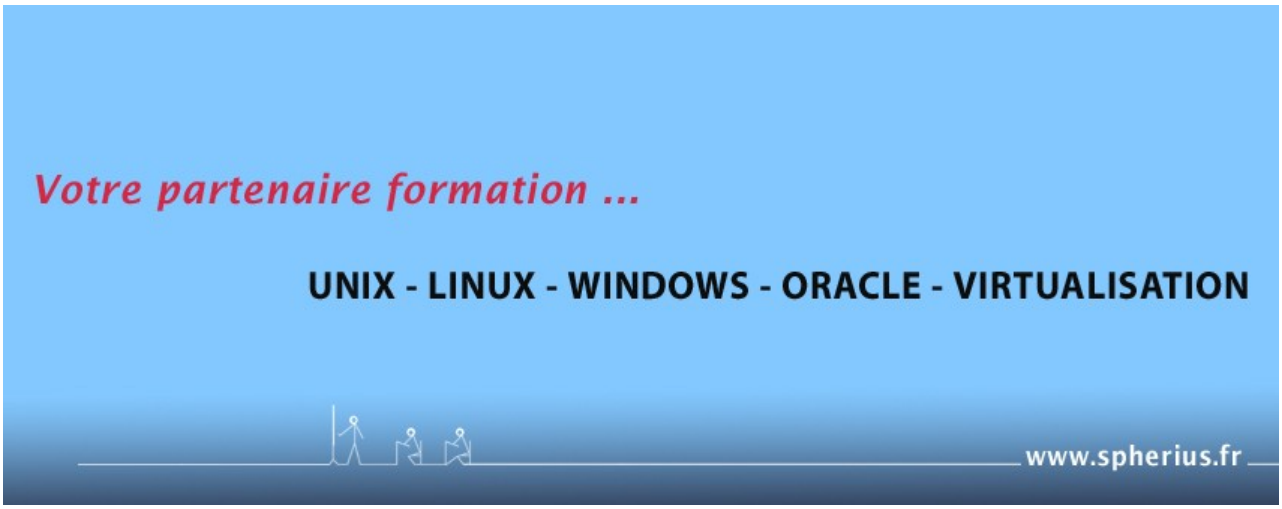
KUBERNETES

EXERCICES



SOMMAIRE

Les auteurs de ce document sont :

- Steeven Herlant,
- Jean-Marc Baranger,
- Theo Schomaker.

Les versions utilisées pour ce support d'exercices sont :

- Ubuntu: 20.04

Les références sont : les documents disponibles sur le site web de Kubernetes.

L'environnement de formation comprend 4 machines :
- admin : machine d'administration pour nous connecter aux nœuds de notre cluster.
- master : machine qui servira comme machine maitre de notre cluster kubernetes.
- worker1 et worker2 : nœuds supplémentaires de notre cluster kubernetes.

Sur toutes les machines le mot de passe de l'utilisateur user1 est user1, celui de l'administrateur root est root.

Les clefs ssh ont été échangé entre les machines pour éviter de saisir le mot de passe à chaque connexion.

Partie 1 : Installation et Configuration de Kubernetes

Exercice 1 : L'installation des pré-requis

Un script a été prévu afin de réaliser l'installation des pré-requis :

- Se connecter en tant que root sur la machine admin (mdp : root)
- Se rendre dans le répertoire /srv/git/formation-kubernetes/scripts
- executer le script : ./install_prerequis.sh

Pour réaliser les opérations manuellement :

Se connecter en SSH en tant que root (mdp : root) sur les 3 machines (master, worker1, worker2) depuis la machine admin.

Installer les paquets nécessaires.
Configurer le dépôt de Kubernetes.
Modifier sysctl pour l'utilisation de Kubernetes.
Activer les modules br_netfilter et overlay.
Enlever la swap.
Editer la fstab pour supprimer la ligne permettant le montage de la swap.
Configurer le dépôt Docker.
Installer containerd.
Mettre en place la configuration de containerd.
Modifier la configuration containerd afin d'utiliser systemd-cgroup.
Redémarrer et activer le service systemd de containerd.
Installer Les paquets nécessaires au fonctionnement du cluster Kubernetes.
Activer Kubelet au démarrage.
Bloquer les versions des paquets.

Exercice 2 : L'installation du cluster

Se connecter via ssh en tant que root (mdp : root) au master.
Initialiser le master (le réseau 10.244.0.0/16 sera utilisé pour les pods).
Noter pour plus tard la commande de join fournie à la fin de l'initialisation.

