TP Kubernetes

Au cours de ce TP, vous allez installer, configurer et administrer un cluster K8S. Puis, vous allez manipuler différents objets K8S (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 20.04.1
- 2 vCPU
- 4GB RAM

Une machine 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. Nous allons utiliser Docker comme Container Runtime.

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=5:19.03.14~3-0~ubuntu-focal
docker-ce-cli=5:19.03.14~3-0~ubuntu-focal containerd.io
kubelet=1.19.0-00 kubeadm=1.19.0-00 kubectl=1.19.0-00
```

  - 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 docker-ce-cli 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

```
$ sudo docker run hello-world
```

- Redémarrez tous les machines

Initialization du Cluster

Une fois les packages Kubernetes et Docker installés, 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 nœuds workers pour rejoindre le cluster.
  - Qu'est-ce que l'option "--pod-network-cidr" permet de faire?
  - Essayez de comprendre les étapes d'initialisation du 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**, lors de l'initialisation du cluster.

Sur le noeud Master

- Verifiez l'etat des nodes

```
$ kubectl get nodes
$ kubectl describe nodes
```

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

- Installez CNI (Container network interface)

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

  - Pour initialiser Flannel, Kubernetes crée un objet de type “DaemonSet”. Pourquoi un objet de type “DaemonSet” est-il créé?

- Re-vérifiez l’état des nœuds
  - Qu'avez-vous remarqué?

- Decrivez les pods du namespace kube-system

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

Validation de l’installation

- Creez un deployment nginx

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

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

```
$ kubectl get pods
$ kubectl get deployments
```

- Créez un port forward et vérifiez son fonctionnement

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

  - Que permet de faire un port-forward?

- Visualisez des logs du Pod

```
$ kubectl logs PODNAME
```

- Créez et vérifiez un service de type NodePort

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

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

La mise à jour du cluster

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

- Que permet de faire l'outil **kubeadm**?

Préparation de la mise à jour

Avant de commencer la mise à jour du cluster, il faut vérifier la version actuelle des éléments de votre cluster. Ensuite, vous devez trouver quelle est la dernière version stable de K8S.

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

```
$ kubectl get nodes
```

- Vérifiez la version d'API du client et serveur

```
$ kubectl version --short
```

- Vérifiez la version de **kubeadm**

```
$ kubeadm version
```

Au moment de la rédaction de ce TP, la dernière version stable de Kubernetes est la **v1.20.1**. Vous allez donc mettre à jour le cluster vers cette version.

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=v1.20.1
$ 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
```

- Exécutez la commande de planification de mise à jour

```
$ sudo kubeadm upgrade plan
```

  - Que vous montre 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 v1.20.1
```

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

```
$ kubectl get nodes
```

  - Que pouvez-vous constater?

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

```
$ export VERSION=v1.20.1
$ export ARCH=amd64
```

  - Installez la nouvelle version de **kubelet**

```
$ curl -sSL
https://dl.k8s.io/release/${VERSION}/bin/linux/${ARCH}/kubel
et > 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**

```
$ kubectl get nodes
```

- Vérifiez la version du **kubectl**

```
$ 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
$ 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

Dans cette section, vous allez déployer quelques applications sur votre cluster.

Creation d'un pod

Pour créer des objets Kubernetes, vous devez créer un fichier de description en format yaml.
Vous commencerez par créer une Pod qui est un objet de base de K8S.

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

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

- Vérifiez si le pod a été bien créé

```
$ kubectl get pods
```

Creation d'un Service

Vous avez créé un Pod mais pour l'instant il n'est pas accessible depuis l'extérieur. 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:** service=web
- **Port**: écoute sur le port 80
- **Target Port:** le port écouté par l'objet sélectionné

- Créez cet objet dans le cluster

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

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

```
$ kubectl get services
```

- Affichez des endpoints du service

```
$ 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

Dans la section précédente, vous avez créé un Pod avec le conteneur Nginx et un Service qui expose ce Pod vers l'extérieur.
Kubernetes vous permet de faire bien plus que cela. Dans cette section, vous allez déployer des applications hautement disponibles avec des mécanismes de mises à jour déclaratives et rollbacks.
Pour bénéficier de ces mécanismes, il faut créer un objet 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

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

  - Pourquoi utilise-t-on l’option “--record”?

