TP Kubernetes

Au cours de ce TP, vous allez installer, configurer et administrer un cluster Kubernetes. Vous allez créer et manipuler différentes objets Kubernetes (Controllers, Pods, Volumes etc).

Creation de l'infrastructure

Dans cette section vous devez créer trois machines virtuelles dans OpenStack avec les caractéristiques suivantes:

- OS Ubuntu 18.04
- 2 vCPU
- 4GB RAM

Une VM sera le Master Node et deux autres seront des Worker Nodes.

Installation et Validation

Dans cette section vous allez installer et valider le fonctionnement du cluster Kubernetes. Votre cluster aura un Master Node et deux Worker Nodes.

Installation

Sur tous les VMs

- Ajoutez la clé GPG et le repository Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

- Ajoutez la clé GPG et le repository Kubernetes

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo apt-key add -
sudo add-apt-repository "deb https://apt.kubernetes.io/
kubernetes-xenial main"
```

- Installez les packages Kubernetes et Docker

```
sudo apt-get install -y docker-ce=18.06.1~ce~3-0~ubuntu
kubelet=1.16.0-00 kubeadm=1.16.0-00 kubectl=1.16.0-00
```

  - Dans cette section vous installez la version 1.16.0 du Kubernetes. Dans les sections suivantes vous allez mettre-à-jour votre cluster.

- Bloquez la mise-à-jour automatique des packages installés précédemment

```
sudo apt-mark hold docker-ce kubelet kubeadm kubectl
```

- Ajoutez la variable NO_PROXY dans les variables d'environnement
  - Ajoutez la ligne dans /etc/environment

```
NO_PROXY=127.0.0.1,10.244.0.0/16,10.96.0.0/12,192.168.0.0/16
```

    - 10.244.0.0/16 - la plage des adresses des PODS dans votre cluster
    - 10.96.0.0/12 - la plage des adresses systeme de Kubernetes

- Configurez Daemon Docker pour l'utilisation du Proxy
  - sudo mkdir /etc/systemd/system/docker.service.d
  - sudo vi /etc/systemd/system/docker.service.d/http-proxy.conf

```
[Service]
Environment="HTTP_PROXY=http://proxy.univ-lyon1.fr:3128/"
Environment="HTTPS_PROXY=http://proxy.univ-lyon1.fr:3128/"
Environment="NO_PROXY=127.0.0.1,10.244.0.0/16,10.96.0.0/12,192
.168.0.0/16"
```

  - sudo systemctl daemon-reload
  - sudo systemctl restart docker

- Testez si Docker arrive a télécharger et lancer un container

Initialization du Cluster

Une fois que les packages Kubernetes et Docker sont installées, vous pouvez initialiser le cluster et installer un CNI (Container Network Interface). Le processus d'initialisation du cluster est très simple. Vous allez utiliser **kubeadm** pour initialiser votre cluster et **flannel** comme le CNI.

Sur le noeud master

- Initialisez votre cluster

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

  - **Attention!** Mémorisez bien le token donné par cette commande, ce token sera utilisé par vos noeuds workers pour rejoindre le cluster.
  - Qu'est-ce que l'option --pod-network-cidr permet de faire?
  - Essayez de comprendre les étapes de l'initialisation d'un cluster Kubernetes.

- Configurez l'outil d'administration **kubectl**

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

  - Qu'est-ce que cet outil permet de faire?

Sur les noeuds worker

- Ajoutez les Workers dans cluster

```
sudo kubeadm join [join_token]
```

  - Le token a été donné par la commande **kubeadm init**.

Sur le noeud Master

- Verifiez l'etat des nodes

```
sudo kubectl get nodes
sudo kubectl describe nodes
```

  - Vérifiez l'état des noeuds.
  - Pourquoi l'etat des noeuds est "NotReady"?

- Installez CNI (Container network interface)

```
sudo kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documenta
tion/kube-flannel.yml
sudo kubectl get pods -n kube-system
```

  - Flannel crée un objet de type “DaemonSet”. Pourquoi un objet de type “DaemonSet” est créé?

- Re-verifiez l’etat des nodes
  - Qu'avez-vous remarqué?

- Decrivez les pods du namespace kube-system

```
sudo kubectl get pods -n kube-system
```

Validation de l’installation

- Creez un deployment nginx

```
sudo kubectl create deployment --image=nginx nginx
```

- Vérifiez que le pod est bien lancé et que le déploiement a été bien créé

```
sudo kubectl get pods
sudo kubectl get deployments
```

- Créez un port forward et vérifiez son fonctionnement

```
sudo kubectl port-forward PODNAME 8081:80 &
curl 127.0.0.1:8081
```

  - Quand faut-il utiliser un port-forward?

- Visualisez des logs du Pod

