Université de Paris diderot - Paris 7

Rapport d'exposé

Programmation comparée

Gestion de cluster

Kubernetes vs. Docker Swarm vs. Nomad

Réalisé par

— Hicham BOUZARA

— Mahdi LARBI

2021/2022

1 Introduction aux conteneurs

Un conteneur est un logiciel qui crée un environnement qui encapsule et contient le code et les dépendances d'une application, son but est d'offrir aux développeurs un moyen de mettre en place leurs applications et de les exécuter sans se préoccuper de l'environnement. Les conteneurs sont basés sur la virtualisation de l'environnement d'exécution.

1.1 Conteneur vs Machine virtuelle

Les conteneurs et les machines virtuelles peuvent sembler avoir les mêmes capacités de virtualisation, mais ils présentent des différences architecturales majeures. Contrairement aux machines virtuelles qui doivent avoir un système d'exploitation invité pour chaque image séparément, les conteneurs reposent tous sur le même système d'exploitation hôte, ce qui les rend extrêmement légers par rapport aux machines virtuelles.

1.2 Conteneur Docker

Il est vrai que Docker a gagné une popularité qui entraîne toujours la confusion du concept de conteneurs pour être une invention de Docker, mais la technologie des conteneurs est apparu en 1979 avec la version 7 d'Unix et le système chroot. Le système chroot isole un processus en limitant l'accès d'une application à un répertoire spécifique, ce répertoire étant composé d'une racine et de répertoires enfants. C'était bien sûr bien avant la sortie initiale de Docker en 2013. Cependant, la technologie des conteneurs est devenue populaire après l'apparition du Cloud Computing, lorsque les grandes entreprises ont commencé à migrer leurs systèmes pour devenir Cloud Native.

2 Introduction à Docker

Docker est une solution qui permet de gérer les conteneurs en simplifiant les tâches de création, démarrage et arrêt des conteneurs et en offrant d'autres fonctionnalités plus avancées qu'on verra par la suite de ce rapport.

2.1 Architecture de Docker

L'architecture de Docker se repose sur une architecture *client-serveur*, Le **Docker Client** envoie des commandes au **Docker Host** pour gérer les conteneurs soit sur la même machine en utilisant *l'interface en ligne de commande (CLI)* ou à distance en utilisant *REST API*. **Docker Host** utilise le **Docker Hub** pour télécharger les images docker. (voir la figure 1)

2.2 Docker Moteur (Engine)

Docker Engine est le coeur du système Docker. Il est installé sur la machine hôte et se repose sur trois composants, *Le serveur* qui gère les conteneurs à partir des commandes qui les reçoit depuis le composant *Client* à travers *l'interface en ligne de commandes ou REST API.*

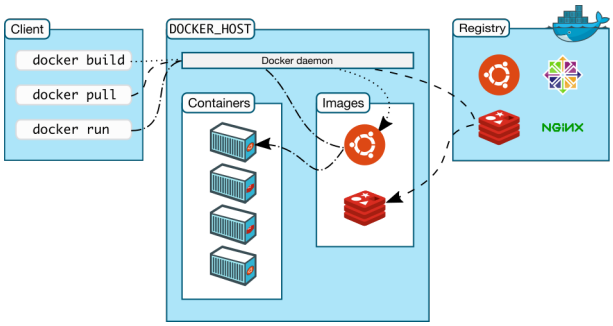

FIGURE 1 – Architecture de Docker

2.3 Objets de Docker

Le système Docker se repose sur les images et les conteneurs, ce qui nous ramène à définir ces objets

2.3.1 Docker image

Docker image est un fichier qui contient un ensemble d'instructions qui permettent la création d'un conteneur Docker, incluant tous les fichiers nécessaires (configuration, code source ...etc). On peux créer une image à partir d'un *Dockerfile* ou en la téléchargeant depuis Docker Hub.

2.3.2 Docker container

Docker container est une instance d'une image Docker, une fois l'image est déployée, elle est exécutée en tant qu'un Docker container.

2.4 Docker Compose et les multi-conteneurs

Prenant un exemple d'une application web qui est basée sur une architecture REST qui sépare notre application en trois couches, Frontend, Backend et la base de données. Chacune de ces couches doit être séparée et déployée sur un conteneur, c'est où Docker Compose entre en jeu, en permettant de gérer et orchestrer des différents conteneurs dans la même machine.

3 Orchestration des conteneurs

3.1 Pourquoi l'orchestration ?

Les conteneurs sont généralement utilisés pour virtualiser l'environnement d'exécution d'une application, cependant, à l'exception des applications à petite échelle, l'architecture des applications logicielles repose sur le principe de la séparation des préoccupations, un exemple célèbre est l'architecture en couches que l'on peut trouver dans la plupart des applications web et mobiles, où la couche d'interface est séparée de la couche métier et de la couche de persistance ou de données, cette séparation devrait de préférence être assortie d'une séparation physique également, c'est là que les multi-conteneurs entrent en jeu, où le développeur d'une application aurait besoin de gérer plusieurs conteneurs dans la machine/serveur hôte.

