TPs : Formation Kubernetes, mise en œuvre

Objectifs :

1. Création de cluster
2. Administration de cluster
3. Configurer et étendre des clusters
4. Déployer des charges de travail sur des clusters : Deployments, Services, Statefulsets
5. Créer des Secrets, ConfigMaps, des Volumes Persistants (PV) et Volumes Persistants Claim (PVC), des StorageClass …
6. Configurer des restrictions d'utilisation de ressources, sur les Nodes, Pods, Namespaces
7. Configurer le stockage | le réseau | la sécurité d'un cluster

1. Les solutions d'installation

1.1. Installation de Microk8s sur une machine Ubuntu

Prérequis : Snap

Installer Snap sur Linux.
Pour les versions d'Ubuntu entre 14.04 LTS (Trusty Tahr) et 15.10 (Wily Werewolf), ainsi que pour les versions Ubuntu qui n'incluent pas snap par défaut, snap peut être installé à partir du Centre logiciel Ubuntu en recherchant snapd.

```
$ sudo apt update
$ sudo apt install snapd
```

Déconnectez-vous et reconnectez-vous, ou redémarrez votre système pour vous assurer que les chemins de snap sont correctement mis à jour.

Pour tester votre système, installez le composant logiciel enfichable hello-world et assurez-vous qu'il fonctionne correctement :

```
$ sudo snap install hello-world
hello-world 6.4 from Canonical✓ installed
```

```
$ hello-world

Hello World!
```

Snap est maintenant installé et prêt à fonctionner !

Installer Microk8s

```
$ sudo snap install microk8s –classic


# Pour connaitre le status de votre microk8s

$ sudo microk8s status

# Ou avec l'option --wait-ready

$ sudo microk8s status --wait-ready


# Pour ne plus avoir à saisir chaque fois sudo, faire :

$ sudo usermod -a -G microk8s $USER

$ sudo chown -f -R $USER ~/.kube

$ source .bashrc

# si changement ne s'opère pas, ouvrir un autre shell


# Pour créer un raccourci, de la commande "microk8s kubectl" ou de "
microk8s.kubectl", faire :

$ sudo snap alias microk8s.kubectl kubectl

# Si vous êtes curieux, vous pouvez diriger la sortie de la commande
"kubectl completion bash" et grep pour "__kubectl" pour voir une liste des
fonctions que vous allez/pouvez renommer.

$ kubectl completion bash | grep __kubectl
```

```
# Pour créer un namespaces, faire :

$ kubectl create namespace development

# Créer un deployment dans le namespace development

$ kubectl create deployment cattle --image=k8s.gcr.io/serve_hostname -n=
development
```

1.2. Les plugins disponibles de Microk8s

```
# Avec la commande « microk8s status », on peut voir le status de
Microk8s : savoir s'il est en running » ou non.

# Mais aussi, on obtient comme résultat la liste de tous les plugins
(« addons ») disponibles sur Microk8s, avec ceux qui sont déjà activés
(« enabled ») de ceux qui ne le sont pas (« disabled »).


ubuntu@kube-node:~/manifests$ microk8s status

microk8s is running

high-availability: no

  datastore master nodes: 127.0.0.1:19001

  datastore standby nodes: none

addons:

  enabled:

    dashboard            # The Kubernetes dashboard

    dns                  # CoreDNS

    fluentd              # Elasticsearch-Fluentd-Kibana logging and

                           monitoring

    ha-cluster           # Configure high availability on the current node

    ingress              # Ingress controller for external access

    metrics-server       # K8s Metrics Server for API access to service

                           metrics

    portainer            # Portainer UI for your Kubernetes cluster

    rbac                 # Role-Based Access Control for authorisation

    storage              # Storage class; allocates storage from host

                           directory

  disabled:

    ambassador           # Ambassador API Gateway and Ingress

    cilium               # SDN, fast with full network policy

    gpu                  # Automatic enablement of Nvidia CUDA

    helm                 # Helm 2 - the package manager for Kubernetes

    helm3                # Helm 3 - Kubernetes package manager

    host-access          # Allow Pods connecting to Host services smoothly

    istio                # Core Istio service mesh services

    jaeger               # Kubernetes Jaeger operator with its simple
```

```
                          config
    keda                # Kubernetes-based Event Driven Autoscaling
    knative             # The Knative framework on Kubernetes.
    kubeflow            # Kubeflow for easy ML deployments
    linkerd             # Linkerd is a service mesh for Kubernetes and
                          other frameworks
    metallb             # Loadbalancer for your Kubernetes cluster
    multus              # Multus CNI enables attaching multiple network
                          interfaces to pods
    prometheus          # Prometheus operator for monitoring and logging
    registry            # Private image registry exposed on
                          localhost:32000
    traefik             # traefik Ingress controller for external access
```