```
sudo kubectl logs PODNAME
```

- Créez et vérifiez le service de type NodePort qui expose le déploiement

```
sudo kubectl expose deployment nginx --port 80 --type NodePort
sudo kubectl get services
curl -I 127.0.0.1:NODE_PORT
```

- Que permet de faire un service dans K8s?
- Que fait un service de type “NodePort”?

La mise-à-jour du cluster

Dans cette section vous devez mettre-à jour votre cluster K8s. Pour effectuer cela vous allez utiliser l’outil **kubeadm.**

- Que permet de faire **kubeadm**?

Vérification de la version du cluster

Avant de commencer la mise-à-jour, il faut détecter la version actuelle des éléments de cluster. Puis il faudra trouver quelle est la dernière version stable de K8s.

- Détectez la version de **kubelet** sur les noeuds

```
sudo kubectl get nodes
```

- Détectez la version d’API du client et serveur

```
sudo kubectl version --short
```

- Detectez la version de **kubeadm**

```
sudo kubeadm version
```

Au moment de la rédaction de ce TP, la dernière version stable de Kubernetes est la **v1.17.3**.

La mise-à-jour du cluster

Maintenant vous pouvez commencer la mise-à-jour de votre cluster.

- Exportez les variables d'environnement suivantes

```
export VERSION=v1.17.3
export ARCH=amd64
```

- Récupérez et installez de la nouvelle version de **kubeadm**

```
curl -sSL
https://dl.k8s.io/release/${VERSION}/bin/linux/${ARCH}/kubeadm >
kubeadm
sudo install -o root -g root -m 0755 ./kubeadm /usr/bin/kubeadm
sudo kubeadm version
```

- Planifiez la mise-à-jour

```
sudo kubeadm upgrade plan
```

  - Que vous affiche cette commande?
  - Si toute l'information affiché vous semble correcte, vous pouvez effectuer la mise-à-jour du cluster

- La mise-à-jour du cluster

```
sudo kubeadm upgrade apply v1.17.3
```

- Vérifiez les versions de **kubelet** sur les noeuds

```
sudo kubectl get nodes
```

  - Que pouvez-vous constater? Pourquoi cela est arrivé?

- Mettez-à-jour le **kubelet**
  - Exportez les variables d'environnement suivantes

```
export VERSION=v1.17.3
export ARCH=amd64
```

  - Installez la nouvelle version de **kubelet**

```
curl -sSL
https://dl.k8s.io/release/${VERSION}/bin/linux/${ARCH}
/kubelet > kubelet
sudo install -o root -g root -m 0755 ./kubelet
/usr/bin/kubelet
sudo systemctl restart kubelet.service
```

  - Vérifiez la mise-à-jour de **kubelet**

```
sudo kubectl get nodes
```

- Verifiez la version du **kubectl**

```
sudo kubectl version
```

- Que pouvez vous constater?

- Mettez-à-jour le **kubectl**

```
curl -sSL
https://dl.k8s.io/release/${VERSION}/bin/linux/${ARCH}/kubectl > kubectl
sudo install -o root -g root -m 0755 ./kubectl /usr/bin/kubectl
sudo kubectl version
```

- Mettez-à-jour tous les noeuds du cluster.

Bravo! Vous avez mis-à-jour votre cluster sans aucune interruption de service!

Utilisation du cluster

Maintenant vous allez déployer quelques applications sur votre cluster.

Creation d'un pod

Dans cette section vous allez créer une Pod qui est un objet de base de Kubernetes.
Pour créer des objets Kubernetes, vous devez créer un fichier de description en format yaml.

Un exemple:

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    service: web
spec:
   containers:
   - name: nginx
     image: nginx
     ports:
        - containerPort: 80
```

Ce fichier décrit un pod qui a les caractéristiques suivantes:

- **Nom**: nginx-pod
- **Label**: service = web
- **Image et nom du conteneur**: nginx
- **Le port du conteneur**: 80

- Créez cet objet dans le cluster

```
sudo kubectl create -f pod.yaml
```

- Vérifiez si le pod a été bien créé

```
sudo kubectl get pods
```

Creation d'un Service

Vous avez créé un Pod mais il n'est pas accessible. Pour rendre le Pod accessible il faut créer un service.

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    service: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Ce fichier décrit un service qui a les caractéristiques suivantes:

- **Nom**: nginx-service
- **Type:** NodePort
- **Selector:** sélectionne les objets avec un label (service=web)
- **Port**: ecoute sur le port 80
- **Target Port:** le port écouté par l'objet sélectionné

- Créez cet objet dans le cluster

```
sudo kubectl create -f service.yaml
```

- Détectez quel port est exposé sur les noeuds pour atteindre ce

service

```
sudo kubectl get services
```

- Affichez des endpoints du service

```
sudo kubectl get endpoints
```

  - Qu'est-ce qui est affiché dans la liste “ENDPOINTS”?

