



RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI
INSTITUT DE FORMATION ET DE
RECHERCHE EN INFORMATIQUE



BP 526 Cotonou Tel : +229 21 14 19 88
<https://www.ifri-uac.bj/> Courriel : contact@ifri.uac.bj

MÉMOIRE

pour l'obtention du

Diplôme de Licence en Informatique

Option : Sécurité Informatique

Présenté par :
Codjo Jean TATA

Déploiement et gestion de conteneurs avec Docker Swarm au sein d'une infrastructure informatique: cas de l'UAC

Sous la supervision :
Ing. Victor OYETOLA

Enseignant à l'Institut de Formation et de Recherche en Informatique
Chef du Service de Promotion des Technologies de l'Information et de Communication de
l'Université d'Abomey-Calavi

Membres du jury :
Dr. Eméry ASSOGBA IFRI Président
Ing. Luc SOUKPO IFRI Examineur
Ing. Eric ATTOU IFRI Rapporteur

Année Académique : 2018-2019

Table des matières

Sigles et Abréviations	v
Glossaire	vi
Dédicace	vii
Remerciements	viii
Résumé/Abstract	ix
Introduction	1
2 Revue de littérature	3
2.1 Généralité sur le deploiement des applications	3
2.1.1 Le cycle de vie du développement logiciel	3
2.2 La virtualisation par conteneurs	4
2.2.1 Définition	4
2.2.2 Vue d'ensemble des conteneurs	5
2.2.3 Achitecture	5
2.3 La virtualisation	6
2.3.1 Définition	6
2.3.2 Achitecture	6
2.4 Comparaison entre machine virtuelle et conteneur	7
2.5 Présentation de l'existant	8
3 Matériels et méthodes	9
3.1 Méthodes	9
3.1.1 Présentation de la technologie Docker	9
3.1.2 Architecture de Docker	10
3.1.3 Les principaux services de Docker	11
3.1.4 Les principaux solutions d'orchestrations de cluster de conteneurs . .	12
3.1.5 Choix de la solution adaptée pour l'UAC	18
3.2 Expériences	19
3.2.1 Oracle VM VirtualBox	19
3.2.2 Ubuntu 20.04	19
3.2.3 Docker Engine	19

3.2.4	Vagrant	20
3.2.5	Ansible	20
3.2.6	Git BASH (facultatif)	20
3.2.7	Exigences d'installation	20
3.3	Mise en œuvre	21
3.3.1	Vagrant et Ansible	21
3.3.2	Déploiement de la Base de donnée sous la forme de conteneur	21
3.3.3	Déploiement de la partie Front End de l'application sous la forme de conteneur	21
3.3.4	Déploiement de la partie Back End de l'application sous la forme de conteneur	22
3.3.5	Duplication des conteneurs	22
4	Résultats et suggestions	24
4.1	Résultats	24
4.1.1	Présentation du Box pour les VM	24
4.1.2	Sortir de l'exécution de Vagrant	25
4.1.3	Présentation du Cluster	25
4.1.4	Présentation des services déployés	26
4.1.5	Accès à l'application à partir du Leader	26
4.1.6	Accès à l'application à partir des Workers	27
4.2	Discussion	27
	Conclusion	28
	Bibliographie	29
	Webographie	30
4.3	Annexes	32
4.3.1	Les fichiers de configuration de Vagrant	32
4.3.2	Les fichiers de configuration de Ansible	34

Table des figures

2.1	Les quatre environnements élémentaires du SDLC	3
2.2	Achitecture de virtualiation par conteneur	5
2.3	Achitecture de la virtualisation	6
2.4	Comparaison de l'utilisation des ressources des deux technologies	8
3.1	Architecture de Docker	10
3.2	Architecture technique de Kubernetes	12
3.3	Architecture technique Redhat OpenShift	13
3.4	Architecture Docker-Swarm Mode	17
3.5	Déploiement de la base donnée sous la forme de conteneur	21
3.6	Déploiement de la partie Front End de l'application sous la forme de conteneur	22
3.7	Déploiement de la partie Back End de l'application sous la forme de conteneur	22
3.8	Duplication de la la partie Front End	22
3.9	Duplication de la Base de donnée	23
3.10	Duplication de la la partie Back End	23
4.1	Box Ubuntu 20.04	24
4.2	Résultat de Vagrant	25
4.3	Swarm Manager	25
4.4	Liste des services déployés	26
4.5	Accès au site à partir de l'adresse IP 192.168.6.12 par le port 808	26
4.6	Accès au site à partir de l'adresse 192.168.6.5 par le port 808	27
4.7	Accès au site à partir de l'adresse 192.168.6.4 par le port 808	27
4.8	Script d'installation	32
4.9	Hosts et adresses IP	32
4.10	Vagrantfile	33
4.11	Script d'installation de docker,vagrant et d'autres packages	34
4.12	Désignation de la machine maitre	34
4.13	Création et initialisation du cluster	35

Liste des tableaux

3.1	Forces et faiblesses Kubernetes	13
3.2	Forces et faiblesses OpenShift	14
3.3	Forces et faiblesses Fleet	15
3.4	Forces et faiblesses Mesos Marathon	15
3.5	Comparaison des solutions open-source d'orchestration de conteneur les plus utilisées	18

Sigles et Abréviations

API :	Application Programming Interface
CaaS :	Containers as a Service
CLI :	Comande Line Interface
CPU :	Central Processing Unit
DoS :	Denial of Service
GPG :	GNU Privacy Guard
IFRI :	Institut de Formation et de Recherche en Informatique
IPv4 / IPv6 :	Internet Protocol version 4,6
LXC :	Linux Containers
OS :	Operating System
PaaS :	Platform as a Service
SaaS :	Software as a Service
SDLC :	Software Development Life Cycle
SPTIC :	Service de Promotion des Technologies de l'Information et de la Communication
UAC :	Université d'Abomey-Calavi
YAML :	Yet Another Markup Language

Glossaire

DevOps :

Il s'agit d'un mouvement en ingénierie informatique et une pratique technique visant à l'unification du développement logiciel et de l'administration des infrastructures informatiques, notamment l'administration système.

Docker :

C'est une plate-forme logicielle qui permet aux développeurs de logiciels d'intégrer facilement l'utilisation de conteneurs dans le processus de développement logiciel.

Docker Engine :

C'est un outil client-serveur sur lequel repose la technologie de container pour prendre en charge les tâches de création d'applications basées container. Le moteur crée un processus daemon server-side permettant d'héberger les images, les containers, les réseaux et les volumes de stockage.

Docker Hub :

C'est un dépôt basé sur le cloud qui est totalement géré par l'organisation Docker; C'est un dépôt en ligne où les images Docker peuvent être publiées et utilisées par les utilisateurs de la communauté Docker.

Docker Swarm :

C'est un groupe de machines physiques ou virtuelles configurées pour se rejoindre dans un cluster qui exécutent l'application Docker.

Dockerfile :

C'est un fichier texte décrivant les différentes étapes permettant de partir d'une base pour aller vers une application fonctionnelle.

Image :

Une image est un package de fichiers exécutables qui contient tout le code, les bibliothèques, le runtime, les binaires et les fichiers de configuration nécessaires pour exécuter une application.

Dédicace

A

mon père **Antoine HOUNDJO**

ma mère **Dénise GBEHO**

mon Feu grand-père **Tegbe GBEHO**

monsieur **Sophonie CHAGAS**

mes frères et sœurs.

Remerciements

Nous ne saurions réalisé ce travail sans exprimer nos vifs remerciements a toute personne ayant contribuer de près ou de loin a la réalisation de ce travail,

- Merci à monsieur Eugène C. EZIN, Directeur de l'Institut de Formation et de Recherche en Informatique ([IFRI](#)) , à tout le corps administratif et professoral qui ont largement contribué à notre formation durant ces trois années d'étude en licence sécurité informatique.
- Nous souhaitons adresser nos remerciements les plus sincères à notre directeur de stage et maitre mémoire M. Victor OYETOLA pour son aide,sa disponibilité,pour le temps consacrer et la rigueur scientifique neccessaire apportée au bon déroulement de ce travail,ainsi qu'à M. Eric ATTOU et tous les informaticiens et personnels administratifs de [SPTIC](#) - [UAC](#).
- Tous nos aînés des promotions précédentes des filières SI (Sécurité Informatique) et GL (Génie Logiciel) qui nous ont aidés à grandir dans cet institut.
- Nous adressons un merci tout particulier à tous les camarades de la promotion réalisant aussi leurs mémoires pour l'entraide dans les moments de faiblesses.
- Enfin nous remercions nos parents, frères, sœurs, et proches qui nous ont soutenus physiquement, moralement et financièrement durant les moments difficiles de ce travail,qu'ils trouvent ici le témoignage de notre reconnaissance.

Merci à tous et à toutes

Résumé

Au cours des dernières années, l'utilisation des technologies de virtualisation dans les infrastructures informatiques a considérablement augmenté. Ceci est dû principalement aux avantages en termes d'efficacité d'utilisation des ressources et de robustesse que procure la virtualisation. La virtualisation par conteneurs docker et la virtualisation par hyperviseurs sont les deux principales technologies apparues sur le marché. Parmi elles, la virtualisation par conteneurs docker se distingue par sa capacité de fournir un environnement virtuel léger et efficace, mais qui nécessite la présence d'un orchestrateur.

Afin de relever les nouveaux défis des technologies du Cloud, les entreprises adoptent des cycles de développement applicatif et de déploiement très rapides. Les approches classiques de déploiement et de sécurité ne sont plus compatibles et l'automatisation des tâches est devenue incontournable. Automatiser toutes les étapes de déploiement d'une application représente une économie en temps et une diminution des risques d'erreurs certaine.

L'objectif principal de ce travail est d'automatiser le déploiement d'une application conteneurisée dans un cluster de machine docker.

Mots clés : conteneurs Docker, déploiement, disponibilité, réplication, Docker Swarm, virtualisation, automatisation.

Abstract

In recent years, the use of virtualization technologies in IT infrastructures has increased significantly. This is mainly due to the advantages in terms of resource efficiency and robustness that virtualization provides. Docker container virtualization and hypervisor virtualization are the two main technologies that have emerged in the market. Among them, docker container virtualization is distinguished by its ability to provide a lightweight and efficient virtual environment, but requires the presence of an orchestrator.

To meet the new challenges of cloud technologies, companies are adopting very rapid application development and deployment cycles. Traditional approaches to deployment and security are no longer compatible and automation of tasks has become a must. Automating all the steps of an application deployment represents a saving in time and a reduction in the risk of errors. The main objective of this work is to automate the deployment of a containerized application in a docker machine cluster.

Key words: Docker containers, deployment, availability, replication, Docker Swarm, virtualization, automation.

Introduction

Depuis l'apparition de l'informatique au milieu du XXIème siècle, la performance des matériels tels que les unités de traitement ou de stockage ainsi que les équipements réseaux, n'a cessé de croître de façon continue et exponentielle. Cette tendance est à l'origine du développement d'applications toujours plus sophistiquées, capables de répondre à de nouveaux besoins utilisateurs et aux exigences qui leurs sont associées en termes de qualité de service, de disponibilité, de fiabilité, de performance et de productivité.

Depuis quelques années maintenant l'université d'Abomey-calavi dispose d'une infrastructure informatique et met à disposition de divers services électroniques à ses enseignants et étudiants, nous pouvons citer entre autres:

- Le Webmail qui est une application web gérant l'envoi et la réception des e-mails.
- OKAPI, c'est l'application qui gère la scolarité des étudiants de l'université
- BEC est la bibliographie des enseignants et chercheurs
- WebTV la plateforme web de diffusion de programmes télévisuels
- E-learning est une plateforme d'apprentissage en ligne etc...