Se connecter en tant que user1 (mdp : user1) sur le master.
Créer le répertoire $HOME/.kube de l'utilisateur user1, appartenant à user1, du groupe user1.
Copier le fichier de configuration /etc/kubernetes/admin.conf dans le répertoire $HOME/.kube de l'utilisateur user1.
Modifier les droits sur le fichier pour qu'il appartienne à l'utilisateur user1 du groupe user1.
Vérifier l'état des noeuds.
Vérifier l'état des pods.
Récupérer les fichiers manifests pour Calico.
Modifier le fichier /tmp/custom-resources.yml afin de prendre en compte la bonne interface réseau, ainsi que le bon réseau pour nos pods.
Déployer Calico.
Vérifier l'état des pods. Qu'est-ce qui a changé ?
Vérifier l'état des noeuds. Qu'est-ce qui a changé ?
Afficher les conteneurs en cours d'execution, utiliser la commande pour containerd suivante :

```
$ sudo ctr --namespace k8s.io containers list
```

Faire joindre les workers dans le cluster (en tant que root, mdp root).
Contrôler que les workers ont bien rejoint le cluster.
Vérifier les pods maintenant présent sur le cluster. Que constatez-vous ?

Partie 2 : Administration du cluster en cli

Exercice 1 : Récupérer des informations

Se connecter en tant que user1 (mdp : user1) sur la machine master.
Afficher l'état des nœuds du cluster de manière détaillée.
Afficher l'aide de kubectl.
Afficher l'aide de kubectl get.
Récupérer les informations des nœuds en formattant la sortie en json.
Afficher les pods présents sur le cluster dans tous les namespaces, en utilisant l'option courte.

Exercice 2 : Premiers pods

Se connecter en tant que user1 sur la machine master.
Lancer un premier pod nommé pod1, à partir de l'image debian, sans autre option.
Vérifier le status du pod. Pourquoi est-il dans ce status ?
Supprimer le pod.
Relancer le même pod, en y attachant cette fois-ci un TTY et de manière interactive.
Afficher le nom du pod.
Quitter le TTY à l'aide d'un <CTRL>+d. Quel est l'état du pod ?
Afficher les informations concernant le pods à l'aide de la sous-commande get pour obtenir l'adresse ip qui lui a été affecté.
Afficher la description du pod.
Afficher les informations concernant le pod au format json, puis au format yaml.
Utiliser les custom-columns pour afficher, le nom et l'adresse ip du pod, ainsi que le nœud sur lequel il tourne.
rappel des colonnes : nom : .metadata.name, ip : .status.podIP et nœud : .spec.nodeName.
Utiliser la sous-commande attach pour se reconnecter au pod, puis se deconnecter et le supprimer.

Exercice 3 : Premiers déploiements

Se connecter en tant que user1 (mdp : user1) sur la machine master.

En ligne de commandes, Créer un déploiement nommé mon-serveur-web, à partir d'une image nginx.

Afficher les informations et descriptions concernant ce déploiement, ainsi que les informations concernant le pod.
Pourquoi le nom du pod n'est pas mon-serveur-web ?
Supprimer le pod. Que constatez vous ?
Un nouveau pod est automatiquement créé par Kubernetes.
Editer le déploiement, pour y ajouter un nouveau conteneur, nommé mon-debian, et basé sur une image debian. A l'execution du conteneur la commande sleep 600 devra être exécuté.

Afficher à nouveau les informations et la description du pod, que constatez-vous ?
Analyser la partie Events dans la description du pod. Que constatez-vous ?
Supprimer le deploiement.

Exercice 4 : Autocompletion

Se connecter en tant que user1 (mdp : user1) sur la machine master.
Vérifier la présence du paquet bash-completion sur le master, l'installer le cas échéant. Il faudra utiliser les droits sudo ou se connecter en root sur la machine master pour vérifier cela.
Ajouter à la fin du .bashrc de l'utilisateur user1 la commande permettant d'activer la completion de kubectl.
Rappel de la commande : source <(kubectl completion bash)
Se delogguer/relogguer pour que la complétion soit prise en compte.
Vérifier le bon fonctionnement en tapant la commande kubectl, suivi de <TAB><TAB>

Partie 3 : L'utilisation des fichiers yaml

Exercice 1 : Exporter les fichiers

Se connecter en tant que user1 (mdp : user1) sur la machine master.
En ligne de commandes, créer un déploiement nommé mon-debian, à partir d'une image debian en version 11.
Vérifier que la version du conteneur est conforme à celle demandée.
Exporter les informations concernant ce déploiement en yaml (l'option -o yaml fonctionne également pour les déploiements). Que constatez vous ?
Dans quel état est le pod ?
Exporter la configuration du déploiement dans un fichier mon-debian.yml.
Supprimer le déploiement.
Editer le fichier généré :

- Changer l'image utilisée pour une image debian 10.
- Ajouter une commande sleep de 60 secondes

Recréer le déploiement.
Vérifier.
Supprimer le déploiement.