- Vous pouvez suivre le processus de déploiement avec la commande

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

- Mettez à l'échelle votre déploiement pour avoir 6 replicas

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

  - N'hésitez pas à utiliser la commande **kubectl describe** pour afficher une description détaillée de votre déploiement

- Vérifiez que votre déploiement a lancé un bon nombre des replicas

```
$ kubectl get deployments
```

Nous avons créé un objet de type “Deployment” qui crée et maintient un nombre des Pods demandées. Ensuite, nous avons augmenté le nombre de Pods à la volée.
“Deployment” peut être vu comme un regroupement des Pods dont le nombre est garanti par K8S.
Comme pour un Pod individuel, pour qu'un “Deployment” soit accessible, un service doit être créé.

deployment-service.yaml

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

- Créez cet objet dans le cluster

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

  - Quel est l'intérêt de la section “selector” dans le fichier yaml?

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

```
$ kubectl get services
```

- Affichez des endpoints du service

```
$ kubectl get endpoints
```

  - Que remarquez-vous?

- Vérifiez que le déploiement est bien accessible depuis n’importe quel noeud

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

Rolling Updates

Imaginez que des développeurs ont publié une nouvelle version de l'application et que vous êtes responsable de la mise à jour.

Pour simuler ce scénario, nous allons changer la version de l’image nginx.

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

```
$ 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 de l’objet

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
```

  - Pour appliquer les changements

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

- Inline (en utilisant la ligne de commande)

```
$ 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 nous avons ralenti le processus de déploiement 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 passée 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 que la nouvelle version n'est plus compatible avec des autres éléments de la stack applicative. Kubernetes vous offre la possibilité d'effectuer un Rollback.

- Effectuez le rollback de déploiement nginx-deployment

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

- Suivez le processus de Rollback

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

Vous pouvez spécifier la version vers laquelle vous souhaitez revenir. Avant cela, vous devez récupérer l'historique des déploiements et choisir la révision du déploiement.

- Récupérez l’historique des déploiements

```
$ kubectl rollout history deployment nginx-deployment
```

Vous pouvez revenir à 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 persistant. Dans cette section vous allez créer et manipuler des volumes persistants.

La création d’un volume et son attribution à un pod se font en plusieurs étapes.

Premièrement, vous devez 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 généralement effectuée 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

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

- Quel est son statut après la création?

```
$ kubectl get pv
```

Vous ne pouvez pas attacher directement un volume à votre Pod.

Kubernetes ajoute une couche d'abstraction - l'objet "PersistentVolumeClaim".

Cette abstraction permet de découpler les volumes configurés et mis à disposition par les administrateurs K8S et les demandes d'espace de stockage des développeurs pour leurs applications.

Pour créer une demande d'un volume, nous allons créer l'objet "PersistentVolumeClaim".

Nous allons demander un volume qui a au moins 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

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

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

```
$ kubectl get pv
```

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

Maintenant, vous pouvez attribuer le "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

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

- Vérifiez que votre pod est lancé

```
$ kubectl get pods
```

- Trouvez un moyen de vérifier que le volume persistent fonctionne

Liveness et Readiness

Kubernetes peut vérifier automatiquement si vos applications répondent aux demandes des utilisateurs avec des sondes Liveness. Si votre application ne répond pas, K8S le détecte et redémarre ou recrée le conteneur.

Kubernetes peut également retenir le trafic entrant jusqu'au moment où votre service a entièrement démarré avec les sondes Readiness.

Dans cette section, vous allez déployer des Pods avec les sondes Liveness et Readiness.

Liveness probes

Pour rappel, les sondes Liveness 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

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

- Déployez un Pod nginx avec la liveness probe qui échoue
  - Par exemple: qui requête un port qui n'est pas écouté par nginx ou un path inexistant

- Surveillez les events et le comportement du Pod

```
$ kubectl get pods
$ 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

Pour rappel, les sondes Readiness permettent de vérifier si un conteneur est prêt à recevoir des demandes. Si la vérification échoue, le trafic ne sera pas dirigé vers ce Pod.

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

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

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

    ```
    $ kubectl get pods
    ```

  - Surveillez la liste des endpoints du service

    ```
    $ kubectl get endpoints
    ```

  - Que remarquez-vous?

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

```
$ kubectl edit pod nginx-nogood
```

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

Bravo! Vous avez fini le TP!