Ces applications sont les plus utilisées par le grand public; étant la première et plus grande université du pays; l'UAC dispose de plusieurs écoles et instituts universitaires dont le nombre d'étudiants augmente chaque année, face à cette croissance massive le personnel du [SP-TIC](#) (Service de Promotion des Technologies de l'Information et de la Communication) réfléchit dans le but d'améliorer les différents services électroniques proposés par l'université, c'est dans cette optique qu'il réfléchit à l'optimisation de l'architecture de déploiement de ses applications web.

Existe-t-il une nouvelle technologie assez performante et optimale qui puisse faciliter le déploiement continu des applications? Comment peut-on s'y prendre pour l'implémenter? Notre objectif principal sera donc de mettre en place un système pouvant aider les développeurs à déployer les plateformes tout en ajoutant d'autres fonctionnalités de façon progressive sans perturber la disponibilité de la plateforme. En somme fait une étude des technologies pouvant maintenir la disponibilité des plateformes tout assurant un déploiement progressif aux développeurs.

Il s'agira en particulier de :

- Chercher à faciliter le déploiement continu des applications;
- Améliorer la scalabilité et la disponibilité des applications;
- Faciliter les mises à jour et les ajouts de fonctionnalités;
- Faire des tests et des simulations pour tester la fiabilité de l'architecture proposée;

Le présent travail est subdivisé en 04 chapitres. Le premier chapitre présente une introduction à notre étude; le second nous permettra de présenter les types de déploiement, leurs différentes architectures et la présentation de l'existant. Le troisième chapitre présente le choix de notre solution à travers son mécanisme de fonctionnement et son implémentation. Les résultats obtenus et les insuffisances liées à notre solution feront l'objet du chapitre 4.

Revue de littérature

2.1 Généralité sur le déploiement des applications

Le déploiement se définit comme la promotion de composants d'un environnement vers le suivant, dans la pratique il s'agit d'introduire une nouvelle solution technique et des services accessibles depuis un navigateur web de façon coordonnée. Un déploiement réussi repose sur une planification prospective, des ressources adéquates, une surveillance et une évaluation continue et une communication solide.

2.1.1 Le cycle de vie du développement logiciel

Le [SDLC](#) (Software Development Life Cycle) désigne toutes les étapes du développement d'un logiciel, de sa conception à sa mise en production sur le marché, il permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés. Pour mener à bien une SDLC un enchaînement des environnements s'impose et il permet d'améliorer et de vérifier la qualité d'une application avant qu'elle ne devienne accessible par les utilisateurs. Il n'y a pas un nombre idéal d'environnements, mais nous pouvons citer l'environnement de *development*, *test d'intégration système*, *test d'acceptation des utilisateurs* et celle de la *production*.

Le diagramme ci-dessous nous montre la chronologie de ces quatre environnements.

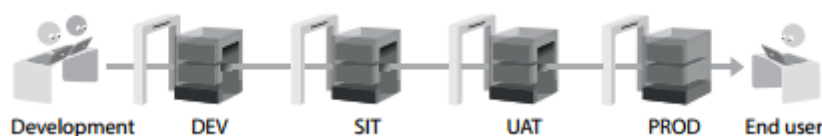


FIGURE 2.1 – Les quatre environnements élémentaires du SDLC

- **Développement (DEV)** dans l'environnement de Développement, les développeurs créent et déploient du code dans un laboratoire où l'application est testée au niveau le

plus élémentaire. Lorsque l'application satisfait certains critères de qualité, elle passe à l'environnement suivant.

- **Test d'intégration système (SIT)** dans l'environnement Test d'intégration système, l'application est testée pour garantir qu'elle fonctionne avec les applications et systèmes existants. Lorsque l'application réussit les tests d'intégration, elle est déployée dans l'environnement suivant.
- **Test d'acceptation des utilisateurs (UAT)** dans l'environnement test d'acceptation des utilisateurs, l'application est testée pour s'assurer qu'elle fournit les fonctionnalités requises pour les utilisateurs finaux. Cet environnement est généralement proche de celui de la production ; Lorsque l'application répond à ces exigences, elle est promue vers l'environnement final.
- **Production (PROD)** dans l'environnement Production, l'application est mise à la disposition des utilisateurs. L'analyse du comportement utilisateur de l'application est obtenue en surveillant la disponibilité et la fonctionnalité de l'application. Toute les mises à jour ou correctifs sont introduites dans l'environnement DEV et suivent le même cycle.

Un déploiement se définit comme la promotion de composants d'une application depuis un environnement vers le suivant, l'on rencontre souvent des problèmes logiciels ou de dépendances lors du passage d'un environnement à un autre, les développeurs passent peine un peu dans la résolution de ces problèmes et bien cela freine parfois la mise en production de l'application.

2.2 La virtualisation par conteneurs

2.2.1 Définition

La virtualisation par conteneurs (également connue sous le nom de conteneurisation) constitue une alternative à la virtualisation système. Il s'agit d'une approche qui s'appuie directement sur les fonctionnalités du noyau (Kernel) pour créer des environnements virtuels (Virtual Environment) isolés les uns des autres. Ces VE sont appelés conteneurs, tandis que les fonctionnalités fournies par le noyau du système d'exploitation sont les groupes de contrôles (cgroups) et les espaces de noms (namespaces). Les namespaces permettent de contrôler et de limiter la quantité de ressources utilisée pour un processus, tandis que les cgroups gèrent les ressources d'un groupe de processus. Un conteneur fournit donc les ressources nécessaires pour exécuter des applications comme si ces dernières étaient les seuls processus en cours d'exécution dans le système d'exploitation de la machine hôte.

2.2.2 Vue d'ensemble des conteneurs

Les conteneurs constituent une solution pour la gestion des ressources entre les applications. Le terme est dérivé des conteneurs d'expédition, une méthode standard pour stocker et expédier tout type de cargaison. Le concept de base des conteneurs est né en 1979 avec la fonction Unix `chroot`. Cette dernière permet de créer plusieurs sessions isolées sur une même machine, de façon à partager parallèlement les ressources de processus et de stockage de la machine. Vu cette caractéristique particulière de la fonction `chroot`, celle-ci peut être considérée comme étant le précurseur de la conteneurisation. Introduite en l'année 2000, la solution FreeBSD Jail a permis de partitionner le système d'exploitation de la machine hôte en des sous-systèmes isolés. Par rapport à `chroot`, la fonctionnalité supplémentaire de l'environnement FreeBSD était la possibilité d'isoler également le système de fichiers, les utilisateurs et le réseau, en les rendant plus sécurisés. Mais cette solution présente des difficultés en termes de mise en oeuvre. L'année suivante a vu le développement d'une solution semblable à FreeBSD Jail appelée Linux VServer. Dans ce nouveau projet, chaque distribution Linux importe ses propres bibliothèques qui supportent les conteneurs [LXC](#), mais qui ne sont pas compatibles entre elles. Cependant, c'est à partir de 2013 que Docker a rendu simple le développement et le déploiement des conteneurs tout en les standardisant. [Docker](#) résout un problème de longue date dans le cloud computing : la portabilité des applications.

2.2.3 Achitecture

La véritable valeur des conteneurs réside dans leur portabilité. En rassemblant tous les fichiers nécessaires pour exécuter une application, une fonctionnalité ou une composante dans une seule image distincte, les conteneurs Linux fournissent de la cohérence et de la prédictibilité lorsqu'ils passent du développement aux tests et enfin en production, ils ont généralement une architecture qui se définit comme suit :

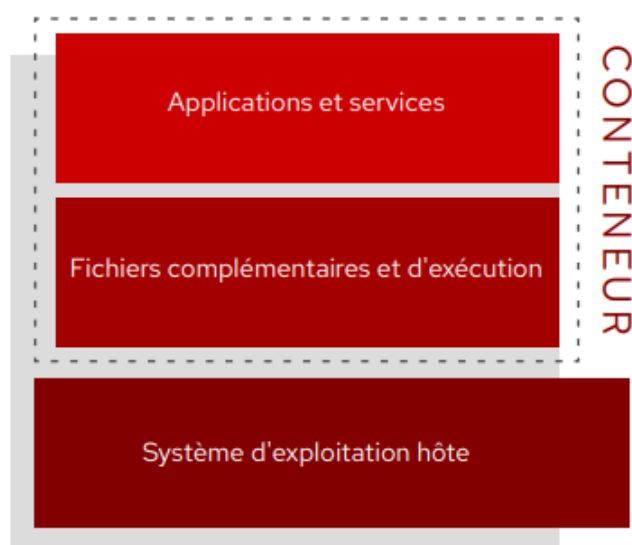


FIGURE 2.2 – Achitecture de virtualisation par conteneur

- *Système d'exploitation hôte* représente l'[OS](#) installé sur le serveur ;

- *Fichiers complémentaires et d'exécution*, ceux-ci représentent l'ensemble des fichiers qui permettent de faire fonctionner les applications, ils sont installés à l'aide d'un **container Engine**¹ ;
- *Les applications et services*, ensemble des applications et services prêt à être démarrés.

2.3 La virtualisation

2.3.1 Définition

La virtualisation consiste à créer une version virtuelle d'un dispositif ou d'une ressource, comme un système d'exploitation, un serveur, un dispositif de stockage ou une ressource réseau. Elle peut donc être considérée comme l'abstraction physique des ressources informatiques. En d'autres termes, les ressources physiques allouées à une machine virtuelle sont abstraites à partir de leurs équivalentes physiques. Chaque dispositif virtuel, qu'il s'agisse d'un disque, d'une interface réseau, d'un réseau local, d'un commutateur, d'un processeur ou d'une mémoire, correspond à une ressource physique sur un système informatique. Les machines virtuelles hébergées par la machine hôte sont donc perçues par ce dernier comme des applications auxquelles il est nécessaire de dédier ou distribuer ses ressources.

2.3.2 Achitecture

La virtualisation vous permet d'exécuter plusieurs ordinateurs séparés sur un seul équipement. Les systèmes d'exploitation et leurs applications partagent des ressources matérielles à partir d'un unique serveur hôte. Tout ceci est possible grâce à l'architecture suivant :

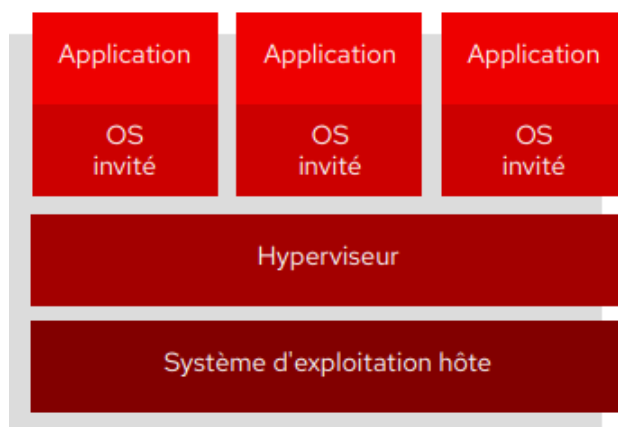


FIGURE 2.3 – Architecture de la virtualisation

- *Système d'exploitation hôte* représente l'OS installé sur le serveur de l'infrastructure ;

¹Container Engine : Container Engine (GKE) est un système de gestion et d'orchestration de conteneurs

- *Hyperviseur* est la couche logicielle qui s'insère entre le matériel et les différents systèmes d'exploitation. Il peut soit gérer lui-même toutes les ressources matérielles du serveur, soit s'appuyer pour cela sur un système d'exploitation existant ;
- *Les applications et OS invité* ; l'ensemble représente une machine virtuelle installée avec son système d'exploitation.

