

Kubernetes



Travaux Pratiques



zenika

Pré-requis

L'ensemble des exercices seront réalisés en s'appuyant sur [kubeadm](#).

Vous avez à votre disposition dans le Lab Strigo trois machines : `control-plane`, `worker-1` et `worker-2`. `docker`, `kubeadm` et `kubectl` sont déjà installés avec autocompletion.

Vous avez accès à un Visual Studio Code via le bouton 'code' du Lab Strigo. Le mot de passe est contenu dans le fichier `~/.config/code-server/config.yaml` et affiché au lancement d'un terminal.

Cheat sheet

Une cheat sheet contenant les commandes Docker et Kubernetes usuelles est présente en fin de document.

TP 1 : Premiers pas

TP 1.1 : Installation

Objectif : démarrer un `kubernetes` sur les VMs mises à disposition.

Vous utiliserez `kubeadm` pour construire votre cluster. Il sera composé de :

- 1 noeud *Control plane*
- 2 noeuds *Worker*

Étapes :

- Regardez l'aide de la commande `kubeadm` pour l'argument `init` : `kubeadm init --help`
- Vérifiez que le noeud remplit les attendus (pour installer kubernetes avec `kubeadm`) :

```
sudo kubeadm init phase preflight
```

Control plane

⚠ Les commandes de cette section seront à exécuter depuis la machine `control-plane`.

`kubeadm` va installer les composants du `control-plane` en utilisant `kubelet` à partir des définitions de `pods`.

Initialisez le *control plane* (depuis la machine `control-plane`) :

```
sudo kubeadm init
```

- `kubeadm` démarre (sur ce noeud) le kubelet en tant service `systemd` avec la bonne configuration

```
# Vérifiez l'état du service kubelet ("q" pour quitter lorsque vous voyez que kubelet est running)
sudo systemctl status kubelet
```

- L'initialisation a notamment créé les fichiers de configurations et identification de votre cluster

```
sudo ls /etc/kubernetes/*.conf
```

- Ainsi que les fichiers de définitions statiques (*manifest*) des `pods` du *Control plane*.

```
sudo ls /etc/kubernetes/manifests/
```

- Constatez qu'il y a différents conteneurs `docker` qui sont maintenant démarrés sur le noeud `control-plane`

```
sudo docker container ls
```

Vous pouvez maintenant utiliser votre cluster. Mais pour que la commande `kubectl` interagisse avec celui-ci, vous devez récupérer le fichier de configuration lui permettant de se connecter avec des droits administrateur :

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

(le `sudo` est utile quand vous n'êtes pas root et vous permet de configurer `kubectl` pour votre utilisateur)

- Vous devriez pouvoir maintenant utiliser `kubectl` pour interagir avec votre cluster
- Vérifiez l'état du cluster avec `kubectl get nodes`
- Vérifiez l'état des pods système du cluster avec `kubectl get pods -n kube-system`

Workers

- Sur la machine `control-plane`, lancez `kubeadm` pour créer un jeton de rattachement au cluster :

```
sudo kubeadm token create --print-join-command
```

- Copiez le résultat de la commande (devrait être de la forme `kubeadm join ...`)
- Connectez-vous à chaque machine `worker-1` et `worker-2` afin de les rallier au cluster en exécutant la commande copiée précédemment, préfixée de `sudo` :

```
sudo kubeadm join ...
```

- Retournez sur le noeud `control-plane` et vérifiez l'état du cluster :

```
kubectl get nodes
```

- Vérifiez que vous avez bien tous les `nodes` dans votre cluster. Si ce n'est pas le cas, c'est qu'il y a eu une erreur lors des `kubeadm join` .

Installation du réseau

Votre cluster n'est pas encore fonctionnel. Vous voyez cela en passant la commande `kubectl get nodes` qui indique les noeuds en `NotReady` .

Il manque encore le `network add-on` pour permettre à vos `pods` de communiquer.

Il y a de nombreuses solutions. Vous pouvez voir cela [ici](#).

Dans cet atelier, vous utiliserez le `CNI` *calico* :

```
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml
```

Vérifiez que les nodes sont maintenant `Ready` , un délai de 2 min peut être nécessaire :

```
kubectl get nodes
```

Le résultat attendu est de la forme, sinon attendre un peu et relancer la commande précédente :

```
kubectl get nodes
```

#	NAME	STATUS	ROLES	AGE	VERSION
#	control-plane	Ready	control-plane,master	2m37s	v1.23.17
#	worker-1	Ready	<none>	61s	v1.23.17
#	worker-2	Ready	<none>	47s	v1.23.17

TP 1.2 : kubectl

Objectif : prise en main de la CLI Kubernetes

Prise en main

- Afficher la liste de toutes les commandes disponibles : `kubectl` , `kubectl get --help`
- Vérifier que la complétion est bien en place si vous l'aviez mise en place auparavant `kubectl g<tab>`
- Afficher les versions du client (kubectl) et du cluster : `kubectl version`
- Afficher la liste des options communes à toutes les commandes : `kubectl options`

Infos sur le cluster

- Afficher le nom du cluster actuellement utilisé : `kubectl config current-context`
- Afficher les différents noeuds du cluster : `kubectl get nodes`

Visualisation des groupes et ressources disponible

- Utiliser `kubectl api-versions` pour afficher les groupes supportés par le cluster
- Utiliser `kubectl api-resources` pour afficher les ressources supportées par le cluster

TP 1.3 : Docker Warmup

Objectifs :

- se remémorer les commandes et les principes des conteneurs
- simuler comment Kubernetes lance les conteneurs

Accéder à l'environnement Docker

- Vérifier que vous pouvez vous connecter au daemon docker
- Vérifier que votre version de **Docker** est compatible avec **Kubernetes**

Démarrage d'un conteneur whoami avec Docker

- Lancer un conteneur :
 - en spécifiant son nom `whoami`
 - en mode détaché
 - à partir de l'image `traefik/whoami:v1.8`
- Récupérer l'ip du conteneur `whoami`
- Lancer la commande `curl <ip-de-whoami>:80/api`
 - Pourquoi cela fonctionne-t-il ?
- Lancer la commande `curl localhost:80/api`
 - Que se passe-t-il ?
 - Pourquoi ?

