CRV

TME 7

L'objectif de ce TME est de déployer une application de type web sur un cluster kubernetes. A partir des images créé dans la séance précédentes.

Kubernetes

Pour utiliser un cluster kubernetes il y a deux solutions principales :

- lancer un cluster minikube sur votre machine (impossible à la ppti)
- utiliser le cluster fourni (demandez les accès au chargé de TME)

Objectifs

Existant

En se basant sur l'application docker suivante :

```yaml
#docker-compose.yml
version: '3'

services:
  main:
    image: redis:7.2
    hostname: redis
    ports:
      - 6379:6379
    networks:
      - redis-replication

  replica:
    image: redis:7.2
    hostname: redis-replica
    deploy:
      replicas: 6
    command: redis-server --slaveof redis 6379
    depends_on:
```

```
      - main
    networks:
      - redis-replication

  server:
    image: arthurescriou/node-redis:1.0.5
    hostname: server
    ports:
      - 8080:8080
    depends_on:
      - main
      - replica
    networks:
      - redis-replication
    environment:
      - REDIS_URL=redis://redis:6379
      - REDIS_REPLICAS_URL=redis://redis-replica:6379

networks:
  redis-replication:
    driver: bridge
```

NB: Vous pouvez utiliser votre images `node-redis` pour remplacer `arthurescriou/node-redis:1.0.5` .

Attention : Ce fichier ne déploit que la base de données et le serveur node.js.

Kubernetes

On cherche à déployer les mêmes composants que la séance précéndentes

Kubernetes nous propose de créer et manipuler différents types d'objets. Dans ce TME on va s'intéresser à seulement certains :

- deployment
- pods
- services

Il y a plusieurs manières de créer ces différents objets, nous allons utiliser des fichiers Yaml que nous allons appliquer avec des lignes de commandes `kubectl` .

Exemple pour créer un deploiement :

```
#node-redis-deployment.yml
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-redis-deployment
  labels:
    app: node-redis
spec:
  replicas: 3
  selector:
    matchLabels:
      app: node-redis
  template:
    metadata:
      labels:
        app: node-redis
    spec:
      containers:
        - name: node-redis
          imagePullPolicy: Always
          image: arthurescriou/node-redis:1.0.5
          ports:
            - containerPort: 8080
```

Ce fichier permet de créer un déploiement qui contiendra des pods avec le conteneur issu de l'image `node-redis` vu précédemment.

Le déploiment créera 3 instance de ce pod. (On n'en a besoin que d'un pour continuer).

Pour utiliser ce yaml : `kubectl create -f node-redis-deployment.yml` .

Si on le modifie plus tard, `kubectl apply -f node-redis-deployment.yml` , pour appliquer les modifications.

Pour supprimer ce qui a été créé `kubectl delete -f node-redis-deployment.yml` .

Objectifs

En s'inspirant de la partie précédentes créer les éléments suivants :

- la base de donnée redis
  - un déploiment (un seul replica)
  - un service exposant le déploiment
- le serveur node redis

    - un déploiment (plusieurs replicas possible)
    - un service exposant le déploiment
- le front end
    - un déploiment
    - un service exposant le déploiment

Bien veiller à connecter les bonnes adresses pour les configurations à chaque services.

(Utiliser la commande `kubectl get services` .)

Comme à la séance précédentes exposez les endpoints nécessaire avec des tunnels.

CI/CD

Si vous ne pouvez pas créer d'images car vous n'avez pas docker sur votre machine vous pouvez créer une CI/CD.

Github actions

Github propose un freetier pour la CI/CD.

Démarche à suivre

- Créer un repository avec le code que vous voulez build dans votre image
- Créer un fichier `build-image.yml` dans le dossier `.github/workflows` :

```yaml
name: build-image

on:
  push:
    branches:
      - 'master'

jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Login to Docker Hub
        uses: docker/login-action@v3
```

```
      with:
        username: $
        password: $
    - name: Build and push
      uses: docker/build-push-action@v5
      with:
        push: true
        tags: $/${IMAGE_NAME}:1.0.0
```

- Spécifiez `DOCKERHUB_USERNAME` et `DOCKERHUB_TOKEN` dans les secrets de votre repository => https://docs.docker.com/security/for-developers/access-tokens/

A chaque commit sur la branche spécifié une nouvelle image sera build et poussé sur dockerhub.