2.4 Comparaison entre machine virtuelle et conteneur

La virtualisation traditionnelle utilise un hyperviseur pour créer de nouvelles machines virtuelles et pour assurer l'isolation entre elles. Toutefois, la conteneurisation ne nécessite l'installation d'un logiciel de gestion de conteneur sur le noyau du système d'exploitation de la machine hôte. Néanmoins, il existe une certaine similitude entre ces deux instances virtuelles : elles sont toutes des systèmes autonomes qui, en réalité, utilisent un système supérieur (celui de la machine hôte) pour réaliser leurs tâches. Cependant, la principale différence entre elles est que la VM doit contenir tout un système d'exploitation (un système invité) alors que les conteneurs se contentent d'utiliser le système d'exploitation sur lequel ils s'exécutent.

- Les machines virtuelles sont idéales pour prendre en charge les applications qui nécessitent toutes les fonctionnalités d'un système d'exploitation lorsque vous souhaitez déployer plusieurs applications sur un serveur ou lorsque vous avez une grande variété de systèmes d'exploitation à gérer. Les conteneurs sont un meilleur choix lorsque votre plus grande priorité est de minimiser le nombre de serveurs que vous utilisez pour plusieurs applications.
- Les conteneurs sont un excellent choix pour raccourcir le cycle de vie des logicielles grâce à leur temps de configuration rapide, il est idéal pour une architecture de micro services ; les micro services divisent une grande application logiciel en de nombreuses fonctions différentes qui peuvent être déployées indépendamment, ce qui accélère le développement et le déploiement des éléments qui favorise la réduction du « Time to market ». ² Les machines virtuelles ont un cycle de vie plus long que les conteneurs et sont mieux utilisées pendant de plus longues périodes.

En outre, ci-dessous, voici une brève comparaison de l'utilisation des ressources de l'infrastructure informatique par ses deux technologies.

² Time to market : c'est le délai nécessaire pour le développement et la mise au point d'un projet ou d'un produit, avant qu'il puisse être lancé sur le marché.

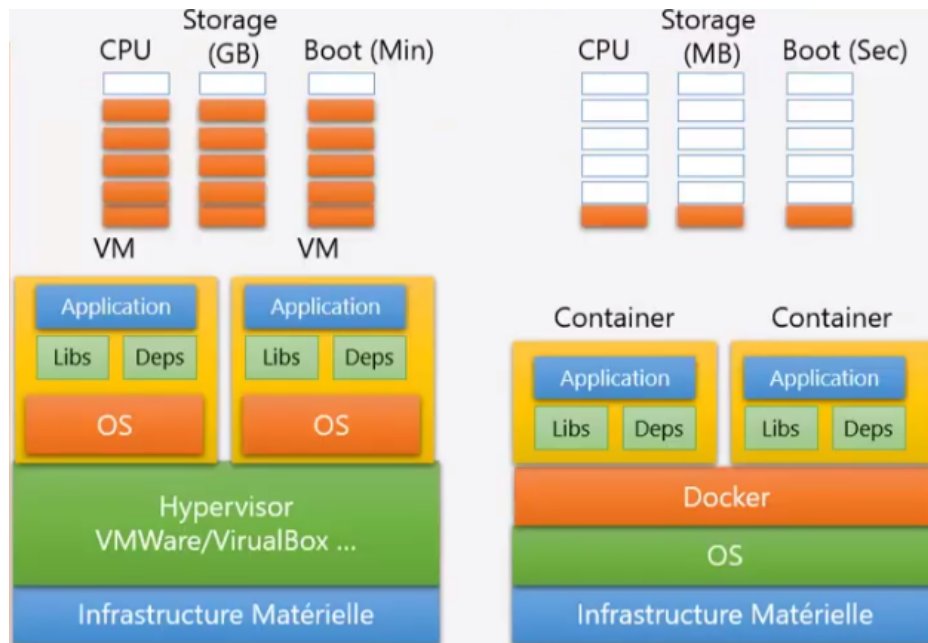


FIGURE 2.4 – Comparaison de l'utilisation des ressources des deux technologies

1. Nous avons d'une part un environnement qui permet de virtualiser une machine physique et d'autre part un environnement qui permet d'exécuter des applications.
2. Les solutions d'hyperviseur les plus connus sous windows sont VMWare et VirtualBox et sous linux on peut citer KVM et XEN ; la solution de conteneurisation la plus utilisée est Docker.
3. Chaque machine virtuelle a son propre système d'exploitation du côté de la virtualisation mais du côté de la conteneurisation tous les conteneurs utilisent le même OS.
4. Une VM consomme beaucoup plus de ressources (CPU, stockage) et prend assez de temps pour booter (quelques minutes), tous les conteneurs utilisent le même kernel OS (linux), consomment peu de ressources et ont un démarrage beaucoup plus rapide (quelques secondes).

2.5 Présentation de l'existant

L'[UAC](#) a été fondée en 1970, elle est de loin la première université du pays et la plus connue dans la sous-région comme à l'étranger. Elle dispose depuis 2007 d'une salle serveur équipée de plusieurs équipements informatiques, notamment les serveurs qui lui permettent d'héberger et de mettre à disposition de ses enseignants et étudiants des plates-formes pour faciliter les procédures d'inscription et autres de l'université. L'UAC dispose d'une infrastructure virtuelle et de quelques conteneurs, mais ceux-ci ne sont pas encore utilisés pour le déploiement des applications. Notre travail s'inscrit dans la logique d'explorer les technologies de déploiement par conteneur, trouver un gestionnaire de conteneur crédible et proposer des solutions de déploiements adaptées aux réalités de l'université.

Matériels et méthodes

3.1 Méthodes

Le déploiement d’une solution de conteneurisation doit passer par une étude détaillée des solutions architecturales, matérielles et sécuritaires existantes pour être en accord avec les réalités africaine; la solution la plus en vogue sur le marché depuis quelques temps est docker qui est de loin la meilleure solution de conteneurisation de notre ère.

3.1.1 Présentation de la technologie Docker

Il s’agit d’un logiciel libre qui permet d’automatiser le déploiement d’applications dans des conteneurs. La firme de recherche sur l’industrie 451 Research a défini le [Docker](#) comme un outil qui permet d’empaqueter une application et ses dépendances dans un conteneur isolé, Ce logiciel pourra aussi être exécuté sur n’importe quel serveur Linux. Et par la suite, la portabilité d’exécution et la flexibilité d’une application s’étendent, que ce soit sur la machine locale, , une machine nue, un Cloud public ou privé etc...

[Docker](#) est utilisé pour créer et gérer des conteneurs, ce qui permet la simplification de la mise en œuvre de systèmes distribués en permettant l’exécution de multiples applications, tâches de fond et autres processus. l’exécution sera de façon autonome sur une seule machine physique ou à travers un éventail de machines isolées. Ceci permet le déploiement des nœuds en tant que ressources sur besoin.

Docker est une évolution basée sur les technologies propriétaires de dot Cloud, depuis Mars 2013 Docker a été distribué en tant que projet open source.

- Au 18 novembre 2013, le projet a été favorisé plus de 7 300 fois sur GitHub (Il présente le 14e projet le plus populaire), avec plus de 900 forks et 200 contributeurs ;
- Au 9 mai 2014, le projet a été favorisé plus de 11 000 fois sur GitHub, avec plus de 1 900 forks et 420 contributeurs ;
- En octobre 2015, le projet a été favorisé plus de 25 000 fois sur GitHub, avec plus de 6 500 forks et 1 100 contributeurs ;

- En septembre 2016, le projet a été favorisé plus de 34 000 fois sur GitHub, avec plus de 10 000 forks et 1 400 contributeurs ;

Docker étend le format de conteneur Linux standard [LXC](#) avec une API de haut niveau proposant une solution de virtualisation qui effectue les processus de façon isolées. [Docker](#) fait usage de [LXC](#), Cgroups, et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur [Docker](#) ne contient pas de système d'exploitation, se basant sur les fonctionnalités du système d'exploitation fournies par l'infrastructure sous-jacente.

3.1.2 Architecture de Docker

La figure suivante montre l'architecture de fonctionnement de [Docker](#).

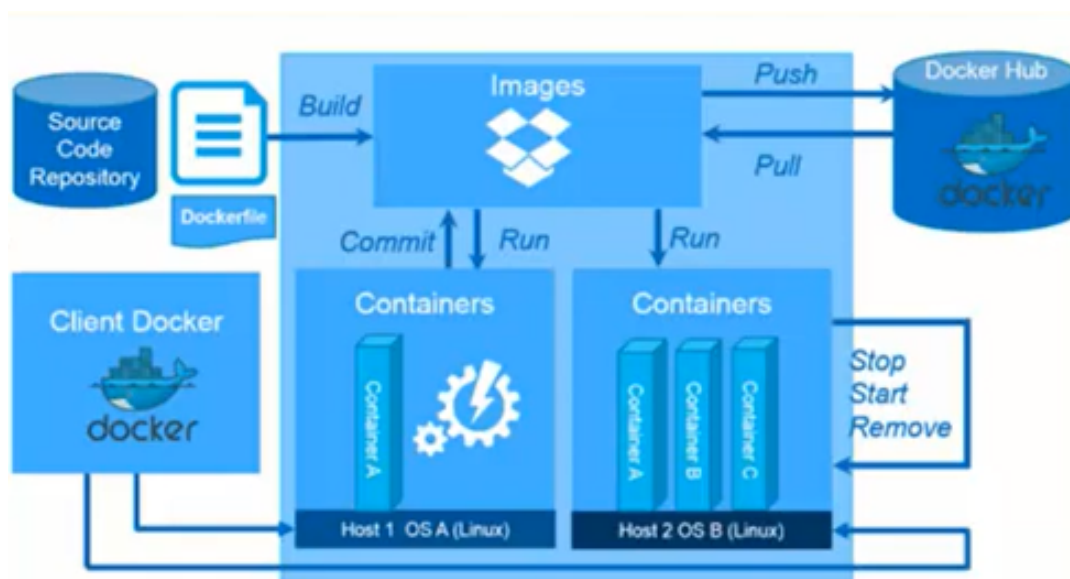


FIGURE 3.1 – Architecture de Docker

Cette architecture se décrit comme suit :

1. Première partie

- Pendant l'écriture du code source du projet, le développeur crée un fichier [Dockerfile](#) dans lequel il écrit les dépendances de son application.
- L'administrateur [DevOps](#) chargé du déploiement crée une image de l'application avec la commande *Build*.
- L'[Image](#) ainsi créée est publiée avec la commande *push* dans le registre docker appelé [Docker Hub](#), la commande *Pull* permet de télécharger et utiliser l'application dans n'importe quel machine.

2. Deuxième partie

- *Run* permet d'exécuter une application dans un conteneur ; pour chaque *Run* exécuter une instance de l'application est créée et démarrée (étant donné que toutes les dépendances de l'application ont été déjà installées l'application fonctionne)

- *Le Client Docker* permet de communiquer avec le [Docker Engine](#)
- Chaque conteneur sont dans des hosts respectifs et On peut effectuer des actions de *Démarrage, Arrêt et Supression*

3.1.3 Les principaux services de Docker

3.1.3.1 Docker Compose

Docker Compose est un outil développé par Docker pour créer les architectures logicielles containerisées. Dans cette logique, chaque brique de l'application (code, base de données, serveur web...) sera hébergée par un container. Cet outil repose sur le langage [YAML](#) (Yet Another Markup Language) pour décrire l'architecture. Une fois celle-ci codée dans un fichier [YAML](#), l'ensemble des services applicatifs seront générés via une commande unique.

