Lab 2 : Créer un premier POD et manipuler son environnement d'orchestration K8s

Ce Lab s'intéresse à la création de l'entité de référence dans un cluster Kubernetes à savoir un `Pod` qui est une représentation logique de un ou plusieurs conteneurs. Nous allons donc voir comment manipuler un `Pod` pour le créer, s'y connecter via son conteneur associé et pour le supprimer. Nous étudierons également comment écrire un fichier de configuration utilisé pour faciliter l'exportation des paramètres d'un `Pod`. Nous verrons également l'intérêt d'utiliser un `Namespace` pour isoler les différents objets créés dans un cluster Kubernetes. Enfin, nous montrerons via une première solution comment accéder à un `Pod` par le réseau.

Ce premier exercice sera aussi l'occasion de manipuler les outils **kubectl** et K9s que nous avons installés lors de la mise en place de notre cluster Kubernetes.

Quel que soit le type d'installation choisi pour la mise en place de votre cluster Kubernetes, toutes les commandes ci-dessous devraient normalement fonctionner. Nous considérons qu'il existe un fichier `k3s.yaml` à la racine du dossier `kubernetes-tutorial/`, si ce n'est pas le cas, merci de reprendre la mise en place d'un cluster Kubernetes. Il est important ensuite de s'assurer que la variable `KUBECONFIG` soit initialisée avec le chemin du fichier d'accès au cluster Kubernetes (`export KUBECONFIG=$PWD/k3s.yaml`).

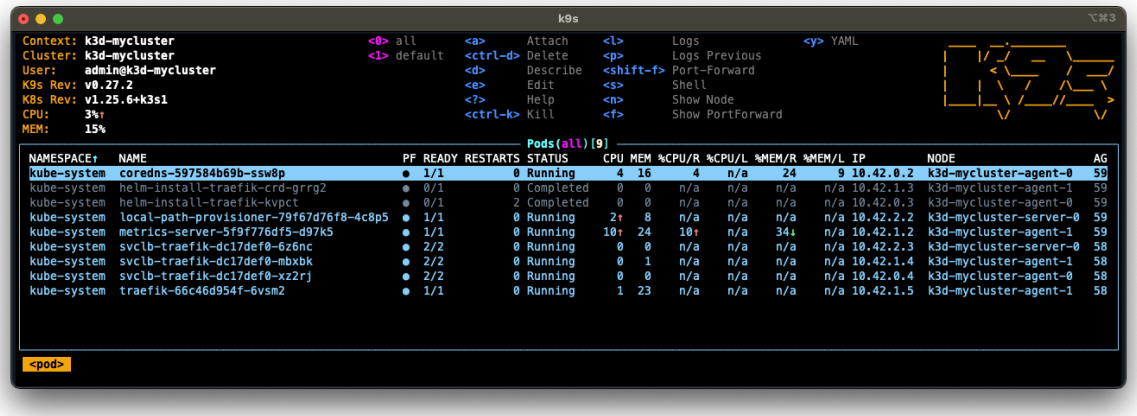
But

- Manipuler un `Pod` (créer, se connecter via son conteneur associé, supprimer)
- Écrire un fichier de configuration pour décrire un objet `Pod`
- Accéder à un `Pod` via le réseau
- Organiser les `Pods` via les `Namespaces`

Étapes à suivre

- Créer deux invites de commande, l'une pour l'exécution de K9s et la seconde pour la saisie des commandes avec l'outil **kubectl**, puis se placer pour chaque invite de commande à la racine du dossier `kubernetes-tutorial/`. Pour faciliter l'identification des invites de commande, nous les appellerons respectivement *k9s* et *kubectl*.
- Depuis l'invite de commande *k9s* :