- Vérifiez que le pod est bien accessible via le port exposé par le service

```
curl 127.0.0.1:[node_port]
```

  - À partir de quel nœud le service est-il accessible?

Creation d’un deployment

Vous avez créé un Pod avec le conteneur Nginx et un Service qui expose ce Pod à l'extérieur.
Kubernetes vous permet de faire bien plus que cela. On souhaite de déployer des applications hautement disponibles avec des mécanismes de mises-à-jour déclaratives et rollbacks. Pour bénéficier de ces mécanismes, il faut utiliser les objets de type “Deployment”.

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: web
  replicas: 3
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
          - containerPort: 80
```

- Créez cet objet dans le cluster

```
sudo kubectl create -f deployment.yaml --record
```

  - Pourquoi on utilise l'option --record?

- Vous pouvez suivre le processus de déploiement grâce à la commande

```
sudo kubectl rollout status deployments nginx-deployment
```

- Scalez votre déploiement jusqu'à 6 replicas

```
sudo kubectl scale deployment nginx-deployment --replicas=6
```

  - N'hesitez pas a utiliser la commande **kubectl describe** pour voir les détails de votre déploiement
- Vérifiez que votre déploiement a lancé un bon nombre des replicas

```
sudo kubectl get deployments
```

Nous avons créé un objet "Deployment" qui crée 3 pods. Ensuite, nous avons augmenté le nombre des pods à la volée.
Pour que le déploiement soit accessible, il faut créer un service qui exposera ce déploiement.

deployment-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

- Créez cet objet dans le cluster

```
sudo kubectl create -f deployment-service.yaml
```

  - Quel est l'intérêt de la section "selector" dans ce fichier yaml?

- Détectez quel port est exposé sur les noeuds pour atteindre le service

```
sudo kubectl get services
```

- Affichez des endpoints du service

```
sudo kubectl get endpoints
```

  - Que remarquez-vous?

- Vérifiez que le déploiement est bien accessible par un port de noeuds

```
curl -I 127.0.0.1:[node_port]
```

Rolling Updates

Imaginez que les développeurs aient publié une nouvelle version de l'application et que vous êtes responsable de la mise-à-jour du déploiement.
Dans notre exemple, nous allons changer la version de l'image nginx.

Pour suivre le processus de la mise-à-jour du déploiement, nous allons ralentir ce processus. Le déploiement sera suspendu pendant 10 secondes après le déploiement de chaque pod.

```
sudo kubectl patch deployment nginx-deployment -p '{"spec": {"minReadySeconds": 10}}'
```

Mettre-à-jour un déploiement

Vous avez deux moyens de mettre-à-jour votre déploiement:

- En modifiant le fichier yaml

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: web
  replicas: 3
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
```

```
      - name: nginx
        image: nginx:1.16.0
        ports:
          - containerPort: 80
```

  - Appliquer les changements

```
sudo kubectl apply -f deployment.yaml
```

- Inline (en utilisant la ligne de commande)

```
sudo kubectl set image deployments/nginx-deployment
nginx=nginx:1.16.0 --v 6
```

- Mettez-à-jour le déploiement et vérifiez que l'application a été bien mise-à-jour

```
curl -I 127.0.0.1:[node_port]
```

  - Exécutez la commande plusieurs fois
  - Comme on a ralenti les déploiements au début de cette section, vous pouvez suivre en temps réel le déploiement de la nouvelle version

```
watch -n 1 curl -I 127.0.0.1:[node_port]
```

Comme vous voyez la mise-à-jour est passé sans aucune interruption de service.

Rollbacks

Imaginez que vous avez mis-à-jour une application en production et que cette application ne fonctionne plus ou la nouvelle version n'est plus compatible avec les autres éléments de la stack applicative. Kubernetes vous offre une possibilité d'effectuer un Rollback.

- Effectuez le rollback de deploiement nginx-deployment

```
sudo kubectl rollout undo deployments nginx-deployment
```

- Suivez le Rollback

```
watch -n 1 curl -I 127.0.0.1:[node_port]
```

Vous pouvez effectuer le Rollback a une version du déploiement précise. Pour cela il faut récupérer l'historique des déploiements.

- Récupérez l'historique des déploiements

```
sudo kubectl rollout history deployment nginx-deployment
```

  - cela est possible grâce à l'option **--record** qu'on a rajouté lorsque on a créé le déploiement

Vous pouvez revenir vers une version particulière du déploiement en utilisant l'option --to-revision.

- Revenez à la révision 2 du déploiement
  - Quelle commande utiliserez-vous?