Démarrage d'un conteneur shell avec Docker

- Lancer un conteneur :
 - en spécifiant son nom `shell`
 - en mode détaché
 - à partir de l'image `zenika/k8s-training-tools:v1`
 - dont le process principal est la commande `sleep infinity`
- Se connecter dans le conteneur `shell` et lancer la commande `curl <ip-de-whoami>:80/api`
 - Pourquoi cela fonctionne-t-il ?
- (toujours depuis le conteneur `shell`) Lancer la commande `curl localhost:80/api`
 - Que se passe-t-il ?
 - Pourquoi ?
- Sortir du conteneur `shell`

Démarrage d'un conteneur shell sidekick du conteneur whoami

- Lancer un conteneur :
 - en spécifiant son nom `whoami-shell`
 - en mode détaché
 - qui utilise le même namespace réseau que le container `whoami` (`--net=container:whoami`)
 - à partir de l'image `zenika/k8s-training-tools:v1`

- dont le process principal est la commande `sleep infinity`
- Se connecter dans le conteneur `whoami-shell` et lancer la commande `curl <ip-de-whoami>:80/api`
 - Pourquoi cela fonctionne-t-il ?
- (toujours depuis le conteneur `whoami-shell`) Lancer la commande `curl localhost:80/api`
 - Que se passe-t-il ?
 - Pourquoi ?
- Lancer la commande `ip addr` , quelle est l'ip du conteneur `whoami` ?
- Lancer la commande `ss -lntp` , quels sont les ports en écoute ?
- Sortir du conteneur `whoami-shell`

Exposer le service whoami en dehors des VMs

- Stopper et détruire le conteneur `whoami`
- Le relancer en exposant le port `80` du conteneur sur le port `8080` de la VM hébergeant Docker
- Vérifier avec un navigateur que le service est accessible (les urls `/` et `/api` doivent répondre)

Nettoyage

- Supprimer les conteneurs :
 - `whoami`
 - `shell`
 - `whoami-shell`

TP 1.4 : kubectl run

Objectif :

- refaire l'équivalent du TP 1.3 avec Kubernetes
- investiguer un problème sur un Pod

Prise en main

- Afficher l'aide en ligne de `kubectl run`

Démarrer un Pod

- Utiliser la commande `kubectl run whoami --image=traefik/whoami:v1.8 --port=80` qui va :
 - démarrer un Pod appelé `whoami`
 - à partir de l'image `traefik/whoami:v1.8`
 - et expose le port `80`
- Surveiller le démarrage du Pod à l'aide de la commande `watch kubectl get po` (**ctrl+c** pour sortir de la boucle **watch**)
- Que constatez-vous ?
- Afficher plus d'informations sur le Pod pour accéder à plus d'informations par la commande `kubectl describe` et récupérer notamment son adresse IP dont vous aurez besoin par la suite
- Vérifier que le composant démarré fonctionne en lançant la commande : `curl <ip-du-pod-whoami>:80/api`

Investiguer un problème lors d'un démarrage de Pod

- Utiliser la commande `kubectl run faulty-whoami --image=containous/whoami:nil --port=80` qui va :
 - démarrer un **Pod** (nous verrons plus en détail par la suite ce qu'est un Pod)
 - dont le nom est `faulty-whoami`
 - à partir de l'image `containous/whoami:nil`
 - et expose le port `80`
- Surveiller le démarrage du Pod à l'aide de la commande `watch kubectl get po` (**ctrl+c** pour sortir de la boucle **watch**)
- Constater que le Pod associé ne démarre pas correctement
- Afficher plus d'informations sur le Pod pour accéder aux logs de sa création avec `kubectl describe` et déterminer la cause du dysfonctionnement constaté

Démarrer un Pod Shell

- Utiliser la commande `kubectl run training-shell --image=zenika/k8s-training-tools:v1 --command -- sleep infinity` qui va :
 - démarrer un Pod appelé `training-shell`
 - à partir de l'image `zenika/k8s-training-tools:v1`
 - dont le process principal est `sleep infinity`
- Surveiller le démarrage du Pod
- Lister tous les Pods tournants actuellement
- Lancer la commande `kubectl exec -ti training-shell -- curl <ip-du-pod-whoami>:80/api`
- Afficher l'aide en ligne pour la commande `kubectl exec`

TP 2 : Pods, labels, annotations, Namespaces

TP 2.1 : Pods

Objectifs : créer un Pod et vérifier son comportement

Prise en main de 'kubectl apply'

- Afficher l'aide en ligne de `kubectl apply` et parcourir les options possibles

Utilisation d'un descripteur au format yaml

- Créer un modèle de descripteur pour un Pod au format `yaml` :
 - nommer le Pod `yaml-pod`
 - à partir d'une image `containous/whoami:latest`
 - exposer le port `80` du conteneur
 - Note : il est possible d'utiliser la ligne suivante pour créer un modèle base dans le fichier `yaml-pod.yaml` :

```
kubectl run POD_NAME --image=IMAGE_NAME --port=PORT_VALUE --dry-run=client -o yaml > yaml-pod.yaml
```
- Instancier le Pod en utilisant le descripteur créé
- Faire un appel `http` sur l'ip du pod sur le port 80 pour vérifier le bon démarrage

Ajout d'un conteneur dans le Pod

- Détruire le pod `yaml-pod`
- Modifier le descripteur du Pod `yaml-pod` pour ajouter un **second conteneur**
 - appelé `shell-in-pod`
 - à partir de l'image `zenika/k8s-training-tools:v1`
 - le process principal du conteneur sera `sleep infinity` (consulter la documentation "[Define a Command and Arguments for a Container](#)" pour trouver la syntaxe permettant de le faire)
 - forcer Kubernetes à vérifier que l'image est à jour en spécifiant le `imagePullPolicy`
- Créer le Pod en utilisant le fichier créé
- Faire un appel `http` sur l'ip du pod sur le port 80 pour vérifier le bon démarrage

Tester le whoami en passant par le conteneur sidecar

- Afficher l'aide en ligne de `kubectl exec`
- Se connecter dans le conteneur `shell-in-pod` du Pod `yaml-pod` pour ouvrir une session bash à l'aide de `kubectl exec`
- Vérifier que l'application `whoami` est bien accessible sur le port `80` de `localhost`

Modifier un Pod

- Afficher les informations détaillées pour le Pod `yaml-pod` avec `kubectl describe` et avec `kubectl get pod POD_NAME -o yaml`
- Modifier le descripteur yaml pour ajouter un label `from-descriptor` avec comme valeur `yaml` (s'appuyer sur l'exemple ci-dessous, source : `workspaces/Lab2/pod--debian-with-label--2.1.yaml`) [nous verrons à quoi servent les labels dans la section suivante]

```
---
apiVersion: v1
```

```
kind: Pod
metadata:
  name: debian-pod-with-labels
  labels:
    my-label: cool
    my-other-label: super
spec:
  containers:
  - name: shelly
    image: debian:10-slim
    imagePullPolicy: Always
    command:
      - bash
      - -c
      - sleep infinity
  restartPolicy: Always
```

- Appliquer la modification sur le Pod `yaml-pod` et vérifier la bonne prise en compte du changement

Visualiser la definition d'un Pod

- Lancer la commande `kubectl get pod yaml-pod -o yaml`
- Observer les différentes sections de la description

Optionnel : utilisation d'un descripteur au format json

- Créer un modèle de descripteur pour un Pod au format `json` à partir de la configuration du Pod précédent
 - nommer le Pod `json-pod`
- Créer le Pod avec `kubectl apply`

