## Déploiement dans Kubernetes

Table des matières

[Déploiement dans Kubernetes 1](#_Toc529553467)

[Installation de Rancher et Kubernetes 1](#_Toc529553468)

[Installation manuelle d’un master Kubernetes et d’un Node séparé 3](#_Toc529553469)

[Configurer Kubernetes avec le Cloud-provider AWS 4](#_Toc529553470)

[*kube-dns is always pending because of weave net crash* 8](#_Toc529553471)

[Quelques commandes du CNI Weave 8](#_Toc529553472)

[Diverses commandes Kubernetes 9](#_Toc529553473)

[Reset complet de Kubernetes et des nodes 9](#_Toc529553474)

[Installation d’un Ingress Controller Kubernetes-Nginx 11](#_Toc529553475)

[Certificat auto-signé : 12](#_Toc529553476)

[Création d’une configuration de déploiement Kubernetes 17](#_Toc529553477)

### Installation de Rancher et Kubernetes

*Note : tutoriel d’installation de Rancher :*

<https://www.weave.works/blog/how-to-supercharge-your-kubernetes-cluster-with-rancher-weave-cloud>

1. Instancier une instance EC2 **'Ubuntu Server 16**' dédiée à Rancher *(T2.medium, sinon pb de mémoire avec les micro et les small pour héberger le serveur + un host (Worker node Kubernetes))*. **Y associer également une EIP**.
2. Mettre à jour les références apt-get et installer Docker :

$ apt-get update

$ apt-get install -y docker.io

1. Installer un conteneur Rancher :

$ docker run -d --restart=unless-stopped -p 8080:8080 rancher/server

1. Se connecter à la console d’administration de Rancher via l’URL http://<IP publique instance EC2> :8080
2. *Dans la console web de Rancher, créer un environnement Kubernetes et ajouter un worker node :*
   * *Cliquer sur "Default"-> "Manage Environments", puis sur le bouton "Add environnment"*
   * *Entrer un nom et une description, sélectionner "Kubernetes" et cliquer sur Ok*
   * *Dans le menu suivant, cliquer sur "Default"-> <Nom de l'environnement créé>*
   * *Un hôte doit être créé : dans la bannière qui apparait, cliquer sur "Add a host"*
   * *Indiquer une clé d'accès d'utilisateur AWS et la région de la VM Rancher installée*
   * *Dans la page suivante, choisir un VPC et un subnet accessibles depuis le serveur Rancher (même subnet privé ou exposé sur Internet).*
   * *Dans la page suivante, choisir de créer un « nouveau Security Group ».*
   * *Dans la page suivante, choisir un nom (ex : 'rancher-kube-host-1'), laisser le reste tel quel et cliquer sur "Create"*
   * *Suivre également l'évolution de la création de l'hôte dans le menu "Infrastructure" -> "Hosts"*
3. Dans la console web de Rancher, créer un environnement Kubernetes et ajouter un worker node :
   * Cliquer sur "Default"-> "Manage Environments", puis sur le bouton "Add environnment"
   * Entrer un nom et une description, sélectionner "Kubernetes" et cliquer sur Ok
   * Dans le menu suivant, cliquer sur "Default"-> *<Nom de l'environnement créé>*
   * Un hôte doit être créé : dans la bannière qui apparait, cliquer sur "Add a host". **Bien vérifier qu'on est dans l'environnement Kubernetes** (et non dans 'Default', cf en haut à gauche).
   * Dans l'hôte à créer, choisir 'Custom', puis saisir l'EIOP dans l'adresse.
   * Copier la commande générée par Rancher et la coller dans l'instance EC2 exécutant Rancher (en gros on créé un Worker node Kubernetes sur le serveur lui-même). Elle doit ressembler à :
   * $ sudo docker run -e CATTLE\_AGENT\_IP="52.210.251.84" --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.11 http://52.210.251.84:8080/v1/scripts/B5FBD6EE765F0FE57896:1514678400000:KID50OYIQ0Rc5dDvYaxGXNrCFk
   * Suivre également l'évolution de la création de l'hôte dans le menu "Infrastructure" -> "Hosts"
4. Une fois l'hôte créé et configuré, configurer la commande kubectl sur l’instance EC2 hébergeant Rancher :
   * Installer la commande ‘kubectl’ :

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

$ sudo apt-get update

$ sudo apt-get install -y kubectl

* + Dans la console web de Rancher, lancer "Kubernetes" -> "CLI".
  + Cliquer sur le bouton « Generate configuration » et copier la configuration.
  + Sur l’instance EC2 hébergeant Rancher, créer le fichier « ~/.kube/config » et y coller la configuration copiée précédemment.
  + Tester avec la commande "kubectl get nodes" sur l’instance EC2 hébergeant Rancher et Jenkins : l'host créé doit apparaitre et être 'Ready'.