1.3. Installation de Cluster Microk8s

Voir la section : « Déploiement d'un cluster Kubernetes (Micrk8s) »

2. Utilisation de Kubernetes/Microk8s

Des fichiers .yaml sont disponibles dans le répertoire partagé dédié aux travaux pratiques.

Ce document est aussi disponible dans ce même répertoire, avec d'autres documents pour le bon déroulement des TPs.

2.1. Premiers essais

Création déploiement et service :

```
$ kubectl create deployment nginx --image=docker.io/library/nginx:latest
$ kubectl get deployments
$ kubectl expose deployment/nginx --type="NodePort" --port 80
$ kubectl get svc
$ kubectl describe service nginx
$ kubectl scale deployments/nginx --replicas=5
$ kubectl get deployments.apps
```

```
$ kubectl get po

$ sudo kubectl set image deployments/nginx
nginx=docker.io/library/nginx:nginx-v3
```

Scale up :

Augmenter le nombre de répliques du déploiement par :

```
$ kubectl scale deployments/nginx --replicas=4

$ kubectl get deployments/nginx
```

Nettoyage :

Si une application n'est plus nécessaire, il suffit de la supprimer.

On peut toutefois ramener le replicas de l'application à 0. Ce qui permet de rendre disponible cette application très rapidement.

Le nettoyage est enfin fait avec :

```
$ kubectl delete deployments/nginx

$ kubectl delete service nginx
```

2.2. Déploiement d'une plateforme de test

i. Validation du poste de travail
ii. Mise en œuvre Microk8s (voir guide d'installation Microk8s sur votre poste de travail)
iii. Tests de fonctionnement (faire l'exercice des premiers essais)

3. Les fichiers descriptifs / manifests

i. Voici quelques exemples de fichiers manifests qui vous permettent de créer des déploiements et services.
ii. Tenter de comprendre votre fichier manifest.

Des questions après création de déploiements et services ?

4. Objets de base

4.1. Avec des fichiers manifests, créer les objets suivants :

i. pod
ii. Deployment
iii. Service
iv. HostPath
v. Persistent volume
vi. Configmap
vii. Secrets

4.2. Refaire l'exercice en déployant les applications « mongoexpress » et « mongodb ».

Mongo-express est une interface d'administration MongoDB basée sur le Web écrite avec Node.js et Express.

Voici un digramme que nous voulons implémenter avec Kubernetes :

Nous pouvons obtenir les images du docker à partir de Dockerhub - mongo / mongo-express.

MongoDB Secret / Deployment / et internal Service

Les secrets (nom d'utilisateur / mot de passe MongoDB) doivent être en place avant tout déploiement MongoDB car les pods MongoDB doivent accéder aux secrets. Alors, créons les secrets avec un encodage en base64 :

```
$ echo -n 'username' | base64
dXNlcm5hbWU=

$ echo -n 'password' | base64
cGFzc3dvcmQ=
```

Voir l'objet « secret » de mongo dans le fichier **mongodb-secret.yaml**
Le déploiement de mongo sera fait avec le fichier **mongodb-deployment.yaml**. Son service interne associé avec le fichier manifest **mongodb-deployment-service.yaml**.

Que remarquez-vous entre les fichiers descriptifs

i. mongodb-deployment.yaml et
ii. mongodb-deployment-service.yaml

Notez que dans la définition du service mongo-service, le type n'est pas spécifié.
Nous utilisons le type de service ClusterIP qui est le type de service par défaut.

