



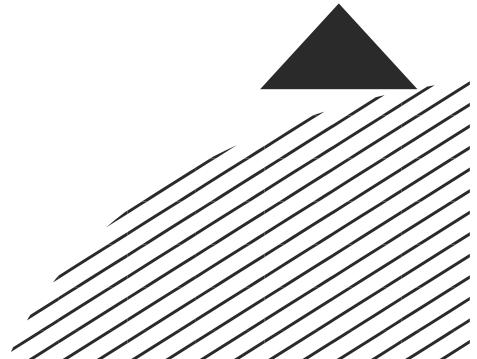
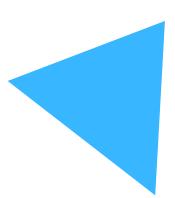
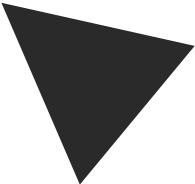


# REMERCIEMENT

On remercie dieu le tout puissant de nous avoir donné la santé et la volonté d'entamer et de terminer ce projet.

Nous tenons à remercier toute l'équipe pédagogique de la faculté des sciences Ain chock et les intervenants professionnels responsables du département de mathématiques et informatique.

Avant d'entamer ce rapport, nous profitons de l'occasion pour exprimer avec un grand plaisir et un grand respect nos remerciement à notre professeur **Mr. RAOUYANE BRAHIM** pour la qualité de son encadrement exceptionnel pour sa patience sa rigueur sa disponibilité son encouragements et ses conseils concernant les missions évoquées dans ce rapport et malgré tout ce que notre pays a vécu à cause de corona virus, on le remercie pour les réunions en ligne organisées durant toute la réalisation du projet et qui nous ont permis de le terminer .



# Liste des figures :

Figure 1:Composants Docker	14
Figure 2: Architecture Docker	15
Figure 3:Architecture de virtualisation	20
Figure4:Architecture de Conteneurisation	20
Figure5:Réseau Bridge	22
Figure6:Réseau Host	23
Figure7: Réseau Null	23
Figure8: Réseau Macvlan	24
Figure9:Réseau Overlay	25
Figure10:Architecture Mesos	31
Figure11: Architecture Swarm	35
Figure 12:Page principale de Portainer	41
Figure 13:Liste des conteneurs dans Portainer	42
Figure 14:Gestion des images dans Portainer	43
Figure 15:Gestion des réseaux dans Portainer	43
Figure 16:Gestion des volumes	44
Figure17:Fonctionnalité de Rancher	45
Figure 18:Page principale de Rancher	45
Figure 19:DuckDns	49
Figure 20:initialisation du nom pour conteneur	50
Figure 21:creation du volume et démarrage du conteneur OpenVPN	50
Figure 22:générer un certificat privé:	51
Figure 23:Démarrage du serveur OpenVPN	51
Figure 24:Générer un certificat client	51
Figure 25:Récupération de la configuration client	52
Figure 26:Creation du volume avec NFS	53
Figure 27:NFS dans Portainer	54
Figure 28:Génère une clé RSA	55
Figure 29:Architecture Kubernetes	57
Figure30:Fonctionnement des déploiements de k8s	61
Figure31: composants des Pods	69
Figure 32:Diagramme de pod	71
Figure 33:Fonctionnement du déploiement	75
Figure34:Exemple fonctionnement d'un Job	76
Figure 35:Fonctionnement de réseau kubernetes	79
Figure36:Fonctionnement du service	81
Figure37:Service ClusterIP	82
Figure 38:Service NodePort	83
Figure 39:service nodePort créé	84
Figure 40:Connexion au service NodePort de l'extérieure	85
Figure 41:Architecture LoadBalancer	85
Figure 42:liste des services créés	86
Figure43:Environnement Cloud	86
Figure44:Environnement Bare-metal	87
Figure45:Architecture de metaLB	88

Figure 46:fichier configmap de MetalLB	89
Figure 47:Service LoadBalancer créé	90
Figure 48 : Connexion au service LoadBalancer de l'extérieur	90
Figure 49: Architecture d'ingress	91
Figure50:PersistentVolumeClaim	94
Figure 51:exécution d'une application simple de serveur web	95
Figure 52:Création du pod pour le trafic entrant	96
Figure 53:Page d'entrée du Dashboard	97
Figure 54:Kubernetes vs Swarm	101
Figure 55:Docker EE Dashboard avec des conteneurs déployés avec Swarm et kubernetes	103
Figure 56: Architecture de docker EE	104