Exercice 2 : Créer ses propres fichiers

Se connecter en tant que user1 sur la machine master.
Créer un fichier appelé mon-premier-pod.yml pour lancer un pod avec les contraintes suivantes :

- nom du pod : mon-premier-pod
- conteneur :
  - image : ubuntu
  - nom : mon-ubuntu
  - commande : sleep 600

Appliquer la configuration.
Exécuter un processus bash dans le conteneur et s'y attacher, puis vérifier la version d'ubuntu utilisée. Utiliser la commande suivante :

```
$ kubectl exec -ti mon-premier-pod -- bash
```

Se déconnecter du pod.
Modifier le fichier de configuration pour utiliser une image ubuntu 18.04, vérifier.
Créer un fichier pour lancer un déploiement avec les contraintes suivantes :

- nom du déploiement : mon-premier-deploiement
- selecteur :
  - matchLabels: app => web
- template :
  - label : app => web
  - conteneur :
    - nom : mon-serveur-web
    - image : nginx

Récupérer l'ip du pod généré ainsi que le nœud sur lequel il a été déployé, et vérifier à l'aide de la commande curl le bon fonctionnement du serveur web, sur le nœud qui héberge le pod.
Supprimer le pod mon-premier-pod et le déploiement mon-premier-deploiement.

Partie 4 : Administration

Exercice 1 : Utilisation des ressources

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie4/exercice1
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie4/exercice1

Adapter le fichier source utilisation-ressources1.yml pour créer un déploiement avec les contraintes suivantes :

- nom du déploiement : utilisation-ressources
- selecteur :
  - matchLabels: stress => test
- template :
  - label : stress => test
  - conteneur :
    - nom : ubuntu1
    - image : ubuntu
    - command : sleep 10 minutes

Se connecter au nœud sur lequel est executé le conteneur et surveiller les ressources à l'aide de la commande top.
Le laisser tourner et noter le load average du nœud ainsi que la consommation RAM.

Sur la machine master en tant que user1 (mdp : user1).
Exécuter un processus bash à l'intérieur du conteneur généré. Faire un update des paquets et installer le paquet stress.
Executer la commande stress -c 1000 pour demander l'utilisation de 1000 CPU simultanés.
Laisser le processus tourner et retourner vérifier l'état de la commande top, notamment le load average. Que constatez-vous ?
Se déconnecter du conteneur.
Adapter le fichier source utilisation-ressources2.yml pour respecter les contraintes suivantes :

- nom du déploiement : utilisation-ressources
- selecteur :
  - matchLabels: stress => test
- template :
  - label : stress => test
  - conteneur :
    - nom : ubuntu1
    - image : ubuntu
    - command : sleep 10 minutes
    - ressources :
      - requests : cpu à 100milliCPU (100m) et RAM à 100Mo (100Mi)

- limits : cpu à 500milliCPU et RAM à 300Mo

Revérifier à l'aide des commandes top et free l'état du worker sur lequel tourne notre pod.
Que constatez-vous maintenant ?
Supprimer le déploiement.

Exercice 2 : Labels, NodeSelector et NodeName

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie4/exercice2
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie4/exercice2

Placer 2 labels sur les workers :

- worker1 : datacenter => Paris
- worker2 : datacenter => Marseille

Vérifier.

Adapter les fichiers du répertoires des sourcess pour créer 4 déploiements avec les contraintes suivantes :

- Un template de pod contiendra un conteneur basé sur une image nginx, tournant sur le datacenter de Paris
- Un template de pod contiendra un conteneur basé sur une image nginx, tournant sur le datacenter de Marseille
- Un template de pod contiendra un conteneur basé sur une image debian, tournant sur le nœud worker1, et la commande sleep 600 permettant de maintenir le conteneur up.
- Un template de pod contiendra un conteneur basé sur une image debian, tournant sur le nœud worker2, et la commande sleep 600 permettant de maintenir le conteneur up.

Vérifier les workers sur lesquels tournent les pods.
Nous avons bien nos pods issus des déploiements nginx-paris et debian-worker1 qui tournent sur worker1, et nginx-marseille et debian-worker2 sur worker2.
Supprimer les 4 déploiements.

Exercice 3 : Namespaces, Contextes

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie4/exercice3
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie4/exercice3

Créer le namespace exercice3.
Adapter le fichier nginx-marseille.yml présent dans le répertoire des sources pour qu'il utilise le namespace exercice3.
Déployer le projet.
Vérifier que le déploiement tourne bien dans le namespace exercice3.

Créer un nouveau contexte qui utilisera le namespace exercice3, et l'utilisateur kubernetes-admin, puis switcher sur ce nouveau contexte. Utiliser les commandes suivantes :

```
$ kubectl config set-context exercice3 --namespace exercice3 --user \
kubernetes-admin --cluster kubernetes
$ kubectl config use-context exercice3
```

Vérifier que le contexte est bien configuré en affichant les pods du namespace actuel (qui doit être le namespace exercice3).
Supprimer le namespace exercice3. Vérifier ensuite l'état des pods et du contexte exercice3. Que constatez-vous ?
Changer de contexte en sélectionnant celui par défaut et supprimer le contexte exercice3.

