TP1 - Cours de virtualisation avancée - *Kubernetes*, introduction

L'objectif de ce TP est de découvrir l'orchestrateur *"Kubernetes"*.

Pour cela, vous allez déployer quelques conteneurs et lancer des commandes pour découvrir le fonctionnement de *Kubernetes*.

Ces conteneurs auront été configurés par mes soins et vous n'aurez donc pas de fichier à éditer.

Pour les TPs de *Kubernetes*, vous allez utiliser un outil nommé *"minikube"*. Cet outil permet la mise en place de *"clusters" K8S* dédiés aux tests. Ce ne sont pas des *clusters* utilisables sereinement en production mais conçus pour permettre la découverte et les tests sans devoir s'embarrasser d'un "vrai" *cluster*, compliqué à mettre en place soit même.

1 Mise en place

Vous allez travailler dans une machine virtuelle *VDN*. De cette façon, vous pourrez utiliser *Docker*, ce qui est nécessaire pour les TPs.

Commencez par lancer une machine virtuelle *VDN*:

```
vdn-start -t -n docker debian-1
```

> **Attention:**
> À chaque fois que vous vous connectez à la machine virtuelle, n'oubliez pas de lancer la commande suivante, sinon vous ne pourrez pas utiliser l'outil `kubectl` qui est l'outil de gestion de *K8S* !
> `export no_proxy="127.0.0.1,.vdn,localhost,192.168.49.1/24,10.96.0.0/12"`

Dans un autre terminal, connectez vous à la machine virtuelle en `root`:

```
vdn-ssh root@debian-1
```

Puis ajouter l'utilisateur `test` au *sudoers* pour lui permettre de lancer des commandes en tant que `root`. Pour cela, ajoutez à la fin du fichier `/etc/sudoers` la ligne:

```
test ALL=(ALL:ALL) ALL
```

Puis changez le mot de passe de l'utilisateur `test` **et mémorisez ce mot de passe, vous en aurez besoin**:

```
passwd test
```

Ensuite, **dans un autre terminal**, connectez-vous via *SSH* à cette machine virtuelle:

```
vdn-ssh test@debian-1
```

Clonez le dépôt du TP dans la machine virtuelle:

```
git clone \
https://codefirst.iut.uca.fr/git/evrard.van_espen/cours_virtualisation_avancee_ETUD.git
```

Ensuite, lancez le script `01_init.sh` présent dans le dépôt pour installer *minikube*.

Une fois que le script est fini et que *minikube* a fini de démarrer, lancez cette commande:

```
./minikube status
```

Le résultat de la commande doit être:

```
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Puis lancez:

```
./kubectl get namespaces
```

Le résultat de la commande doit être:

```
NAME              STATUS   AGE
default           Active   11m
kube-node-lease   Active   11m
kube-public       Active   11m
kube-system       Active   11m
```

Enfin, **dans un autre terminal**, lancez:

```
./minikube tunnel
```

Cela va vous demander le mot de passe que vous avez attribué à l'utilisateur `test`. Ce terminal doit afficher:

```
Status:
 machine: minikube
 pid: 10418
 route: 10.96.0.0/12 -> 192.168.49.2
 minikube: Running
 services: []
    errors:
  minikube: no errors
  router: no errors
  loadbalancer emulator: no errors
```

Laissez ce terminal tourner, **n'y touchez plus !**

> Si tout s'est déroulé comme il faut, cela veut dire que tout est installé correctement et que vous pourrez faire les TPs comme il faut.

2 Votre mission

Vous allez découvrir comment gérer *K8S* au moyen de son outil ligne de commande `kubectl`. Vous allez donc devoir déployer des ressources et vérifier leur fonctionnement.

2.1 Déployer et vérifier les ressources

Les ressources sont configurées dans le fichier du dépôt `02_ressources/01_all.yaml`, jetez-y un œil ;)

Ce fichier défini un *pod* et un *service* permettant l'accès au service présent dans le *pod*. L'accès se fait via un *NodePort*, qui permet l'ouverture d'un port local sur la machine à destination du service *HTTP* présent dans le *pod*. Cependant, ce n'est pas une belle manière de faire les choses, vous changerez cela dans la seconde partie de ce TP.

Pour commencer, appliquez les configurations du fichier. Ensuite, listez les *pods*, vous devez obtenir un résultat similaire à:

```
NAME                                READY   STATUS    RESTARTS   AGE
hello-world-6cffc7d974-5pgc8   1/1     Running   0          9m38s
```

Puis listez les services, vous devez obtenir un résultat similaire à:

```
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-world   NodePort    10.109.243.202   <none>        4242:30000/TCP   8s
```

Si c'est le cas, bravo, vous avez déployé vos premiers *pods* et *services*. Vous pouvez tentez de communiquer avec le service en tapant:

```
curl http://192.168.49.2:30000
```

Si tout s'est bien passé, vous devriez avoir un résultat tel que:

```
Request served by hello-world-bdd88c654-kcg8p

HTTP/1.1 GET /

Host: 10.96.123.46
Accept: */*
User-Agent: curl/7.88.1
```

2.2 Maintenant, à vous de jouer

À présent, vous allez devoir modifier la configuration pour rendre tout cela un peu plus "propre".

Tâches à réaliser:

- ne plus utiliser un simple *Pod* mais plutôt un *Deployment* avec un nombre de *replicas* à 3;
- ne pas utiliser un *NodePort* mais plutôt un *LoadBalancer* qui redirigea le trafic sur les différents *Pods* du *Deployment*.

Pour vérifier le bon fonctionnement du *LoadBalancer*, commencez par récupérer son adresse "externe" et vous pourrez faire:

```
curl http://<external ip>
```

Cela vous donnera une sortie telle que:

```
Request served by hello-world-<PARTIE QUI CHANGE>

HTTP/1.1 GET /

Host: 10.101.132.184
Accept: */*
User-Agent: curl/7.88.1
```

Si `<PARTIE QUI CHANGE>` est différente d'une requête à l'autre, c'est que votre mission est réussie !

3 À l'aide !

Basez-vous sur le cours ainsi que les documentations suivantes pour y parvenir:

- `https://kubernetes.io/fr/docs/concepts/services-networking/service/`
- `https://kubernetes.io/docs/concepts/workloads/pods/`
- `https://kubernetes.io/docs/concepts/workloads/controllers/deployment/`