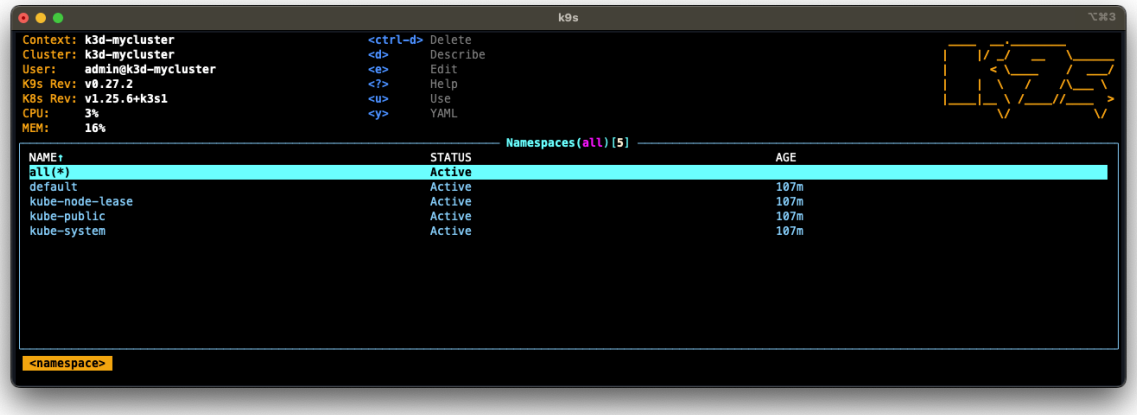
```
$ export KUBECONFIG=$PWD/k3s.yaml
$ k9s
```



L'outil K9s affiche tous les objets créés au sein du cluster. L'affichage au démarrage donne l'ensemble des objets de type `Pod`. Les `Pods` déjà présents concernent le fonctionnement interne de Kubernetes. Ces `Pods` sont identifiables par le `Namespace` appelé `kube-system` (colonne `NAMESPACE`).

L'utilisation K9s est très proche de l'éditeur de texte **Vim**. Deux modes sont disponibles : *commande* et *recherche*. Pour saisir une commande, la touche `:` doit être utilisée. Ce mode est identifiable par la forme du prompt 🐶> qui représente un chien de race Bigle (enfin je crois). L'ensemble des commandes est disponible via le raccourci `CTRL + a` ou via la commande `:aliases`. Pour effectuer une recherche, la touche `/` doit être utilisée. Ce mode est identifiable par la forme du prompt 🐩> qui est un chien de race Caniche (il n'y a pas à se tromper là).

- Depuis l'outil K9s, afficher la liste des `Namespaces` via la commande `:namespaces`.



Le résultat est identique selon que vous utilisez le singulier ou le pluriel. `:namespaces` donne le même résultat que `:namespace`.

Un ensemble de `Namespace` est donné par l'outil K9s. Ces `Namespaces` existent déjà car ils sont utilisés pour le fonctionnement interne de Kubernetes.

- Depuis l'outil K9s, retourner à l'affichage de la liste des `Pods` via la commande `:pods`.

Nous allons obtenir les mêmes informations depuis l'outil **kubectl**. Cependant, contrairement à l'outil K9s, elles ne pourront être mises à jour qu'en exécutant plusieurs fois la même ligne de commande. En effet, K9s est réactif et tout changement sur le cluster Kubernetes est automatiquement affiché à l'utilisateur (sous condition qu'il se trouve dans la bonne commande).

- Depuis l'invite de commande *kubectl* :

```
$ export KUBECONFIG=$PWD/k3s.yaml
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                      READY   STATUS      RESTARTS   AGE
kube-system   local-path-provisioner-79f67d76f8-flwh9   1/1     Running     0          115m
kube-system   coredns-597584b69b-kwbdc                  1/1     Running     0          115m
kube-system   helm-install-traefik-crd-67bd4            0/1     Completed   0          115m
kube-system   svclb-traefik-86c9da09-xdmwv              2/2     Running     0          115m
kube-system   svclb-traefik-86c9da09-mhm2n              2/2     Running     0          115m
kube-system   helm-install-traefik-k6j29                0/1     Completed   1          115m
kube-system   svclb-traefik-86c9da09-zs7tn              2/2     Running     0          115m
kube-system   traefik-66c46d954f-pzwrl                  1/1     Running     0          115m
kube-system   metrics-server-5f9f776df5-2rgdh           1/1     Running     0          115m
```

L'option `get` permet de récupérer les informations de l'objet passé en paramètre `pods`. Le paramètre `--all-namespaces` indique que tous les `Namespaces` sont considérés.