Volumes

Parfois vos Pods auront besoin d'avoir un stockage persistent. Dans cette section vous allez créer et manipuler des volumes persistants.

Création d'un volume et son attribution à un pod se passe en plusieurs étapes.

Premièrement, il faudra créer un objet "Persistent Volume". Dans le cadre de ce TP, nous allons créer un volume "local" (un répertoire sur le host). Cette tâche est principalement effectué par l'administrateur du cluster.

pv.yaml

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

- Créez cet objet dans le cluster

```
sudo kubectl create -f pv.yaml
```

- Quel est son statut après la creation?

```
sudo kubectl get pv
```

Vous ne pouvez pas utiliser directement le volume dans votre Pod. Kubernetes ajoute une couche d'abstraction - l'objet "PersistentVolumeClaim". Cette abstraction permet de découpler les volumes qui sont configurés et rendus disponibles par les administrateurs et des demandes de l'espace de stockage faites par les développeurs pour leurs applications.

Pour créer une demande d'un volume nous allons créer l'objet "PersistentVolumeClaim". Nous allons demander un volume qui a minimum 3 Giga de stockage et qui peut fournir un accès en lecture-écriture à au moins un nœud.

pvc.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

- Créez cet objet dans le cluster

```
sudo kubectl create -f pvc.yaml
```

- Quel est le statut du volume persistent après la création de la claim?

```
sudo kubectl get pv
```

- Est-ce que deux claims peuvent utiliser le même volume persistent?

Finalement, vous pouvez attribuer votre "PersistantVolumeClaim" à un Pod.

pvc-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
    - image: mongo
      name: mongodb
      volumeMounts:
        - name: mongodb-data
          mountPath: /data/db
      ports:
        - containerPort: 27017
          protocol: TCP
  volumes:
    - name: mongodb-data
      persistentVolumeClaim:
       claimName: task-pv-claim
```

- Créez cet objet dans le cluster

  ```
  sudo kubectl create -f pvc-pod.yaml
  ```

- Vérifiez que votre pod est lancé

  ```
  sudo kubectl get pods
  ```

- Trouvez un moyen de vérifier que le volume persistent fonctionne

Liveness et Readiness

Kubernetes grâce aux sondes Liveness peut automatiquement vérifier si vos applications répondent aux requêtes utilisateurs. Si votre application ne répond pas aux requêtes K8s détecte cela et redémarre ou recrée le containeur. Kubernetes permet de retenir le trafic entrant jusqu'à le démarrage complet de votre service avec les sondes Readiness. Dans cette section vous allez déployer des pods avec les sondes de Liveness et Readiness.

Liveness probes

Liveness probes permettent de vérifier si un conteneur est en vie et répond bien aux requêtes.

liveness-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: liveness
spec:
  containers:
  - image: nginx
    name: nginx
    livenessProbe:
       httpGet:
         path: /
         port: 80
```

- Créez cet objet dans le cluster

  ```
  sudo kubectl create -f liveness-pod.yaml
  ```

- Déployez un Pod nginx avec la liveness probe qui échoue
  - Par exemple: qui requete un path nginx nonexistent

- Surveillez les events et le comportement du Pod

  ```
  sudo kubectl get pods
  sudo kubectl describe pod POD_NAME
  ```

- Quel est le statut du Pod?

- Combien de fois Kubernetes essaye de redémarrer le Pod avant de conclure qu'il est défaillant?

Readiness probe

Readiness probes est un mécanisme dans K8s qui vous permet de vérifier si le conteneur est prêt à recevoir les demandes des clients. Si la vérification échoue, le conteneur ne sera pas redémarré, mais le trafic ne sera pas redirigé vers ce conteneur.

readiness.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-readiness
spec:
  type: NodePort
  ports:
```

```
  - port: 80
    targetPort: 80
  selector:
    app: nginx-readiness
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-good
  labels:
      app: nginx-readiness
spec:
  containers:
  - name: nginx
    image: nginx
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-nogood
  labels:
      app: nginx-readiness
spec:
  containers:
  - name: nginx
    image: nginx:1.222
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

- Créez cet objet dans le cluster

```
sudo kubectl create -f readiness.yaml
```

- Etudiez le comportement des pods avec readiness check
  - Surveillez les pods

    ```
    sudo kubectl get pods
    ```

  - Surveillez la liste des endpoints du service

    ```
    sudo kubectl get endpoints
    ```

  - Que remarquez-vous?

- Est-ce que le service repond aux requetes?
  - Comment pouvez-vous expliquer un tel comportement?
- Trouvez et corrigez l’erreur
  - Utilisez la commande

    ```
    sudo kubectl edit pod nginx-nogood
    ```

- Surveillez l'état des pods et la liste des endpoints du service
  - Que remarquez-vous?

Vous avez fini le TP! Bravo!