Vous remarquerez que les points de terminaison du service mongodb ont la même adresse IP que celle du pod MongoDB avec un port 27017.

Mongo express ConfigMap / Deployment et / external Service

De même avec les secrets, une configmap doit déjà être dans notre cluster pour que notre mongo-express puisse l'utiliser comme référence.
Notez ici que le **database_url** est le même que le nom du service mongodb, mongodb-service

i. Créez le configmap
ii. Le deployment du mongo express et
iii. le service associé.

Remarque :
Le containerPort est le port ouvert sur le serveur MongoDB pour monngo-express.

Le LoadBalancer attribue à un service une adresse IP externe afin qu'il reçoive des requêtes externes.

Contrairement au service interne qui a un Cluster-IP (interne) par défaut, le type de service LoadBalancer a une IP supplémentaire (externe). Comme le montre la sortie de la commande

```
$ kubectl get svc | grep mongo-express-service
```

, il est actuellement en <pending>, ce qui signifie qu'il n'est pas encore affecté.
Ceci est spécifique à Microk8s, sinon, il devrait en avoir un.
Par exemple, dans des environnements tels que AWS ou GCP, etc.

Observations :

```
$ kubectl describe service mongodb-service
$ kubectl get pod -o wide
$ kubectl get all | grep mongo
$ kubectl logs mongo-express-deployment-78fcf796b8-f6mwd
$ kubectl get svc
```

Habituellement, le type de service **LoadBalancer** expose le service en externe à l'aide de l'équilibreur de charge d'un fournisseur de cloud. Pour Microk8s, avec le plugin ingress activé, on peut accéder au service à l'externe sur le port spécifique de l'objet ingress créé.
Ce point sera abordé plus en profondeur dans la suite du cours.

5. Déploiement d'un cluster Kubernetes (Microk8s)

Nous allons créer des machines virtuelles qui vont constituer les nœuds de notre cluster, avec Multipass.

Multipass : c'est « VM Ubuntu à la demande pour n'importe quel poste de travail ».
Il permet d'obtenir une machine virtuelle Ubuntu instantanée avec une seule commande. Multipass peut lancer et exécuter des machines virtuelles et les configurer avec cloud-init comme un cloud public. Vous pouvez donc prototyper gratuitement le lancement de votre cloud localement.

Installer Multipass sur Linux (Ubuntu) :

```
$ sudo snap install Multipass
```

Chaque VM Multipass va tourner sur un Ubuntu 18.04 LTS ou Ubuntu 20.04 LTS.

Créer une VM Ubuntu avec Multipass :

```
$ multipass launch --name kube-node --mem 6G --disk 50G –cpus 2
```

Installer Docker (optionnel) car Microk8s vient avec Containerd (qui est un conteneur runtime)
Vérifier la visibilité réseau

Installer Microk8s sur chaque VM créée
(Voir guide d'installation de microk8s)

```
$ multipass exec kube-node -- sudo snap install microk8s –classic
# Ouvrir le shell de votre VM :
$ multipass shell kube-node
```

Joindre les différents nœuds par :
Déploiement : Ajouter un nœud worker au master-node
Lancer « microk8s add-node » sur le master (# Noter bien l'output de cette commande).

```
Join node with:
microk8s join ip-172-31-20-243:25000/DDOkUupkmaBezNnMheTBqFYHLWINGDbf

If the node you are adding is not reachable through the default
interface you can use one of the following:

microk8s join 10.1.84.0:25000/DDOkUupkmaBezNnMheTBqFYHLWINGDbf
microk8s join 10.22.254.77:25000/DDOkUupkmaBezNnMheTBqFYHLWINGDbf
```

Lancer microk8s join ip … sur le worker-node :

```
microk8s join ip-172-31-20-243:25000/DDOkUupkmaBezNnMheTBqFYHLWINGDbf
```

```
# faire sur chaque nœud :
$ kubectl get nodes
```

Observations :

6. Exploiter Kubernetes

Le **kubelet** utilise des « liveness probes » (sondes de vivacité) pour savoir quand redémarrer un conteneur.

