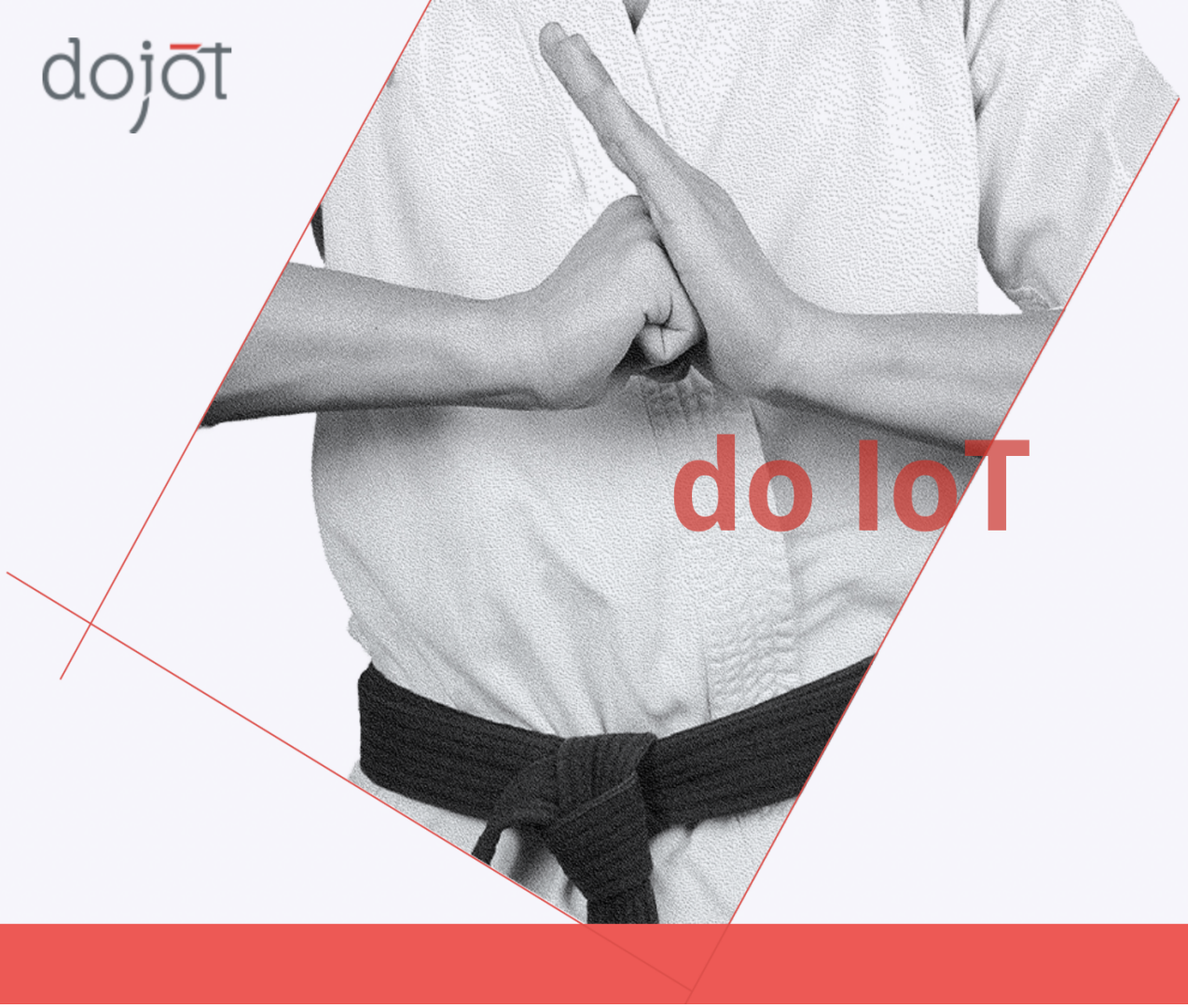
dojot



Dojot v0.6.0

Guia de Instalação

Instalando a dojot no Kubernetes

Última modificação: 06/05/2021

SUMÁRIO

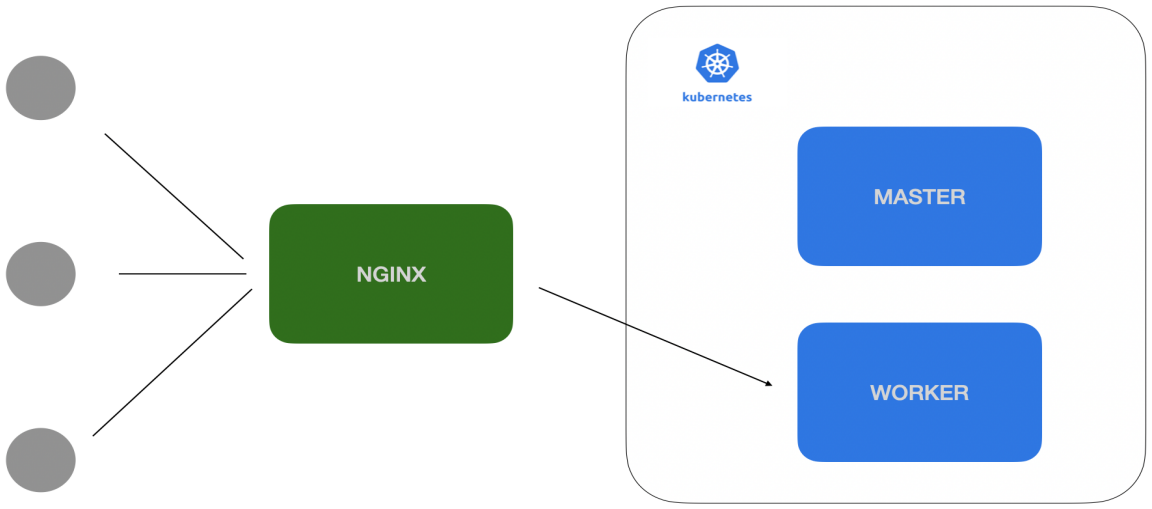
Introdução

Este documento aborda a configuração da dojot v0.6.0 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, também