Retours d’expériences sur l’enseignement du Cloud en école d’ingénieurs

Yves Denneulin

Contexte et contenus

Cursus de concepts et utilisation du Cloud en 3A d'école d'ingénieurs depuis ~10 ans

Élèves avec déjà un gros bagage informatique : système, réseau, Shell, programmation

CM de concepts fondamentaux : IaaS, PaaS, SaaS/FaaS, virtualisation, architectures physiques, sécurité, consommation énergétique

TPs : services de base (VM, K/V), containerisation, orchestration, micro-services, prise en main d'un environnement hyperscalers

Gros projet possible tout au long du semestre pour aller beaucoup plus loin

Début de l'année : 3h de CM puis 3h de TP

Séance de 3h : 1,5h CM, 1,5h TP, 3 séances sur conteneurisation et 2 sur les microservices

Plateformes utilisées et déroulement

Historiquement OpenStack

Hyperscalers : Google, AWS pour découverte (VM, K/V) et SaaS (learnerlab)

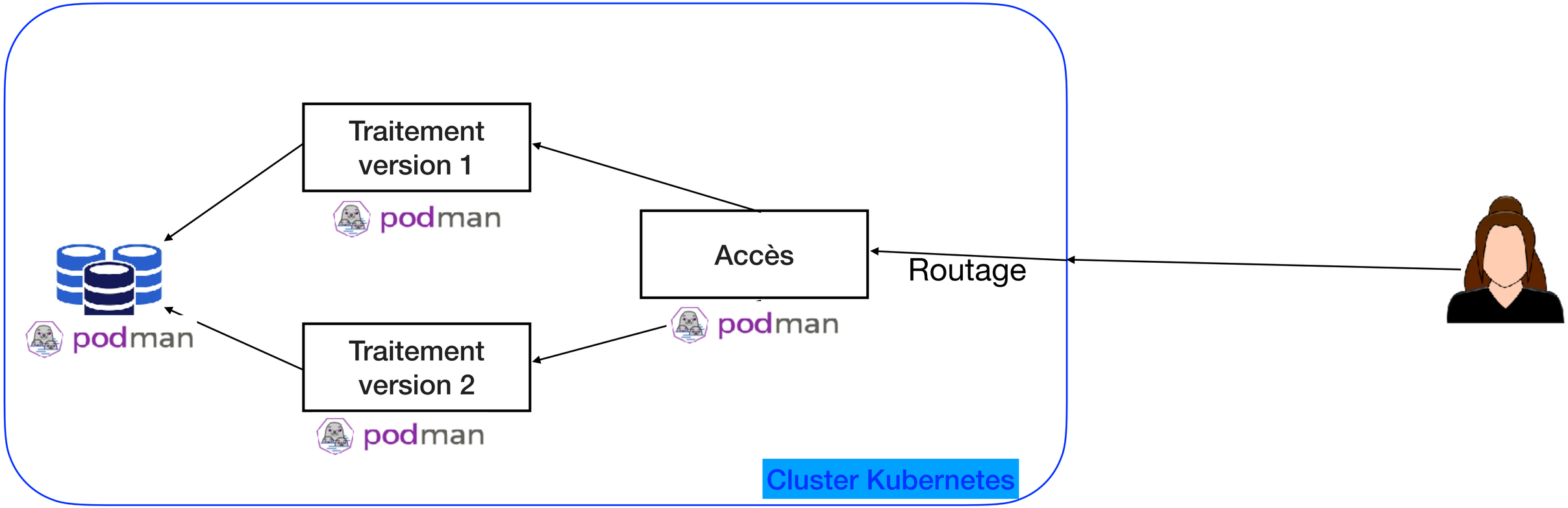
Machines individuelles ou machines de l’école : containérisation et orchestration

Volonté de les laisser essayer et explorer : lien vers des docs, pas d’évaluation, difficulté progressive

TP Kubernetes, TP FaaS

Application typique sur un cloud

À la fin de cette séquence vous aurez créé et déployé sur Kubernetes une application fonctionnant de la manière suivante dans le cloud



Outils de containerisation

Docker ou podman peuvent être utilisés pour ce TP.

