos acima. Através do control node, executamos o playbook do Ansible para a instalação e configuração do kubernetes, que será utilizado para orquestrar os containers docker da dojot.

É interessante alterar o hostname das máquinas que farão parte do cluster kubernetes para facilitar a visualização dos nós quando necessário. No Ubuntu 18.04 utilizamos o comando “hostnamectl”. Para alterar o hostname da máquina que será o master node do kubernetes utilizamos o seguinte comando (dentro do master node):

```
$ sudo hostnamectl set-hostname k8s-master
```

E para alterar o hostname da máquina que será o worker node usamos (dentro do worker node):

```
$ sudo hostnamectl set-hostname k8s-worker
```

Para executar o playbook do Ansible com sucesso, precisamos informar os hosts onde o k8s será instalado e configurado.

Configuramos isso no arquivo “*inventories/example_local/hosts.yaml*” representado abaixo:

```
---
all:
  hosts:
    ...
  localhost:
    ansible_connection: local
    ansible_python.version.major: 3
  worker_host:
    ansible_host: WORKER_HOST_ADDRESS #change to the host of worker node on k8s
  children:
    ...
```

```
k8s-nodes:
  children:
    master_nodes:
      hosts:
        localhost:
    worker_nodes:
      hosts:
        worker_host:
...

```

Ocultamos o restante do conteúdo do arquivo com "... " para dar mais foco no que vamos utilizar agora.

Nesse arquivo, configuramos os hosts que receberão a instalação do k8s. Como exemplo, consideramos que o master node tem o ip 192.168.0.10 e o worker node tem o ip 192.168.0.11. Por padrão, o arquivo hosts.yaml aponta o master para localhost, então devemos adicionar um "master_host" e apontá-lo para o IP da máquina que terá o papel de master do cluster. Dessa forma, o arquivo ficaria assim (As partes destacadas em negrito são as que devem ser alteradas):

```
---
all:
  hosts:
    ...
    localhost:
      ansible_connection: local
      ansible_python.version.major: 3
      master_host:
      ansible_host: 192.168.0.10
      worker_host:
      ansible_host: 192.168.0.11
  children:
    ...
    k8s-nodes:
      children:
        master_nodes:
          hosts:
            master_host:
        worker_nodes:
          hosts:
            worker_host:
...

```

Podemos também ter mais de um worker node, como no exemplo abaixo:

```
---
all:
  hosts:
    ...
    master_host:
      ansible_host: 192.168.0.10
    worker_host:

```

```

    ansible_host: 192.168.0.11
  worker_host_two:
    ansible_host: 192.168.0.12
children:
  ...
  k8s-nodes:
    children:
      master_nodes:
        hosts:
          master_host:
      worker_nodes:
        hosts:
          worker_host:
          worker_host_two:
  ...

```

É importante saber que o playbook do Ansible para instalação e configuração do kubernetes utiliza os grupos "k8s-nodes", "master_nodes" e "worker_nodes", como destacado abaixo. Sendo assim, todos os hosts abaixo desses grupos serão afetados e adicionados ao cluster. Os hosts que estão abaixo de "master_nodes" serão configurados como master node no kubernetes, e os hosts abaixo do grupo "worker_nodes" serão configurados como worker nodes.

```

...
  k8s-nodes:
    children:
      master_nodes:
        hosts:
          master_host:
      worker_nodes:
        hosts:
          worker_host:
          worker_host_two:
  ...