é recomendado no mínimo 2 workers.

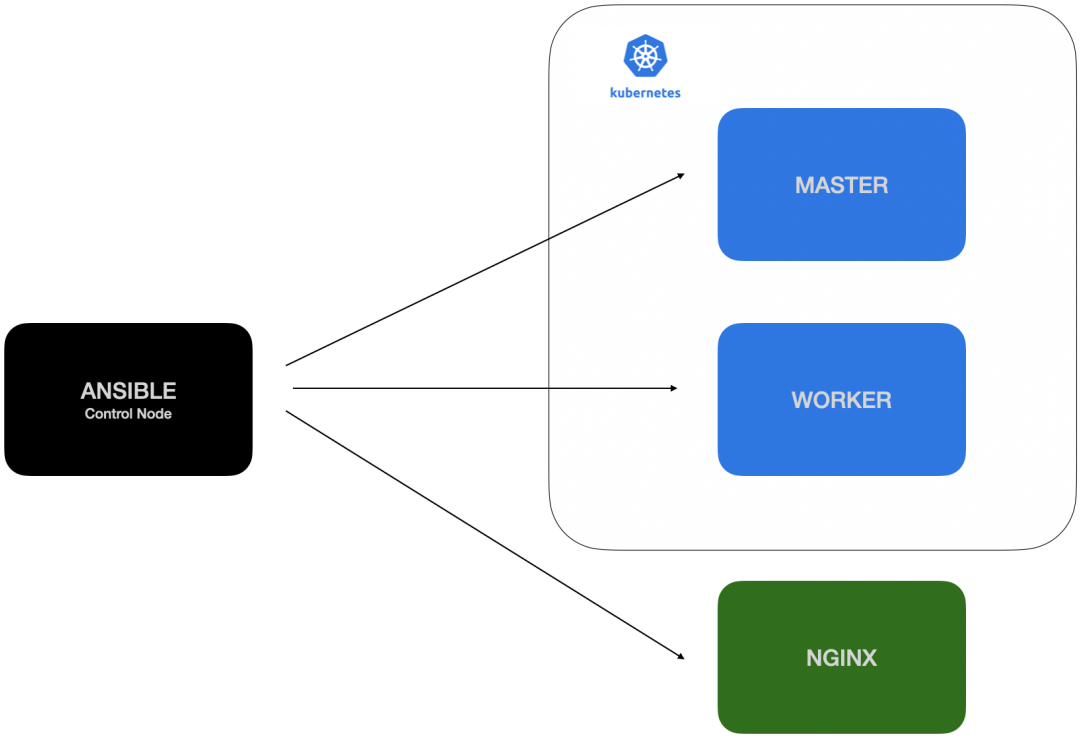
É importante lembrar que mesmo para um ambiente simples, é de extrema importância o mapeamento de volumes no cluster, para que em caso de falhas, o sistema consiga manter o estado e os dados dos contêineres.