Les deux fonctionnent de manière à peu près similaire.

Podman est installé sur les machines de l'Ensimag, si vous voulez utiliser Docker Il faut l'installer sur votre machine. Dans ce cas je vous conseille d'installer Docker Desktop qui propose une interface graphique qui simplifie l'utilisation et embarque aussi un cluster Kubernetes.

Votre premier Container

Un serveur web

Source du container (Dockerfile)

```
FROM nginx
```

Construction de l'image

```
$podman build . -t mon_nginx
```

Vérifier que l'image existe en local

```
$podman images
```

Exécuter le container

```
$podman run mon_nginx
$podman ps
```

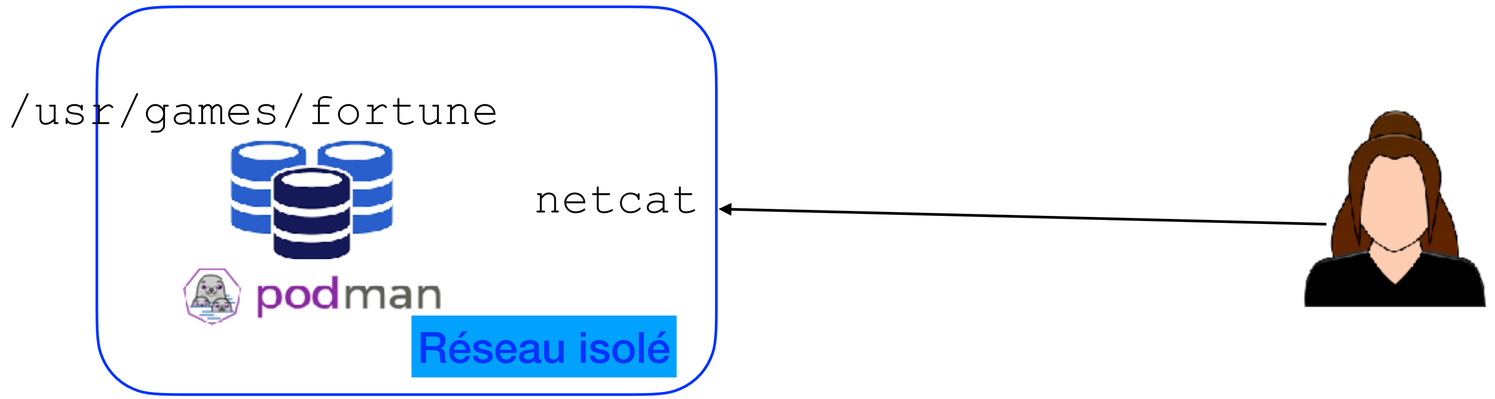
Changer la page d'accueil

Modifier le Dockerfile

```
COPY …
(/usr/share/nginx/html/index.html)
```

Première étape

Concevoir un container et l'interroger



Dockerfile :

```
FROM debian
RUN apt-get update && \
    apt-get -y install bash fortune
netcat-traditional

CMD [ "bash","-c" ,"while [ true ] ; do /
usr/games/fortune | nc -l -p 8088 -q 1 ;
done" ]
```

dans le répertoire où se trouve ce fichier Dockerfile

`docker build . -t mon_container`

crée une image appelée mon_container

puis

`docker run mon_container`

l'exécute avec des options éventuelles (—port …)

Vous exécutez un container qui génère un texte et se met en attente sur un port TCP.

Vous pouvez utiliser indifféremment docker ou podman

Utiliser Kubernetes sur les ensipc

1. Positionner les variables d'environnement XDG_CONFIG_HOME et XDG_DATA_HOME sur un espace perso sur /scratch

```
export XDG_CONFIG_HOME=/scratch/$USER/config
export XDG_DATA_HOME=/scratch/$USER/data
```

2. Avoir MINIKUBE_HOME dans le répertoire /scratch/$USER

```
export MINIKUBE_HOME=/scratch/$USER/.minikube
```

3. Configurer minikube pour être en rootless

```
minikube config set rootless true
```

4. Démarrer minikube avec podman

```
minikube start --driver=podman
```

5. Vérifier que le cluster fonctionne

```
kubectl get nodes
```

Utiliser Kubernetes avec Docker Desktop

Vous avez juste à démarrer le cluster Kubernetes fourni, un client kubectl est installé avec.

```
kubectl get nodes
```

Pour vérifier que le cluster est bien démarré.