```

Com o arquivo "hosts.yaml" devidamente editado, podemos executar o playbook do ansible para configurar o cluster k8s:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local k8s.yaml
```

No exemplo acima, o Ansible irá conectar nas máquinas remotamente via SSH utilizando o usuário "dojot". O Ansible também vai perguntar o password do usuário "dojot" no terminal. Outras abordagens podem ser utilizadas para melhorar a segurança.

O Ansible mostrará no log tudo que está executando e alterando. Todos os comandos na cor "amarela" significam que algo foi alterado na máquina remota. A cor verde significa que nada foi alterado e a cor vermelha significa erro. Não devemos utilizar o cluster caso algum erro seja acusado no log.

No final da execução do playbook, podemos acessar o nó master do kubernetes e executar o comando `kubectl get nodes`. Se a saída for parecida com essa, significa que o cluster foi criado corretamente (Pode ser que o worker node leve um tempo até ficar como o status "Ready"):

<i>NAME</i>	<i>STATUS</i>	<i>ROLES</i>	<i>AGE</i>	<i>VERSION</i>
<i>k8s-master</i>	<i>Ready</i>	<i>master</i>	<i>0d</i>	<i>v1.19.8</i>
<i>k8s-worker</i>	<i>Ready</i>	<i><none></i>	<i>0d</i>	<i>v1.19.8</i>

Lembrando que o "NAME" do nó no kubernetes é o hostname da máquina onde o kubernetes foi instalado.

Agora, com nosso cluster configurado, precisamos criar e mapear os volumes da dojot para que os dados sejam persistidos e também para que seja possível restaurar o estado da aplicação, caso algum container ou algum nó do cluster apresentar algum problema.

Existem diversas maneiras e abordagens para configurar volumes no kubernetes. Nesse documento, vamos abordar a criação de volumes locais.

Para que os volumes funcionem corretamente, precisamos primeiramente colocar alguns labels no nosso worker node. Para isso, devemos configurar o arquivo `inventories/example_local/group_vars/all/labels.yaml` trocando as variáveis `"k8s-worker-nodename"` pelo nome do nosso worker node (Nome que aparece quando você roda o comando `"kubectl get nodes"`). Dessa forma o arquivo fica da seguinte maneira:

```
...
dojot_dojot_nodenames:
  - k8s-worker

dojot_kafka_nodenames:
  - k8s-worker

dojot_x509_nodenames:
  - k8s-worker

dojot_vernemq_nodenames:
  - k8s-worker
...
```

Com o arquivo "labels.yaml" devidamente editado, podemos executar o playbook do ansible para configurar os labels:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local label.yaml
```

Após a execução do playbook, podemos rodar o comando `"kubectl get nodes --show-labels"` para ver se deu tudo certo. O campo "labels" do worker node deve ter o seguinte conteúdo:

```
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,dojot.components/dojot=enabled,dojot.components/kafka=enabled,dojot.components/vernemq=enabled,dojot.components/x509=enabled,kubernetes.io/arch=amd64,kubernetes.io/hostname=eri-worker,kubernetes.io/os=linux
```

Com os labels devidamente configurados, podemos prosseguir com a criação dos volumes. Os volumes locais utilizam o disco dos nodes do cluster como storage, sendo assim, nosso playbook irá criar diretórios no cluster para que os volumes sejam mapeados corretamente. Por padrão os diretórios serão criados no seguintes caminhos:

Node	Diretório
worker	/mnt/data/kong
worker	/mnt/data/minio
worker	/mnt/data/influxdb
worker	/mnt/data/mongodb
worker	/mnt/data/postgres
worker	/mnt/data/prometheus
worker	/mnt/data/kafka_ws
worker	/mnt/data/kafka
worker	/mnt/data/zookeeper/data
worker	/mnt/data/zookeeper/log
worker	/mnt/data/ejbca
worker	/mnt/data/certificate_acl

Precisamos agora fazer algumas alterações no arquivo de configuração de volumes “inventories/example_local/group_vars/all/volumes.yaml”, onde iremos alterar a variável “dojot_storage_class_name” para que tudo funcione corretamente. Dessa forma o arquivo ficaria da seguinte maneira:

```
...
dojot_storage_class_name: "local-storage"

dojot_volume_directory: "/mnt/data"
...
```

Se você desejar alterar o caminho dos diretórios dos volumes, você pode alterar a variável “dojot_volume_directory” apontando para o caminho desejado como no exemplo abaixo:

```
...
dojot_volume_directory: "/home/dojot"
...
```

Como nessa abordagem estamos utilizando apenas um worker node, os diretórios dos volumes serão todos criados dentro dele. Se você desejar isolar o volume de um serviço em um node específico, basta editar o arquivo “inventories/example_local/hosts.yaml” informando os IP’s abaixo do grupo “volume-nodes” como no exemplo abaixo, onde os volumes do kafka e x509 serão criados em worker nodes diferentes:

```

...
volume-nodes:
  children:
    dojot_nodes:
      hosts:
        worker_host:
    kafka_nodes:
      hosts:
        worker_host_two:
    x509_nodes:
      hosts:
        worker_host_three:
...