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 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 Ansbile:



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. 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 v0.6.0 -b v0.6.0
```

Nesse exemplo, mudamos a tag para a versão v0.6.0 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 1.17 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:
    ...
    master_host:
      ansible_host: MASTER_HOST_ADDRESS #change to the host of master node on k8s
```

```
    worker_host:
      ansible_host: WORKER_HOST_ADDRESS #change to the host of worker node on k8s
  children:
    ...
    k8s-nodes:
      children:
        master_nodes:
          hosts:
            master_host:
        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, considerando que o "master" node tem o ip "192.168.0.10" e o "worker" node tem o ip 192.168.0.11. O arquivo hosts.yaml ficaria da seguinte forma:

```
---
all:
  hosts:
    ...
    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”):

```
NAME          STATUS   ROLES    AGE   VERSION
k8s-master    Ready    master   0d    v1.17.3
k8s-worker    Ready    <none>   0d    v1.17.3
```

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.

Os volumes locais utilizam o disco dos nodes do cluster como storage. Sendo assim, precisamos criar diretórios no cluster para que os volumes sejam mapeados corretamente. Precisamos então acessar o worker node via SSH e criar os diretórios localmente. O administrador do cluster pode criar os diretórios onde achar mais interessante, mas os exemplos que estão no diretório *"local_storage_example/volumes", do repositório “ansible-dojot” baixado anteriormente no control node,* apontam para os seguintes locais dentro dos nós:

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

Com base nessa tabela, os diretórios devem ser criados dentro do worker node como no exemplo abaixo:

$ mkdir -p /mnt/data/kong

No exemplo acima, assumimos que o único worker node do kubernetes possui o label "***dojot.components/group=dojot***", pois sem isso o volume não poderá ser criado e configurado de acordo com o yaml de criação do volume. Para adicionar o label no nó worker do kubernetes podemos utilizar o seguinte comando (dentro do master node):

$ kubectl label nodes k8s-worker ***dojot.components/group=dojot***

Para que os scripts de criação de volumes de exemplo possam ser utilizados, é necessário a criação dos diretórios nos locais apontados acima. Caso contrário, será necessário entrar nos arquivos do diretório *"local_storage_example/volumes"* e alterar o parâmetro path dos volumes como no exemplo abaixo, onde o diretório de volume do PostgreSQL foi alterado para *"/home/anderson/postgres"*:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv-postgres
  labels:
    type: local
    app: dojot
    db: postgres
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /home/anderson/postgres
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: dojot.components/group
          operator: In
          values:
          - dojot
```

Um outro ponto que devemos alterar no diretório *"local_storage_example/volumes"* são as labels dos arquivos "kafka.yaml", "x509_identity_mgmt.yaml", "zookeeper_data.yaml" e "zookeeper_log.yaml". Para isso basta substituir o trecho "kafka" ou "x509" por "dojot" como representado em destaque no exemplo abaixo:

```
...
 nodeAffinity:
    required:
```

```
nodeSelectorTerms:
- matchExpressions:
  - key: dojot.components/group
    operator: In
    values:
    - dojot
```

Depois de ajustar os arquivos corretamente, vamos agora copiá-los do control node para o master node do kubernetes para que possamos aplicá-los com o kubectl. Podemos utilizar o seguinte comando (dentro do control node) para copiar os arquivos:

$ scp -r local_storage_example/volumes dojot@192.168.0.10:~/

Com esse comando, copiamos o diretório "volumes" que está dentro de "local_storage_example" para dentro do diretório raiz do usuário "dojot" do host master (192.168.0.10).

Com todos os arquivos de criação de volumes copiados, podemos então acessar nó master do kubernetes por SSH e executar o seguinte comando para criar os volumes:

$ ssh dojot@192.168.0.10
$ kubectl apply -f volumes

Todos os arquivos do diretório "volumes" serão aplicados e todos os volumes serão criados. Para visualizar o status dos volumes podemos utilizar o comando *"kubectl get pv*". A seguinte lista será exibida:

```
cpqd@ryo-master:~/leonardo/ansible-dojot/local_storage_example$ kubectl get pv
NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS    REASON   AGE
local-pv-influxdb             2Gi        RWO            Retain           Available           local-storage            11s
local-pv-kafka                2Gi        RWO            Retain           Available           local-storage            11s
local-pv-kafka-ws             5Mi        ROX            Retain           Available           local-storage            11s
local-pv-kong                 5Mi        ROX            Retain           Available           local-storage            11s
local-pv-minio                2Gi        RWO            Retain           Available           local-storage            11s
local-pv-mongo                2Gi        RWO            Retain           Available           local-storage            11s
local-pv-postgres             2Gi        RWO            Retain           Available           local-storage            11s
local-pv-prometheus           2Gi        RWO            Retain           Available           local-storage            10s
local-pv-x509-identity-mgmt   10Mi       RWX            Retain           Available           local-storage            10s
local-pv-zk-data              1Gi        RWO            Retain           Available           local-storage            10s
local-pv-zk-log               1Gi        RWO            Retain           Available           local-storage            10s
```

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 que fica em i*nventories/example_local/group_vars/all/dojot.yaml*, contém estes dados:

```
dojot_namespace: dojot
dojot_version: v0.6.0
dojot_domain_name: 102.168.0.12
dojot_storage_class_name: "local-storage"
dojot_kubernetes_rbac: true
```

```
dojot_zk_persistent_volumes: true
dojot_psql_persistent_volumes: true
dojot_influxdb_persistent_volumes: true
dojot_x509_identity_mgmt_persistent_volumes: true
dojot_mongodb_persistent_volumes: true
dojot_minio_persistent_volumes: true
dojot_kafka_persistent_volumes: true
dojot_kafka_ws_persistent_volumes: true
dojot_apigw_persistent_volumes: true

dojot_guiv2_enabled: false
dojot_auth_email_enabled: false
dojot_insecure_mqtt: 'true'
dojot_vernemq_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
  mqtt: 30004
  mqtts: 30005
  lwm2m:
    coap: 30006
    coaps: 30007
    file_server: 30008
    file_servers: 30009
    http: 30010

dojot_enable_node_affinity: false
dojot_node_label:
    dojot: dojot
    x509: x509
   kafka: kafka
   vernemq: vernemq

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_storage_class_name**: utilizada para especificar o storage class utilizado pelos serviços da dojot. Na verdade, quando configuramos e criamos nossos volumes, esse storage class já foi criado com o nome “local-storage”, então só precisamos informá-lo no arquivo de variáveis. No momento do deployment os serviços vão utilizar o storage dessa classe.