### Installation manuelle d’un master Kubernetes et d’un Node séparé

INSTALLATION DU MASTER ET DU WORKER-NODE :

2 instances EC2 à déployer : 1 pour le master (kubeadm, etc.) et 1 pour un workernode (peuvent être des T2 micro).

Les commandes suivantes sont à exécuter **sur le master et sur le worker** :

$ apt-get update -y

$ swapoff -a // Doit être saisi à chaque démarrage de l’instance ?

$ hostnamectl set-hostname kube-master *// ou kube-worker*

$ vi /etc/hosts *//Y declarer les hostname des 2 serveurs*

$ apt-get install apt-transport-https ca-certificates curl software-properties-common -y

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

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

$ apt-get update -y

$ apt-get install docker-ce -y

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

$ echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ apt-get update -y

$ sysctl net.bridge.bridge-nf-call-iptables=1 // Pour le CNI wave.net // Doit être saisi à chaque démarrage de l’instance ? Fonctinne avec Ubuntu 16 mais pas Ubuntu 18 ?

$ apt-get install kubelet kubeadm kubectl -y

Sur le **master** uniquement :

*Optionnel : pour deployer des pods sur le master (non recommandé).*

*$ kubectl taint nodes --all node-role.kubernetes.io/master-*

$ kubeadm init *--pod-network-cidr=192.168.1.0/24*

*Note : noter la commande à exécuter sur le worker node pour le joindre au master.*

$ mkdir -p $HOME/.kube

$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ chown $(id -u):$(id -g) $HOME/.kube/config

OU $ *mkdir -p $HOME/.kube; cp -i /etc/kubernetes/admin.conf $HOME/.kube/config; chown $(id -u):$(id -g) $HOME/.kube/config*

$ kubectl get pods --all-namespaces

-> Attendre que tout soit 'running', sauf le service CoreDNS.

Installation du CNI pour la gestion du réseau (ici, Weaver) :

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

*Note : pour modifier le CIDR :*

*$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC\_RANGE=10.32.0.0/24"*

$ kubectl get pods --all-namespaces

-> Tout doit être 'running'

*Optionnel : à tester ?*

*$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml*

Joindre le premier **Worker node** avec la commande 'Join' *(commande à passer sur le Worker uniquement)* :

$ kubectl join (cf commande copiée précédemment lors du ‘kubeadm init’) :

Sur le master uniquement :

$ kubectl get nodes

-> Le master node et le worker doivent tous les 2 être 'ready'

### Configurer Kubernetes avec le Cloud-provider AWS

Suivant le tuto : <https://blog.scottlowe.org/2018/09/28/setting-up-the-kubernetes-aws-cloud-provider/>

Sur chacune des instances EC2 appartenant au cluster Kubernetes :

$ kubectl set-hostname <EC2 Private DNS (cf description de l’instance EC2)>

*A ajouter dans le /etc/hosts ?*

Création de 2 Policies dans IAM qui seront utilisées en tant qu’Instance Profile par les instances EC2 hébergeant le master et le worker.

**kube-master-InstanceProfile-Policy** :

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"autoscaling:DescribeAutoScalingGroups",