# Liste des tableaux :

Tableau 1:Composants de docker .....	14
Tableau 2:Architecture de docker .....	16
Tableau 3:Commandes pour gérer les Conteneurs .....	17
Tableau 4:Swarm & Swarm Mode .....	33
Tableau 5:Swarmkit & swarm mode.....	34
Tableau 6 : Quelques commandes docker swarm.....	38
Tableau 7:Objets de kubernetes.....	56
Tableau 8:Composants de kubernetes .....	57
Tableau 9:Composants du Master .....	58
Tableau 10:Composants du Worker .....	59
Tableau 11:Quelque produit de GCP .....	59
Tableau 12:Quelque API de GCP.....	60
Tableau 13:Composants de la syntaxe de Kubectl .....	62
Tableau 14:Quelque commande de kubectl .....	68
Tableau 15:Differentes status d'un pod.....	70
Tableau 16:Types des Jobs .....	77
Tableau 17:Fonctionnalités d'Octant.....	99
Tableau 18:Kubernetes Vs Swarm .....	100
Tableau 19:Fonctionnalité de docker EE.....	104

## Résumé :

---

Ce présent rapport fait état du travail effectué sur le logiciel Opensource Docker, une technologie de conteneurisation qui permet la création et l'utilisation de conteneurs linux.

Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.

Ce rapport d'écrit le principe de conteneurisation avec Docker et l'orchestration des conteneurs en utilisant Docker swarm et kubernetes.

Ce rapport portera sur deux parties :

- **Etat de l'art** : étude théorique de cette technologie.
- **Etude comparative de docker swarm et Kubernetes les solutions d'orchestration des conteneurs.**

## **Abstract :**

---

This report reports on the work done on Docker open source software, a containerization technology that allows the creation and use of Linux containers.

Docker is a tool that can package an application and its dependencies in an isolated container, which can be run on any server.

This report describes the principle of containerization with Docker and the orchestration of containers using Docker swarm and kubernetes. This report will focus on two parts :

- **State of the art:** Theoretical study of this technology
- **Comparative study of Docker swarm and kubernetes the solutions of the orchestration of containers**

# Table des matières :

<b>Introduction générale :</b> .....	<b>11</b>
<b>Partie 1 : L'état de l'art .....</b>	<b>12</b>
I.    Technologie Conteneur Docker .....	13
1.    Conteneurs : .....	16
2.    Images : .....	17
3.    Dockerfile : .....	17
4.    Volume : .....	17
5.    Docker-compose : .....	18
6.    Docker Stack : .....	18
7.    Virtualisation VS Conteneurisation .....	19
II.    Docker networking .....	21
1.    Différents types de réseau .....	21
2.    Réseau Single-Host.....	26
3.    Réseau Multi-Host .....	27
III.    Conclusion .....	27
<b>Partie 2 : Etude des solutions d'orchestration des conteneurs.....</b>	<b>28</b>
I.    Introduction : .....	29
II.    Orchestration des conteneurs : .....	29
1.    Propriétés : .....	30
2.    Outils d'Orchestration des conteneurs : .....	30
III.    Docker Swarm .....	32
1.    Présentation : .....	32
2.    Cluster Swarm : .....	34
3.    Services: .....	38
4.    Configuration de DNS dans Docker Swarm: .....	39
5.    Interfaces Graphiques: .....	40
6.    Sécurisation: .....	46
I.    Kubernetes .....	56
1.    Composants & Objets Kubernetes .....	56
1.    Google Cloud Platform (GCP) : .....	59
2.    Commande de configuration Kubectl .....	60
3.    Création et gestion d'un cluster multi-nœud : .....	62
4.    Gestion et manipulation d'un pod .....	68

5. Gestion et manipulation des Deployments .....	75
6. Jobs & Cronjobs .....	75
7. Mise en réseau de cluster Kubernetes .....	78
8. Gestion des volumes : .....	92
9. Sécurité .....	95
10. Interfaces Graphiques : .....	96
II. Kubernetes VS Docker Swarm.....	99
1. Différence entre Docker swarm et Kubernetes : .....	99
2. Interopérabilité des deux orchestrateurs :.....	102
Conclusion générale : .....	<b>105</b>
Références: .....	<b>108</b>