- Affichons maintenant la liste des `Namespaces` de notre cluster Kubernetes, depuis l'invite de commande *kubectl* :

```
$ kubectl get namespace
NAME              STATUS   AGE
default           Active   117m
kube-system       Active   117m
kube-public       Active   117m
kube-node-lease   Active   117m
```

Il est maintenant temps de créer notre premier `Pod` qui, pour rappel, est une représentation logique de un ou plusieurs conteneurs.

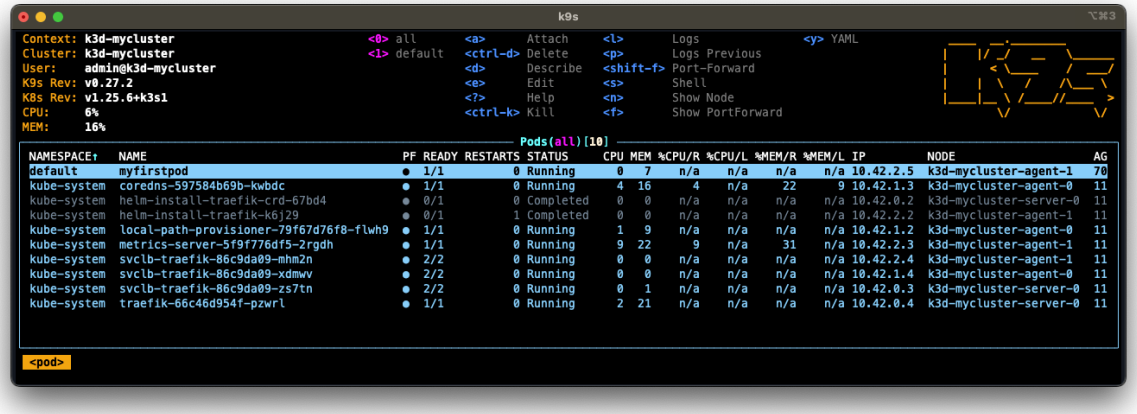
- Dans l'exemple qui va suivre, nous allons créer un `Pod` avec un conteneur basé sur l'image du serveur web Nginx. Depuis l'invite de commande *kubectl* :

```
$ kubectl run myfirstpod --image=nginx:latest
pod/myfirstpod created
```

- Pour s'assurer que le `Pod` a été créé :

```
$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
myfirstpod   1/1     Running   0          34s
```

Depuis l'outil K9s, vous devriez obtenir le résultat suivant (commande `:pods` si vous n'affichez pas la liste des `Pods`).



Puisque notre premier `Pod` a été créé et déployé sans problème, nous allons vérifier si la page web par défaut de Nginx est retournée après une requête HTTP. L'accès depuis l'extérieur d'un cluster K8s à un `Pod` se fait généralement par les services. Toutefois, nous découvrirons les services plus tard dans les prochains exercices. En attendant, nous allons utiliser une technique d'exposition des `Pods` via une redirection des ports. Cette technique ne peut être mise en place que par l'intermédiaire des outils **kubectl** et K9s. Cette redirection des ports n'est à utiliser que pour les phases de test, **ne jamais utiliser cette technique pour la mise en production de vos microservices**.

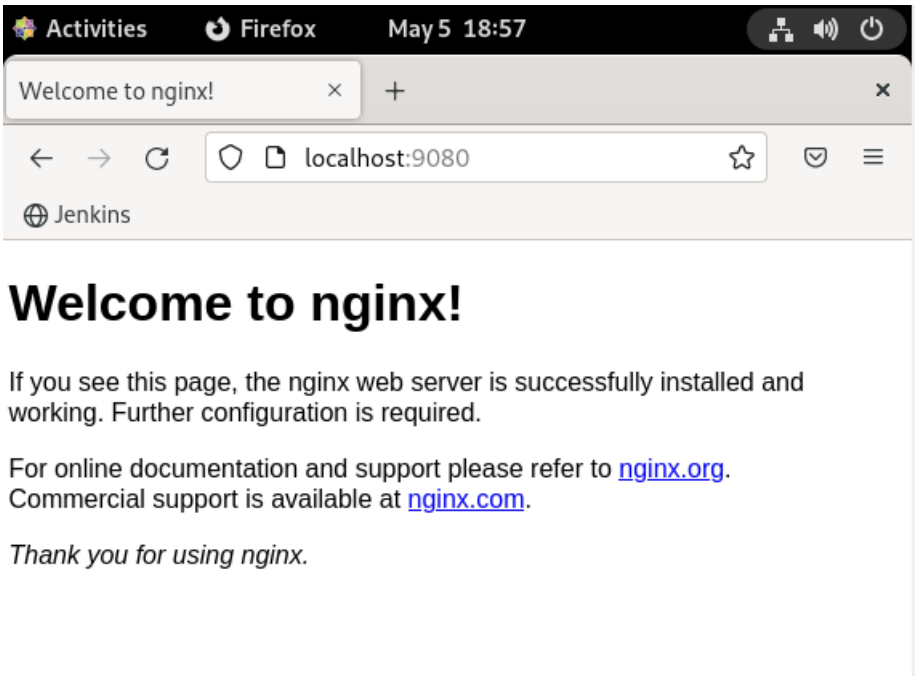
- Depuis l'invite de commande *kubectl* :

```
$ kubectl port-forward myfirstpod 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
```