TP 2.2 : Labels

Objectifs :

- afficher et ajouter des labels sur un Pod et un Node
- utiliser des labels pour sélectionner des Pods
- utiliser les labels pour contrôler le scheduling d'un Pod

Prise en main

- Afficher l'aide en ligne de `kubectl label`
- Afficher les labels des Pods existants
- Ajouter les labels `release=stable` et `stack=market` aux Pods `yaml-pod` et `json-pod` (s'il est présent)
- Modifier le label `release=stable` en `release=unstable` pour le Pod `yaml-pod`
- Afficher la description complète du Pod `yaml-pod` et vérifier que les labels attendus sont bien présents

Afficher les labels des Pods

- Afficher tous les labels de tous les Pods
- Afficher les labels `run`, `release` et `stack` dans des colonnes dédiées
- N'afficher que les Pods qui ont le label `stack` positionné
- N'afficher que les Pods qui n'ont pas le label `release` positionné
- Afficher les Pods pour lesquels le label `run` vaut `whoami` ou `yaml-pod`
- Afficher les Pods pour lesquels le label `run` vaut `whoami` ou `yaml-pod` et pour lesquels `stack` n'est pas spécifié

Utilisation du nodeSelector

- Lister les labels actuels des noeuds du cluster
- Créer un descripteur de Pod :
 - nom : `light-sleeper`
 - 1 conteneur :
 - Utilisation de l'image `debian:10-slim`
 - Process principal : `bash -c 'sleep 3d'`
 - Labels :
 - `from-descriptor = yaml`
 - Contraindre le placement du Pod sur un noeud qui porte le label `container-runtime=docker`
- Créer le Pod
- Lister tous les Pods dans un terminal en parallèle avec `watch kubectl get pods`
- Quel est l'état du Pod `light-sleeper` ? Pourquoi ?
- Ajouter le label `container-runtime=docker` sur le noeud `worker-1`
- Vérifier que le Pod `light-sleeper` est bien démarré et vérifier sur quel noeud

TP 2.3 : Annotations et Namespaces

Objectifs :

- afficher et ajouter des annotations sur un Pod
- afficher les Namespaces
- créer un Namespace
- créer des ressources dans un Namespace défini

Annotations

- Consulter l'aide en ligne de la commande `kubectl annotate`
- Consulter les annotations existantes sur le Pod `yaml-pod`
- Ajouter l'annotation `super.mycompany.com/ci-build-number=722` au Pod `yaml-pod`
- Vérifiez que l'annotation est bien présente sur le Pod

Namespaces

- Afficher la liste de tous les Namespaces
- Afficher tous les Pods du Namespace `kube-system`
- Afficher tous les Pods de tous les Namespaces

Création d'un nouveau Namespace

- Créer un nouveau Namespace `my-sandbox` par le biais d'un fichier descripteur définissant le label `sandbox="true"`
- Instancier le Pod `yaml-pod` dans le Namespace `my-sandbox` sans modifier le descripteur du Pod
- Lister tous les pods ayant le label `run=yaml-pod` de tous les namespaces
- Optionnel : modifier le descripteur de déploiement du Pod `json-pod` afin de l'instancier dans le Namespace `my-sandbox`
- Lister les Pods du Namespace `my-sandbox`
- Supprimer le Namespace `my-sandbox`

TP 2.4 : logs

Objectifs : consulter les logs d'un Pod

Prise en main

- Consulter l'aide en ligne de `kubectl logs`
- Consulter les logs du conteneur `yaml-pod` du Pod `yaml-pod` créé lors du Lab 2.1
- Créer un Pod à partir du descripteur suivant (source : `workspaces/Lab2/pod--whoami-and-clock--2.4.yml`)

```
---
apiVersion: v1
kind: Pod
metadata:
  name: whoami-and-clock
spec:
  containers:
    - name: whoami
      image: containous/whoami:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 80
    - name: clock
      image: debian:10-slim
      command:
        - bash
        - -c
        - while true; do date | tee /dev/stderr; sleep 1; done
```

- Consulter les logs du conteneur `clock` du Pod `whoami-and-clock`
- Expérimenter les options `--follow`, `--tail` et `--since`

TP 2.5 : Init Containers

Objectifs : ajouter un *Init Containers*



Expérimentations

- Consulter les specs des **Init Containers** avec `kubectl explain`
- Repartir du descripteur du Pod `whoami-and-clock` pour créer un nouveau Pod `whoami-and-clock-with-init` :
 - Ajouter un **Init Container** :
 - appelé `timer`
 - qui utilise l'image `debian:10-slim`
 - qui boucle en affichant la date toutes les 1 secondes pendant 15 secondes (avec la commande `bash` suivante `for i in {1..15}; do date; sleep 1s; done` par exemple)
- Lancer le Pod `whoami-and-clock-with-init`
- Vérifier avec un `watch kubectl get po whoami-and-clock-with-init` que le Pod met plus de 15 secondes pour se lancer
- Accéder aux logs du conteneur `timer` et vérifier qu'ils contiennent l'affichage de la date comme attendu

TP 2.6 : Cycle de vie

Objectifs : comprendre la relance automatique des conteneurs

Expérimentations

- Exécuter la commande `watch kubectl get po -o wide` dans un terminal
- Déterminer le noeud sur lequel est le pod `whoami-and-clock` est déployé
-  Se connecter sur ce noeud
- Identifier le conteneur docker correspondant au conteneur `clock` du Pod `whoami-and-clock` :
`docker container ls --filter name=clock_whoami-and-clock`
- Terminer le conteneur identifié précédemment (avec `docker stop` ou `docker kill`)
-  Revenir sur le noeud `control-plane`
- Vérifier que Kubernetes redémarre bien le conteneur dans le Pod `whoami-and-clock` et que le compteur de **restart** a évolué

Restart Policy

- Créer un Pod à partir du descripteur suivant (source : `workspaces/Lab2/pod--restart-policy-rp-check--2.6.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: rp-check
  labels:
    app: rpchk
spec:
  restartPolicy: Always
  containers:
  - name: rp-check
    image: debian:10-slim
    command:
      - bash
      - -c
      - sleep 15s
```

- Surveiller la liste des Pods avec un `watch kubectl get po`
- Que constatez-vous ?
- Supprimer le Pod créé
- Modifier la valeur de `restartPolicy` à `OnFailure`
- Recréer le Pod
- Surveiller la liste des Pods avec un `watch kubectl get po`
- Que constatez-vous ?
- Afficher la liste des Pods.
- Supprimer le Pod `rp-check`
- Ajuster la commande dans le descripteur, remplacer `sleep 15s` par `sleep 15s ; false` (Note : la commande `false` retourne un `status code` qui vaut 1)
- Recréer le Pod `rp-check`
- Surveiller la liste des Pods avec un `watch kubectl get po`

- Que constatez-vous ?

TP 2.7 : Liveness probes

Objectifs : mettre en oeuvre une sonde de *liveness* et vérifier le fonctionnement

Liveness

- Créer un descripteur avec le contenu suivant (source : `workspaces/Lab2/pod--liveness-probe-starter--2.7.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: liveness-http
  labels:
    test: liveness
  annotations:
    what-do-you-like: crash-with-error-500
spec:
  containers:
    - name: i-am-alive
      image: k8s.gcr.io/e2e-test-images/agnhost:2.21
      args:
        - liveness
```