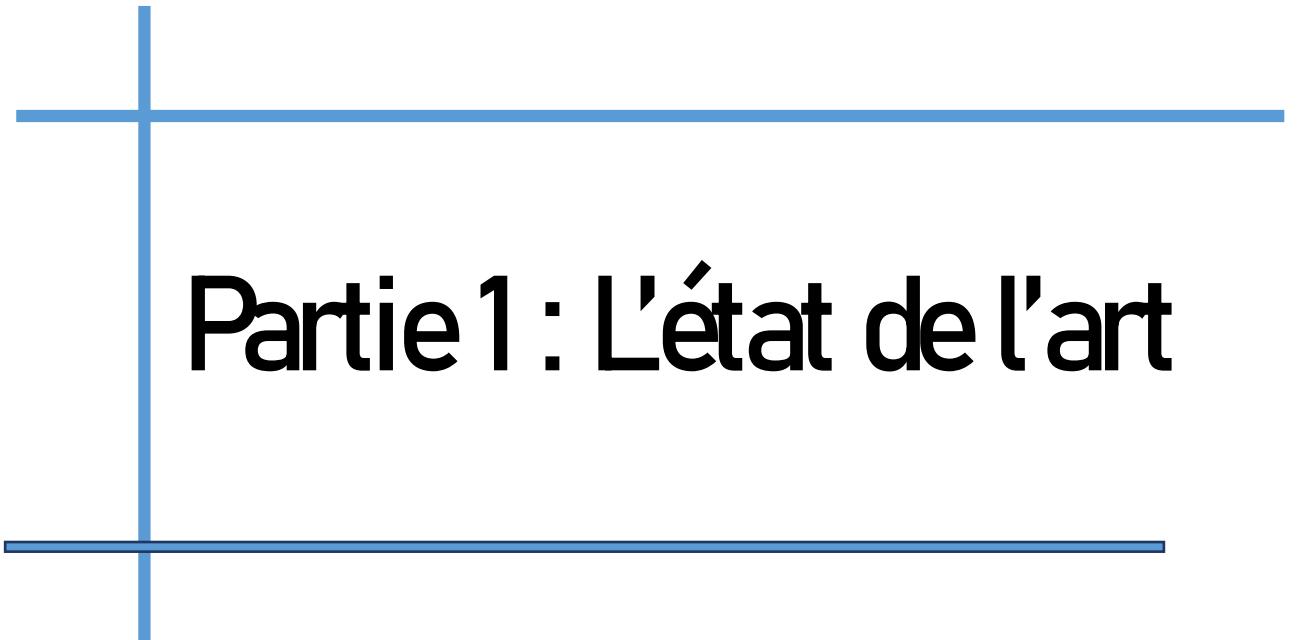
# Introduction générale :

Un jour un développeur a passé toute une nuit à coder tranquillement sur son ordinateur. Après des heures de travail, il a testé son application, l'application a bien fonctionné .Le lendemain développeur a transféré son projet sur un autre ordinateur, mais malheureusement, l'application n'a pas fonctionné, c'est à cause d'un problème d'environnement. Deux systèmes peuvent avoir des différences de version sur les dépendances ou encore des bibliothèques manquantes.

Dans cet exemple, le problème se limite à 2 systèmes, mais imaginez une équipe de 10 personnes avec des ordinateurs sous OS X, Linux ou même Windows, un serveur de test sous Ubuntu 20.04, et un serveur de production sous CentOS 7. Assurer le fonctionnement de leur application sur tous ces environnements peut s'avérer être un vrai cauchemar.

Les dockers viennent proposer une solution à ce type de problème (et au problème d'hétérogénéité) en général. Il s'agit d'une plateforme qui permet d'exécuter le code à l'intérieur d'un conteneur indépendamment de la machine sur laquelle on est. Un conteneur ressemble à une machine virtuelle sauf qu'il n'inclut pas un système d'exploitation ce qui lui permet de s'exécuter en quelque secondes et d'être beaucoup plus léger, mais, cette plateforme a des insuffisances. En effet, la création des conteneurs nécessite la saisie des plusieurs lignes de commandes. Par la suite l'utilisateur va mettre beaucoup de temps pour finir une tâche (création, modification, suppression) et beaucoup de concentration. Le temps sera facturé et l'entreprise va subir les coûts. Afin, d'optimiser les coûts et éviter que les employés se lassent, on a besoin de trouver des interfaces graphiques pour la gestion des conteneurs du docker, qui permet de créer , modifier ,publier et supprimer ces conteneurs tout en diminuant le temps alloué pour effectuer une telle tâche.