Par exemple, les liveness probes peuvent intercepter un blocage, où une application est en cours d'exécution, mais incapable de progresser. Le redémarrage d'un conteneur dans un tel état peut aider à rendre l'application plus disponible malgré les bogues.

Le kubelet utilise des « readiness probes » (sondes de disponibilité) pour savoir quand un conteneur est prêt à commencer à accepter du trafic.

Un pod est considéré comme prêt lorsque tous ses conteneurs sont prêts. Une utilisation de ce signal est de contrôler quels pods sont utilisés comme backends pour les services. Lorsqu'un pod n'est pas prêt, il est supprimé des équilibreurs de charge de service.

Le kubelet utilise des « startup probes » (sondes de démarrage) pour savoir quand une application conteneur a démarré. Si une telle probe est configurée, elle désactive les vérifications de liveness et de readiness jusqu'à ce qu'elle réussisse, en s'assurant que ces probes n'interfèrent pas avec le démarrage de l'application.

Cela peut être utilisé pour adopter des contrôles de liveness sur les conteneurs à démarrage lent, en évitant qu'ils ne soient tués par le kubelet avant qu'ils ne soient opérationnels.

Déploiement d'une application :

i. Mise en place d'affinité sur un nœud
ii. Contrainte sur une machine
iii. Mise en œuvre de healthcheck

7. Création de conteneur personnalisé

Conteneuriser un service python :

Ci-dessous la structure de notre programme :

```
app
├── requirements.txt
└── src
    └── server.py
```

server.py est un simple service Flask.

```
from flask import Flask
```

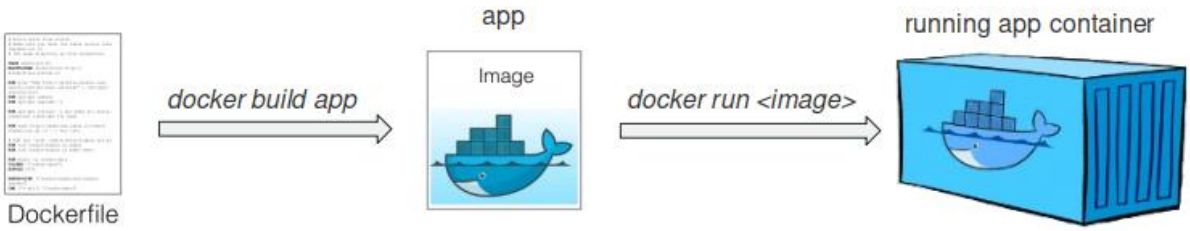
```
server = Flask(__name__)


@server.route("/")
 def hello():
    return "Hello World!"
if __name__ == "__main__":
   server.run(host='0.0.0.0')
```

Quant au fichier requirements.txt, il contient les dépendances à installer avant l'exécution du programme.

requirements.txt

```
Flask==1.1.1
```

DockerFile, en secours pour la conteunairisation :



Dockerfile

```
# set base image (host OS)
FROM python:3.8

# set the working directory in the container
WORKDIR /code

# copy the dependencies file to the working directory
COPY requirements.txt .

# install dependencies
RUN pip install -r requirements.txt
```

```
# copy the content of the local src directory to the working directory
COPY src/ .

# command to run on container start
CMD [ "python", "./server.py" ]
```

Build et exécution de l'image

```
$ docker build -t myimage .
$ docker images
$ docker ps
$ docker run -d -p 5000:5000 myimage
$ curl http://localhost:5000
```

Création et automatisation d'images personnalisées :

i. Mise en œuvre d'un registre privé
ii. Créer un conteneur Mariadb
iii. Créer un conteneur Jboss

8. Kubernetes en production

Ingress / Ingress contrôler :

i. Mise en œuvre du frontal Ingress
ii. Autoscaling du frontal Web

Gestion des accès :

i. Création d'un namespace
ii. Ajout d'un utilisateur à ce namespace
iii. Gérer les droits d'accès d'une application