3.1.3.2 Docker Hub

[Docker Hub](#) est un service en ligne, conçu pour faciliter l'échange d'applications containerisées. Hébergeant plus de 100 000 images de containers, cet espace est aussi intégré à GitHub. Il recouvre de très nombreux domaines (analytics, bases de données, frameworks, monitoring, outils de [DevOps](#), sécurité, stockage, systèmes d'exploitation ...etc). Qualifiées d'officielles, certaines images sont directement maintenues par [Docker](#). D'autres sont proposées par des contributeurs. Testées et éprouvées par Docker, une cinquantaine sont certifiées par ce dernier. La société de San Francisco commercialise par ailleurs une version du [Docker Hub](#) installable localement (le Docker Hub Enterprise). Enfin [Docker](#) a lancé une boutique en ligne d'applications pour offrir aux éditeurs un canal commercial pour distribuer leurs applications sous forme de containers.

3.1.3.3 Docker Swarm

[Docker Swarm](#) est une plateforme [CaaS](#) (Containers as a Service), elle a été construite Pour faciliter le management des architectures complexes. Baptisée Docker Enterprise, elle a été cédée à Mirantis en novembre 2019. Elle comprend les principaux outils nécessaires pour gérer le déploiement, le pilotage, la sécurité et le monitoring de tels environnements. Côté gestion de cluster, Docker Entreprise intègre à la fois Swarm, son moteur d'orchestration maison et offre une flexibilité dans le choix du moteur d'orchestration. La société de San Francisco entent proposer une plateforme capable de gérer de manière fédérée des applications qu'elles soient basées sur les grands services cloud des moteurs meilleurs d'orchestration de conteneurs.

L'interface de ligne de commande du moteur [Docker](#) est utilisée pour créer un essaim de moteurs Docker où les services d'application peuvent être déployés.

3.1.4 Les principaux solutions d'orchestrations de cluster de conteneurs

Plusieurs solutions d'orchestration des conteneurs existent sur le marché, des plus utilisées au plus moindres, certaines sont open source et d'autres propriétaires. Nous pouvons citer entre autres :

3.1.4.1 Kubernetes

Kubernetes est une plate-forme open source qui a été initialement conçue par Google et maintenant maintenue par la Cloud Native Computing Foundation. Kubernetes prend en charge à la fois la configuration déclarative et l'automatisation. Il peut aider à automatiser le déploiement, la mise à l'échelle et la gestion de la charge de travail et des services conteneurisés. Cette architecture nous résume le mode de fonctionnement de kubernetes.

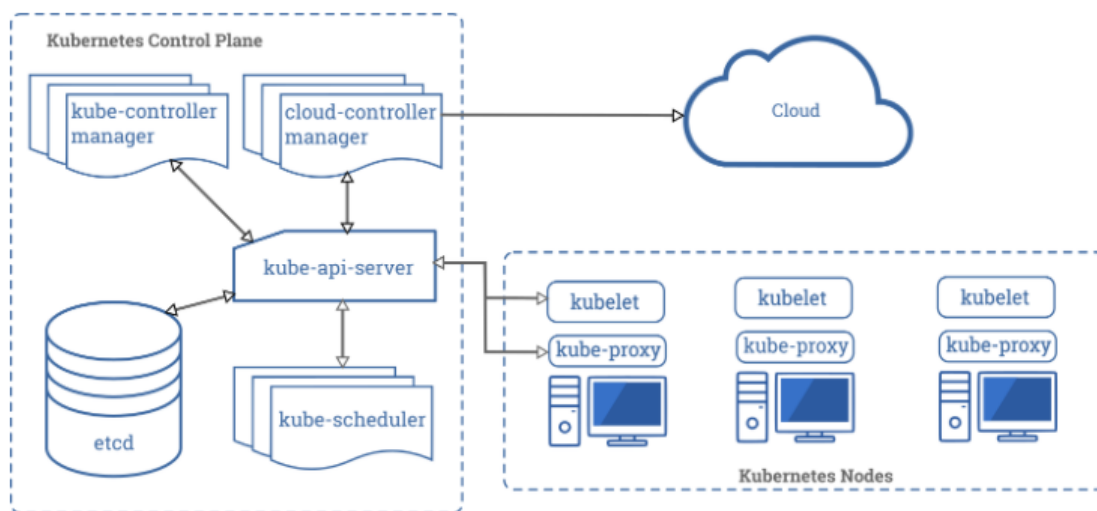


FIGURE 3.2 – Architecture technique de Kubernetes

L'[API](#) Kubernetes aide à établir la communication entre les utilisateurs, les composants du cluster et les composants tiers externes. Le plan de contrôle Kubernetes et les nœuds s'exécutent sur un groupe de nœuds qui forment ensemble le cluster. La charge de travail de l'application se compose d'un ou de plusieurs pods qui s'exécutent sur des nœuds de travail. Le plan de contrôle gère les pods et les nœuds de travail.

Les avantages liés à l'utilisation de cette solution d'orchestration sont :

- Découverte de services, équilibrage de charge et Orchestration du stockage
- Déploiements et annulations automatisés
- Mise à l'échelle horizontale et gestion des secrets et de la configuration
- L'auto-guérison, exécution par lots et double pile [IPv4 / IPv6](#)

En résumé, voici les forces et les faiblesses de kubernetes résumés dans ce tableau ci-dessous.

Points forts	Points faibles
Orchestrateur mature et riche fonctionnellement Intégration aux principales plateformes cloud	Complexité dans l'appréhension Approche solutions clés en main repoussant

TABLE 3.1 – Forces et faiblesses Kubernetes

3.1.4.2 OpenShift

Redhat OpenShift [PaaS](#) aide à l'automatisé des applications sur des ressources sécurisées et évolutives dans des environnements de cloud hybride. Il fournit des plates-formes de niveau entreprise pour la création, le déploiement et la gestion d'applications conteneurisées.

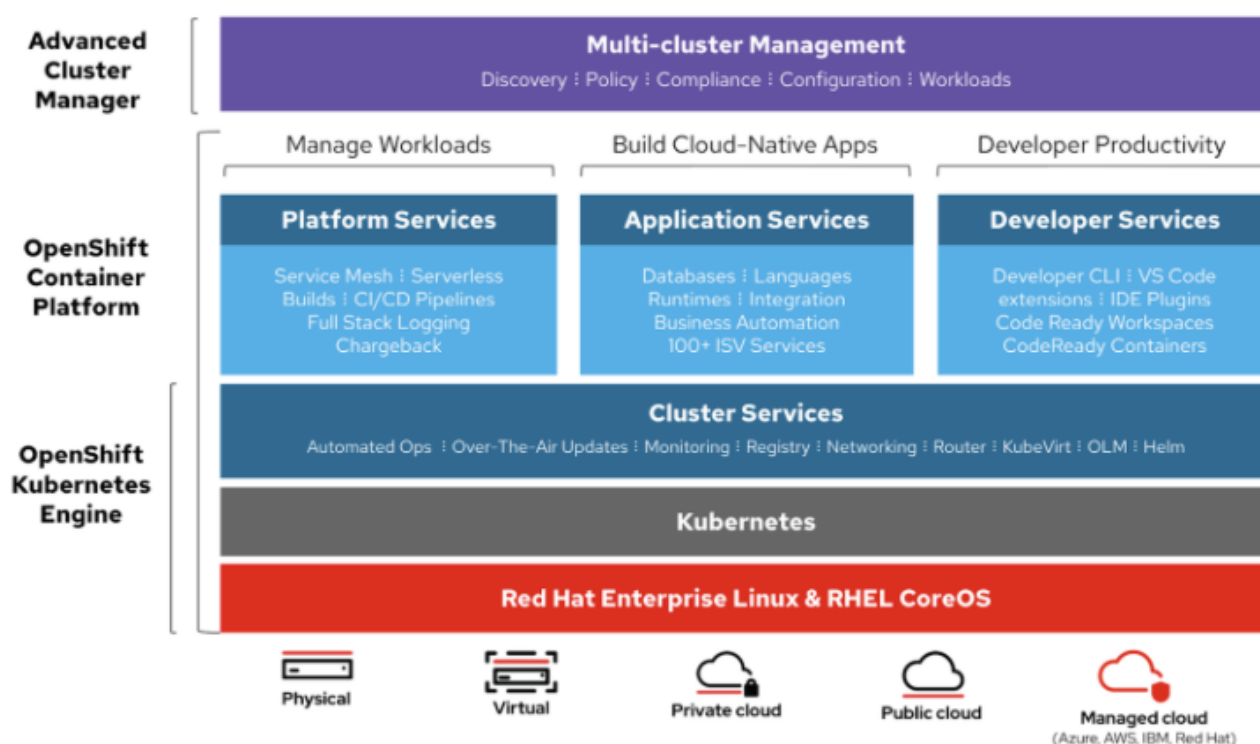


FIGURE 3.3 – Architecture technique Redhat OpenShift

Il est basé sur le moteur Redhat Enterprise Linux et Kubernetes. OpenShift dispose de diverses fonctionnalités pour gérer les clusters via l'interface utilisateur et la CLI.

Redhat propose OpenShift dans deux autres variantes,

- **OpenShift en ligne** : offert en tant que logiciel en tant que service ([SaaS](#))
- **OpenShift dédié** : offerts en tant que services gérés OpenShift Origin (Origin Community Distribution) est un projet communautaire open source en amont qui est utilisé dans OpenShift Container Platform, OpenShift Online et OpenShift Dedicated.

En résumé, voici les forces et les faiblesses de openshift résumés dans ce tableau ci-dessous.

Points forts	Points faibles
Flexible et dynamique Basée sur les technologies open source	Plus de compétence pour la mise en place Difficulté de migration vers d'autres orchestreurs

TABLE 3.2 – Forces et faiblesses OpenShift

3.1.4.3 Nomade

Nomade est un orchestrateur de charges de travail simple, flexible et facile à utiliser pour déployer et gérer des conteneurs et des applications non conteneurisées sur site et dans les clouds à grande échelle. Nomade fonctionne comme un binaire unique avec une faible empreinte de ressources (35 Mo) et pris en charge sur macOS, Windows, Linux.

Nomade se caractérise généralement par ces différents critères :

- Simple et fiable
- Modernisation des applications héritées sans réécriture
- Fédération facile à grande échelle
- Evolutivité éprouvée
- Multi-Cloud avec facilité
- Intégrations natives avec Terraform, Consul et Vault.

Nomade orchestrate des applications de tout type (pas seulement des conteneurs). Il fournit une prise en charge de première classe pour [Docker](#), Windows, Java et les machines virtuelles.

3.1.4.4 Fleet

Fleet est l'orchestrateur livré en standard avec la distribution Linux CoreOS, un OS allégé taillé pour le cloud ; il repose sur Etcd (un système de clé-valeur distribué qui une fois installé sur les nœuds d'un cluster va lui soumettre des "jobs" et les répartit sur les nœuds. Sorti depuis 2013, il a une approche multi technologies semblable à Mesos Apache. Et comme lui, gérer des conteneurs n'était pas sa vocation originelle.

Sur un cluster de serveurs, Fleet va exécuter des commandes système et donc appeler n'importe quel processus Linux ou, notamment le programme [Docker](#).

En quelques termes, nous avons résumé les forces et les faiblesses de Fleet dans le tableau suivant :

Points forts	Points faibles
Antériorité	Périmètre fonctionnel limité
Approche multi technologies	Communauté de contributeurs en déclin

TABLE 3.3 – Forces et faiblesses Fleet

Fleet fait de l'orchestration bas niveau et est très orienté administrateur système , il ne soutient pas la comparaison en termes de capacité et de scalabilité comparé aux autres orchestrateurs.