D'où, notre projet consiste à bien comprendre les principes de Docker et de quelques orchestrateurs de conteneurs tel que Docker swarm et kubernetes tout en utilisant des machines locales ainsi que des interfaces graphiques pour chacun d'eux. Après nous réalisons une étude comparative des deux orchestrateurs pour savoir le meilleur à utiliser, et la fin du projet nous essayons de trouver une solution pour interopérabilité des deux orchestrateurs.



# **Partie 1 : L'état de l'art**

## I. Technologie Conteneur Docker

Docker est un système de gestion de conteneurs. Il vous permet de "packager" une application ou un site web avec tout son environnement et ses dépendances dans un conteneur, qui peut ensuite être facilement et simplement géré : transfert vers un autre serveur, évolutivité, mise à jour.

Docker a été écrit dans le langage de programmation Go et est sorti en 2013. Initialement, il ne fonctionnait qu'avec les systèmes Linux, mais pour le moment, il peut également être utilisé sur Windows et MacOs. Bien que le projet soit relativement nouveau, il est déjà largement utilisé par de nombreux experts et continue de gagner en popularité.

Une partie importante de l'écosystème Docker est le [Docker Hub](#), un référentiel d'images de conteneurs ouvert. Vous y trouverez des dizaines d'applications prêtes à l'emploi de développeurs officiels. Parmi eux se trouvent nginx, MySQL, Apache, Gitlab, Redmine, Elasticsearch, Jenkins et autres.

Le 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.

### 1- Flux de travail de Docker

Le Docker Engine (Docker Engine) est l'un de ses principaux composants. Il est responsable du fonctionnement de la plateforme Docker dans son ensemble. À la base, il s'agit d'une application client-serveur composée de trois composants principaux (**voir figure 1**):

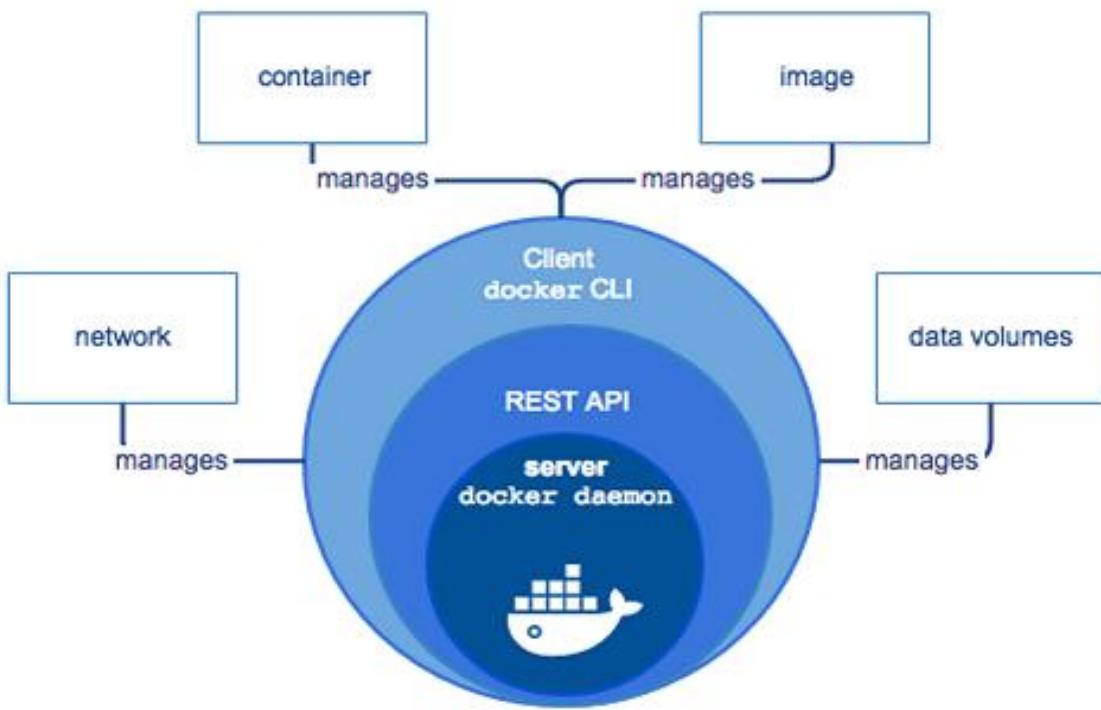


Figure 1: Composants Docker

Le client Docker et le démon peuvent fonctionner sur le même système. Nous pouvons également connecter un client Docker à Docker daemon distant. De plus, en utilisant une API REST, le client et Docker daemon communiquent via des sockets UNIX ou une interface réseau.