- Ajouter une sonde de type *liveness* pour le conteneur `i-am-alive` :
 - de type `httpGet`
 - sur le port `8080` du conteneur
 - qui teste le chemin `/healthz`
 - délai initial : 10s
 - timeout : 1s
 - le nombre minimum d'échecs pour que la sonde soit considérée en état **échec** : 5
- Créer le Pod
- Observer les états successifs du Pod
- Afficher les événements associés au Pod

TP 3 : Controllers

TP 3.1 : ReplicaSets

Objectifs :

- créer un *ReplicaSet*
- augmenter le nombre d'instance de *Pod*
- migrer les *Pod* d'un *ReplicaSet* à un autre

Nettoyage

- Supprimer tous les Pods et Services du Namespace **default** à l'aide de la commande `kubectl delete all --all` (ne pas s'inquiéter de la suppression du service `kubernetes`)
- Optionnel : observer les différents états des ressources avec la commande `watch kubectl get pod,rs --show-labels`

Premier ReplicaSet

- Créer un *ReplicaSet* appelé `nginx` à partir du descripteur suivant (source : `workspaces/Lab3/rs--nginx-simple--3.1.yml`) :

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      level: novice
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
        level: novice
    spec:
      containers:
        - name: nginx
          image: nginx:alpine
          ports:
            - containerPort: 80
```

- Surveiller l'avancement de la création du *RS* et des Pods associés avec la commande `watch kubectl get pod,rs --show-labels`
- Supprimer un des Pods créés et vérifier que le *RS* fait son travail
- Créer un Pod à partir du fichier `workspaces/Lab3/pod--looks-like-a-nginx--3.1.yml`
- Modifier le label `app` du dernier Pod créé en remplaçant `imposteur` par `nginx` en utilisant la commande `kubectl label` et observer

ReplicaSet et Pod selector

- Créer un *ReplicaSet* à partir du descripteur suivant (source : `workspaces/Lab3/rs--nginx-cannot-create--3.1.yml`) :

```
---
apiVersion: apps/v1
kind: ReplicaSet
```

```

metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
      level: intermediate
  template:
    metadata:
      name: frontend
      labels:
        app: frontend
        level: novice
    spec:
      containers:
        - name: nginx-fe
          image: nginx:alpine
          ports:
            - containerPort: 80

```

- Que se passe-t-il ?
- Corriger le *sélecteur* dans le descripteur et créer le *RS*
- Vérifiez que le ReplicaSet a été créé correctement

Modifications de ReplicaSet

- Modifier le template du *RS* `frontend` en changeant l'image *nginx* utilisée pour `nginx:stable-alpine`
- Appliquer la nouvelle configuration du *RS*
- Vérifier qu'il n'y a pas de recréation des Pods
- Modifier avec `kubectl label` les labels sur les Pods `frontend-xxxx` en surchargeant la valeur pour la clé *level* à `advanced`
- Que se passe-t-il ? Pourquoi ?
- Vérifier que l'image docker utilisée par les nouveaux Pods créés est bien `nginx:stable-alpine`

Mise à l'échelle d'un ReplicaSet

- Modifier le nombre de replicas du *RS* `frontend` en le passant à 3 en utilisant `kubectl scale`
- Modifier le nombre de replicas du *RS* `frontend` en le passant à 4 en utilisant `kubectl apply`
- Utiliser `kubectl describe pod frontend-xxxx` sur l'un des Pods associés au ReplicaSet frontend et noter la valeur du champ `Controlled By`
- Utiliser `kubectl get pod frontend-xxxx -o yaml` sur l'un des Pods associés au ReplicaSet frontend et noter la valeur du champ `.metadata.ownerReferences`

Suppression d'un ReplicaSet

- Supprimer le *RS* `nginx` en supprimant aussi ses Pods
- Supprimer le *RS* `frontend` *sans supprimer* ses Pods
- Utiliser `kubectl describe pod frontend-xxxx` sur l'un des Pods associés au ReplicaSet frontend et noter que le champ `Controlled By` n'est plus présent
- Utiliser `kubectl get pod frontend-xxxx -o yaml` sur l'un des Pods associés au ReplicaSet frontend et noter que le champ `.metadata.ownerReferences` n'est plus présent

Migration de Pods d'un ReplicaSet à un autre (Optionnel)

Nous avons désormais plusieurs Pods `frontend-*` orphelins, c'est à dire sans *ReplicaSet*. L'objectif est de les rattacher à un nouveau *ReplicaSet* nommé `frontend-rebirth`.

- Créer un *ReplicaSet* (appelé `frontend-rebirth`) pour piloter les Pods auparavant gérés par le RS `frontend` (`app=frontend, level=novice`)
- Vérifier l'état du *ReplicaSet* avec `kubectl get rs frontend-rebirth`
- Utiliser `kubectl describe pod frontend-xxxx` sur l'un des Pods associés au *ReplicaSet* frontend et noter la valeur du champ `Controlled By`
- Utiliser `kubectl get pod frontend-xxxx -o yaml` sur l'un des Pods associés au *ReplicaSet* frontend et noter la valeur du champ `.metadata.ownerReferences`

TP 3.2 : Jobs et CronJobs

Objectifs : créer un *Job* puis un *CronJob*

Premier Job

- Créer un *Job* à partir du descripteur suivant (source : `workspaces/Lab3/job--compute-pi--3.2.yml`) :

```
---
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      name: pi
    spec:
      containers:
        - name: pi
          image: perl:5.34.0
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

- Accéder aux logs du *Job* une fois celui-ci terminé
- Afficher les informations détaillées sur le *Job*

Un peu de parallélisme

- Faites les modifications pour que le *Job* soit lancé 5 fois, avec 2 occurrences en parallèle

CronJob (Optionnel)

