Rapport de TP – Introduction à Kubernetes

Samy Mehamli

1 Architecture monolithique et micro-services

Historiquement, les applications étaient conçues sous forme **monolithique**, c'est-à-dire regroupant l'ensemble des fonctionnalités dans un seul bloc logiciel. Dans ce modèle, une erreur dans une partie de l'application peut entraîner l'arrêt complet du système.

L'architecture **micro-services** propose une approche différente : l'application est découpée en plusieurs services indépendants, chacun réalisant une tâche précise et communiquant avec les autres via le réseau.

Les principaux avantages de cette architecture sont :

— l'isolation des erreurs ;
— le déploiement indépendant des services ;
— une scalabilité ciblée selon la charge ;
— une flexibilité technologique accrue.

Cependant, la gestion manuelle de nombreux conteneurs devient rapidement complexe. Kubernetes répond à ce besoin en automatisant le déploiement et la gestion des applications conteneurisées.

2 Présentation de Kubernetes

Kubernetes est une plateforme open source d'orchestration de conteneurs. Elle permet de déployer, gérer, mettre à l'échelle et superviser automatiquement des applications conteneurisées sur un cluster de machines.

Kubernetes repose sur la notion d'**état souhaité** : l'utilisateur décrit ce qu'il veut (nombre de réplicas, image à utiliser, ports), et Kubernetes se charge d'atteindre et de maintenir cet état.

3 Architecture d'un cluster Kubernetes

Un cluster Kubernetes est composé de deux types de nœuds.

3.1 Control Plane

Le **control plane** (anciennement appelé master) représente le cerveau du cluster. Il est responsable de :

— recevoir les requêtes de l'utilisateur via l'API Kubernetes ;
— planifier les pods sur les nœuds disponibles ;
— surveiller l'état du cluster ;
— redémarrer automatiquement les composants défaillants.

3.2 Worker Nodes

Les **worker nodes** sont les machines qui exécutent réellement les conteneurs. Chaque worker héberge des pods et fournit les ressources nécessaires (CPU, mémoire, réseau).

4 Pods

Le **pod** est l'unité de base de Kubernetes. Il contient un ou plusieurs conteneurs (le plus souvent un seul) qui partagent :

— la même adresse IP ;

— le même espace de stockage via des volumes.

Les conteneurs d'un pod sont toujours déployés ensemble sur le même worker node.

Analogie :

— Pod : appartement

— Conteneurs : colocataires

Ils partagent le même réseau et le même espace, tout en pouvant avoir des rôles différents.

5 Environnement de test : Minikube

Pour ce TP, Kubernetes est utilisé via **Minikube**, un outil permettant de lancer un cluster Kubernetes localement sur une machine de développement.

Minikube peut s'appuyer sur Docker pour simuler un cluster, ce qui permet de tester Kubernetes sans infrastructure dédiée.

5.1 Démarrage du cluster

```
minikube start
```

Cette commande démarre un cluster Kubernetes local.

5.2 Liste des nœuds

```
kubectl get nodes
```

```
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   6h43m   v1.34.0
```

Un seul nœud est présent car il s'agit d'un environnement de test. En production, un cluster contient généralement plusieurs worker nodes afin d'assurer la haute disponibilité.

6 Déploiement d'une application

6.1 Création d'un deployment

```
kubectl create deployment hello-nginx --image=nginx
```

Cette commande crée un **deployment** nommé `hello-nginx`, basé sur l'image Docker `nginx`. Le deployment est responsable de la création, de la supervision et de la réplication des pods.

6.2 Liste des pods

```
kubectl get pods
```

7 Scalabilité

Kubernetes permet d'ajuster dynamiquement le nombre de pods associés à un deployment.

```
kubectl scale deployment hello-nginx --replicas=3
```

```
NAME                           READY   STATUS    AGE
hello-nginx-xxxxx              1/1     Running   5s
hello-nginx-yyyyy              1/1     Running   6m
hello-nginx-zzzzz              1/1     Running   5s
```

Kubernetes veille automatiquement à maintenir le nombre de pods demandé, même en cas de défaillance.

8 Exposition du service

8.1 Création d'un service

```
kubectl expose deployment hello-nginx --type=NodePort --port 80
```

Un service de type **NodePort** permet d'exposer une application à l'extérieur du cluster via un port du nœud.

8.2 Accès à l'application

```
minikube service hello-nginx --url
```

Cette commande fournit directement l'URL permettant d'accéder à l'application depuis un navigateur web.

9 Nettoyage et arrêt du cluster

9.1 Suppression des ressources

```
kubectl delete service hello-nginx
kubectl delete deployment hello-nginx
```

9.2 Arrêt du cluster

```
minikube stop
```

10 Conclusion

Ce TP a permis de découvrir les concepts fondamentaux de Kubernetes : architecture du cluster, pods, deployments, services et scalabilité. Kubernetes constitue un outil essentiel pour le déploiement d'architectures micro-services robustes et hautement disponibles, et représente une étape clé vers l'orchestration avancée d'applications conteneurisées.