Docker Daemon	processus d'arrière-plan persistant qui gère les images Docker, les conteneurs, les réseaux et les volumes de stockage. Docker Daemon écoute constamment les demandes de l'API Docker et les traite
Docker Engine REST API	une API est utilisée par les applications pour interagir avec Docker Daemon. Il est accessible par un client http.
Docker CLI	client d'interface de ligne de commande pour interagir avec Docker Daemon. Il simplifie considérablement la façon dont vous gérez les instances de conteneur et est l'une des principales raisons pour lesquelles les développeurs adorent utiliser Docker.

Tableau 1: Composants de docker

## 2- Architecture de Docker :

L'architecture de Docker utilise un modèle client-serveur et se compose des composants Docker Client, Docker Host, Docker Object et Docker Registry / Hub (**voir figure 2**). Examinons chacun d'eux en détail :

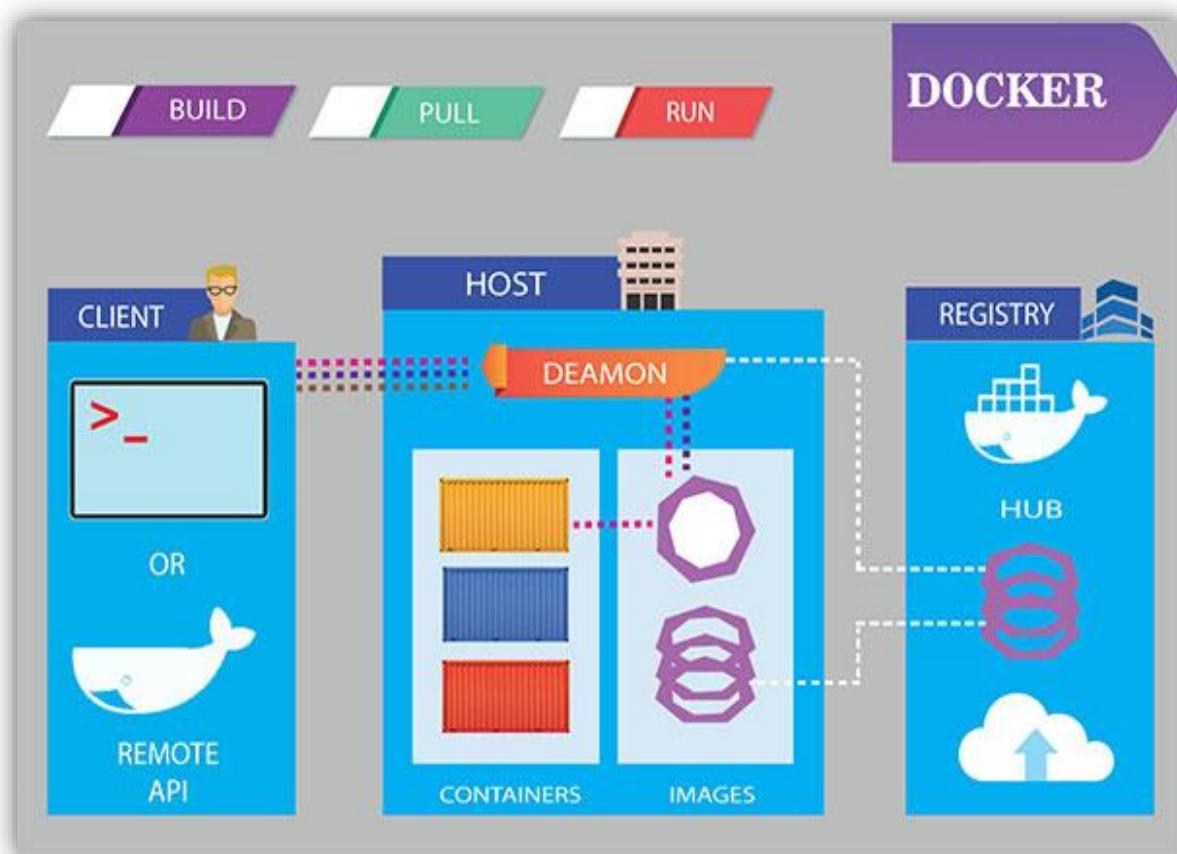


Figure 2: Architecture Docker

Docker Client	Les utilisateurs Docker peuvent interagir avec Docker via un client. Lorsqu'une commande docker s'exécute, le client les envoie au démon dockerd, qui les exécute. L'API Docker est utilisée par les commandes Docker. Il est possible pour le client Docker de communiquer avec plusieurs démons.
Docker Host	L'hôte Docker fournit un environnement complet pour exécuter et exécuter des applications. Il comprend Docker Daemon, les images, les conteneurs, les réseaux et le stockage. , Daemon est responsable de toutes les actions liées aux conteneurs et reçoit les commandes via la CLI ou l'API REST. Il peut également communiquer avec d'autres démons pour gérer ses services.

Docker Registry	Docker Registry sont des services qui fournissent des emplacements à partir desquels vous pouvez stocker et télécharger des images. En d'autres termes, Docker Registry contient des référentiels Docker qui hébergent une ou plusieurs images Docker. Les registres publics comprennent deux composants, à savoir le Docker Hub et Docker Cloud. Vous pouvez également utiliser des registres privés. Les commandes les plus courantes lorsque vous travaillez avec des registres incluent: docker push, docker pull, docker run.
-----------------	--

Tableau 2:Architecture de docker

## 1. Conteneurs :

Un conteneur Docker est une instance d'exécution d'une image. À partir d'une image, vous pouvez créer plusieurs conteneurs (tous exécutant l'exemple d'application) sur plusieurs plateformes Docker.

Un conteneur s'exécute comme un processus discret sur la machine hôte. Étant donné que le conteneur s'exécute sans qu'il soit nécessaire de démarrer un système d'exploitation invité, il est léger et limite les ressources (par exemple, la mémoire) nécessaires pour le laisser fonctionner.