- Créer le *CronJob* à partir du descripteur suivant (source : `workspaces/Lab3/cron--hello-from-k8s-cluster--3.2.yml`) :

```
---
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: debian:10-slim
              args:
                - bash
                - -c
                - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

- Après avoir créé le *CronJob*, récupérer son statut avec `kubectl get cronjob`
- Surveiller le premier *Job* associé : `kubectl get jobs --watch`
- Récupérer le statut du *CronJob* une fois que le premier *Job* est passé
- Penser à supprimer le *CronJob* pour arrêter l'exécution des *Jobs* pour le reste de la formation

TP 4 : Services

TP 4.1 : Services et Service Discovery

Objectif : créer un service et vérifier son comportement

Préparation

- Créer le *ReplicaSet* `whoami` à partir du descripteur suivant (source : `workspaces/Lab4/rs--whoami--4.1.yml`) :

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: whoami
spec:
  selector:
    matchLabels:
      app: whoami
      level: expert
  replicas: 2
  template:
    metadata:
      labels:
        app: whoami
        level: expert
    spec:
      containers:
        - name: whoami
          image: containous/whoami:latest
          ports:
            - name: main-port
              containerPort: 80
```

Création du premier Service

- Créer le descripteur permettant de créer un *Service* appelé `whoami` qui écoute sur le port `8080` et va rediriger le trafic vers tous les Pods qui correspondent à `app=whoami, level=expert` sur le port `80` (le Service devra aussi porter les labels `app=whoami, level=expert`)
- Lister les Services du namespace `default`
- Afficher les informations détaillées du Service `whoami`
- Quelle est l'ip du Service dans le cluster ?

Créer un Pod "rebond"

- Créer un Pod appelé `gateway` à partir du descripteur suivant (source : `workspaces/Lab4/pod--gateway.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: gateway
  labels:
    app: gw
spec:
  containers:
    - name: shell-in-pod
      image: zenika/k8s-training-tools:v1
      command:
        - bash
```

```
- -c
- sleep infinity
```

- Se connecter dans le Pod pour exécuter les commandes (en remplaçant `clusterIp-du-svc-whoami` par la ClusterIP de votre Service)
 - Ping du Service (qui ne fonctionnera pas) : `ping -c 1 -W 1 <clusterIp-du-svc-whoami>`
 - Ping HTTP du Service : `httping -c 1 -t 1 <clusterIp-du-svc-whoami>:8080`
 - Requête le Service : `curl <clusterIp-du-svc-whoami>:8080/api`

Ca marche encore quand on kill les Pods ?

- Depuis le Pod `gateway`, exécuter plusieurs fois la commande `curl <clusterIp-du-svc-whoami>:8080/api` et vérifier que la clé `hostname` retourne des valeurs différentes
- Supprimer les 2 Pods associés au *ReplicaSet* `whoami`
- Depuis le Pod `gateway`, exécuter plusieurs fois la commande `curl <clusterIp-du-svc-whoami>:8080/api` et vérifier que la clé `hostname` retourne des valeurs différentes des précédentes

Et si on augmente le nombre de replicas ?

- Passer le *ReplicaSet* `whoami` à 5 replicas
- Depuis le Pod `gateway`, exécuter plusieurs fois la commande `curl <clusterIp-du-svc-whoami>:8080/api` et vérifier que la clé `hostname` retourne des valeurs supplémentaires par rapport au cas précédent

Utilisation du DNS

- Effectuer les mêmes requêtes curl mais en utilisant les noms DNS complet (*fqdn*) et court

Service avec plusieurs ports (Optionnel)

- Créer le *ReplicaSet* `multi-ports` à partir du descripteur suivant (source : `workspaces/Lab4/rs---multi-ports.yml`) :

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: multi-ports
spec:
  replicas: 1
  selector:
    matchLabels:
      app: multi-ports
  template:
    metadata:
      labels:
        app: multi-ports
    spec:
      containers:
        - name: multi-ports-app
          image: alpine:3.12
          command:
            - ash
            - -c
            - (while true; do echo "web $(date)" | nc -l -p 80; done & while true; do echo "admin $(date -
Iseconds)" | nc -l -p 1234; done)
          ports:
            - name: web-port
              containerPort: 80
            - name: admin-port
              containerPort: 1234
```

- Créer le Service `multi-ports` :

- qui redirige vers les Pods avec `app=multi-ports`
- qui expose le port `web-port` sur le port 80 et le port `admin-port` sur le port 8000
- Tester depuis le Pod `gateway` :
 - `curl ip-du-service:80` doit retourner une chaîne de caractères de la forme `"web samedi 20 avril 2022, 12:15:48 (UTC+0200)"`
 - `curl ip-du-service:8000` doit retourner une chaîne de caractères de la forme `"admin 2022-04-20T12:16:57+02:00"`

TP 4.2 : NodePort et LoadBalancer

Objectifs : rendre accessible un Service via 2 types d'expositions

NodePort

- Modifier le Service `whoami` en un Service de type *NodePort* sans spécifier la valeur du nodePort
- Vérifier avec `kubectl get svc whoami`
- Tester le Service depuis l'extérieur de vos machines en utilisant le `nodePort`
- Afficher les informations détaillées du Service `whoami`

LoadBalancer

Demo Formateur

TP 4.3 : Ingress

Objectifs : installer un Ingress Controller puis l'utiliser

Ingress Controller

- Installer l'Ingress Controller basé sur Nginx via la commande suivante :

```
BASE_URL=https://raw.githubusercontent.com/kubernetes/ingress-nginx
URL_PATH=controller-v1.0.0/deploy/static/provider/baremetal
kubectl apply -f ${BASE_URL}/${URL_PATH}/deploy.yaml
```

- Attendre que le controller soit démarré en lançant la commande suivante :

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

- Définir l'IngressClass `nginx` comme étant celle par défaut :

```
kubectl patch ingressclass nginx -p '{"metadata": {"annotations":{"ingressclass.kubernetes.io/is-default-class":"true"}}}'
```

- Afficher tous les objets créés dans le Namespace `ingress-nginx` avec le label `app.kubernetes.io/name=ingress-nginx` avec la commande suivante :

```
kubectl get all -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx
```

- Noter le port http du service NodePort `ingress-nginx-controller`, nous y ferons référence via le nom `NGINX_NODEPORT`

Ingress whoami

Remarque : Pour le reste du TP, remplacer `FIXME`, par l'IP d'une machine du cluster (`echo ${PUBLIC_IP}`).