L'option `port-forward` permet de créer un pont entre la machine locale (depuis le port 8080) et notre `Pod` (vers le port 80). À l'éxécution de cette commande, le processus **kubectl** devient bloquant.

Vous pouvez également utiliser l'option `--address 0.0.0.0` pour exposer à toutes les adresses IP de la machine.

- Ouvrir la page par défaut de Nginx depuis un navigateur : http://localhost:8080.



- Arrêter le processus bloquant **kubectl** depuis l'invite de commande *kubectl* via `CTRL+C`.

Puisqu'un `Pod` est une représentation logique de un ou plusieurs conteneurs, nous pouvons exécuter une commande directement sur les conteneurs d'un `Pod`.

- Depuis l'invite de commande *kubectl* :

```
$ kubectl exec -it myfirstpod -- /bin/bash
root@myfirstpod:/# ls
bin  boot  dev docker-entrypoint.d  docker-entrypoint.sh  etc        home
lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

L'option `exec` permet d'exécuter une commande sur un conteneur d'un `Pod`. Comme un `Pod` peut représenter plusieurs conteneurs, il est possible d'indiquer le conteneur sur lequel la commande doit être exécutée. S'il n'y a pas de conteneur d'indiquer en paramètre, le conteneur par défaut choisi sera le premier. Dans notre cas, comme il n'y a qu'un conteneur dans notre `Pod`, c'est le conteneur associé à l'image Nginx qui sera utilisé. Les options `i` et `t` indiquent que la commande à exécuter sera interactive et le résultat sera affiché sur le teminal courant. Ces options sont identiques à Docker. L'option `--` précise que les paramètres qui suivront concerneront la commande à exécuter par le conteneur. Dans cet exemple, nous ouvrons un prompt depuis le conteneur.

Nous allons changer la page HTML par défaut, en modifiant le contenu du fichier */usr/share/nginx/html/index.html*.

- Toujours depuis l'invite de commande *kubectl*, vous devriez toujours être dans le prompt du conteneur Nginx :

```
root@myfirstpod:/# echo "Modification de la page web par defaut" >
/usr/share/nginx/html/index.html
root@myfirstpod:/# exit
$ kubectl port-forward myfirstpod 8080:80
```

- Depuis une nouvelle invite de commande, nous allons récupérer le contenu de la page web via l'outil cURL :

```
$ curl http://localhost:8080
Modification de la page web par defaut
```

- Arrêter le pont réseau entre la machine locale et le `Pod` via `CTRL+C` puis supprimer le `Pod` depuis l'invite de commande *kubectl* :

```
$ kubectl delete pods myfirstpod
pod "myfirstpod" deleted
```

Pour l'instant, nous avons créé un `Pod` via l'option `run` de l'outil **kubectl**. Nous allons utiliser un fichier de configuration basée sur une syntaxte YAML.

- Créer dans le répertoire *exercice1-pod-tools/* un fichier appelé `mypod.yaml` en ajoutant le contenu suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer-1
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: mycontainer-2
    image: alpine:latest
    command: ["watch", "wget", "-qO-", "localhost"]
```

Ce fichier de configuration décrit un objet de type `Pod`. Deux conteneurs sont déclarés. Le premier est un conteneur basé sur l'image du serveur web Nginx (identique aux instructions précédentes) et le second est un conteneur basé sur l'image d'une distribution minimaliste Linux Alpine. Pour ce second conteneur, une commande récupère la page HTML du premier conteneur toutes les deux secondes (`watch`). La communication entre des conteneurs d'un même `Pod` se fait via `localhost`.