Mais les grandes entreprises travaillent sur des systèmes qui sont beaucoup plus grands que ceux qui peuvent être gérés dans une seule machine hôte, avec parfois des milliers de conteneurs qui composent les parties complètes du système, en particulier avec l'adoption d'architectures comme l'architecture microservice et l'architecture modulaire, où chaque microservice ou module est une entité indépendante qui communique avec d'autres par le biais de son interface prédéfinie, la nécessité de former un cluster distrbué de conteneurs soulève, et avec elle vient la nécessité d'utiliser un outil qui les orchestre.

3.2 Objectifs

L'orchestration de conteneurs répond au besoin d'automatisation de la gestion du cycle de vie des conteneurs. L'orchestrateur nous donne la possibilité d'avoir les fonctionnalités suivantes :

— Configuration des conteneurs.
— Provisionnement et déploiement des conteneurs
— Assurer la disponibilité et l'évolutivité (auto-scaling).
— Gérer l'équilibrage de charge, la découverte de services et le trafic réseau des conteneurs.
— Surveillance de la santé.
— Gestion des mises à jour et des retours en arrière des conteneurs

4 Outils d'orchestrations

4.1 Docker Swarm

Docker Swarm est le premier gestionnaire de conteneurs Docker, lancé en 2014 par la société Docker et qui est devenu une fonctionnalité native et intégrée au Docker Engine.

L'architecture de Docker Swarm est basée sur une architecture de cluster. Le cluster Swarm est constitué d'un *noeud gestionnaire* qui permet de gérer les différents noeuds et l'état du cluster et des *noeuds esclaves* qui reçoivent des commandes à partir du noeud gestionnaire pour exécuter des conteneurs.

4.2 Kubernetes

Abrégé K8s, Kubernetes est un système d'orchestration de conteneurs open-source. Google a initialement conçu Kubernetes et l'a publié en juin 2014, mais la Cloud Native Computing Foundation maintient désormais le projet. En termes de communauté, Kubernetes a le point fort, car il a été fondé par Google et est soutenu par un grand nombre de développeurs.

En termes d'architecture, il n'y a pas de différences majeures entre Kubernetes et d'autres alternatives connues d'orchestration de conteneurs telles que Docker Swarm ou HashiCorp Nomad, elles reposent toutes sur une architecture maître-esclave avec des noms différents pour les différents outils. Le maître et les travailleurs sont des nœuds, un nœud est une machine hôte. Le nœud maître est celui que le développeur utilise pour gérer et superviser le cluster en utilisant son serveur API pour communiquer avec lui via CLI, le nœud maître a quelques composants comme le gestionnaire de contrôleur qui gère les différents nœuds de travailleurs, et l'ordonnanceur pour diviser les tâches sur les travailleurs. De l'autre

côté, chaque travailleur possède un Kubelet, un logiciel qui joue le rôle d'interface entre le travailleur et le maître.

4.3 Nomad

Nomad est crée par la société HashiCorp en septembre 2015, sa particularité c'est qu'il prend en charge les applications virtualisées, conteneurisées ou autonomes.

L'architecture de Nomad se repose également sur une architecture de cluster, ell est consituée d'un *Serveur Nomad* qui est le coeur du cluster Nomad qui permet de gérer les différentes tâches, *le Nomad Leader* qui est un serveur Nomad qui gère les autres noeuds et maintient l'état du cluster, *le Nomad follower* qui est abonné au Nomad Leader et offre une capacité au cluster de gérer autres noeuds et *les Nomad clients* sont des noeuds qui exécutent les différentes tâches qui lui sont attribuées.

5 Comparaison des outils d'orchestration

Kubernetes	Docker Swarm	Nomad
Installation et configuration		
L'installation du cluster est **difficile** et **compliquée**, il faut installer des applications tierces pour le faire simplement.	L'installation du cluster est **simple** avec deux commandes, initialiser le cluster (init) et le rejoindre (join).	L'installation du cluster est **simple** et il existe des outils externes pour configurer plus rapidement un cluster.
Interface graphique		
Kubernetes Dashboard.	Aucun (CLI/API).	Nomad Dashboard.
Scalabilité et Autoscaling		
Le scaling est très facile avec **Autoscaling**.	Le scaling est plus rapide avec un Scaling **manuel**.	Le scaling est très facile avec **Autoscaling**.
Load Balancing		
Une configuration **manuelle** est requise pour l'activer.	Par défaut, est **automatique**.	Une configuration **manuelle** est requise pour l'activer.
Mises à jour et Rollbacks		
Mettre à jour les **pods** l'un après l'autre. Un rollback **automatique** en cas d'erreur.	Mettre à jour les **conteneurs** l'un après l'autre. Un rollback **manuel** en cas d'erreur.	Mettre à jour les **services** l'un après l'autre. Un rollback **automatique** en cas d'erreur.
Manipulation des Volumes		
Les volumes sont partagés entre les conteneurs qui sont dans le même **pod**.	Les volumes sont partagés entre les conteneurs qui sont dans le même **noeud**.	Les volumes sont partagés entre les conteneurs qui sont dans le même **noeud**.
Monitoring		
Intégré dans Kubernetes.	Il faut installer des applications tierces.	Intégré dans Nomad.