- Exposez le Service `whoami` par un *Ingress* qui devra répondre sur l'url `whoami.FIXME.nip.io`
- Vérifiez que l'*Ingress* est correctement configuré avec `kubectl get ingress`
- Testez l'url `http://whoami.FIXME.nip.io:<NGINX_NODEPORT>/`, vérifiez en rafraîchissant la page que vous arrivez bien alternativement sur les différents Pods du Service (voir `Hostname`)
- Observer les logs du Pod `ingress-nginx-controller-...` du Namespace `ingress-nginx`

TP 4.4 : Sonde Readiness

Objectifs :

- mettre en place une sonde *Readiness*
- simuler une erreur sur les Pods

Simuler une erreur sur les Pods du ReplicaSet whoami

- Depuis le Pod `gateway`, vérifier que le code de retour HTTP en GET sur l'URL `http://whoami:8080/health` est 200 :
`curl --head whoami:8080/health`
- Lancer la commande suivante pour changer ce code de retour HTTP à 500 pour tous les Pods derrière le Service `whoami` :

```
kubectl get endpoints whoami --output jsonpath='{range .subsets[0].addresses[*]}{.ip}{"\n"}{end}' | \
while read ip; do echo "Send 500 to ${ip}/health"; kubectl exec gateway -- curl -si --data 500
${ip}/health ; done
```

- Depuis le Pod `gateway`, vérifier que le code de retour HTTP en GET est bien maintenant 500

Ajout d'une sonde Readiness sur les Pods du ReplicaSet whoami

- Éditez le descripteur du ReplicaSet `whoami`
- Ajoutez la sonde Readiness suivante à la définition du conteneur `whoami` :

```
readinessProbe:
  httpGet:
    path: /health
    port: 80
```

- Appliquez les modifications au ReplicaSet
- Optionnel : la colonne *READY* des *Pods* affiche toujours `1/1`, pourquoi ?
- Supprimez les *Pods* `whoami` existants
- Vérifiez que la colonne *READY* des nouveaux *Pods* a bien la valeur `1/1` (avec `kubectl get po`)
- Depuis le Pod `gateway`, exécutez la commande `curl -s --data 500 whoami:8080/health`
- Observer que la valeur de la colonne *READY* passe à `0/1` au bout d'une dizaine de secondes pour l'un des Pods
- Faire le nécessaire pour que tous les Pods `whoami` aient dans leur colonne *READY* la valeur `0/1`
- Vérifier avec `kubectl` (Optionnel : et en testant `http://whoami.FIXME.nip.io:<NGINX_NODEPORT>/`)
- Depuis le Pod `gateway`, exécutez la commande `curl -s --data 200 <ip_d_un_pod_whoami>:80/health`
- Observer la colonne *READY* repasser à `1/1` au bout d'une dizaine de secondes du Pod concerné
- Optionnel : Constaté que lorsque l'on consulte `http://whoami.FIXME.nip.io:<NGINX_NODEPORT>/` c'est toujours ce même Pod qui est retourné
- Faire le nécessaire pour remettre en marche tous les Pods `whoami`. Leur colonne *READY* doit repasser à `1/1`

TP 4.5 : Services headless

Objectifs : tester la mise en place d'un service *Headless*

Restaurer la config précédente du ReplicaSet `whoami`

- Restaurer la configuration précédent du ReplicaSet `whoami` sans la sonde *Readiness*
- Supprimer les *Pods* pour s'assurer qu'ils soient créés sans la sonde

`whoami` headless

- Créer un nouveau Service `whoami-headless` qui redirige le trafic vers tous les Pods qui correspondent à `app=whoami, level=expert` sur le port *80*, mais de type *Headless*
- Lancer la commande `kubectl exec gateway -- nslookup whoami-headless` et vérifier que le résultat est bien celui attendu

TP 4.6 : Port forward

Objectif : Accéder au Pod whoami en utilisant le port-forward

- Afficher l'aide en ligne pour `kubectl port-forward`
- Utiliser la commande `kubectl run whoami --image=containous/whoami:latest --port=80` pour lancer directement un Pod `whoami`
- Utiliser la commande `kubectl port-forward` pour forwarder le port 80 du Pod `whoami` vers le port 8888 local
- Vérifier que l'url `http://localhost:8888/api` retourne le résultat attendu avec la commande `curl http://localhost:8888/api`.
- Supprimer le Pod `whoami`

TP 5 : ConfigMaps et Secrets

TP 5.1 : ConfigMaps

Objectifs : créer des ConfigMaps et les consommer depuis un Pod

Valeurs littérales

- Créer la *ConfigMap* `special-config` qui doit contenir les clés/valeurs suivantes
 - `special.k=cereales`
 - `special.ite=kubernetes`
- Vérifier le contenu de la ConfigMap

Import de fichier

- Créer la *ConfigMap* `game-config` à partir du contenu du répertoire `workspaces/Lab5/configs`
- Vérifier le contenu de la ConfigMap

Utilisation en tant que variable d'env

- Créer un Pod à partir du descripteur suivant (source : `workspaces/Lab5/pod--cm-one-venv--5.1.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-cm-one-venv
spec:
  containers:
    - name: witness
      image: debian:10-slim
      command: [ "bash", "-c", "env ; sleep infinity" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.k
```

- Afficher les logs du Pod et vérifier que la variable d'environnement `SPECIAL_LEVEL_KEY` est bien définie

Importer toutes les clés d'une ConfigMap

- Créer la *ConfigMap* `special-config-for-venv` qui doit contenir les clés/valeurs suivantes
 - `SPECIAL_K=cereales`
 - `SPECIAL_ITE=kubernetes`
- Créer un Pod à partir du descripteur suivant (source : `workspaces/Lab5/pod--cm-all-vars--5.1.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-cm-all-vars
spec:
  containers:
    - name: neo
      image: debian:10-slim
      command: [ "bash", "-c", "env ; sleep infinity" ]
```

```
envFrom:
  - configMapRef:
      name: special-config-for-env
```

- Afficher les logs du Pod et vérifier que toutes les variables d'environnement SPECIAL_* sont bien définies

Monter un volume à partir d'une ConfigMap

- Créer un Pod à partir du descripteur suivant (source : `workspaces/Lab5/pod--cm-as-vol--5.1.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: cm-as-vol
spec:
  volumes:
    - name: config-volume
      configMap:
        name: game-config
  containers:
    - name: droopy
      image: debian:10-slim
      command: [ "bash", "-c", "ls /etc/config/ ; sleep infinity" ]
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
```

- Se connecter dans le Pod et vérifier le contenu des fichiers dans /etc/config

TP 5.2 : Secrets

Objectifs : créer des Secret et les lire depuis un Pod

Créer un Secret

- Créer un Secret appelé `secret-identities` avec les clés valeurs suivantes :
 - `spiderman`=Peter Parker
 - `superman`=Clark Kent

Utiliser le Secret / variable d'env

- Créer le Pod à partir du descripteur suivant (source : `workspaces/Lab5/pod--civil-war-expose-secret-identity--5.2.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: civil-war
spec:
  containers:
    - name: revelation
      image: debian:10-slim
      command: [ "bash", "-c", "env ; sleep infinity" ]
      env:
        - name: SPIDERMAN_IS
          valueFrom:
            secretKeyRef:
              name: secret-identities
              key: spiderman
```

- Vérifier dans le Pod que la variable d'environnement est valorisée comme attendu

Utiliser le Secret / fichiers

- Créer le Pod à partir du descripteur suivant (source : `workspaces/Lab5/pod--dc-vs-marvel--5.2.yml`) :

```
---
apiVersion: v1
kind: Pod
metadata:
  name: dc-vs-marvel
spec:
  volumes:
    - name: for-your-eyes-only
      secret:
        secretName: secret-identities
  containers:
    - name: revelations
      image: debian:10-slim
      command: [ "bash", "-c", "grep '^' /etc/revelations/* ; sleep infinity" ]
      volumeMounts:
        - name: for-your-eyes-only
          mountPath: /etc/revelations
          readOnly: true
```