Exercice 4 : Scaling

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie4/exercice4
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie4/exercice4

Créer un nouveau namespace exercice4.
Adapter le fichier scaling.yml présent dans le répertoire des sources avec les contraintes suivantes :

- nom du déploiement : scaling
- replicas : 2
- namespace : exercice4
- selecteur :
  - matchLabels: app => scale
- template :
  - label : app => scale
  - conteneur :
    - nom : scale-nginx
    - image : nginx
    - ressources :
      - requests : cpu à 100milliCPU et RAM à 100Mo
      - limits : cpu à 500milliCPU et RAM à 300Mo

Déployer le projet.
Vérifier que le déploiement respecte les contraintes demandées, notamment le nombre de replicas. Quelle est la répartition sur le cluster ?
Modifier en ligne de commande le nombre de replicas pour le passer à 5. Quelle est maintenant la répartition ?
Supprimer le namespace exercice4.

Partie 5 : Les Services

Exercice 1 : ClusterIp

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie5/exercice1
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie5/exercice1

Créer le namespace services.
Adapter le fichier deploy-cluster-ip.yml présent dans le répertoire des sources pour que cela respecte les contraintes suivantes :

- nom du déploiement : deploy-cluster-ip
- replicas : 2
- namespace : services
- selecteur :
  - matchLabels: app => cluster-ip
- template :
  - label : app => cluster-ip
  - conteneur :
    - nom : cluster-ip-nginx
    - image : nginx

Déployer le projet.
Vérifier que chaque worker fait bien tourner un pod.
Se connecter au pod du worker1 en executant un bash.
Rappel de la commande :

```
$ kubectl exec -ti -n services cluster-ip-$podId -- bash
```

Copier le fichier index-worker1.html sur le pod tournant sur worker1, dans /usr/share/nginx/html/index.html. Le fichier se trouve dans le répertoire des sources.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Worker1</title>
  </head>
  <body>
    <h1>Vous etes sur Worker1</h1>
  </body>
</html>
```

Faire la même chose sur le pod tournant sur worker2, et utiliser le fichier index-worker2.html.
Le fichier se trouve dans le répertoire des sources.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Worker2</title>
  </head>
  <body>
    <h1>Vous etes sur Worker2</h1>
```

```
  </body>
</html>
```

Créer maintenant un service de type ClusterIp, en adaptant le fichier svc-cluster-ip.yml présent dans le répertoire des sources, avec les contraintes suivantes :

- nom du service : svc-cluster-ip
- namespace : services
- type : ClusterIP
- port d'écoute : 8000
- port utilisé par le pod : 80
- selecteur : app => cluster-ip

Récupérer les ips des pods.
Vérifier par un curl le bon fonctionnement du serveur web nginx depuis ces 2 ips. Vérifier à l'aide du texte des pages que nous contactons bien le bon serveur web.
Récupérer l'ip affectée à notre service.
Lancer un curl sur cette ip, en veillant bien à utiliser le port définit lors de la création du service.
Répéter plusieurs fois la commande. Que constatez-vous ?
Augmenter le nombre de replicas à 3. Utiliser la commande suivante :

```
# kubectl scale deploy -n services  --replicas=3 deploy-cluster-ip
```

Répéter le curl. Que constatez-vous ?

Exercice 2 : NodePort

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie5/exercice2
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie5/exercice2

Créer un déploiement de conteneur apache. Adapter le fichier deploy-node-port.yml, présent dans le répertoire des sources, et respecter les contraintes suivantes :

- nom du déploiement : deploy-node-port
- replicas : 2
- namespace : services
- selecteur :
  - matchLabels: app => node-port
- template :
  - label : app => node-port
  - conteneur :
    - nom : node-port-apache
    - image : httpd

Vérifier que les pods fonctionnent normalement et repérer les nœuds sur lesquels ils tournent.
Adapter le fichier svc-node-port.yml pour créer le service suivant :

- nom du service : svc-node-port
- namespace : services
- type : NodePort

- NodePort : 30001
- port d'écoute : 8001
- port utilisé par le pod : 80
- selecteur : app => node-port

Vérifier l'état du service.
Copier le fichier index-worker1.html sur le pod tournant sur worker1, dans /usr/local/apache2/htdocs/index.html. Le fichier se trouve dans le répertoire des sources.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Worker1</title>
  </head>
  <body>
    <h1>Vous etes sur Worker1</h1>
  </body>
</html>
```

Copier le fichier index-worker2.html sur le pod tournant sur worker2, dans /usr/local/apache2/htdocs/index.html. Le fichier se trouve dans le répertoire des sources.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Worker2</title>
  </head>
  <body>
    <h1>Vous etes sur Worker2</h1>
  </body>
</html>
```

Récupérer l'ip du master.
Comme tout à l'heure, lancer des curls consécutifs sur l'ip du master et le port présent dans la configuration du service. Que constatez-vous ?
Supprimer le namespace services.