- **dojot_*_persistent_volumes**: existe uma variável para cada serviço que pode utilizar persistência de dados em volumes no kubernetes. Se a variável tem o valor “true” então o volume é criado para o serviço durante o deployment.

- **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 serão utilizados pela dojot. Para isso, devemos editar o arquivo que fica em i*nventories/example_local/group_vars/all/services.yaml:*

```
---
#Some optional services that can be enabled or not on deployment
```

```
optional:
  #InfluxDB Services
  influxdb: false
  influxdb_storer: false #It will only be enabled if InfluxDB is enabled as well
  influxdb_retriever: false #It will only be enabled if InfluxDB is enabled as well
  #MongoDB Services
  history: true
  persister: true
```

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 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.

No modo padrão, utilizando esse comando, a dojot será disponibilizada utilizando o VerneMQ como Iot Agent. Temos ainda mais duas opções de Iot Agents que podem ser utilizadas na dojot. O Iot Agent Mosca e o LWM2M.

Utilizando o comando abaixo, podemos fazer deploy da dojot utilizando o VerneMQ e o LWM2M juntos:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local deploy.yaml --tags "vernemq, lwm2m"
```

Utilizando esse outro comando, podemos fazer deploy utilizando ao Iot Agent Mosca e o LWM2M:

```
$ ansible-playbook -K -k -u dojot -i inventories/example_local deploy.yaml --tags "mosca, lwm2m"
```

Podemos também fazer deploy de apenas um Iot Agent:

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

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.

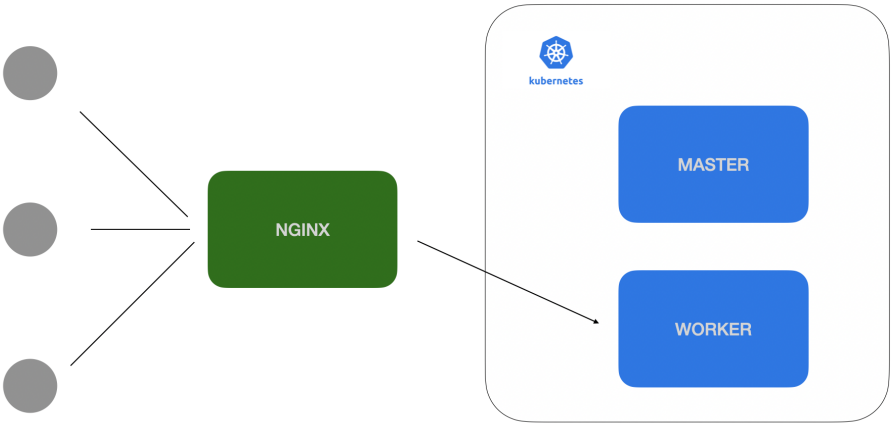
A saída desse comando é uma lista com todos os microsserviços da dojot e o status de cada um:

```
NAME                                    READY   STATUS      RESTARTS   AGE
auth-57579c6989-49hdx                   2/2     Running     0          5d1h
backstage-59865469d4-k454p              1/1     Running     2          5d1h
cron-fb9947cf-jkf4k                     1/1     Running     0          5d1h
data-broker-5cfdf576df-tbrsb            1/1     Running     0          5d1h
data-broker-redis-5f967dbcdb-pb79z      1/1     Running     0          5d1h
data-manager-744c6fd9d5-fchjj           1/1     Running     0          5d1h
device-manager-7cfb8755f-mjfnw          1/1     Running     0          5d1h
device-manager-redis-6645bd65b6-5zk7p   1/1     Running     0          5d1h
flowbroker-db77847fd-pwvbh              3/3     Running     2          5d1h
gui-6f7db8cd58-x6l44                    1/1     Running     0          5d1h
gui-v2-77d8967c5b-cld64                 1/1     Running     0          5d1h
history-6549c8fcc7-mjcc6                1/1     Running     0          5d1h
image-manager-7975dd669b-9hbxc          1/1     Running     0          5d1h
k2v-bridge-0                            1/1     Running     4          5d1h
kafka-server-0                          1/1     Running     0          5d1h
kafka-ws-7d89697f9f-psnk8               1/1     Running     0          5d1h
kafka-ws-redis-0                        1/1     Running     0          5d1h
kafka2ftp-7dd99c6c44-fcrb9              1/1     Running     1          5d1h
kong-c574d7897-j99gr                    1/1     Running     0          5d1h
kong-migrate-s42k4                      0/1     Completed   0          5d1h
kong-migrate-up-wjxgc                   0/1     Completed   0          5d1h
kong-route-config-kbgmh                 0/1     Completed   1          5d1h
minio-0                                 1/1     Running     0          5d1h
mongodb-0                               1/1     Running     0          5d1h
persister-75664cb8cb-gzx52              1/1     Running     0          5d1h
postgres-0                              1/1     Running     0          5d1h
rabbitmq-bff4b85df-xch72                1/1     Running     0          5d1h
v2k-bridge-0                            1/1     Running     4          5d1h
vernemq-k8s-0                           1/1     Running     5          5d1h
vernemq-k8s-deployment-69d556df7b-h6cqh 1/1     Running     0          5d1h
vmq-operator-6774ff9ff4-nnwng           1/1     Running     0          5d1h
x509-identity-mgmt-7f76fff569-sfs2t     1/1     Running     0          5d1h
zookeeper-0                             1/1     Running     0          5d1h
```

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:

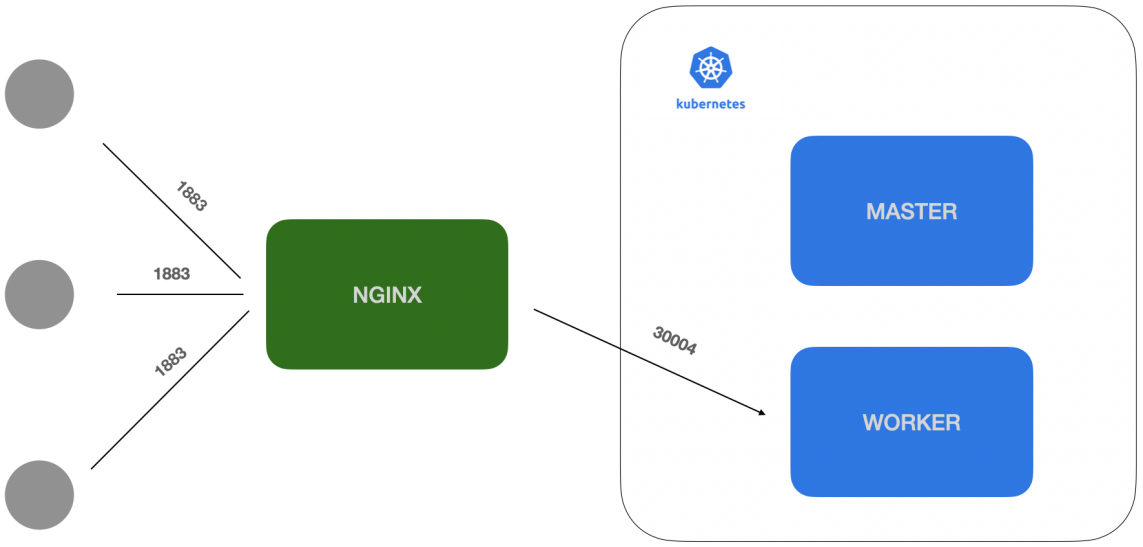


Temos um playbook para instalação e configuração do NGINX. Mas antes de executá-lo, precisamos configurar o nosso arquivo hosts.yaml novamente para informar ao Ansible o host da máquina que terá o NGINX instalado e também qual será o nó worker responsável por hospedar os micro-serviços da dojot. Para 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 Iot 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.