- Pour créer ce `Pod` dans notre cluster :

```
$ kubectl apply -f exercice1-pod-tools/mypod.yaml
pod/mypod created
```

L'option `apply` permet d'appliquer un fichier de configuration au cluster K8s.

- Vérifions que le `Pod` a été créé dans le cluster K8s :

```
$ kubectl get pods mypod -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP          NODE
mypod   2/2     Running   0          23s   10.42.2.6   k3d-mycluster-agent-1
```

Nous introduisons le paramètre `-o` dans l'option `get` qui permet d'obtenir des informations plus détaillées. Nous remarquons également que les deux conteneurs sont en cours d'exécution (`2/2`). Enfin, le `Pod` est déployé dans le nœud de travail `k3d-mycluster-agent-1`.

- Une autre option intéressante proposée par **kubectl** est `describe` qui permet d'obtenir un détail complet des informations d'un `Pod` :

```
$ kubectl describe pods mypod
Name:             mypod
Namespace:        default
Priority:         0
Service Account:  default
Node:             k3d-mycluster-agent-1/172.29.0.4
Start Time:       Tue, 07 Feb 2023 14:49:58 +0100
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.42.2.6
IPs:
  IP:  10.42.2.6
Containers:
  mycontainer-1:
…
```

- Comme le `Pod` *mypod* dispose de deux conteneurs (*mycontainer-1* et *mycontainer-2*), nous montrons comment exécuter une commande en choisissant un conteneur. Depuis l'invite de commande *kubectl* :

```
$ kubectl exec -it mypod -c mycontainer-1 -- /bin/sh -c "echo 'Helloworld
from K3s' > /usr/share/nginx/html/index.html"
```

Le choix du conteneur se fait via l'option `-c` et dont le nom a été défini dans le fichier *mypod.yaml*. Contrairement à l'exécution précédente d'une commande dans un conteneur, nous modifierons directement le fichier */usr/share/nginx/html/index.html* sans passer par un prompt interactif.

- Pour vérifier que la page web par défaut a été modifiée, nous utiliserons l'option `logs` de **kubectl** qui comme son nom l'indique permet d'afficher le contenu de la sortie console :

```
$ kubectl logs mypod --tail=10 -f -c mycontainer-2
Every 2.0s: wget -qO- localhost                                 2023-02-07
13:53:08

Helloworld from K3s
```

L'option `-f` permet d'afficher en continu l'arrivée de nouveaux messages sur la sortie console. L'option `-c`, déjà utilisée, permet de désigner le conteneur dans lequel nous souhaitons l'affichage des messages de la sortie console. Enfin `--tail=10` n'affichera que les dix dernières lignes.

Nous allons nous intéresser au concept de `Namespace` qui permet de regrouper des `Pods` par projet, par équipe ou par famille. Actuellement, nous n'avons pas utilisé explicitement de `Namespace` lors de la création des `Pods`. Si aucun `Namespace` n'est précisé, un `Pod` sera automatiquement placé dans le `Namespace` intitulé `default`.

L'association d'un `Pod` à un `Namespace` peut être faite soit dans le fichier de configuration, soit depuis la commande **kubectl**. Il est préférable d'utiliser la seconde technique car cela permet d'utiliser un même fichier de configuration dans des `Namespaces` différents. À noter que ce principe sera le même pour les autres types d'objets (`Service`, `PersistentVolume`, etc.).

- Commencer par supprimer le `Pod` précédemment créé :

```
$ kubectl delete pods mypod
pod "mypod" deleted
```

- Créer dans le répertoire *exercice1-pod-tools/* un fichier appelé *mynamespaceexercice1.yaml* en ajoutant le contenu suivant :

```
apiVersion: v1
kind: Namespace
metadata:
  name: mynamespaceexercice1
```

- Pour créer ce `Namespace` dans notre cluster :

```
$ kubectl apply -f exercice1-pod-tools/mynamespaceexercice1.yaml
namespace/mynamespaceexercice1 created
```

- Nous pouvons maintenant recréer notre `Pod` dans ce `Namespace` :

```
$ kubectl apply -f exercice1-pod-tools/mypod.yaml -n mynamespaceexercice1
pod/mypod created
```

L'option `-n` sert à préciser le `Namespace` qui contiendra notre `Pod`.