Les conteneurs isolent le logiciel de son environnement et garantissent son fonctionnement uniforme malgré les différences, par exemple, entre le développement et le transfert.

Les conteneurs Docker ont plusieurs caractéristiques principales. **Ils permettent :**

- La portabilité d'applications
- L'isolation de processus
- Prévention de modification venant de l'extérieur
- La gestion de l'utilisation des ressources

En plus, ils nécessitent beaucoup moins de ressources que les machines virtuelles traditionnelles utilisées pour le déploiement d'applications isolés. **Ils ne permettent pas :**

Le mélange avec d'autre processus

1. L'exécution sur des systèmes d'exploitation différents
2. La vulnérabilité aux attaques et l'utilisation abusives des ressources du système hôte.exit

Pour gérer les conteneurs, il existe plusieurs commandes, parmi ces commandes :

Création et lancement d'un conteneur	<code>\$docker container run -di -name centos centos:latest</code>
Lister les conteneurs	<code>\$docker container ls ou bien \$ docker container ls -a</code>
Arrêter un conteneur	<code>\$docker container stop [container ID or NAME]</code>
Création d'un conteneur qui fait un ping sur google.com	<code>\$docker container run -d debian ping google.com</code>
Suppression d'un conteneur	<code>\$docker container rm [container ID]</code>

Tableau 3:Commandes pour gérer les Conteneurs

## 2. Images :

Une image de conteneur Docker est un package logiciel exécutable autonome et léger qui comprend tout le nécessaire pour exécuter une application : code, runtime, outils système, bibliothèques système et paramètres.

L'image peut ensuite être déployée dans n'importe quel environnement Docker et exécutable en tant que conteneur.

## 3. Dockerfile :

Le Dockerfile est un fichier qui contient toutes les instructions pour créer une image, comme des métadonnées (Mainteneur, label, etc.), ou même les commandes à exécuter pour installer un logiciel.

## 4. Volume :

Les conteneurs Docker n'impliquent aucun type de stockage de données permanent, cependant, il y a souvent des situations où cela est nécessaire. Le volume Docker est destiné à résoudre ces problèmes.

Volumes sont un mécanisme de stockage des données créées et utilisées par les conteneurs Docker (de la machine hôte au conteneur).

Un volume Docker est initialisé lors de la création d'un conteneur et est monté dans le conteneur comme un système de fichiers. Si l'image du conteneur contient déjà des données dans le répertoire dans lequel est monté le système de fichiers, ces données sont copiées dans

le nouveau volume lors de son initialisation. Les volumes de données ont l'avantage particulier de survivre à l'arrêt du conteneur et par défaut, Docker n'efface pas les volumes, même lors de la destruction d'un conteneur associé. En fait, il est même impossible d'effacer un volume tant que ce dernier est référencé par un conteneur.