3.1.4.5 Mesos Marathon

Mesos fait partie de la sphère des projets open source sous licence Apache. Ce méta-framework est conçu pour faire tourner des infrastructures de gestion de cluster. Mesos gère plusieurs machines physiques ou virtuelles et répartit la charge en fonction des ressources mutualisées (CPU, RAM, disques...), Mesos est fréquemment utilisé pour les projets de Big Data. Avec le succès de Docker, Mesos a décidé de se tourner vers l'orchestration de containers. il utilise pour cela Marathon, une interface proposée par la société Mesosphere pour la gestion de cluster, il a la capacité d'exécuter des charges de travail conteneurisées et non conteneurisées.

En quelques termes, nous avons résumé les forces et les faiblesses de Mesos Marathon dans le tableau suivant :

Points forts	Points faibles
L'orchestrateur le plus ancien	En perte de vitesse comparé aux autres technologies
Soutien de la fondation Apache	Travail d'intégration

TABLE 3.4 – Forces et faiblesses Mesos Marathon

Mesos offre une technologie éprouvée en production depuis plusieurs années notamment pour les clouds privés". Avec la particularité d'avoir une approche multi-technologies. Marathon peut aussi bien démarrer des containers que lancer des commandes systèmes (commande Linux, application Java, serveur application etc...).

Même s'il n'est pas un orchestreur de base il présente néanmoins des fonctionnalités intéressantes, nous pouvons citer entre autres :

- Haute Disponibilité
- Applications avec état
- Belle et puissante interface utilisateur
- Découverte de services et équilibrage de charge
- Journalisation
- API REST

3.1.4.6 Cloudify

Cloudify est un outil d'orchestration cloud open source pour l'automatisation du déploiement et la gestion du cycle de vie des conteneurs et des microservices. Il fournit des fonctionnalités telles que les clusters à la demande, la réparation automatique et la mise à l'échelle au niveau de l'infrastructure. Il peut aussi gérer l'infrastructure de conteneurs et orchestrer les services qui s'exécutent sur des plates-formes de conteneurs.

Il peut être facilement intégré aux gestionnaires de conteneurs basés sur [Docker](#) et à d'autres orchestreurs de conteneurs :

- Docker
- Docker Swarm
- Docker Compose
- Kubernetes
- Apache Mesos
- etc...

Cloudify aide à créer, réparer, mettre à l'échelle et supprimer des clusters de conteneurs. L'orchestration de conteneurs est essentielle pour fournir une infrastructure évolutive et hautement disponible sur laquelle les gestionnaires de conteneurs peuvent s'exécuter. Cloudify offre la possibilité d'orchestrer des services hétérogènes sur plusieurs plates-formes.

3.1.4.7 Docker Swarm

Depuis sa version 1.12, sortie en juin 2016, [Docker](#) intègre nativement son outil d'orchestration Swarm. Ce dernier reprend la même notion de maître et esclave présente dans la plupart des solutions d'orchestrations, appelée ici manager et worker, pour gérer le cycle de vie des containers dans un cluster.

[Docker Swarm](#) utilise un modèle déclaratif. Vous pouvez définir l'état souhaité du service et Docker conservera cet état. Docker Enterprise Edition a intégré Kubernetes à Swarm, [Docker](#) offre désormais une flexibilité dans le choix du moteur d'orchestration. L'interface de ligne de commande du moteur [Docker](#) est utilisée pour créer un essaim de moteurs [Docker](#) où les services d'application peuvent être déployés.

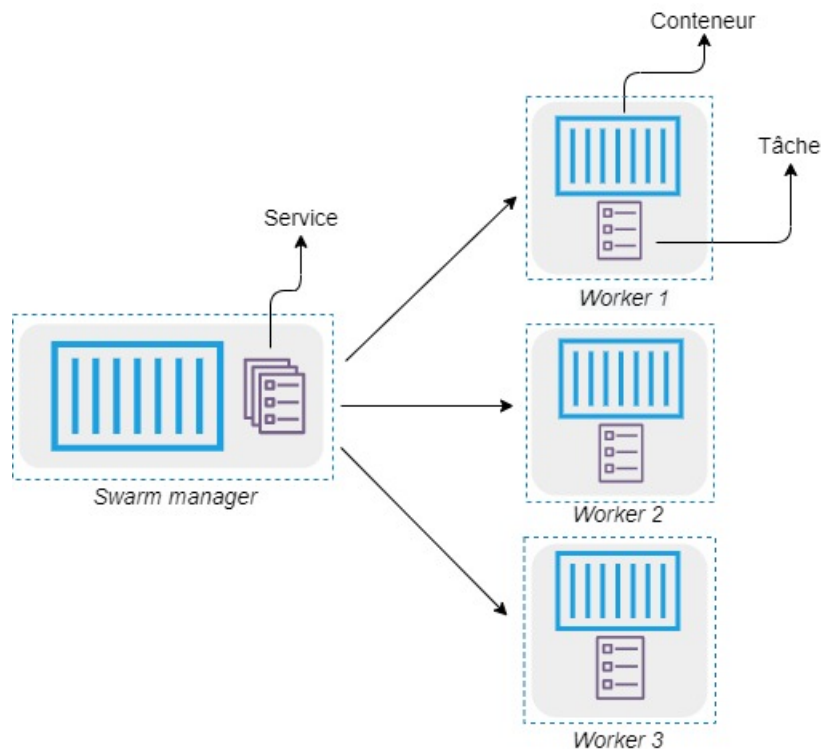


FIGURE 3.4 – Architecture Docker-Swarm Mode

Les commandes [Docker](#) sont utilisés pour interagir avec le cluster. Les machines qui rejoignent le cluster pour former le nœud sont appelées *Worker* et le gestionnaire Swarm gère les activités du cluster.

[Docker Swarm](#) se compose de deux éléments principaux :

- **Manager** - les nœuds de gestionnaire attribuent des tâches aux nœuds de travail dans l'essaim. Un leader est élu sur la base d'un algorithme de consensus Raft. Le chef gère toutes les décisions de gestion de l'essaim et d'orchestration des tâches pour l'essaim.
- **Nœud de travail** - Le nœud de travail reçoit les tâches du nœud de gestionnaire et les exécute.

Les avantages liés à l'utilisation de cette solution d'orchestration sont :

- Gestion des clusters intégrée à Docker Engine
- Conception décentralisée
- Modèle de service déclaratif
- écaillage
- Réconciliation d'état souhaitée
- Réseau multi-hôte

- Découverte de service
- L'équilibrage de charge
- Sécurisé par défaut
- Mises à jour progressives

3.1.5 Choix de la solution adaptée pour l'UAC

Nous avons entrepris de faire une comparaison de ses différents orchestrateurs basée sur certains exigences spécifiques.

De toutes les recherches effectuées pendant notre étude, nous nous sommes concentrés sur les technologies les plus utilisées pour faire le choix de celle que nous utiliseront pour la réalisation de notre projet.

Quelques exemples des critères de sélections :

- Faciliter de déploiement et compatibilité avec Docker
- Minimiser les risques liés à l'aspect sécurité
- Commandes simples et faciles à utiliser
- Moins de ressources déployées pour la mise en place etc...

Critères	Kubernetes	Apache MESOS	OpenShift	Docker-Swarm
Faciliter de déploiement de l'outil avec le moteur Docker	Oui	Oui	Oui	Oui
Moins de risques liés à l'aspect sécurité	Oui	Oui	Oui	Oui
Commandes simples et faciles à utiliser	Non	Non	Non	Oui
Moins de ressources déployée pour la mise en place	Oui	Oui	Oui	Oui
Version commerciale	Oui	Oui	Oui	Oui
Communauté	Oui	Non	Oui	Oui
Intégré à Docker Engine	Non	Non	Non	Oui

TABLE 3.5 – Comparaison des solutions open-source d'orchestration de conteneur les plus utilisées

Après comparaison des solutions d'orchestration, nous constatons que la solution [Docker Swarm](#) satisfait à toutes nos exigences. C'est donc elle que nous implémenterons en tant que solution d'orchestration de conteneur.

3.2 Expériences

Pour les besoins des expériences, nous avons eu à utiliser plusieurs outils pour mettre en place le système de gestion de conteneurs basé sur la technologie [Docker Swarm](#), voici une brève description de ces différents outils.

3.2.1 Oracle VM VirtualBox

VirtualBox est un puissant produit de virtualisation x86 et AMD64 / Intel64 ; il est un produit hautement performant et extrêmement riche en fonctionnalités. C'est aussi la seule solution professionnelle disponible gratuitement en tant que logiciel Open Source sous les termes de la GNU General Public License (GPL). VirtualBox fonctionne sur les hôtes Windows, Linux, Macintosh et Solaris.

3.2.2 Ubuntu 20.04

Ubuntu est un système d'exploitation GNU/Linux basé sur la distribution Linux Debian. Il est développé, commercialisé et maintenu pour les ordinateurs individuels, les serveurs et les objets connectés par la société Canonical.

C'est la distribution la plus consultée sur Internet, et le système d'exploitation le plus utilisé sur les systèmes Cloud ainsi que sur les serveurs informatiques. Nous l'avons utilisé comme système d'exploitation de base pour notre moteur docker étant donné qu'il est basé et fonctionne sur le système Linux.

3.2.3 Docker Engine

[Docker Engine](#) est le moteur de gestion de conteneur de la technologie docker, il est open source et permet de créer et de conteneuriser vos applications. [Docker Engine](#) agit comme une application client-serveur avec :

- Un serveur avec un processus démon de longue durée *dockerd*
- API qui spécifient les interfaces que les programmes peuvent utiliser pour parler et instruire le démon Docker.
- Un client d'interface de ligne de commande (CLI) *docker*

La [CLI](#) utilise les *API Docker* pour contrôler ou interagir avec le démon [Docker](#) via des scripts ou des commandes [CLI](#) directes. De nombreuses autres applications [Docker](#) utilisent l'[API](#) et la [CLI](#) sous-jacentes. Le démon crée et gère des objets Docker, tels que des images, des conteneurs, des réseaux et des volumes.

3.2.4 Vagrant

Vagrant est un logiciel libre et open-source pour la création et la configuration des environnements de développement virtuel. Il peut être considéré comme un wrapper autour de logiciels de virtualisation comme VirtualBox. Depuis la version 1.1, Vagrant n'impose plus VirtualBox, mais fonctionne également avec d'autres logiciels de virtualisation tels que VMware; la version 1.6 fournit un support natif des conteneurs Docker à l'exécution, au lieu d'un système d'exploitation entièrement virtualisé. Cela permet de réduire la charge en ressources puisque Docker utilise des conteneurs Linux légers.

3.2.5 Ansible

Ansible est une plate-forme logicielle libre pour la configuration et la gestion des ordinateurs. Elle combine le déploiement de logiciels multi-nœuds, l'exécution des tâches ad-hoc, et la gestion de configuration. Elle gère les différents nœuds à travers SSH et ne nécessite l'installation d'aucun logiciel supplémentaire sur ceux-ci. Les modules communiquent via la sortie standard en notation JSON et peuvent être écrits dans n'importe quel langage de programmation. Le système utilise YAML pour exprimer des descriptions réutilisables de systèmes appelées playbook.

3.2.6 Git BASH (facultatif)

Git est un ensemble d'utilitaires de ligne de commande conçus pour s'exécuter dans un environnement de ligne de commande de type Unix. Les systèmes d'exploitation modernes comme Linux et macOS comprennent tous deux des terminaux de ligne de commande Unix intégrés. Cela fait de Linux et de macOS des systèmes d'exploitation complémentaires lorsqu'on travaille avec Git. Microsoft Windows utilise plutôt l'invite de commande Windows, un environnement de terminal non Unix.