- Vérifier dans le Pod que les fichiers contiennent les informations attendues

TP 6 : Deployments

TP 6.1 : Deployments

Objectifs : simuler la mise à jour d'une application en rolling update

Déploiement v1

- Consulter le descripteur de *Deployment* `workspaces/Lab6/deploy--zenika-v1--6.1.yml` :

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: zenika
  labels:
    app: zenika
spec:
  replicas: 3
  minReadySeconds: 5
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      app: zenika
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: zenika
    spec:
      containers:
        - name: app
          image: zenika/k8s-training-deploy:v1
          ports:
            - name: main
              containerPort: 8080
          livenessProbe:
            httpGet:
              path: /.healthcheck
              port: 8080
            periodSeconds: 5
            initialDelaySeconds: 2
          readinessProbe:
            httpGet:
              path: /.readicheck
              port: 8080
            periodSeconds: 5
            initialDelaySeconds: 5
```

- Lancer la commande `kubectl apply -f workspaces/Lab6/deploy--zenika-v1--6.1.yml` pour initier le Deployment
- Vérifier l'état du Deployment, du ReplicaSet et des Pods associés

Service

- Créer le Service associé à partir du descripteur suivant (source : `workspaces/Lab6/svc--zenika.yml`) :

```
---
apiVersion: v1
kind: Service
metadata:
```

```
name: zenika-svc
labels:
  app: zenika
spec:
  selector:
    app: zenika
    version: "1"
  ports:
    - name: main
      protocol: TCP
      port: 80
      targetPort: 8080
```

- Recréer si nécessaire le *Pod* `gateway` utilisé dans le TP4
- Connectez vous dans le Pod gateway et lancer la commande
`while true; do curl --silent http://zenika-svc; sleep 1; echo; done`
- (laisser cette commande tourner)

Mise à jour du déploiement en v2

- Mettre à jour le descripteur de Deployment pour utiliser l'image avec le tag v2
- Mettre à jour le Deployment
- Lancer la commande `kubectl rollout status deployment zenika` de temps en temps pour voir les infos remontées
- Vérifier l'état du Deployment, du ReplicaSet et des Pods associés
- Vérifier dans le résultat de la boucle lancée dans le Pod gateway que l'application est mise à jour
- Lancer la commande `kubectl rollout history deploy zenika` pour constater les différentes versions des déploiements enregistrés

TP 6.2 : Canary

Objectifs : Mettre en place un canary de 20%

Remarque : Pour le reste du TP :

- remplacer `FIXME`, par l'IP d'une machine du cluster (`echo ${PUBLIC_IP}`).
- remplacer `NGINX_NODEPORT` par le port HTTP du service NodePort `ingress-nginx-controller` du Namespace `ingress-nginx`
- Effectuer un rollback du Deployment du TP précédent
- Créer un second Deployment `zenika-v2` basé sur le précédent Deployment en modifiant :
 - le `metadata.name` en `zenika-v2`
 - les labels `version` en `"2"`
 - le tag de l'image en `v2`
- Déployer un second Service `zenika-svc-v2` pointant sur les Pods du Deployment `zenika-v2` (se baser sur le Service `zenika-svc` en modifiant la valeur du label `version`)
- Déployer l'Ingress de la v1 avec le fichier `workspaces/Lab6/canary-v1.yml` (penser à remplacer le `FIXME`)
- Créer un second Ingress `zenika-ing-v2` basé sur le précédent Ingress mais pointant sur le Service `zenika-svc-v2` et définissant les annotations de canary NGINX pour recevoir 20% du trafic
- Tester l'url `http://deploy.FIXME.nip.io:NGINX_NODEPORT/` et vérifier le canary testing

TP 7 : Opérateur

TP 7.1 : Opérateur prometheus

Objectifs : Installer et utiliser l'opérateur prometheus

Découverte et installation de l'opérateur

- Rendez vous sur la page de l'opérateur prometheus
- Explorer rapidement le [README.md](#) et [bundle.yaml](#)
- Lancer la commande

```
kubectl apply -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/master/bundle.yaml
```

pour installer l'opérateur
- Vérifier l'état la présence des nouvelles ressources sur le Cluster

Déploiement d'une stack Prometheus (démonstrateur)

- Suivre le tutorial officiel : <https://github.com/prometheus-operator/prometheus-operator/blob/master/Documentation/user-guides/getting-started.md>

TP 8 : Affinity, taints et tolerations

8.1 : Affinity

Dans cet atelier, nous allons déployer wordpress sur le cluster. Nous allons utiliser les descripteurs qui se trouvent dans `Lab8/affinity/`. Ces Deployments auront chacun un Pod et nous voulons que ces Pods tournent sur le même Node. Nous allons utiliser `affinity.podAffinity.requiredDuringSchedulingIgnoredDuringExecution`

- Créer le Secret pour le password de la base de données Mysql :
`kubectl create secret generic mysql-pass --from-literal=password=PASSWORD`
- Créer les Deployments : `kubectl apply -f workspaces/Lab8/affinity/`
- Vérifier que `wordpress` et `wordpress-mysql` ne sont pas sur le même Node :
`kubectl get pods -l app=wordpress -o wide`
- Modifier les descripteurs pour que les Pods soit schedulés sur le même Node

Anti-affinity

- Nous allons réutiliser les Pods déployés dans l'exercice précédent
- Adapter le descripteur dans `workspaces/Lab8/anti-affinity/mysql-backup.yaml`
 - Ajouter de l'anti-affinity pour s'assurer que le Pod sera créé sur un autre Node.
- Créer le Pod : `kubectl apply -f workspaces/Lab8/anti-affinity/`
- Vérifier que le Pod wordpress-backup est bien sur un autre Node : `kubectl get pods -l app=wordpress -o wide`

Nettoyage

- Nettoyer les ressources créées pour wordpress : `kubectl delete all -l app=wordpress`

8.2 : Taints & Tolerations

- Créer un Pod en utilisant le descripteur `workspaces/Lab8/taints/shell-pod.yaml` :
 - Noter le `nodeSelector` pour s'assurer que le Pod démarre sur le noeud `control-plane`