- Pour lister les `Pods` d'un `Namespace` donné, il faudra soit spécifier le `Namespace` via l'option `n` soit utiliser l'option `-all-namespace` :

```
$ kubectl get pods
No resources found in default namespace.

$ kubectl get pods -n mynamespaceexercice1
NAME    READY   STATUS    RESTARTS   AGE
mypod   2/2     Running   0          24s
```

```
$ kubectl get pods --all-namespaces
```

```
[root@localhost ~]# kubectl get pods --all-namespaces
NAMESPACE            NAME                                      READY   STATUS              RESTARTS         AGE
kube-system          local-path-provisioner-79f67d76f8-zgj5l   1/1     Running             0                78m
kube-system          coredns-597584b69b-t7mqp                  1/1     Running             0                78m
kube-system          metrics-server-5f9f776df5-h4fdb           1/1     Running             0                78m
kube-system          helm-install-traefik-crd-blwn2            0/1     Completed           0                78m
kube-system          helm-install-traefik-tllkn                0/1     Completed           6                78m
kube-system          traefik-66c46d954f-95dgj                  1/1     Running             0                66m
kube-system          svclb-traefik-d5fc38d6-8s4ss              0/2     CrashLoopBackOff    34 (4m57s ago)   66m
kube-system          svclb-traefik-d5fc38d6-4ssdq              0/2     CrashLoopBackOff    34 (4m35s ago)   66m
kube-system          svclb-traefik-d5fc38d6-496wt              0/2     CrashLoopBackOff    34 (4m24s ago)   66m
mynamespaceexercice1_ mypod                                    0/2     ContainerCreating   0                43s
```

Vous remarquerez dans la première commande que seuls les `Pods` dans le `Namespace` par défaut sont listés, sauf qu'il n'y en a pas. La deuxième commmande liste les `Pods` pour le `Namespace` `mynamespaceexercice1`. Enfin la troisième commande liste tous les `Pods` quelque soit son `Namespace`.

- Si vous supprimez un `Namespace`, tous les objets qu'il contient seront supprimés.

```
$ kubectl delete namespace mynamespaceexercice1
namespace "mynamespaceexercice1" deleted
```

- Vérifier si le `Pod` a été supprimé :

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                      READY   STATUS      RESTARTS      AGE
kube-system   local-path-provisioner-79f67d76f8-flwh9   1/1     Running     0             137m
kube-system   coredns-597584b69b-kwbdc                  1/1     Running     0             137m
kube-system   helm-install-traefik-crd-67bd4            0/1     Completed   0             137m
kube-system   svclb-traefik-86c9da09-xdmwv              2/2     Running     0             136m
kube-system   svclb-traefik-86c9da09-mhm2n              2/2     Running     0             136m
kube-system   helm-install-traefik-k6j29                0/1     Completed   1             137m
kube-system   svclb-traefik-86c9da09-zs7tn              2/2     Running     0             136m
kube-system   traefik-66c46d954f-pzwrl                  1/1     Running     0             136m
kube-system   metrics-server-5f9f776df5-2rgdh           1/1     Running     0             137m
```

Bilan de l'exercice

À cette étape, vous savez :

- manipuler les outils **kubectl** et K9s ;
- créer un `Pod` avec et sans un fichier de configuration ;
- interagir avec un `Pod` pour exécuter une commande sur un des ses conteneurs ;
- tester en créant un pont entre la machine locale et un `Pod` (uniquement pour faire des tests !!!) ;
- créer des `Namespaces` pour organiser ses déploiements.

Pour continuer sur les concepts présentés dans cet exercice, nous proposons les expérimentations suivantes :

- créer un `Pod` basé sur une image Apache HTTP et modifier le contenu du répertoire (*/var/www/html*);
- créer plusieurs `Pods` dans un `Namespace` et le supprimer.