"autoscaling:DescribeLaunchConfigurations",

"autoscaling:DescribeTags",

"ec2:DescribeInstances",

"ec2:DescribeRegions",

"ec2:DescribeRouteTables",

"ec2:DescribeSecurityGroups",

"ec2:DescribeSubnets",

"ec2:DescribeVolumes",

"ec2:CreateSecurityGroup",

"ec2:CreateTags",

"ec2:CreateVolume",

"ec2:ModifyInstanceAttribute",

"ec2:ModifyVolume",

"ec2:AttachVolume",

"ec2:AuthorizeSecurityGroupIngress",

"ec2:CreateRoute",

"ec2:DeleteRoute",

"ec2:DeleteSecurityGroup",

"ec2:DeleteVolume",

"ec2:DetachVolume",

"ec2:RevokeSecurityGroupIngress",

"ec2:DescribeVpcs",

"elasticloadbalancing:AddTags",

"elasticloadbalancing:AttachLoadBalancerToSubnets",

"elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",

"elasticloadbalancing:CreateLoadBalancer",

"elasticloadbalancing:CreateLoadBalancerPolicy",

"elasticloadbalancing:CreateLoadBalancerListeners",

"elasticloadbalancing:ConfigureHealthCheck",

"elasticloadbalancing:DeleteLoadBalancer",

"elasticloadbalancing:DeleteLoadBalancerListeners",

"elasticloadbalancing:DescribeLoadBalancers",

"elasticloadbalancing:DescribeLoadBalancerAttributes",

"elasticloadbalancing:DetachLoadBalancerFromSubnets",

"elasticloadbalancing:DeregisterInstancesFromLoadBalancer",

"elasticloadbalancing:ModifyLoadBalancerAttributes",

"elasticloadbalancing:RegisterInstancesWithLoadBalancer",

"elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",

"elasticloadbalancing:AddTags",

"elasticloadbalancing:CreateListener",

"elasticloadbalancing:CreateTargetGroup",

"elasticloadbalancing:DeleteListener",

"elasticloadbalancing:DeleteTargetGroup",

"elasticloadbalancing:DescribeListeners",

"elasticloadbalancing:DescribeLoadBalancerPolicies",

"elasticloadbalancing:DescribeTargetGroups",

"elasticloadbalancing:DescribeTargetHealth",

"elasticloadbalancing:ModifyListener",

"elasticloadbalancing:ModifyTargetGroup",

"elasticloadbalancing:RegisterTargets",

"elasticloadbalancing:SetLoadBalancerPoliciesOfListener",

"iam:CreateServiceLinkedRole",

"kms:DescribeKey"

],

"Resource": [

"\*"

]

}

]

}

**kube-node-InstanceProfile-Policy** :

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ec2:DescribeInstances",

"ec2:DescribeRegions",

"ecr:GetAuthorizationToken",

"ecr:BatchCheckLayerAvailability",

"ecr:GetDownloadUrlForLayer",

"ecr:GetRepositoryPolicy",

"ecr:DescribeRepositories",

"ecr:ListImages",

"ecr:BatchGetImage"

],

"Resource": "\*"

}

]

}

**TAGS instances et resources** :

Change **EC2** instance tag, KEY=**KubernetesCluster**, VALUE=**k8s**

Key=kubernetes.io/cluster/<cluster name> (par défaut, le ‘<cluster name>’ est ‘kubernetes’)

Value=shared

Créer les ressources suivantes :

VPC

Subnet

Route table

Internet Gateway

SecurityGroup (tag)

Instance EC2 (tag) / Ne pointe que sur le Security Group créé (pas d’autre)/ association de profile

***Hostname****:*

*It is important that the node name (as seen in kubectl get nodes) match the private-dns-name property of the EC2 instance.*

*Ajouter l’argument suivant aux commandes kubeadm-join et kubeadm -init :*

--node-name="$(hostname -f 2>/dev/null || curl http://169.254.169.254/latest/meta-data/local-hostname)"

Autres tutos :

<https://github.com/kubernetes/kubernetes/issues/57718>

<https://docs.google.com/document/d/17d4qinC_HnIwrK0GHnRlD1FKkTNdN__VO4TH9-EzbIY/edit>

kubeadm config file to specify the Cloud Provider to configure during kubeadm init :

***kubeadmin\_conf.yaml****:*

$ kubeadm init –cloud-provider=aws --node-name="$(hostname -f 2>/dev/null || curl <http://169.254.169.254/latest/meta-data/local-hostname>)"

$ kubectl get pods --all-namespaces

-> Attendre que tout soit 'running', sauf le service CoreDNS.

Installation du CNI pour la gestion du réseau (ici, Weaver) :

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"

*Note : pour modifier le CIDR :*

*$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC\_RANGE=10.32.0.0/24"*

$ kubectl get pods --all-namespaces

-> Tout doit être 'running'

$ kubeadm init --config kubeadmin\_conf.yaml

Autres :

/etc/kubernetes/aws.conf

/etc/kubernetes/cloud-config.conf :

*Vérifier la correspondance des tags instances EC2 et contenu du fichier :*

[Global]

KubernetesClusterTag=k8s

KubernetesClusterID=k8s

## *kube-dns is always pending because of weave net crash*

*Fix cgroups setting.*

* *Add Environment="KUBELET\_CGROUP\_ARGS=--cgroup-driver=cgroupfs" to /etc/systemd/system/kubelet.service.d/10-kubeadm.conf.*
* *Restart****kubelet****service.*
* *sudo systemctl deamon-reload*
* *sudo systemctl restart kubelet*

*Install weave net and launch before creating****kubernetes****cluster.*

*Note : noter la commande à exécuter sur le worker node pour le joindre au master.*

$ mkdir -p $HOME/.kube

$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ chown $(id -u):$(id -g) $HOME/.kube/config

OU $ *mkdir -p $HOME/.kube; cp -i /etc/kubernetes/admin.conf $HOME/.kube/config; chown $(id -u):$(id -g) $HOME/.kube/config*

$ kubectl get pods --all-namespaces

-> Attendre que tout soit 'running', sauf le service CoreDNS.