```
---
apiVersion: v1
kind: Pod
metadata:
  name: shell
  labels:
    name: shell
spec:
  nodeSelector:
    kubernetes.io/hostname: control-plane
  containers:
  - name: shell
    image: debian:10
    command:
      - bash
      - -c
      - "sleep infinity"
```

- Regarder l'état du pod avec `kubectl get pods`
 - Quel est l'état du Pod ? Pourquoi ?
- Modifier le descripteur pour que le Pod puisse tourner sur `control-plane`
 - Pour lister les taints du Node : `kubectl describe node master`

TP 9 : Kustomize

TP 9.1 : Kustomize

Remarque : Pour le reste du TP :

- remplacer `FIXME`, par l'IP d'une machine du cluster (`echo ${PUBLIC_IP}`).
- remplacer `NGINX_NODEPORT` par le port HTTP du service NodePort `ingress-nginx-controller` du Namespace `ingress-nginx`
- Créer un Namespace `dockercoins` : `kubectl create ns dockercoins`
- Observer la structure des fichiers présents dans le répertoire `workspaces/Lab9`
- Visualiser le contenu du fichier `workspaces/Lab9/base/kustomization.yaml` fourni
- Remplacer la valeur `FIXME` dans le fichier `workspaces/Lab9/base/kustomization.yaml`
- Lancer la commande `kubectl kustomize workspaces/Lab9/base` et observer le résultat.
- Appliquer la configuration de base : `kubectl apply --kustomize workspaces/Lab9/base`
- Vérifier l'application dans votre navigateur à l'adresse `http://dockercoins.<PUBLIC_IP>.nip.io:NGINX_NODEPORT`
- Appliquer la configuration de l'overlay `perfs` : `kubectl apply --kustomize workspaces/Lab9/overlays/perfs`
- Surveiller la création des Pods dans le Namespace `dockercoins`
- Appliquer la configuration de l'overlay `upgrade` : `kubectl apply --kustomize workspaces/Lab9/overlays/upgrade`
- Supprimer l'application : `kubectl delete --kustomize workspaces/Lab9/overlays/upgrade`

TP 9.2 : Démo ArgoCD

Suivre ce [tutoriel](#)

- Modification du service argocd-server en LoadBalancer (étape 3 du getting started)
- Récupération du mot de passe généré tel qu'indiqué dans la doc
- Création du projet et déploiement de l'application [kustomize-guestbook](#)

Annexe

Recommandation : commande 'watch' et windows

Lors de plusieurs exercices vous serez invité à lancer une commande (`kubectl` ou autre) précédée de la commande unix `watch`. `watch` est une commande permettant d'exécuter un programme périodiquement en affichant le résultat à l'écran ce qui permet de voir l'évolution des résultats de cette commande au cours du temps. Il n'y a pas d'équivalent de `watch` sous windows, cependant plusieurs options sont possibles pour les utilisateurs windows :

- Utiliser une session locale [MobaXterm](#), qui lance un shell [Cygwin](#) où la commande `watch` est disponible parce que préinstallé. Note: [MobaXterm](#) fait aussi partie des logiciels recommandés pour lancer une connexion ssh, voir section suivante.
- Installer le paquet `watch` sous [Cygwin](#) si ce dernier (ou équivalent) est déjà disponible sur votre poste
- Passer par la commande *PowerShell* `Watch` fournie par le module dont le code source est disponible dans le répertoire `watch-for-windows-ps` existant dans le zip fourni pour démarrer les Labs. Pour pouvoir utiliser ce module, déposer le fichier `Watch.ps1` dans le répertoire `$home\Documents\WindowsPowerShell\Modules\Watch`, `$home` représentant le répertoire de base de votre utilisateur courant (`C:\Users\<username>` par exemple). Vous devrez ensuite lancer toutes les commandes `kubectl` depuis une console *PowerShell*. Ce module `Watch` ne fonctionne cependant pas tout à fait comme la commande `watch` linux, vous devrez ajouter un paramètre supplémentaire pour spécifier la fréquence de rafraichissement (qui vaut 2 secondes par défaut avec `watch` sous linux) : `watch 2 kubectl get pods`
- Si aucune des options précédentes n'est possible, vous pourrez remplacer les commandes `watch kubectl` par `kubectl get <xx> -w` qui permet de rester en attente de modification dans la commande `kubectl` mais avec un affichage moins lisible qu'en passant par la commande `watch`

Recommandation : sous windows, utilisation d'un terminal type

MobaXterm/PuTTY pour les commandes ssh

- Si vous êtes sous windows, nous vous conseillons fortement de passer par un terminal type [MobaXterm](#) ou [PuTTY](#) au lieu d'utiliser `minikube ssh` lorsque demandé dans la suite des Labs pour faciliter l'édition des lignes de commandes (ou vous allez vous heurter potentiellement à l'impossibilité de corriger les commandes shell que vous aller taper)
- Pour configurer les accès à la VM `minikube` en passant par *MobaXterm* ou *PuTTY*:
 - lancer `minikube ssh-key` pour visualiser le chemin vers la clé ssh à utiliser pour vous connecter à la VM
 - convertir la clé au format attendu par `PuTTY` / `MobaXterm`
 - lancer `minikube ip` pour connaître l'ip de la VM `minikube`
 - créer une nouvelle connexion avec ces informations (username : `docker`)

Commandes usuelles pour interagir avec Docker/Kubernetes

Docker

Créer un conteneur en mode interactif : `docker container run -it ubuntu bash`

Créer un conteneur en mode interactif en modifiant l'entrypoint : `docker container run --entrypoint bash -it ubuntu`

Lancer un conteneur en mode détaché : `docker container run -d debian:10-slim sleep infinity`

Afficher les conteneurs en cours d'exécution : `docker container ls`

Ouvrir un shell dans un conteneur en mode interactif : `docker container exec -it <CONTAINER_NAME> bash`

Construire une image Docker : `docker image build . -t mon_image:mon_tag`

Kubernetes

Pods

Créer un Pod en mode interactif `kubectl run -ti --image=ubuntu --restart=Never --rm bash`

Lister les Pods en cours d'exécution du Namespace `staging` : `kubectl get pods -n staging`

Lister les Pods en cours d'exécution avec leur IP et noeud d'exécution : `kubectl get pods -o wide`

Afficher le manifest `yaml` d'un Pod : `kubectl get pod <POD_NAME> -o yaml`

Ouvrir un shell dans un Pod/conteneur en mode interactif : `kubectl exec -it <POD_NAME> -- bash`

Si le Pod a plusieurs conteneurs, utiliser cette commande : `kubectl exec -it <POD_NAME> -c <CONTAINER_NAME> -- bash`

Afficher les logs d'un Pod : `kubectl logs <POD_NAME>`

Afficher et suivre les logs d'un Pod : `kubectl logs <POD_NAME> -f`

Afficher les logs d'un conteneur spécifique d'un Pod : `kubectl logs <POD_NAME> -c <CONTAINER_NAME>`

Afficher les informations complètes d'un Pod (dont les événements) : `kubectl describe pod <POD_NAME>`

Namespaces

Lister les Namespaces : `kubectl get ns`

Crée un Namespace : `kubectl create ns <NOM_NAMESPACE>`

Service

Lister les Services : `kubectl get services`

Lister les Endpoints : `kubectl get endpoints`