3.2.7 Exigences d'installation

3.2.7.1 Ubuntu 19.04

Ubuntu est le système d'exploitation le plus utilisé sur les systèmes Cloud pour la mise en place d'infrastructures informatiques. Elle existe en plusieurs versions notamment pour des utilisateurs finaux et en version serveur pour les infrastructures informatiques. En outre pour la mise en place d'un système de gestion de conteneur avec Docker engine. Les versions recommandées selon la documentation officielle de Docker sont :

- Ubuntu Groovy 20.10
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)
- Ubuntu Xenial 16.04 (LTS)

La version serveur est recommandée pour réduire l'utilisation des ressources ; mais la version avec interface graphique est aussi utilisable .

3.2.7.2 Docker Engine

L'installation du gestionnaire de conteneur [Docker](#) nécessite en amont la configuration d'un référentiel [Docker](#) en ajoutant la clé [GPG](#) officielle de [Docker](#) et taper les commande nécessaire pour configurer le référentiel stable . Il est recommandé par la communauté de faire l'installation du gestionnaire à partir du script d'installation qui se trouve sur le site officielle de la documentation. Ce script prends en compte tout les pré-requis avant l'installation de docker.

3.3 Mise en œuvre

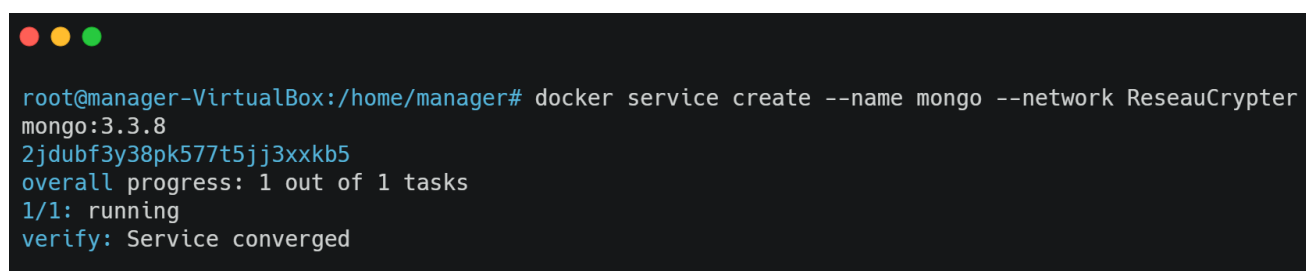
3.3.1 Vagrant et Ansible

Pour mener à bien notre projet nous avons utilisé vagrant et Ansible qui nous ont permis automatisé une grande partie du processus de déploiement des conteneurs grâce à l'édition des différents fichiers de configuration et la création des différentes tâches. En annexe nous avons les différents fichiers conçus et édités à cet effet.

3.3.2 Déploiement de la Base de donnée sous la forme de conteneur

Nous tester le bon fonctionnement de notre cluster nous avons déployé une image du registre docker.

A travers une image [Docker](#) du cloud [Docker Hub](#), nous créons un service sur le swarm manager avec cette commande sous la forme de conteneur Docker à travers un réseau crypté en précisant la version qu'on désire utiliser.

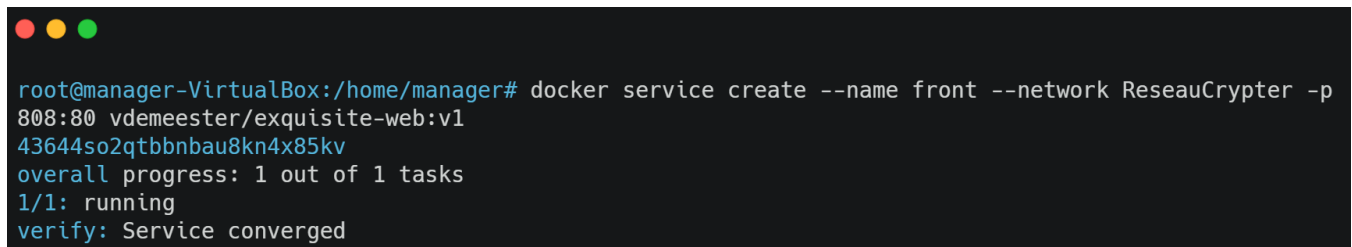


```
root@manager-VirtualBox:/home/manager# docker service create --name mongo --network ReseauCrypter
mongo:3.3.8
2jdubf3y38pk577t5jj3xxkb5
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
```

FIGURE 3.5 – Déploiement de la base donnée sous la forme de conteneur

3.3.3 Déploiement de la partie Front End de l'application sous la forme de conteneur

Nous déployons la partie Front de notre application de test sous la forme de conteneur par notre réseau crypté en précisant une **version** d'utilisation, et un mappage de port **808 : 80**.



```

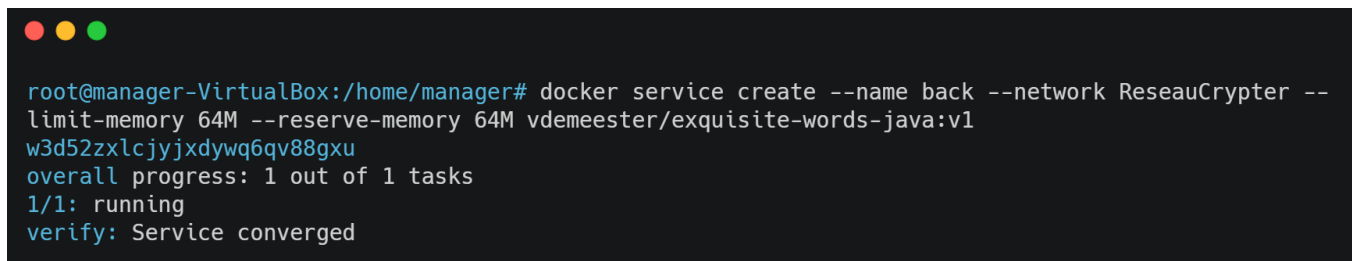
root@manager-VirtualBox:/home/manager# docker service create --name front --network ReseauCrypter -p
808:80 vdemeester/exquisite-web:v1
43644so2qtbbnbau8kn4x85kv
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged

```

FIGURE 3.6 – Déploiement de la partie Front End de l’application sous la forme de conteneur

3.3.4 Déploiement de la partie Back End de l’application sous la forme de conteneur

Nous déployons la partie Back End de notre application de test sous la forme de conteneur par notre réseau crypté en précisant une **version** d’utilisation et un espace mémoire.



```

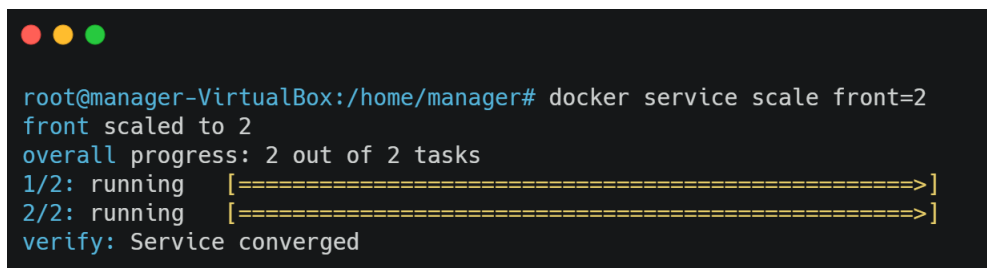
root@manager-VirtualBox:/home/manager# docker service create --name back --network ReseauCrypter --
limit-memory 64M --reserve-memory 64M vdemeester/exquisite-words-java:v1
w3d52zxlcjyjxdyqw6qv88gxu
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged

```

FIGURE 3.7 – Déploiement de la partie Back End de l’application sous la forme de conteneur

3.3.5 Duplication des conteneurs

Nous allons dupliquer les conteneurs déployer pour optimiser l’accès à notre application



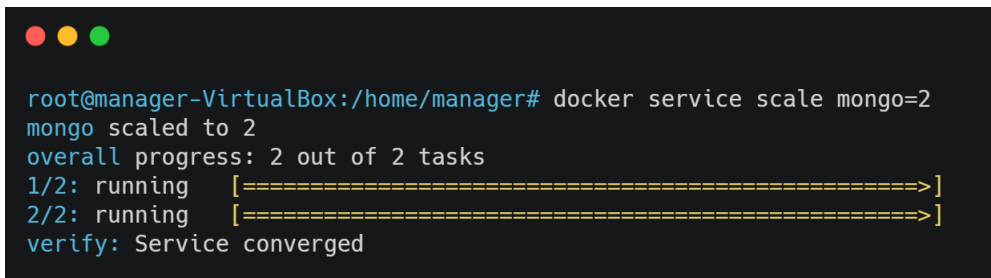
```

root@manager-VirtualBox:/home/manager# docker service scale front=2
front scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged

```

FIGURE 3.8 – Duplication de la la partie Front End

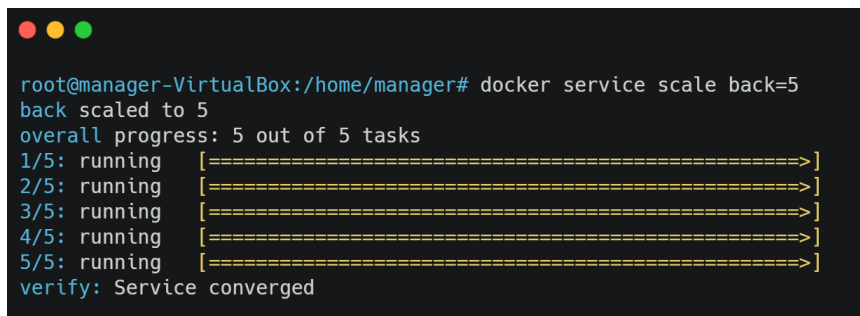
Duplication de la base de Donnée



```
root@manager-VirtualBox:/home/manager# docker service scale mongo=2
mongo scaled to 2
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged
```

FIGURE 3.9 – Duplication de la Base de donnée

Duplication de la partie Back End



```
root@manager-VirtualBox:/home/manager# docker service scale back=5
back scaled to 5
overall progress: 5 out of 5 tasks
1/5: running [=====>]
2/5: running [=====>]
3/5: running [=====>]
4/5: running [=====>]
5/5: running [=====>]
verify: Service converged
```

FIGURE 3.10 – Duplication de la la partie Back End

Résultats et suggestions

Après l'installation et la configuration du moteur Docker sur les différents machines serveurs et la création des différents conteneurs pour le déploiement de notre application de test; nous avons obtenu les resultats suivants :

4.1 Résultats

4.1.1 Présentation du Box pour les VM

```
jb@jb-HP-250-G5-Notebook-PC:~$ vagrant box add bento/ubuntu-20.04 --provider virtualbox
==> box: Loading metadata for box 'bento/ubuntu-20.04'
box: URL: https://vagrantcloud.com/bento/ubuntu-20.04
==> box: Adding box 'bento/ubuntu-20.04' (v202107.07.0) for provider: virtualbox
box: Downloading: https://vagrantcloud.com/bento/boxes/ubuntu-20.04/versions/202107.07.0/providers/virtualbox.box
==> box: Box download is resuming from prior download progress
==> box: Successfully added box 'bento/ubuntu-20.04' (v202107.07.0) for 'virtualbox'!
```

FIGURE 4.1 – Box Ubuntu 20.04

4.1.2 Sortir de l'exécution de Vagrant

```

jb@jb-HP-250-G5-Notebook-PC:~/swarm-vagrant-ansible/Vagrant$ vagrant up
Bringing machine 'swarm-maitre-1' up with 'virtualbox' provider...
Bringing machine 'swarm-maitre-2' up with 'virtualbox' provider...
Bringing machine 'swarm-worker-1' up with 'virtualbox' provider...
Bringing machine 'swarm-worker-2' up with 'virtualbox' provider...
==> swarm-maitre-1: Importing base box 'bento/ubuntu-20.04'...
==> swarm-maitre-1: Matching MAC address for NAT networking...
==> swarm-maitre-1: Checking if box 'bento/ubuntu-20.04' version '202107.07.0' is up to date...
==> swarm-maitre-1: Setting the name of the VM: Vagrant_swarm-maitre-1_1626086178588_43413
Vagrant is currently configured to create VirtualBox synced folders with
the 'SharedFoldersEnableSymlinksCreate' option enabled. If the Vagrant
guest is not trusted, you may want to disable this option. For more
information on this option, please refer to the VirtualBox manual:

  https://www.virtualbox.org/manual/ch04.html#sharedfolders

This option can be disabled globally with an environment variable:

  VAGRANT_DISABLE_VBOXSYMLINKCREATE=1

or on a per folder basis within the Vagrantfile:

  config.vm.synced_folder '/host/path', '/guest/path', SharedFoldersEnableSymlinksCreate: false
==> swarm-maitre-1: Clearing any previously set network interfaces...
==> swarm-maitre-1: Preparing network interfaces based on configuration...
swarm-maitre-1: Adapter 1: nat
swarm-maitre-1: Adapter 2: hostonly
==> swarm-maitre-1: Forwarding ports...
swarm-maitre-1: 22 (guest) => 2222 (host) (adapter 1)
==> swarm-maitre-1: Running 'pre-boot' VM customizations...
==> swarm-maitre-1: Booting VM...
==> swarm-maitre-1: Waiting for machine to boot. This may take a few minutes...
swarm-maitre-1: SSH address: 127.0.0.1:2222
swarm-maitre-1: SSH username: vagrant
swarm-maitre-1: SSH auth method: private key
swarm-maitre-1: Warning: Connection reset. Retrying...