Il est possible de créer un volume avec la commande `$docker volume create`

Il est aussi possible de créer ou d'attacher un volume à un conteneur via les commandes docker create et docker run command. Il est à noter que, par défaut, un volume est monté en mode lecture-écriture, mais qu'il est possible de limiter l'accès au seul mode lecture. La commande ci-dessous crée par exemple un conteneur nommé web depuis l'image training/webapp avec un volume baptisé /webapp stocké sur l'hôte local.

`$docker run -d -P --name web -v /webapp training/webapp python app.py`

## 5. Docker-compose :

Docker Compose est un outil inclus avec Docker. Il est conçu pour résoudre les problèmes liés au déploiement de projets. Il permet de définir le comportement de vos conteneurs et d'exécuter des applications Docker à conteneurs multiples. La configuration se fait à partir d'un fichier YAML, et ensuite, avec une seule commande, vous créez et démarrez tous vos conteneurs de votre configuration.

### Différence entre Docker et Docker Compose :

**Docker** est utilisé pour gérer les conteneurs individuels (services) qui composent l'application.

**Docker Compose** est utilisé pour gérer simultanément plusieurs conteneurs qui composent l'application. Cet outil offre les mêmes fonctionnalités que Docker, mais vous permet de travailler avec des applications plus complexes.

## 6. Docker Stack :

Docker Stack se situe à un niveau supérieur à celui des conteneurs Docker et permet de gérer l'orchestration de plusieurs conteneurs sur plusieurs machines. Docker Stack est exécuté sur un Docker Swarm, qui est essentiellement un groupe de machines exécutant le démon Docker, qui sont regroupées, regroupant essentiellement des ressources.

### 1- Fonctionnalités :

La fonctionnalité Docker Stack est incluse avec le moteur Docker. Vous n'avez pas besoin d'installer de packages supplémentaires pour l'utiliser. Le déploiement de piles Docker fait

partie du mode Swarm. Il prend en charge les mêmes types de fichiers de composition, mais la gestion se produit dans le code Go, à l'intérieur du Docker Engine.

## 2- Docker stack VS Docker Compose

Docker Compose est un outil officiel qui vous aide à gérer vos conteneurs Docker en vous permettant de tout définir via un docker-compose. Fichier yaml [docker stack est une commande intégrée à la Docker CLI](#). Il vous permet de gérer un cluster de conteneurs Docker via Docker Swarm

# 7. Virtualisation VS Conteneurisation

## 1- Conteneurisation :

La conteneurisation, à ne pas confondre avec la virtualisation est une méthode permettant d'exécuter une application dans un environnement virtuel dans une seule zone appelée Conteneur. Seul l'environnement d'exécution est virtualisé dans un conteneur (systèmes de fichiers, réseau, processeur, mémoire vive, ...). L'application exécuter dans cet environnement virtuel stocke tous les fichiers, bibliothèque et librairies dans le conteneur .Ensuite notre conteneur se connecte au noyau (kernel) d'un système d'exploitation. Il n'est donc pas nécessaire comme pour les machines virtuelle, d'installer un nouveau système d'exploitation. Ici le noyau gèrent les ressources de l'ordinateur et permet au différents composants matériels et logiciels de communiquer entre eux. La conteneurisation rend donc le déplacement d'application virtuelle plus simple entre des systèmes d'exploitation identiques et demande moins de ressources de mémoire, de RAM, de CPU, etc.

## 2- Virtualisation :

Contrairement à son homologue la virtualisation, chaque VM (machine virtuelle) embarque un OS ce qui implique un éventuel coût en mémoire et en ressources. Et ne parlons pas des éventuels coûts en licence. A la différence d'un conteneur qui se lance à travers l'os, une VM à besoin d'un hyperviseur pour se lancer. Il existe plusieurs moyen de crée des VM, VMware Virtual box pour citer les plus connues.

## 3- Différence entre Conteneurisation et Virtualisation :

Bien sur la conteneurisation à de nombreux avantages par rapport à la virtualisation mais on ne peut pas affirmer que cette technologie est 100% parfaite, elle a aussi sont lots

d'inconvénients.