### Quelques commandes du CNI Weave

DEBUGG WEAVE :

$ kubectl get pods -n kube-system -o wide *// Affiche également sur quel node le conteneur weave tourne pour le debuger par la suite*

$ kubectl get pods -n kube-system -l name=weave-net

$ kubectl logs -n kube-system <POD Weave> weave

$ kubectl exec -n kube-system <POD Weave> -c weave -- /home/weave/weave --local status

$ kubectl logs <POD Weave> -n kube-system weave-npc

Voir les IPs assignées aux pods des applications :

$ kubectl get pods -o wide | grep nginx

Désinstaller Weave (à faire sur TOUS les noeuds) :

*$ weave reset // ? Requière un script weave ?*

*$ curl -L git.io/weave -o /usr/local/bin/weave*

*$ chmod a+x /usr/local/bin/weave*

$ ip link delete weave

$ rm /opt/cni/bin/weave-\*

*Supprimer également les images Docker de weave*

### Diverses commandes Kubernetes

Reset de Kubeadm :

$ kubeadm reset && rm -rf ~/.kube/ && kubeadm init --pod-network-cidr=172.18.1.0/24

Changer CIDR de kube-proxy :

*Cf* [*https://github.com/kubernetes/kops/issues/4674*](https://github.com/kubernetes/kops/issues/4674)

### Reset complet de Kubernetes et des nodes

*A faire sur le master :*

# Drain and delete the nodes (for each node you have)

kubectl drain kube-master --delete-local-data --force --ignore-daemonsets

kubectl delete node kube-master

# Reset the deployment

kubeadm reset

*A faire sur chaque nœud (master et workers) :*

## Reset the nodes and weave

sudo curl -L git.io/weave -o /usr/local/bin/weave

sudo chmod a+x /usr/local/bin/weave

sudo kubeadm reset && rm -rf ~/.kube/

sudo weave reset --force

## Clean weave binaries

sudo rm /opt/cni/bin/weave-\*

## Flush iptables rules on all nodes and restart Docker

iptables -P INPUT ACCEPT

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

iptables -t nat -F

iptables -t mangle -F

iptables -F

iptables -X

systemctl restart docker

*+ supprimer les images Docker Weave sur les nœuds master et worker*

## Installation d’un Ingress Controller Kubernetes-Nginx

*Création des ressources ‘mandatory’ :*

*$ kubectl apply -f* [*https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml*](https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml)

*$ kubectl apply -f* [*https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/cloud-generic.yaml*](https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/cloud-generic.yaml)

*Vérification du fonctionnement de l’Ingress Controller :*

*$ kubectl get pods --all-namespaces | grep ingress-nginx*

*-> Le conteneur doit être ‘running’*

Selon le **tutoriel** suivant :

<https://hackernoon.com/setting-up-nginx-ingress-on-kubernetes-2b733d8d2f45>

*(*[*https://ryaneschinger.com/blog/automatic-dns-kubernetes-ingresses-externaldns/*](https://ryaneschinger.com/blog/automatic-dns-kubernetes-ingresses-externaldns/)

[*https://akomljen.com/kubernetes-nginx-ingress-controller/*](https://akomljen.com/kubernetes-nginx-ingress-controller/)*)*

**Test de l’ingress-controller** via le déploiement d’une application test :

**nginx-deployment.yaml :**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.15.4

ports:

- containerPort: 80

Déployer la conf :

$ kubectl create -f nginx-deployment.yaml

**nginx-service-lb.yaml** :

apiVersion: v1

kind: Service

metadata:

name: nginx-service-lb

spec:

type: LoadBalancer

ports:

- name: http

port: 80

targetPort: 80

protocol: TCP

*externalIPs:*

*- 172.18.1.51*

selector:

app: nginx

Déployer la conf :

$ kubectl create -f nginx-service-lb.yaml

### Certificat auto-signé :

$ mkdir /etc/ssl/private

$ chmod 700 /etc/ssl/private

$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt

FQDN = ingress.maddoudou.click

$ openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048

Modifier /etc/hosts pour ajouter :

<IP privée instance EC2> ingress.maddoudou.click

Déclaration des certificats dans un secret Kubernetes :

$ kubectl create secret tls tls-certificate --key /etc/ssl/private/ingress-nginx-selfsigned.key --cert /etc/ssl/certs/ingress-nginx-selfsigned.crt