```

FIGURE 4.2 – Résultat de Vagrant

4.1.3 Présentation du Cluster

```

root@manager-VirtualBox:/home/manager# docker node ls

```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
pk4efprqzyxme4mboiiv8urp	jb	Ready	Active		19.03.13
zb3bdmj0yo9hbdmnydolna4ef	jb	Ready	Active		19.03.13
x8ptbpd2xcwpob2lovps76w1 *	manager-VirtualBox	Ready	Active	Leader	19.03.13

FIGURE 4.3 – Swarm Manager

Nous avons notre cluster de trois machines avec comme **Leader** manager-VirtualBox; chaque machine du cluster a un identifiant spécifique qui lui a été attribué par [Docker Engine](#) et toute les serveurs composantes du **cluster** sont actifs.

4.1.4 Présentation des services déployés

Nous voyons les différentes parties de notre application de test et le nombre de duplication ; Chaque partie est appelée service. Chaque service a un **ID** donné par **Docker Engine**. On a l'**Image** par lequel chaque service est crée et le **port** d'accès.

```
root@manager-VirtualBox:/home/manager# docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
4w4vdcqmiwbx	back	replicated	5/5	vdemeester/exquisite-words-java:v1	
2ekx3i4i275w	front	replicated	2/2	vdemeester/exquisite-web:v1	*:808->80/tcp
heqztegxwv5b	mongo	replicated	2/2	mongo:3.3.8	

FIGURE 4.4 – Liste des services déployés

4.1.5 Accès à l'application à partir du Leader

Nous avons accès à notre site de test par l'adresse IP de notre serveur principal.

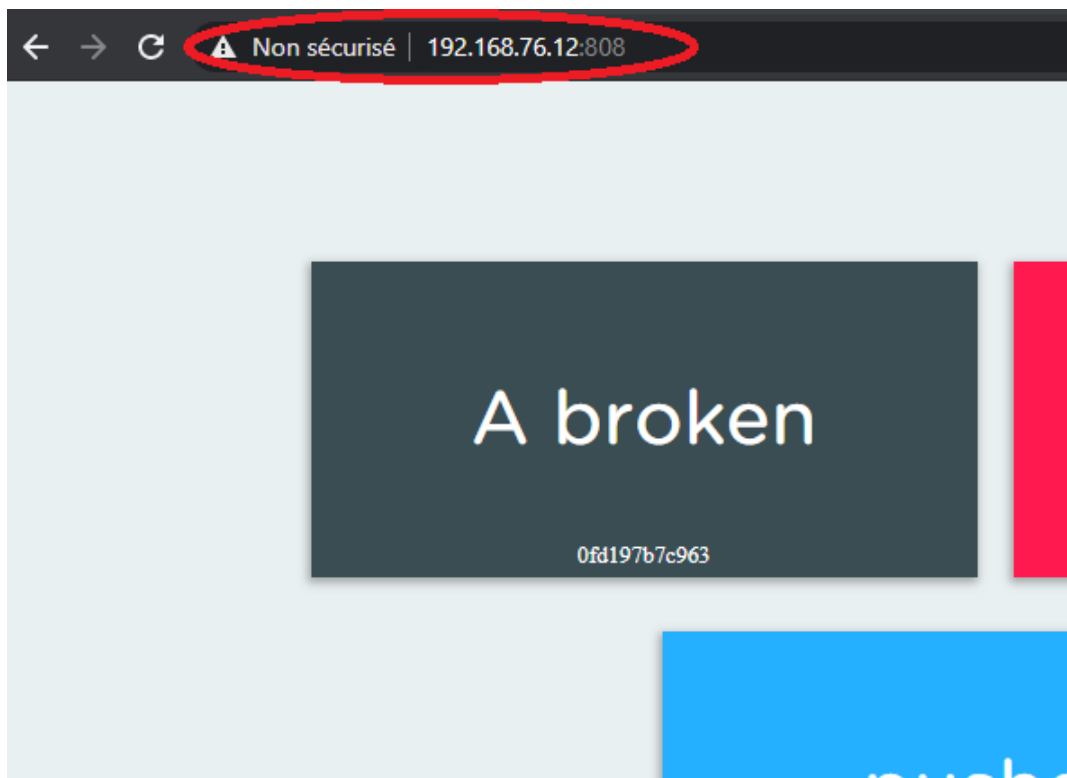


FIGURE 4.5 – Accès au site à partir de l'adresse IP 192.168.6.12 par le port 808

4.1.6 Accès à l'application à partir des Workers

Nous accédons au même site à partir des adresses IP des Workers 192.168.76.5 et 192.168.76.4 de notre cluster

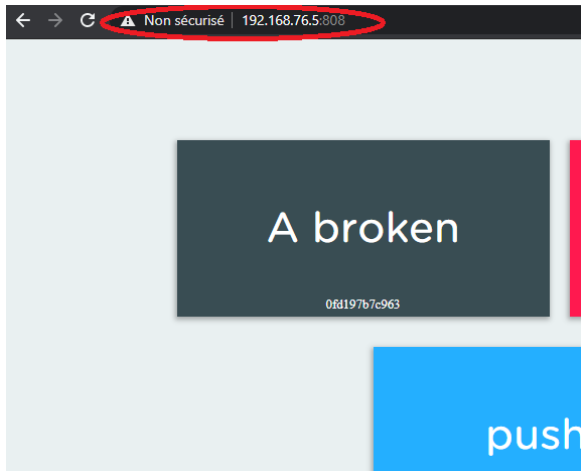


FIGURE 4.6 – Accès au site à partir de l'adresse 192.168.6.5 par le port 808

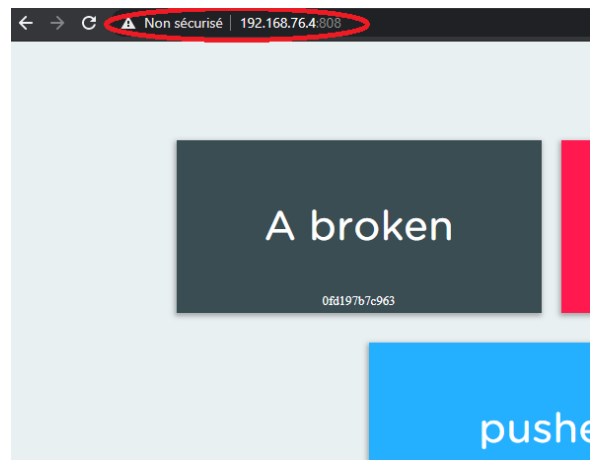


FIGURE 4.7 – Accès au site à partir de l'adresse 192.168.6.4 par le port 808

4.2 Discussion

Notre projet est né de plusieurs constats généraux de l'ordre des principes de la sécurité informatique pour la disponibilité des services web, notre solution présente [Docker Swarm](#) comme un orchestrateur de conteneurs déployés sur plusieurs machines hôtes. Les avantages d'une telle solution incluent :

- Coordonnés les conteneurs en leur affectant des tâches spécifiques
- Gérer et suivre l'état des différents conteneurs présents dans le cluster
- Redistribuer les tâches en cas de besoins
- Assurer la scalabilité et flexibilité de l'infrastructure, en augmentant ou diminuant le nombre de clusters mis en services
- Gérer et mettre à jour les applications sur plusieurs conteneurs

Notre solution élimine tout besoin pour l'administrateur système d'installer un autre outil pour la gestion de ses applications conteneurisées. Le mode swarm de docker permet de mettre en place une architecture de load balancing pour optimiser la disponibilité des applications, tout ceci est automatiser par les outils vagrant et ansible. Enfin, grâce au fichier de log, et système de veille technologie en place l'administrateur pourrait veiller aux attaques de types [DoS](#) et modifier s'il le faut le rôle des différents constituants du cluster si cela s'avère nécessaire pour maintenir la haute disponibilité des applications du système.

Conclusion

Ces six dernières années ont vu le développement d'une nouvelle méthode de virtualisation dans les infrastructures informatiques et cloud computing : *la conteneurisation Docker*. Le principe fondateur de cette approche moderne est de créer des environnements virtuels au niveau applicatif. Ceci révolutionne la virtualisation traditionnelle qui consiste à la création d'environnements virtuels au niveau de l'infrastructure. Cette approche de virtualisation légère a pour principaux avantages l'utilisation efficace des ressources du fournisseur, l'amélioration des performances et la réduction des coûts pour les utilisateurs ; Ceci implique l'apparition de nouvelles formes d'architectures qui organisent les services des infrastructures informatiques au sein de conteneurs prêts à être instanciés dans des machines virtuelles ou physiques.

Toutefois, l'utilisation des conteneurs Docker ce type d'environnement n'est pas sans défi. En effet, plusieurs difficultés doivent être résolues, parmi elles on peut citer *l'orchestration* (gestion des déploiements, migrations, mises à jour, etc.) et la garantie de *la disponibilité* de service (tolérance aux pannes). Comme nous l'avons présenté, le défi d'orchestration de conteneurs Docker est résolu par l'adoption [Docker Swarm](#) comme plateforme évolutive d'orchestration des applications conteneurisées. Quant à la tolérance au panne, l'étude des travaux de recherche effectués dans ce domaine montre que le mécanisme de réplication des machines est une solution à peut aider à résoudre ce problème.

Bien que notre solution réponde au besoin énoncer, il faut noter certaines insuffisances de [Docker Swarm](#) telles pas de garanties solides sur l'état du cluster et la faible tolérance aux pannes franches comme l'arrêt d'un nœud de travail ou d'un conteneur.

Nous envisageons dans nos travaux futurs revoir les caractéristiques de d'autres orchestrateurs pour pallier aux insuffisances de Swarm afin de rendre plus robuste le système mis en place.

Bibliographie

- [1] Gor Mack Diouf, *Une plate forme de conteneurs Docker Tolérance aux fautes Byzantines*. (Université de Québec à Montréal), Avril 2019
- [2] Fabrizio Soppelsa, Chanwit Kaewkasi , *Native Docker Clustering with Swarm*. (BIRMINGHAM-MUMBAI)
- [3] Randall Smith, *Docker Orchestration*. (BIRMINGHAM-MUMBAI), 2017
- [4] J. Kovács, P. Kacsuk, M. Emődi, *Advances in Engineering Software*, (Elsevier), 2018
- [5] Sean P. Kane, Karl Matthias, *Docker : Up Running : Shipping Reliable Containers in Production*, 2018
- [6] Randall Smith, *The Docker Book : Containerization is the new virtualization*. (Turnbull), 2014
- [7] Srdjan Grubor, *Deployment with Docker : Apply continuous integration models, deploy applications quicker, and scale at large by putting Docker to work*. (Turnbull), 2017
- [8] Viktor Farcic, *The DevOps 2.1 Toolkit : Docker Swarm*. (BIRMINGHAM-MUMBAI), 2017

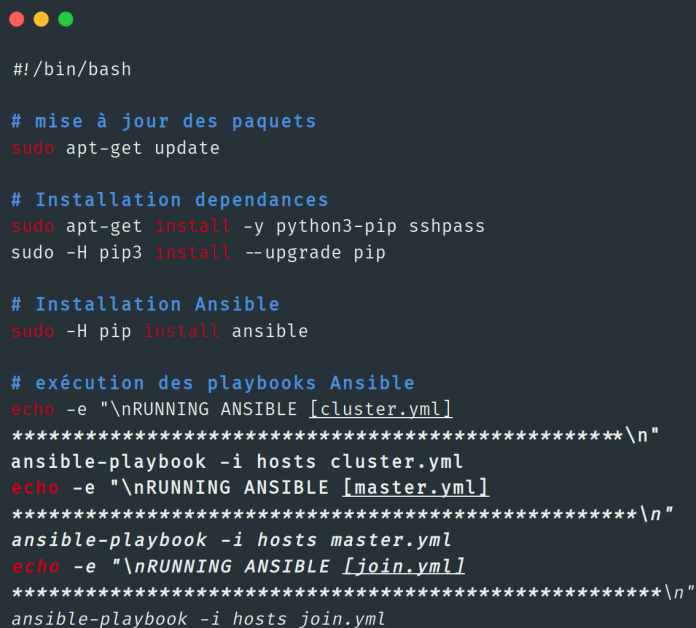
Webographie

- [9] <https://scholar.google.fr/schhp?hl=fr>
- [10] La Documentation complète de Docker <https://docs.docker.com/engine/swarm/key-concepts/>.
- [11] Asad Ali, Administrateur système, Cloud Computing. *14 outils d'orchestration de conteneurs pour DevOps*, Publié le Juillet 27 2020. <https://geekflare.com/fr/container-orchestration-software/#anchor-docker-swarm>, consulté le 5 juillet 2020.
- [12] Xavier Biseul ,JDN. Mis à jour le 30 Novembre 2016 15 :26 *Comparatif : quatre orchestrateurs de containers à la loupe*. <https://www.journaldunet.com/solutions/cloud-computing/1188929-comparatif-qui-va-gagner-la-guerre-des-orchestrateurs-de-containers/1189008-tableau-de-synthese>, consulté le 17 Décembre 2020.
- [13] Yohan Gracia, Technologie, Docker. Mis à jour le 12 Avril 2020. *Docker et Docker-Compose : les commandes de base à connaître*. <https://www.padok.fr/blog/docker-docker-compose-commandes-connaître>, consulté le 25 10 Novembre 2020.
- [14] Ajdaini-hatim. *Comprendre, Gérer et Manipuler un cluster Docker Swarm* <https://devopssec.fr/article/comprendre-gerer-manipuler-un-cluster-docker-swarm>, consulté le 30 Décembre 2020.
- [15] Oracle VM VirtualBox *Oracle VM VirtualBox Datasheet*. <https://www.oracle.com/fr/a/ocom/docs/oracle-vm-virtualbox-ds-1655169.pdf>, consulté le 11 Janvier 2021.
- [16] Jean-François Pillou -le mardi 26 mai 2015. *Cycle de vie d'un logiciel*. <https://www.commentcamarche.net/contents/473-cycle-de-vie-d-un-logiciel>, consulté le 15 Décembre 2020.
- [17] Savaram Ravindra, le 3 Septembre 2018. *Kubernetes vs. Docker Swarm : What's the Difference?*. <https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/>, consulté le 15 Décembre 2020.

-
- [18] Claus Pahl, Brian Lee, - a Technology Review. *Containers and Clusters for Edge Cloud Architectures*. <https://thenewstack.io/kubernetes-vs-docker-swarm-what-s-the-difference/>, consulté le 4 Février 2021.
- [19] Youssfi Mohamed -Laboratoire Signaux Systèmes Distribués et intelligence Artificielle, Université Hassan II Casablanca, Maroc. *Docker*. https://www.researchgate.net/profile/Youssfi_Mohamed, consulté le 20 Décembre 2020.
- [20] Youssfi Mohamed -Laboratoire Signaux Systèmes Distribués et intelligence Artificielle, Université Hassan II Casablanca, Maroc . *Orchestration des conteneurs avec Docker Swarm*. <https://fr.slideshare.net/mohamedyoussfi9>, consulté le 27 Novembre 2020.
- [21] Définition de plusieurs termes et Mots clés <https://fr.wikipedia.org/wiki/Accueil>
- [22] Lien git <https://github.com/itkg-ppottie/packer-ansible-vagrant/tree/master/doc>
- [23] Lien git https://github.com/itkg-ppottie/packer-ansible-vagrant/blob/master/doc/vagrant_ansible.md
- [24] Lien git <https://github.com/acuto/swarm-vagrant-ansible>
- [25] Lien git <https://www.ictjournal.ch/articles/2020-02-12/devsecops-et-processus-de-deploiement-dune-application-web>

4.3 Annexes

4.3.1 Les fichiers de configuration de Vagrant