Partie 6 : Les Daemonsets

Exercice 1 :

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie6/exercice1
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie6/exercice1

Créer un nouveau namespace daemon-set.
Créer un déploiement de conteneur centos.
Adapter le fichier deploy-daemon.yml, à partir du fichier deploy-daemon.yml présent dans le répertoire des sources, et respecter les contraintes suivantes :

- nom du déploiement : deploy-daemon
- namespace : daemon-set
- selecteur :
  - matchLabels: app => my-centos
- template :
  - label : app => my-centos
  - conteneur :
    - nom : centos
    - image : centos
    - commande : sleep de 600 secondes

Ajouter dans la section spec, la section NodeSelector avec comme mot-clef work avec la valeur « true ».
Afficher les informations du daemonset. Que constatez-vous ?
Mettre en place le label work=true sur le worker2, puis vérifier l'état du daemonset, ainsi que celui des éventuels pods créés.
Supprimer maintenant la section « NodeSelector »du fichier YAML, puis appliquez la configuration. Que constatez-vous ?
Supprimer le namespace daemonset.

Partie 7 : Les Volumes

Exercice 1 : HostPath

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie7/exercice1
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie7/exercice1

Créer le namespace volumes.
Créer un répertoire /srv/hostpath sur chaque worker.
Se connecter en tant que user1 sur worker1 et worker2 et effectuer les manipulations suivantes via la commande sudo :
Créer un répertoire /srv/hostpath.
Changer le propriétaire du répertoire par user1, ainsi que le groupe propriétaire, par user1 également.
Créer le fichier index.html dans le répertoire /srv/hostpath sur worker1 et worker2 avec le contenu suivant :
sur worker1 :

```
<h1>Je suis le HostPath de Worker1</h1>
```

sur worker2 :

```
<h1>Je suis le HostPath de Worker2</h1>
```

Créer un nouveau déploiement, en adaptant le fichier deploy-host-path.yml, présent dans le répertoire source, et en y implément un volume de type hostPath, à l'aide du répertoire /srv/hostpath précédemment créé sur worker1 et worker2 :

- nom du déploiement : deploy-host-path
- namespace : volumes
- selecteur :
  - matchLabels: app => host-path
- template :
  - label : app => host-path
  - conteneur :
    - nom : host-path-nginx
    - image : nginx
  - NodeName : worker1
  - volume :
    - nom : mon-host-path
    - point de montage : /usr/share/nginx/html

Vérifier que le pod créé fonctionne correctement, et noter son ip ainsi que le nœud sur lequel il tourne.
Lancer un curl sur le serveur web. Que constatez-vous ?

Modifier le fichier de configuration pour que le pod tourne sur l'autre nœud (à l'aide de la directive « nodeName » vue précédemment). Récupérer l'ip du nouveau pod et lancer de nouveau un curl. Que constatez-vous ?

Supprimer le déploiement.

Exercice 2 : Persistent Volume Claim

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie7/exercice2
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie7/exercice2

Pour cet exercice, nous allons installer un serveur nfs sur le master.
Installer sur le master le paquet nfs-kernel-server.

```
$ sudo apt install -y nfs-kernel-server
```

Installer sur les workers le paquet nfs-common (il devrait déjà être présent).

```
$ ssh worker1 sudo apt install -y nfs-common
$ ssh worker2 sudo apt install -y nfs-common
```

Créer le répertoire /srv/exports sur le master et changer le propriétaire par user1.

```
$ sudo mkdir /srv/exports
$ sudo chown user1: /srv/exports
```

Editer le fichier de configuration du serveur nfs /etc/exports et insérer la ligne suivante (sur le master):

```
/srv/exports 192.168.56.0/24(rw,sync,no_root_squash)
```

Activer et redémarrer le serveur nfs.

```
$ sudo systemctl enable nfs-kernel-server
$ sudo systemctl restart nfs-kernel-server
```

Vérifier que le serveur NFS est opérationnel :

```
$ showmount -e
Export list for master :
/srv/exports        192.168.56.0/24
```

Nous avons maintenant un serveur NFS opérationnel sur le master.

Créer maintenant un PersistentVolume en adaptant le fichier pv-nfs.yml, présent dans le répertoire des sources, avec les contraintes suivantes :

- nom : pv-nfs
- namespace : volumes
- storageClass : manual
- capacity : 1G
- accessModes : ReadWriteMany
- nfs :
  - server : adresse de votre serveur nfs
  - path : chemin du partage

Appliquer la configuration.
Vérifier l'état du PersistentVolume.
Créer maintenant le PersistentVolumeClaim associé,-en adaptant le fichier pvc-nfs.yml, présent dans le répertoire des sources, pour lui affecter 500 Mo.
Vérifier que le PersistentVolumeClaim a été correctement créé.
Sur le master, créer le fichier /srv/exports/index.html, avec le contenu suivant :

```
<h1>Bienvenue sur mon site web.</h1>
```

Créer maintenant 2 déploiements en adaptant les fichiers deploy-nfs-nginx.html et deploy-nfs-httpd.yml, présents dans les répertoires des sources.