```

Com todos os arquivos devidamente configurados, podemos executar o playbook do ansible para criar os volumes:

```

$ ansible-playbook -K -k -u dojot -i inventories/example_local volume.yaml

```

Para visualizar o status dos volumes podemos utilizar o comando "kubectl get pv" dentro do master node. A seguinte lista será exibida:

```

cpqd@eri-master:~$ kubectl get pv

```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
local-pv-certificate-acl	5Mi	ROX	Retain	Bound	dojot/certificate-acl	local-storage		3d20h
local-pv-influxdb	2Gi	RWO	Retain	Available		local-storage		3d20h
local-pv-kafka	2Gi	RWO	Retain	Bound	dojot/kafka-volume-kafka-server-0	local-storage		3d20h
local-pv-kafka-ws	5Mi	ROX	Retain	Bound	dojot/kafka-ws	local-storage		3d20h
local-pv-kong	5Mi	ROX	Retain	Bound	dojot/kong	local-storage		3d20h
local-pv-minio	2Gi	RWO	Retain	Bound	dojot/minio-volume-minio-0	local-storage		3d20h
local-pv-mongo	2Gi	RWO	Retain	Bound	dojot/mongodb-volume-mongodb-0	local-storage		3d20h
local-pv-postgres	2Gi	RWO	Retain	Bound	dojot/psql-volume-postgres-0	local-storage		3d20h
local-pv-prometheus	2Gi	RWO	Retain	Available		local-storage		3d20h
local-pv-x509-ejbca	10Mi	RWX	Retain	Bound	dojot/x509-ejbca	local-storage		3d20h
local-pv-x509-identity-mgmt	10Mi	RWX	Retain	Bound	dojot/x509-identity-mgmt	local-storage		3d20h
local-pv-zk-data	1Gi	RWO	Retain	Bound	dojot/zk-volume-data-zookeeper-0	local-storage		3d20h
local-pv-zk-log	1Gi	RWO	Retain	Bound	dojot/zk-volume-log-zookeeper-0	local-storage		3d20h

É de extrema importância que os volumes sejam criados corretamente, pois sem eles não será possível fazer deploy da dojot.

Deploy da dojot

Para fazer deploy da dojot, precisamos primeiro configurar as variáveis que o playbook irá usar. Nosso playbook toma decisões de configuração com base em variáveis. O arquivo de variáveis fica em `inventories/example_local/group_vars/all/dojot.yaml`. Precisamos então alterar as variáveis destacadas em negrito da seguinte maneira:

```

dojot_namespace: dojot
dojot_version: v0.7.1
dojot_domain_name: 102.168.0.12

```

```

dojot_kubernetes_rbac: true
dojot_gui2_enabled: false
dojot_auth_email_enabled: false
dojot_insecure_mqtt: 'true'
dojot_venremq_replicas: 1
dojot_bridges_replicas: 1

```

```
dojot_x509_identity_mgmt_replicas: 1
dojot_kafka_ws_enable_tls: false
dojot_apigw_enable_mutual_tls: false
```

dojot_fixed_nodeports_enabled: true

dojot_nodeports:

apigw:

http: 30001

https: 30002

metrics: 30003

mqtts: 30004

lwm2m:

coap: 30006

coaps: 30007

file_server: 30008

file_servers: 30009

http: 30010

dojot_enable_node_affinity: false

dojot_enable_locust_exporter: false

dojot_locust_exporter:

ip: 127.0.0.1

port: 9646

Vamos agora detalhar as alterações explicando o papel de cada variável no deployment:

- **dojot_domain_name:** utilizada por alguns serviços internos e principalmente pelo serviço que gera certificados para os dispositivos. Vamos subir um balanceador de carga para receber as conexões dos dispositivos, então nessa variável precisa ter o endereço do host do balanceador de carga (Nginx), ou o domínio que aponta para o host do balanceador.
- **dojot_fixed_nodeports_enabled:** define se os serviços da dojot com NodePort utilizam portas fixas. Para esse ambiente precisamos que as portas sejam fixas, pois precisamos utilizar um balanceador de carga que balanceia as requisições nessas portas. É possível modificar o número dessas portas através da variável “*dojot_nodeports*”.

Também precisamos alterar o arquivo `hosts.yaml` que fica em `"inventories/example_local/hosts.yaml"`, caso ainda não tenha sido alterado. Nesse arquivo temos a configuração do host onde o playbook de deploy será executado. O arquivo deve ficar da seguinte forma:

```
---
```

```
all:
```

```
hosts:
```

```
...
```

```
  master_host:
```

```
    ansible_host: 192.168.0.10
```

```
children:
```

```
...
dojot-k8s:
  hosts:
    master_host:
...

```

Precisamos alterar o grupo "dojot-k8s" para que o playbook seja executado no nó master do nosso cluster kubernetes. Outra estratégia seria utilizar um arquivo de configuração local do kubernetes, mas não vamos abordar esse assunto nesse documento.

Um passo opcional por agora seria escolher quais bancos de dados e quais lot agents serão utilizados pela dojot. Para isso, devemos editar o arquivo que fica em *inventories/example_local/group_vars/all/services.yaml*:

```
---
optional:
  #InfluxDB Services
  influxdb: false
  influxdb_storer: false
  influxdb_retriever: false

  #MongoDB Services
  history: true
  persister: true

  #lotAgents
  vernemq: true
  mosca: false
  lwm2m: false

```

Para escolher quais bancos usar, basta alterar o valor da respectiva variável de "false" para "true" e vice versa. Lembrando que os serviços "influxdb_storer" e "influxdb_retriever" dependem do serviço "influxdb" estar com o valor "true" para funcionarem.

Com relação aos lot agents a abordagem é a mesma. A única restrição deles é que não é possível usar o "vernemq" junto com o "mosca". Se essas duas variáveis estiverem como "true" o "vernemq" terá prioridade sobre o "mosca".

Com nosso arquivo de variáveis e nosso arquivo de inventário configurados, podemos agora executar o playbook de deployment da dojot utilizando o seguinte comando:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local deploy.yaml
```

Lembrando que o parâmetro "-u" informa o usuário do host master.

Ao executar o playbook, o Ansible fará o deploy da dojot em nosso cluster. Os logs informam o status de cada task e no final da execução todos os microsserviços da dojot estarão disponíveis no cluster.

Apesar de os serviços estarem disponíveis, precisamos monitorar o status de cada um. O deployment foi realizado, mas os microsserviços possuem dependências entre eles e precisam subir os serviços dentro do container. Podemos acessar o nó master do kubernetes por ssh e executar o seguinte comando para acompanhar o status dos serviços:

```
$ ssh dojot@192.168.0.10  
$ watch kubectl get pods -n dojot
```

No exemplo acima, acessamos o nó master do kubernetes por ssh e executamos o comando para listar todos os pods da dojot. Para sair do modo "watch" é só usar Ctrl+c.

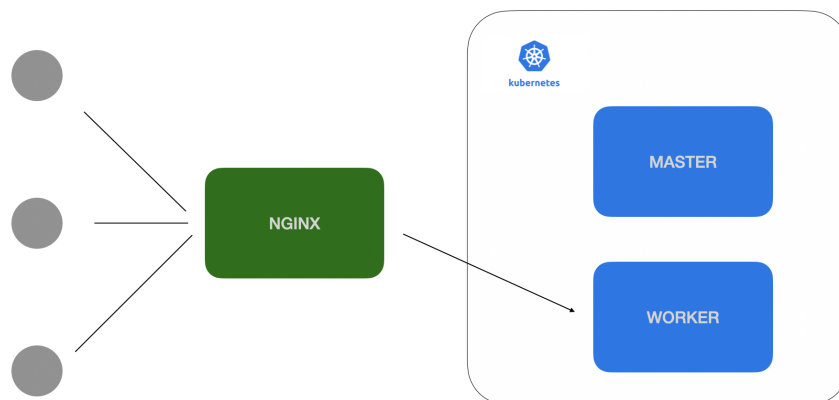
A saída desse comando é uma lista com todos os microsserviços da dojot e o status de cada um:

```
cpqd@eri-master:~$ kubectl get pods  
NAME                                READY   STATUS    RESTARTS   AGE  
auth-dd96ccdc-b-nwsd9                2/2     Running   0           3d18h  
backstage-94c96946d-7vkv5            1/1     Running   0           3d18h  
certificate-acl-654bb6cc49-z7987     1/1     Running   0           3d18h  
certificate-acl-redis-0              1/1     Running   0           3d18h  
cron-95c466b6f-224sf                 1/1     Running   0           3d18h  
data-broker-7d6c6df89f-fv9rd         1/1     Running   0           3d18h  
data-broker-redis-5f786b5f95-58r77   1/1     Running   0           3d18h  
data-manager-68b5457c7-92rcx         1/1     Running   0           3d18h  
device-manager-6fc57964bf-52692      1/1     Running   0           3d18h  
device-manager-redis-578c78755f-sr5s1 1/1     Running   0           3d18h  
flowbroker-9977b5df9-nkqn8           3/3     Running   0           3d18h  
gui-569946456-htc6j                  1/1     Running   0           3d18h  
gui-v2-77f8cc85c-h2tgn                1/1     Running   0           3d18h  
history-747d45f5c9-7gl5x             1/1     Running   0           3d18h  
image-manager-7dc49fcf44-9gqkx       1/1     Running   0           3d18h  
k2v-bridge-0                          2/2     Running   9           3d18h  
kafka-server-0                        1/1     Running   0           3d18h  
kafka-ws-7f5f5c994-6npdd              1/1     Running   0           3d18h  
kafka-ws-redis-0                      1/1     Running   0           3d18h  
kafka2ftp-bffc999d6-89hmp             1/1     Running   0           3d18h  
kong-5b7dd7655c-q846v                 1/1     Running   0           3d18h  
kong-migrate-tlc41                    0/1     Completed 0           3d18h  
kong-migrate-up-9dt42                  0/1     Completed 0           3d18h  
kong-route-config-sqvq9                0/1     Completed 1           3d18h  
minio-0                                 1/1     Running   0           3d18h  
mongodb-0                               1/1     Running   0           3d18h  
persister-7bc96b8b5d-25x2w            1/1     Running   0           3d18h  
postgres-0                             1/1     Running   0           3d18h  
rabbitmq-f55955684-zwpvd              1/1     Running   0           3d18h  
v2k-bridge-0                          2/2     Running   9           3d18h  
vernemq-k8s-0                          2/2     Running   8           3d18h  
vernemq-k8s-deployment-75cd7dff7-6kt56 1/1     Running   0           3d18h  
vmq-operator-6c6bb588b5-zqrz2         1/1     Running   0           3d18h  
x509-ejbca-847bfb4b4b-cq6dl           1/1     Running   0           3d18h  
x509-identity-mgmt-68df95bb75-4wp77   1/1     Running   0           3d18h  
zookeeper-0                            1/1     Running   0           3d18h
```

O status de cada serviço no seu cluster precisa ser o mesmo dos serviços mostrados acima. Isso pode levar alguns minutos para acontecer. Após esse período a dojot já estará pronta para ser utilizada, mas para acessá-la precisamos ainda configurar o load balancer.

Configuração do load balancer

Para utilizar a dojot com pouca carga de dispositivos e para que possamos utilizar também o LWM2M, devemos configurar o NGINX como balanceador de carga. Esse balanceador receberá todas as requisições dos clientes e fará o balanceamento de carga entre os nós worker do kubernetes, ou seja, nosso cluster não é acessado diretamente pelo cliente. Como mostrado na imagem abaixo:

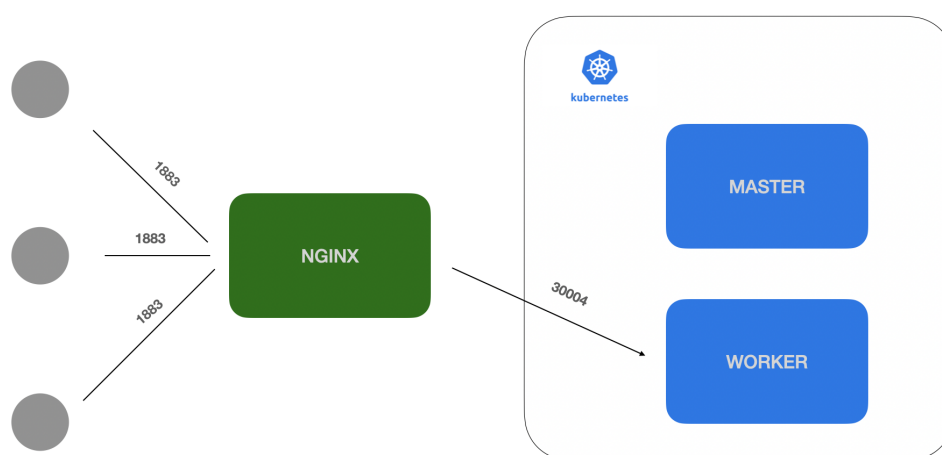


Temos um playbook para instalação e configuração do NGINX. Mas antes de executá-lo, precisamos configurar o nosso arquivo `hosts.yaml` novamente para informar ao Ansible o host da máquina que terá o NGINX instalado e também qual será o nó worker responsável por hospedar os micro-serviços da dojot. Para isso editamos o arquivo de inventário que está em "`inventories/example_local/hosts.yaml`":