Deuxième étape

Exécuter le container fortune dans un cluster Kubernetes

`kubectl run fortnc —image=fort_nc -image-pull-policy=Never`

Démarre un container dans un pod

`kubectl expose pod fortnc --port=8088 --name=svcfort`

Crée un **service** rendu par le pod

`kubectl port-forward service/svcfort 8080`

Exporte le service sur le port 8080 de la machine locale

`kubectl get svc/svcfort -o yaml`

Permet d'avoir la configuration complète du service

Avec minikube, il faut exporter les images des containers dans le cluster Kubernetes

```
podman save localhost/fort_nc:1 -o /tmp/fort_nc.tar
minikube image load /tmp/fort_nc.tar
```

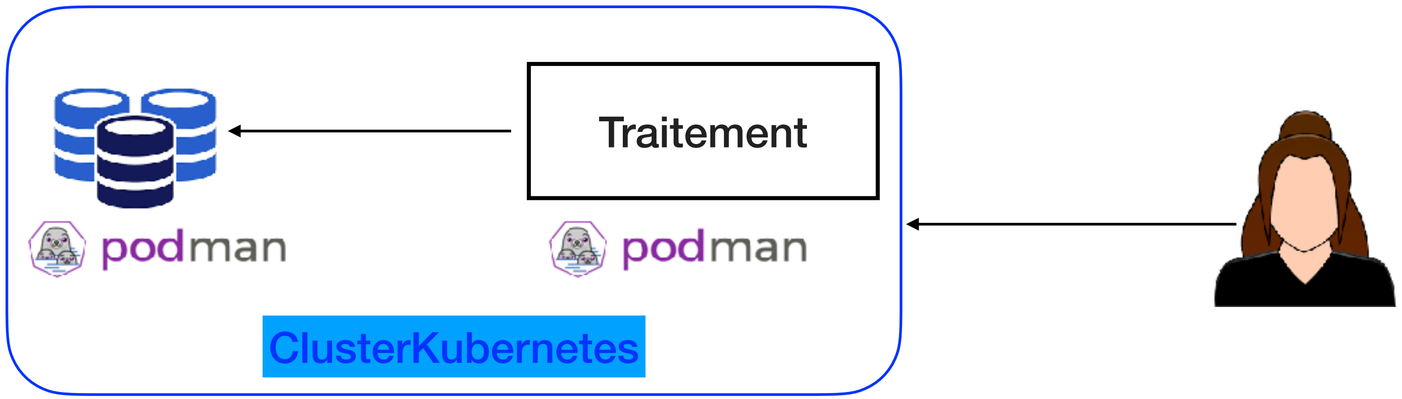
permet d'avoir l'image dans le cluster sous le nom `localhost/fort_nc:1`

Troisième étape

Ajouter une phase de traitement dans un container différent, faire communiquer les deux containers et interroger le traitement

```
FROM debian
RUN apt-get update && \
    apt-get -y install bash fortune netcat-traditional

CMD [ "bash","-c" , "while [ true ] ; nc localhost 8088 | tr '[:lower:]' '[:upper:]') | nc -l -p 8080 -q 1; done" ]
```