- Le premier sur base d'image nginx, avec le volume monté dans /usr/share/nginx/html, en le faisant tourner sur worker1.
- Le deuxième sur base d'image httpd, avec le volume monté dans /usr/local/apache2/htdocs, en le faisant tourner sur worker2.

Vérifier que les pods fonctionnent correctement et respectent bien les contraintes.
Noter leurs adresses ip respectives.
Lancer un curl sur les 2 serveurs web. Que constatez-vous ?
Modifier le fichier html pour qu'il affiche « Bonjour et Bienvenue » sur le master. Relancer les curls. Que constatez-vous ?
Les modifications sont bien prises en compte.

Exercice 3 : ConfigMaps et Secrets

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie7/exercice3
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie7/exercice3

Adapter le fichier de configuration conf-mariadb.yml, présent dans le répertoire des sources, pour créer le configMap suivant :
nom : conf-mariadb
namespace : volumes
data :
MYSQL_DATABASE : dbweb
MYSQL_USER : web

Vérifier que le configMap a été correctement créé, et qu'il contient les bonnes variables.

Adapter le fichier de configuration secret-mariadb.yml, présent dans le répertoire des sources, pour créer le Secret suivant :
nom : secret-mariadb
namespace : volumes
data :
MYSQL_ROOT_PASSWORD: MonSuperMdpRoot
MYSQL_PASSWORD : MonSuperMdp
Vérifier.
Créer maintenant un déploiement, en adaptant le fichier deploy-ma-db.yml, présent dans le répertoire des sources, et basé sur une image mariadb. Passer dans l'environnement du conteneur le configmap et le secret précédemment créés.
Appliquer la configuration.
Vérifier que le déploiement s'est correctement déroulé.
Se connecter au conteneur.
Depuis le conteneur, se connecter à l'instance mariadb en utilisant la commande du même nom, en root et en user web, et en saisissant les mots de passes créés plus tôt.
Au prompt de mariadb, vérifier que la database dbweb a bien été créée à l'aide de la commande « show databases ; », puis quitter à l'aide de la commande « quit ; ».

Se déconnecter du pod

```
# <CTRL-D>
```

Supprimer le namespace volumes.

Partie 8 : Sécurité

Exercice 1 : RBAC

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie8/exercice1
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie8/exercice1

Créer le namespace rbac.
Créer un nouveau Rôle en adaptant le fichier de configuration my-rbac-list.yml, présent dans le répertoire des sources, avec les contraintes suivantes :

- nom : my-rbac-list
- namespace : rbac
- règles :
  - apiGroup : group core
  - ressources : pods
  - opérations : list

Appliquer la configuration. Vérifier avec les commandes adéquates.
Créer un nouveau Rôle, nommé my-rbac-all, en adaptant le fichier my-rbac-all1.yml, présent dans le répertoire des sources, avec les mêmes contraintes que précédemment , mais en lui donnant cette fois-ci les opération get, watch et list.
Appliquer la configuration. Vérifier avec les commandes adéquates
Créer 2 servicesAccounts, user-list et user-all, dans le namespace rbac en adpatant les fichiers user-list.yml et user-all.yml, présents dans les répertoires des sources.
Le premier servira pour le Rôle avec l'opération list, le second pour celui avec les 3 opérations, list, wet et watch.
Appliquer les fichiers.
Vérifier que les Rôles et ServiceAccounts ont été correctement créés.
Adapter les fichiers role-binding-list.yml et role-binding-all.yml, présents dans le répertoire des sources, afin de créer 2 RoleBinding. Ces RoleBinding serviront à lier les ServiceAccount et les Roles précédemment créés :

- role-binding-all = ServiceAccount : user-all, Role : my-rbac-all
- role-binding-list = ServiceAccount : user-list, Role : my-rbac-list

Appliquer la configuration.
Vérifier que les RoleBindings ont été correctement créés et qu'ils respectent bien les contraintes avec la commande suivante :

```
$ kubectl get rolebindings -n rbac -o wide
```

Récupérer le nom des tokens générés lors de la création des serviceAccounts grâce à la commande suivante :

```
$ kubectl describe sa -n rbac
```

Créer un déploiement , en adaptant le fichier deploy-list.yml présent dans le répertoire des sources, basé sur une image Debian, en lui fournissant le serviceAccount user-list. Le token associé à ce ServiceAccount sera monté dans /etc/secrets/user-list-token. Le certificat de l'AC sera monté dans /etc/secrets/ca.crt. Nous n'oublierons pas d'exécuter la commande sleep 600 afin de garder le pod up.