```
#!/bin/bash

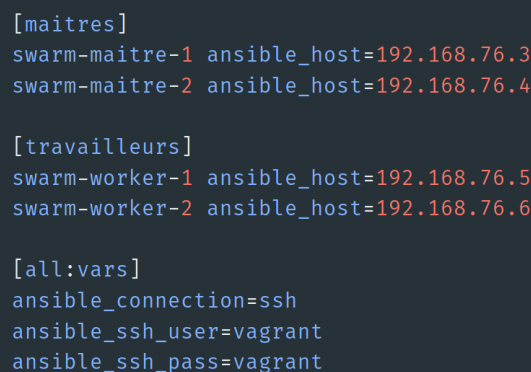
# mise à jour des paquets
sudo apt-get update

# Installation dependances
sudo apt-get install -y python3-pip sshpass
sudo -H pip3 install --upgrade pip

# Installation Ansible
sudo -H pip install ansible

# exécution des playbooks Ansible
echo -e "\nRUNNING ANSIBLE [cluster.yml]"
*****\n"
ansible-playbook -i hosts cluster.yml
echo -e "\nRUNNING ANSIBLE [master.yml]"
*****\n"
ansible-playbook -i hosts master.yml
echo -e "\nRUNNING ANSIBLE [join.yml]"
*****\n"
ansible-playbook -i hosts join.yml
```

FIGURE 4.8 – Script d’installation



```
[maitres]
swarm-maitre-1 ansible_host=192.168.76.3
swarm-maitre-2 ansible_host=192.168.76.4

[travailleurs]
swarm-worker-1 ansible_host=192.168.76.5
swarm-worker-2 ansible_host=192.168.76.6

[all:vars]
ansible_connection=ssh
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
```

FIGURE 4.9 – Hosts et adresses IP

```

# -*- mode: ruby -*-
# vi: set ft=ruby :
nodes = [
  # Les machines du cluster
  { :hostname => 'swarm-maitre-1', :ip => '192.168.76.3', :ram => 1024, :cpus => 1 },
  { :hostname => 'swarm-maitre-2', :ip => '192.168.76.4', :ram => 1024, :cpus => 1 },
  { :hostname => 'swarm-worker-1', :ip => '192.168.76.5', :ram => 1024, :cpus => 1 },
  { :hostname => 'swarm-worker-2', :ip => '192.168.76.6', :ram => 1024, :cpus => 1 }
]

Vagrant.configure("2") do |config|

  config.ssh.insert_key = false
  # Mise en place d'une connexion ssh pour la communication entre les machines
  config.ssh.forward_agent = true
  # Provision des nœuds
  nodes.each do |node|
    config.vm.define node[:hostname] do |nodeconfig|
      nodeconfig.vm.box = "bento/ubuntu-20.04";
      nodeconfig.vm.hostname = node[:hostname] + ".box"
      nodeconfig.vm.network :private_network, ip: node[:ip]
      memory = node[:ram] ? node[:ram] : 1024;
      cpus = node[:cpus] ? node[:cpus] : 1;
      nodeconfig.vm.provider :virtualbox do |vb|
        vb.customize [
          "modifyvm", :id,
          "--memory", memory.to_s,
          "--cpus", cpus.to_s
        ]
      end
    end
  end
end

# Considérons, swarm-worker-2 comme le serveur Ansible
config.vm.define "swarm-worker-2" do |ansible|
  # Provision playbook Ansible
  ansible.vm.provision "file", source: "../Ansible", destination: "$HOME"
  # Installation d'Ansible et configuration des nœuds
  ansible.vm.provision "shell", path: "ansible.sh"
  ansible.inventory_path = "provisioning/hosts"
end
end

```

FIGURE 4.10 – Vagrantfile

4.3.2 Les fichiers de configuration de Ansible

```
---
# Téléchargement, installation de Docker engine
- hosts: all
  become: true
  vars_files:
    - vars.yml
  strategy: free

  tasks:
    - name: Add the docker signing key
      apt_key: url=https://download.docker.com/linux/ubuntu/gpg state=present

    - name: Add the docker apt repo
      apt_repository: repo='deb [arch=amd64] https://download.docker.com/linux/ubuntu focal
stable' state=present

    - name: Install packages
      apt:
        name: "{{ PACKAGES }}"
        state: present
        update_cache: true
        force: yes

    - name: Add user vagrant to the docker group
      user:
        name: vagrant
        groups: docker
        append: yes
```

FIGURE 4.11 – Script d’installation de docker,vagrant et d’autres packages

```
---
#Choix du maitre principal et initialisation du cluster
- hosts: swarm-maitre-1
  become: true

  tasks:
    - name: Initialize the cluster
      shell: docker swarm init --advertise-addr 192.168.76.3 >> cluster_initialized.txt
      args:
        chdir: $HOME
        creates: cluster_initialized.txt
```

FIGURE 4.12 – Désignation de la machine maitre

```

---
# Rejoindre le cluster docker swarm
- hosts: swarm-maitre-1
  become: true
  gather_facts: false

  tasks:
    - name: Get master join command
      shell: docker swarm join-token manager
      register: master_join_command_raw

    - name: Set master join command
      set_fact:
        master_join_command: "{{ master_join_command_raw.stdout_lines[2] }}"

    - name: Get worker join command
      shell: docker swarm join-token worker
      register: worker_join_command_raw

    - name: Set worker join command
      set_fact:
        worker_join_command: "{{ worker_join_command_raw.stdout_lines[2] }}"

- hosts: swarm-maitre-2
  become: true

  tasks:
    - name: Master joins cluster
      shell: "{{ hostvars['swarm-maitre-1'].master_join_command }}" >> node_joined.txt
      args:
        chdir: $HOME
        creates: node_joined.txt

- hosts: workers
  become: true

  tasks:
    - name: Workers join cluster
      shell: "{{ hostvars['swarm-maitre-1'].worker_join_command }}" >> node_joined.txt
      args:
        chdir: $HOME
        creates: node_joined.txt

```

FIGURE 4.13 – Création et initialisation du cluster