Kubernetes : mettre les deux containers dans le même Pod

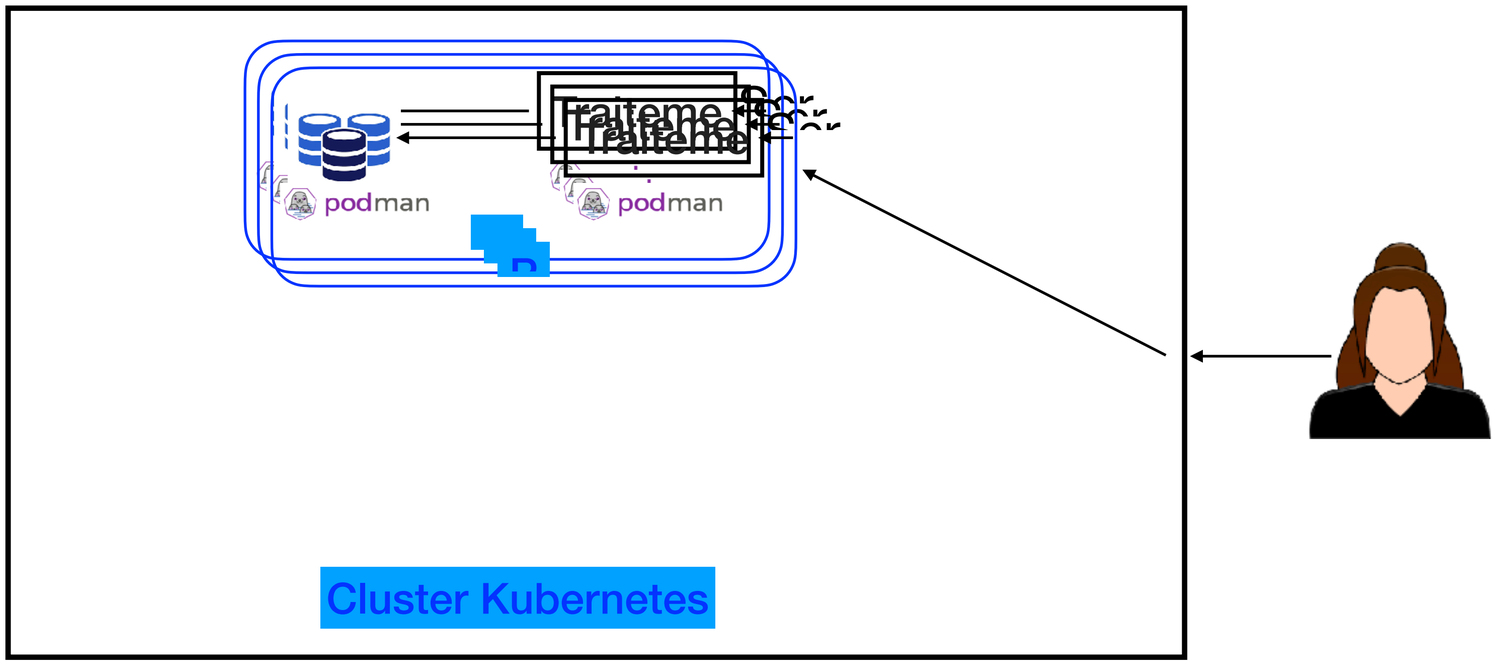
`kubectl get pod fort -o yaml`

donne une description complète du pod à compléter puis à appliquer avec la commande