Créer un deuxième déploiement, en adaptant le fichier deploy-all.yml basé sur une image Ubuntu, en lui fournissant cette fois-ci le serviceAccount user-all, et en définissant à nouveau l'utilisation du token (qui sera monté dans /etc/secrets/user-all-token) et du certificat de l'AC. Encore une fois, il nous faudra executer la commande « sleep 600 » pour que notre pod reste up.
Vérifier le bon fonctionnement des déploiements.
Afficher les pods.
Se connecter sur le pod debian à l'aide de la commande suivante :

```
$  kubectl exec -ti -n rbac my-debian-list-$podId -- bash
```

Vérifier la présence des secrets en listant le contenu du répertoire /etc/secrets.

Installer le paquet curl à l'aide de la commande apt.

Accéder à l'api via curl et les secrets, et afficher la liste des pods à l'aide des commandes suivantes :

```
# token=$(cat /etc/secrets/user-list-token)
# curl -H "Authorization: Bearer $token" --cacert \
/etc/secrets/ca.crt \
https://kubernetes.default/api/v1/namespaces/rbac/pods
```

Essayer maintenant d'obtenir les informations d'un des pods du namespace, ainsi que ces évènements. Que constatez-vous ?
Informations d'un pod :
https://kubernetes.default/api/v1/namespaces/rbac/pods/$nomDuPod
Évènements d'un pod :
https://kubernetes.default/api/v1/watch/namespaces/rbac/pods/$nomDuPod
Se déconnecter du pod.

Répéter les mêmes opérations sur le pod Ubuntu. Que constatez-vous ?
Se déconnecter du pod.

Éditer le fichier de configuration my-rbac-all.yml du rôle my-rbac-all et retirer l'opération watch. Appliquer les changements et vérifier l'accès à l'API depuis le pod. Que constatez-vous ?

Supprimer le namespace rbac.

Exercice 2 : Les quotas

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie8/exercice2
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie8/exercice2
Créer le namespace quotas.
Adapter le fichier de configuration ma-limite.yml présent dans le répertoire des sources, pour créer une ressource LimitRange, avec les limitations suivantes :

- nom : ma-limite
- namespace : quotas
- type : pod
  - max :
  - cpu : 1
  - ram : 100Mo
- type : container
  - cpu : 0.5
  - ram : 50Mo
  - defaultRequest :
    - cpu : 0.25
    - ram : 25Mo
  - maxLimitRequestRatio :
    - cpu : 4
    - ram : 2

Appliquer la configuration.
Vérifier que la ressource a été correctement créée.

Consulter le fichier deploy-my-nginx.yml, présent dans le répertoire des sources, puis appliquer la configuration.
Vérifier les quotas appliqués sur le conteneur du pod déployé. Pourquoi ces quotas ?

Modifier le fichier deploy-my-nginx.yml pour ajouter la section resources sur le déploiement du conteneur my-nginx pour que les requests cpu et ram soient positionnés respectivement à 2 et 150Mo.
Appliquer les changements.
Vérifier à nouveau l'état du conteneur de pod déployé, ainsi que l'état du déploiement. Que constatez-vous ? Constater que le déploiement ne se fait pas à cause des limites qui sont dépassées.
Supprimer le déploiement ainsi que la ressource LimitRange.

Exercice 3 : ResourceQuota

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie8/exercice3
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie8/exercice3

Modifier le fichier de configuration deploy-rs-quotas.yml pour déployer une ressource ResourceQuota, en demandant un maximum de 4 pods, et l'utilisation de 2 cpus dans le namespace quotas.
Appliquer la configuration.

Créer maintenant un nouveau déploiement, avec les contraintes suivantes :

- nom : deploy-rs-quotas
- namespace : quotas
- replicas : 2
- selecteur :
  - matchLabels: app => rs-quotas
- template :
  - label : app => rs-quotas
  - conteneur :
    - nom : apache-rs-quota
    - image : httpd
    - requests :
      - cpu : 1
      - memory : 100M

Appliquer la configuration.
Vérifier que le déploiement c'est correctement déroulé.
Passer maintenant le nombre de réplicas à 4, en ligne de commande.
Vérifier l'état du déploiement.
Constater que les 2 conteneurs suuplémentaires crées sont dans l'état Pending.
Que ce passe-t-il ? Pourquoi ?

Supprimer le namespace quotas.

Exercice 4 : Accès réseaux

Se connecter en tant que user1 sur la machine master.
Les fichiers sources se trouvent ici :
/srv/git/formation-kubernetes/sources/partie8/exercice4
Les fichiers corrections se trouvent ici :
/srv/git/formation-kubernetes/corrections/partie8/exercice4

Créer le namespace frontend.
Créer le namespace backend.
Affecter le label type=backend au namespace backend.
Affecter le label type=frontend au namespace frontend.