$ kubectl create secret generic tls-dhparam --from-file=/etc/ssl/certs/dhparam.pem

**default-backend-deployment.yaml**:

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: default-http-backend

spec:

replicas: 1

template:

metadata:

labels:

app: default-http-backend

spec:

terminationGracePeriodSeconds: 60

containers:

- name: default-http-backend

# Any image is permissable as long as:

# 1. It serves a 404 page at /

# 2. It serves 200 on a /healthz endpoint

image: gcr.io/google\_containers/defaultbackend:1.0

livenessProbe:

httpGet:

path: /healthz

port: 8080

scheme: HTTP

initialDelaySeconds: 30

timeoutSeconds: 5

ports:

- containerPort: 8080

resources:

limits:

cpu: 10m

memory: 20Mi

requests:

cpu: 10m

memory: 20Mi

---

kind: Service

apiVersion: v1

metadata:

name: default-http-backend

spec:

selector:

app: default-http-backend

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: NodePort

Déployer la conf :

$ kubectl create -f default-backend-deployment.yaml

Créer un rôle permettant au conteneur nginx-ingress (prochain fichier de configuration) de cible de default-backend) :

**default-privilege-add.yaml** :

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: default-privilege-add

rules:

- apiGroups: [""]

resources:

- services

- endpoints

- pods

verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: default-privilege-add

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: default-privilege-add

subjects:

- kind: ServiceAccount

name: default

namespace: default

Déployer le rôle :

$ kubectl apply -f default-privilege-add.yaml

Création du contrôleur nginx :

**nginx-controller.yaml** :

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-ingress-controller

spec:

replicas: 1

revisionHistoryLimit: 3

template:

metadata:

labels:

k8s-app: nginx-ingress-lb

spec:

containers:

- args:

- /nginx-ingress-controller

- "--default-backend-service=$(POD\_NAMESPACE)/default-http-backend"

- "--default-ssl-certificate=$(POD\_NAMESPACE)/tls-certificate"

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

image: "gcr.io/google\_containers/nginx-ingress-controller:0.9.0-beta.5"

imagePullPolicy: Always

livenessProbe:

httpGet:

path: /healthz

port: 10254

scheme: HTTP

initialDelaySeconds: 10

timeoutSeconds: 5

name: nginx-ingress-controller

ports:

- containerPort: 80

name: http

protocol: TCP

- containerPort: 443

name: https

protocol: TCP

volumeMounts:

- mountPath: /etc/nginx-ssl/dhparam

name: tls-dhparam-vol

terminationGracePeriodSeconds: 60

volumes:

- name: tls-dhparam-vol

secret:

secretName: tls-dhparam

---

apiVersion: v1

kind: Service

metadata:

name: nginx-ingress

spec:

type: LoadBalancer

ports:

- name: http

port: 80

targetPort: http

- name: https

port: 443

targetPort: https

selector:

k8s-app: nginx-ingress-lb

### Création d’une configuration de déploiement Kubernetes

1. Vérifier que la commande kubectl fonctionne et dialogue bien avec le serveur :

$ kubectl version

*Ne doit pas retourner d’erreurs de connexion*

1. Vérifier qu’au moins 1 worker node est actif :

$ kubectl get nodes

*Doit retourner au moins 1 nœud ‘ready’.*

1. Créer un déploiement (par exemple nommé ‘api-javaspringboot’). :

$ kubectl get nodes

#### Quelques commandes Kubernetes :

$ kubectl create -f api-javaspringboot-small-deployment.yaml

$ kubectl delete deployment nginx-deployment

Debuger un problème de déploiement de POD :

$ kubectl describe pod <ID du pod> -n kube-system

Obtenir le status d'un déploiement :

$ kubectl rollout status deployment.v1.apps/<nom déploiement>

Supprimer un déploiement et les pods déployés :

$ kubectl describe deployment <nom déploiement>

Exposer un déploiement via un service à la volée :

$ kubectl expose deployment hello-world --type=LoadBalancer --name=my-service

Vérifier les endpoints d’un service :

$ kubectl get endpoints <service>

Tenter de démarrer kubelet avec ces infos pour eviter les evictions de pod :

$ kubectl expose deployment hello-world --type=LoadBalancer --name=my-service

#### Exemple de fichier de déploiement :

Fichier "**api-deploy.yaml**" :

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-javaspringboot

labels:

app: API-javaSpringboot

spec:

replicas: 1

selector:

matchLabels:

app: API-javaSpringboot

template:

metadata:

labels:

app: API-javaSpringboot

spec:

containers:

- name: api-javaspringboot

image: devops.maddoudou.click:5000/api-javaspringboot:latest

ports:

- containerPort: 80