Architecture logicielle, API, services web et interopérabilité

J3 - Approfondir l'orchestration avec Kubernetes

Partie 0 — Découvrir Kubernetes

Introduction

Nous avons utilisé Docker Swarm dans les derniers TP. Comme abordé, Swarm n'est pas très utilisé en production. Il est largement dépassé par Kubernetes, qui est beaucoup plus complet. Et donc plus complexe.

Dans ce TP, nous allons déployer notre application Django *watchlist* dans un cluster Kubernetes local avec **Minikube**, gérer les réplicas, la base de données, et tester la haute disponibilité.

Les pré-requis

Voici les éléments dont nous aurons besoin pour ce TP, à récupérer dans les cours précédents :

- Docker installé sur votre machine

Consignes de rendu

Faites un rapport simple des différentes commandes utilisées, étape par étape dans un document (word, markdown sur Github, etc.) avec des captures pour montrer que ça marche.
Envoyez à `gael@skalab.fr` avant la fin du TP à 16h.

Partie 1 - Préparer notre application pour Docker

Exercice 1 ⭐⭐

Créez un Dockerfile qui fait tourner Django et qui expose le port TCP/8000 en partant d'une image Python.
Confirmer que tout est ok : http://127.0.0.1:8000 doit afficher votre app watchlist.

Tips :

- préparer le code de l'"application" Django sur votre ordi
- installer Django dans le container en utilisant pip avec un fichier requirements.txt
- injecter l'application dans l'image (COPY)
- Choisissez la bonne commande à lancer (CMD) au démarrage du container
- Utiliser Gunicorn avec WSGI

Exercice 2 ⭐

Déposez votre image sur Docker Hub.
Kubernetes doit pouvoir récupérer l'image depuis un registry.

Partie 2 - Installer et comprendre Minikube

Exercice 3 ⭐

Installer :

- Docker
- kubectl
- Minikube

Puis trouver une commande pour démarrer le cluster et contrôler que le node est prêt.

Exercice 4 ⭐

Afficher les informations du cluster :

```
kubectl cluster-info
kubectl get pods -A
```

Question : Pourquoi voit-on déjà des pods alors que nous n’avons rien déployé ?

Partie 3 - Déployer une application

Contrairement à Swarm (docker stack), Kubernetes fonctionne avec des manifests YAML.

Exercice 5 - Créer un Deployment ⭐

Créer un fichier `watchlist-deployment.yaml` pour déployer votre application watchlist. Votre fichier de configuration devra prendre en compte les infos suivantes :

- nom d'app watchlist
- utiliser votre image Docker via le Docker Hub
- exposer votre image via le port 8000
- réplication sur 2 pods

Déployer votre configuration : `kubectl apply -f watchlist-deployment.yaml`

Exercice 6 - Exposer votre application ⭐

Créer un fichier `watchlist-service.yaml` pour exposer votre application sur le port 80. Appliquer cette configuration.

Obtenez l'URL avec `minikube service watchlist-service`

L'application doit être accessible.

Petit point pédagogique

Dans Docker Swarm :

- Service = orchestration
- Replica = nombre d'instances

Dans Kubernetes :

- **Pod** = unité minimale
- **Deployment** = gestion des pods
- **Service** = point d'accès réseau
- **ReplicaSet** = gestion des réplicas (automatique via Deployment)

Exercice 7 - Scale ⭐

Trouver une commande pour changer le nombre de réplicas de votre déploiement sans modifier le YAML.

Vérifier avec `kubectl get pods`

Que remarquez-vous lors du scale down ?

Exercice 8 - Ajouter PostgreSQL ⭐⭐

Comme dans le TP Swarm : SQLite ne fonctionne pas dans un cluster comme celui-ci. Nous allons devoir mettre en place une base de données PostgreSQL accessible par tous nos containers.

Créer un fichier `postgresql-deployment.yaml` pour déployer 1 replicas basé sur une image postgres:15.
Créer également le service associé `postgresql-service.yaml` qui expose le port PostgreSQL (5432).

Appliquez ces configurations.

Modifier les variables d'environnement dans le Deployment Django de l'app watchlist pour pointer vers la base de donnée PostgreSQL.

Exercice 9 - Ajouter un volume persistant ⭐⭐

Créez un `PersistentVolumeClaim` et ajoutez le dans le déploiement PostgreSQL.

Question : pourquoi ne peut-on pas utiliser un simple volume comme en Docker ?

Partie 4 - Haute disponibilité

Exercice 10 ⭐

Supprimer un pod :

```
kubectl delete pod NOM_DU_POD
```

Que se passe-t-il ? Pourquoi ?

Exercice 11 ⭐

Simuler un gros crash :

```
minikube stop
minikube start
```

Vos pods sont-ils toujours là ? Pourquoi ?

Exercice 12 ⭐

Trouver une commande pour activer un dashboard minikube, équivalent au Visualizer Swarm.

Conclusion

Docker Swarm :

- Simple
- Rapide
  Très accessible

Kubernetes :

- Plus complexe
- Plus verbeux
- Mais standard industriel
- Extrêmement puissant

Aujourd’hui vous savez :

- Créer un Deployment
- Exposer une app avec un Service
- Gérer les replicas
- Ajouter une base PostgreSQL
- Ajouter un volume persistant
- Comprendre l’auto-healing

C’est fini, bravo !