Afficher le fichier deploy-my-nginx.yml, présent dans le répertoire des sources, et vérifier qu'il respecte les contraintes suivantes :

- nom : deploy-my-nginx
- namespace : frontend
- selecteur :
  - matchLabels: app => frontend
- template :
  - label : app => frontend,auth=db
  - conteneur :
    - nom : my-nginx
    - image : nginx

Appliquer la configuration.
Vérifier que le déploiement s'est correctement déroulé.

Créer également le service nodePort associé en modifiant le fichier svc-my-nginx.yml présent dans le répertoire des sources, afin de pouvoir accéder au serveur web via le port 30001 de l'ip publique du master. Nous utiliserons le port 8001 comme port interne, et le port destination sur le pod est le port 80.
Appliquer la configuration.
Vérifier que le service fonctionne correctement.

Afficher le fichier deploy-my-db.yml, présent dans le répertoire des sources, et vérifier qu'il respecte les contraintes suivantes :

- nom : deploy-my-db
- namespace : backend
- selecteur :
  - matchLabels: app => backend
- template :
  - label : app => backend
  - conteneur :
    - nom : my-db
    - image : mariadb
- env:

- name: MYSQL_DATABASE
- value: webdb
- name: MYSQL_USER
- value: webuser
- name: MYSQL_ROOT_PASSWORD
- value: mdpRoot
- name: MYSQL_PASSWORD
- value: mdp

Appliquer la configuration.
Vérifier que le déploiement fonctionne correctement. Récupérer l'adresse ip du conteneur my-db
Se connecter au conteneur nginx à l'aide de la commande suivante :

```
$ kubectl exec -it -n frontend deploy-my-nginx-$podId-- bash
```

effectuer une mise à jour du dépôt et installer les paquets mycli et iputils-ping.
Essayer de se connecter à la base webdb à l'aide la commande suivante (mdp pour la base : mdp) :

```
# mycli -u webuser -h $ipConteneurBDD webdb
```

Se déconnecter du pod.

Afficher le fichier deploy-my-api.yml, présent dans le répertoire des sources, et vérifier qu'il respecte les contraintes suivantes :

- nom : deploy-my-api
- namespace : frontend
- selecteur :
  - matchLabels: app => api
- template :
  - label : app => api
  - conteneur :
    - nom : my-api
    - image : debian
    - command : sleep 600

Appliquer la configuration.
Vérifier une nouvelle fois que le déploiement a réussi.
Se connecter sur le conteneur du pod my-api avec la commande suivante :

```
$ kubectl exec -ti -n frontend deploy-my-api-$podId -- bash
```

Effectuer une mise à jour et installer le paquet iputils-ping.
Exécuter un ping sur les conteneurs nginx et mariadb, ainsi que sur google.fr.
Se déconnecter du pod.

Modifier le fichier api-net-policy.yml, présent dans le répertoire des sources, pour mettre en place une police réseau afin de respecter les contraintes suivantes :

- appliquer sur : deploy-my-nginx
- connexions entrantes : uniquement depuis les pods du namespace ayant le label app = api sur le port 80 en TCP
- connexions sortantes : uniquement vers les pods ayant un label app=backend dans le namespace ayant le label type=backend sur le port 3306 en TCP.

Appliquer la configuration.

Essayer d'exécuter un curl sur l'ip du master sur le port 30001 du service. Que constatez-vous ?

Se connecter sur le conteneur my-api à l'aide de la commande suivante :

```
$ kubectl exec -it -n frontend deploy-my-api-$podId-- bash
```

Installer le paquet curl (mettre à jour la liste des paquets si nécessaire).
Executer un curl sur l'adresse ip du conteneur nginx. Que constatez vous ?
Se déconnecter du conteneur.

Se connecter sur le conteneur my-nginx à l'aide de la commande suivante :

```
$ kubectl exec -it -n frontend deploy-my-nginx-$podId-- bash
```

Re-executer la commande permettant de se connecter à la base de données du conteneur mariadb. Essayer également d'executer un ping sur l'adresse ip du conteneur mariadb. Que constatez-vous ?
Se déconnecter du conteneur.

Mettre à jour le fichier mariadb-net-policy.yml, présent dans le répertoire des sources, afin de respecter les contraintes suivantes :

- appliquer sur : deploy-my-mariadb
- connexions entrantes : uniquement depuis les pods ayant un label auth=db, quelque soit le namespace
- connexions sortantes : aucunes

Appliquer la configuration.
Se connecter au conteneur mariadb à l'aide de la commande suivante :

```
$ kubectl exec -it -n backend deploy-my-db-$podId-- bash
```

Executer la commande « apt update ». Constater que cela ne fonctionne pas.
Se déconnecter du conteneur.

Se connecter sur le conteneur nginx à laide de la commande suivante :

```
$ kubectl exec -it -n frontend deploy-my-nginx-$podId-- bash
```

Relancer la commande mycli pour se connecter à la base de données. Que constatez-vous ?
Se déconnecter du conteneur.
Supprimer les namespaces backend et frontend.