# Travaux Pratiques: Kubernetes - Minikube

## Introduction

Dans ce TP nous allons déployer un cluster Kubernetes mononode.

Nous montrerons comment :

- comment utiliser minikube
- déployer un pod
- scaler ce pod
- accéder à un service

Pré-requis :

- Connaître les bases de Docker
- Connaître les bases de Kubernetes
- Connaître la CLI Kubectl

## Préparation

Nous allons tout d'abord récupérer la CLI kubectl, nécessaire à Minikube.

### Windows

Utiliser l'installeur en suivant les étapes sur :
https://minikube.sigs.k8s.io/docs/start/

### Mac

```
$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/darwin/amd64/kubectl"
$ chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl
$ kubectl version --client
```

Il faut ensuite récupérer le binaire de Minikube

```
$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
darwin-amd64
$ sudo install minikube-darwin-amd64 /usr/local/bin/minikube
$ minikube version
minikube version: v1.6.2
```

## Lancement de minikube

Lancer minikube:

```
$ minikube start
```

En cas ou ça marche pas essayer de Lancer minikube avec Virtualbox comme driver :

```
$ minikube start --vm-driver=virtualbox
```

## Cluster info

Kubectl est utilisé pour interagir avec le cluster. Pour voir l'état du cluster :

```
$ kubectl cluster-info
```

Lister les nodes :

```
$ kubectl get nodes
```

## Déploiement de conteneurs

Kubectl permet d'interagir avec Kubernetes :

```
$ kubectl create deployment first-deployment --image=katacoda/docker-http-
server
```

Lister les Deployments :

```
$ kubectl get deploy
```

Lister les pods :

```
$ kubectl get pods
```

Il est possible d'exposer le port du conteneur sur la VM minikube en utilisant un service de type *NodePort*. La commande suivante expose le port 80 du conteneur sur un port aléatoire de l'host :

```
$ kubectl expose deployment first-deployment --port=80 --type=NodePort
```

Récupérer le port du services associé :

```
$ kubectl get svc
```

Récupérer l'IP de la VM minikube :

```
$ kubectl get nodes -o wide
```

Accéder au service via curl :

```
curl -v $MINIKUBE_IP:NODE_PORT
```

## Scaler un *deployment*

La commande `kubectl scale` permet d'ajuster le nombre de replica d'un *deployment*:

```
$ kubectl scale --replicas=3 deployment first-deployment
```

Listez ensuite les pods disponibles:

```
$ kubectl get pods
```

Il y a maintenant 3 pods disponibles pour ce *Deployment*

Decrivez ensuite l'objet de type *Service* associé et regarder la partie *endpoints*:

```
$ kubectl describe svc first-deployment
```

```
$ kubectl get endpoints
```

Que remarquez vous ?

## Installation du dashboard

Pour démarrer le dashboard sur minikube :

```
minikube addons enable dashboard
```

Vérifier le déploiement du dashboard :

```
$ kubectl -n kubernetes-dashboard get pods
$ kubectl -n kubernetes-dashboard get svc
```

Comme pour notre premier déploiement, il est possible d'exposer le service associé au dashboard via un *NodePort* :

```
$ kubectl -n kubernetes-dashboard edit svc kubernetes-dashboard
```

Il faut ensuite modifier le type de service en type `NodePort`.

De la même manière vue précédemment, récupérez l'IP de la VM minikube ainsi que le *NodePort* et accédez au dashboard via un navigateur.

## Les fichiers YAML

### Différence entre un pod et déploiement

Nous utiliserons l'image `helloworld` de Docker comme démonstrateur. Celle ci expose un service web sur le port 80.

Nous allons d'abord créer un objet de type *Pod*.

Dans un fichier `hello-world-pod.yaml` :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: helloworld
  labels:
    app: helloworld
spec:
  containers:
  - name: helloworld
    image: particule/helloworld
    ports:
    - containerPort: 80
```

Pour appliquer ce fichier sur le cluster :

```
$ kubectl apply -f hello-world-pod.yaml
```

Vérifiez que votre pod est en marche avec `kubectl get pods`.

Supprimez votre pod avec `kubectl delete pod nom_de_votre_pod`.

Est-il correctement supprimé ?

Nous allons maintenant créer un objet de type *Deployment*

Dans un fichier `helloworld-de.yml`.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworld
  labels:
    app: helloworld
spec:
  replicas: 1
  selector:
    matchLabels:
      app: helloworld
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: helloworld
        image: particule/helloworld
        ports:
        - containerPort: 80
```

Pour appliquer ce fichier sur le cluster :

```
$ kubectl apply -f helloworld-de.yaml
```

Vérifiez que votre pod est en marche avec `kubectl get pods`.

Supprimez votre pod avec `kubectl delete pod nom_de_votre_pod`.

Est-il correctement supprimé ? Pourquoi ?

## Création d'un service

Les services peuvent également être créés via des fichiers YAML. Créez un service de type *NodePort* comme vu précédemment avec `kubectl run`. Par exemple dans un fichier `helloworld-svc.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: helloworld-svc
  labels:
    app: helloworld
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: helloworld
```

Les objets kubernetes sont déployés avec `kubectl` et notamment avec `kubectl apply` :

```
$ kubectl apply -f helloworld-svc.yaml
```

Comme pour tous les objets, il est possible de décrire le service :

```
$ kubectl describe svc helloworld-svc
```

## Scaler un *deployment* en YAML

Pour scaler le *deployment* sans utiliser la commande `kubectl scale`, éditez le fichier YAML et changer le nombre de replicas puis appliquez le fichier sur le cluster.