```
---
all:
  hosts:
    ...
    nginx:
      ansible_host: 192.168.0.12
  ...
children:
  ...
  apigw_nodes:
    hosts:
      192.168.0.11:
  mqtt_nodes:
    hosts:
      192.168.0.11:
  lwm2m_nodes:
    hosts:
      192.168.0.11:
  metrics_nodes:
    hosts:
      192.168.0.11:
```

No exemplo acima, assumimos que o host do NGINX será "192.168.0.12" e que possuímos apenas um nó worker (192.168.0.11). Caso seu cluster tenha mais de um nó worker você tem a opção de definir em qual nó deseja que cada um desses micro-serviços rode. Essas são as únicas configurações que precisamos fazer.

Lembrando que já alteramos o nosso arquivo de variáveis anteriormente para que o kubernetes utilize portas fixas para os serviços com acesso externo. O playbook do NGINX utiliza essa mesma configuração, ou seja, se o serviço lot Agent VerneMQ expõe a porta de acesso para MQTT em 30004, o NGINX vai receber a conexão na porta 1883 e redirecionar para a porta 30004 do cluster kubernetes. Como mostrado na imagem abaixo:



Podemos então executar nosso playbook para que as configurações acima sejam aplicadas e para que nosso NGINX seja configurado:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local nginx.yaml
```

Novamente, podemos acompanhar o resultado da execução do playbook pelos logs. Todos os logs na cor amarela significam que algo foi alterado na máquina remota, a cor verde significa que nada foi alterado e a cor vermelha significa que algo deu erro. Se os logs não demonstrarem nenhum erro na execução, a dojot já estará acessível aos clientes através do NGINX, ou seja, para acessarmos a interface gráfica da dojot, na nossa configuração de exemplo, utilizaríamos o seguinte endereço no navegador: <http://192.168.0.12>.

O NGINX redireciona todas as conexões recebidas na porta 80 para a GUI da dojot que está rodando dentro do cluster.