`kubectl apply -f monpod.yaml`

Quatrième étape

Replication du pod



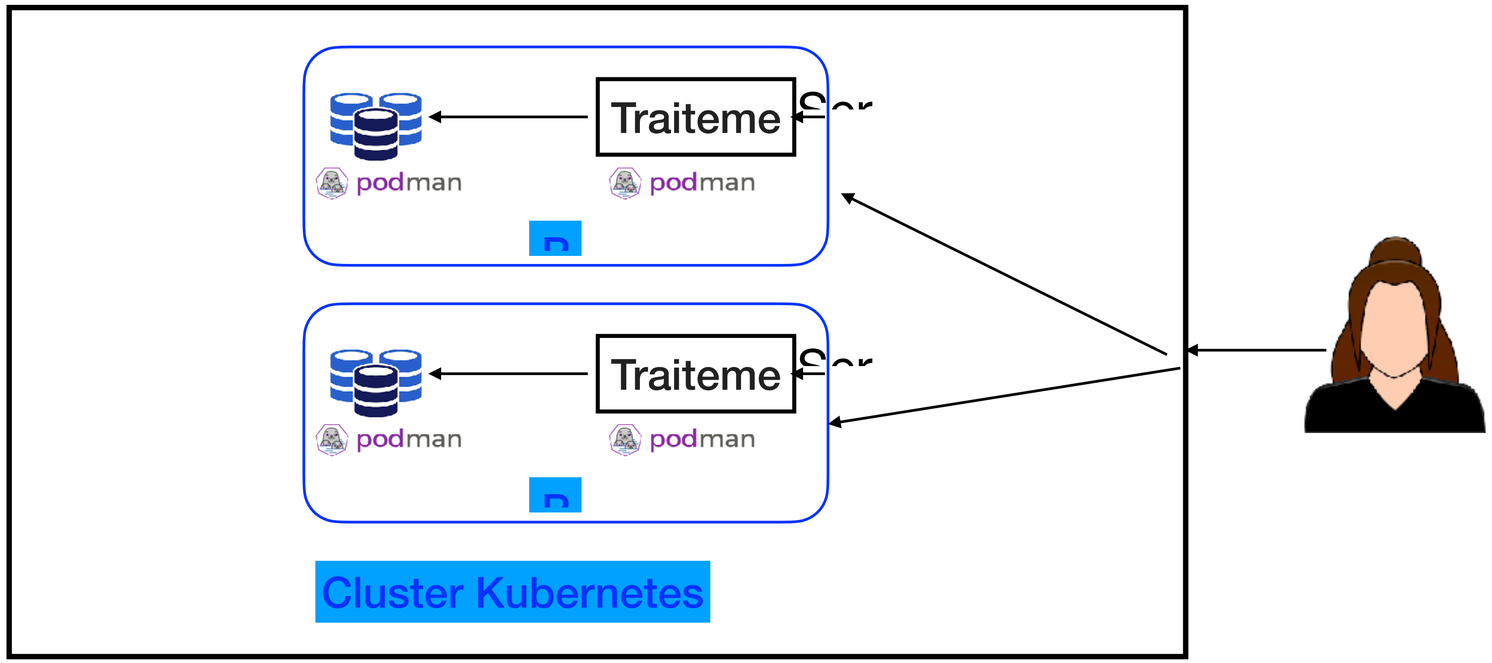
Créer un déploiement pour le pod concerné

Tuer des containers appartenant à un pod, tuer un pod

Observer ce qu’il se passe

Cinquième étape

Deux traitements différents



Les deux traitements sont derrière le même service Kubernetes.

On doit avoir répartition à peu près équitable entre les deux traitements.

TP Function aaS

TP *Function as a Service*

Le TP FaaS se déroule en utilisant AWS. Vous avez du recevoir un message vous permettant de vous connecter pour lancer un Learner Lab AWS dans lequel vous pourrez faire les tutoriels suivants :

- celui de base pour créer et exécuter une fonction Lambda minimale : lien
- un plus avancé où vous configurerez une fonction qui s'exécutera lors d'une écriture dans un bucket S3 : lien

Attention : pour les deux tutoriels lorsqu'un rôle vous est demandé vous ne pouvez pas en créer un mais devez utiliser le rôle LabRole qui est le seul existant dans un learner lab.

Fait sur AWS, difficulté d’avoir un environnement complet dans lequel travailler avec un coût d’apprentissage pas trop important

Préparation d’un TP KNative en cours pour l’année prochaine

TP FaaS

Create a serverless file-processing app

One of the most common use cases for Lambda is to perform file processing tasks. For example, you might use a Lambda function to automatically create PDF files from HTML files or images, or to create thumbnails when a user uploads an image.

In this example, you create an app which automatically encrypts PDF files when they are uploaded to an Amazon Simple Storage Service (Amazon S3) bucket. To implement this app, you create the following resources:

- An S3 bucket for users to upload PDF files to
- A Lambda function in Python which reads the uploaded file and creates an encrypted, password-protected version of it
- A second S3 bucket for Lambda to save the encrypted file in

You also create an AWS Identity and Access Management (IAM) policy to give your Lambda function permission to perform read and write operations on your S3 buckets.