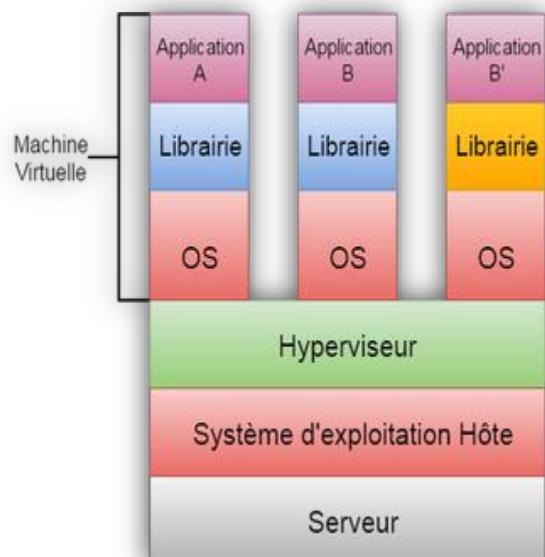


Figure 3: Architecture de virtualisation

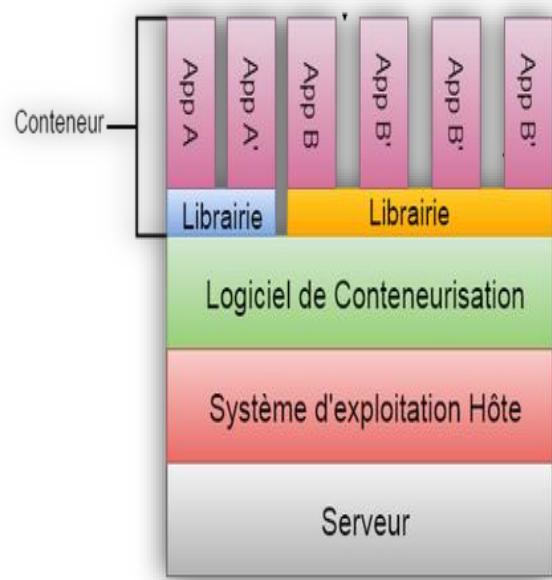


Figure 4: Architecture de Conteneurisation

Ces deux schémas si dessus, d'un environnement de virtualisation à gauche et d'un environnement de conteneurisation à droite permettent de bien identifier les différences entre ces deux technologies.

#### a- Avantage :

Les conteneurs partagent un seul et unique système d'exploitation, de ce fait, l'échange de données entre les conteneurs est plus simple et plus rapides que pour les VM. De plus comme chaque conteneur de contient pas de système d'exploitation propre à lui, les conteneurs sont donc réduits et prennent moins de place et moins de ressource Serveur (environ 10 fois plus petit qu'un VM). Le temps de création et de suppression d'un conteneur est par la même occasion réduit. Les conteneurs facilitent l'évolution technique, par exemple si dans un environnement de VM, l'on souhaite faire évoluer les OS de plusieurs VM, il faut le faire manuellement sur chaque Machines. Ce problème n'est pas présent pour la conteneurisation car toute l'infrastructure repose sur un seul Système d'exploitation.

#### b- Inconvénients :

Mais la conteneurisation peut avoir quelques inconvénients, comme tous les conteneurs ne reposent que sur un seul système d'exploitation, la diversification des systèmes d'exploitation n'est pas possible avec la conteneurisation, « où est plus compliquée à mettre en place ». Les conteneurs sont isolées pour assurer la sécurité et empêcher les malwares de se transmettre entre les conteneurs, mais il est évident que les machines virtuelles seront toujours plus

isolées que les conteneurs. Même si les conteneurs ont un grand nombre d'avantages, leur apparition ne sonne pas la fin de la virtualisation. Aujourd'hui, les machines virtuelles sont intégrées dans de nombreuses entreprises comportant des réseaux de grande taille. Pour utiliser entièrement la technologie de conteneurisation, ces entreprises devraient remanier tous leur système informatique ce qui est impensable. Mais de nouvelles entreprises ont vu le potentiel de la conteneurisation et on donc créer leur système informatique en fonction.

**Remarque :** Docker ne vient pas pour remplacer les machines virtuelles. Dans la pratique on utilise les deux :

- Les machines virtuelles pour virtualiser les machines
- Utiliser Docker pour isoler les environnements d'exécution des applications dans des machines virtuelles

**Conclusion :**

Une architecture à base de conteneurs offre une solution de compromis. Le conteneur offre l'isolation permettant à un développeur d'embarquer l'ensemble des dépendances logicielles dont il a besoin (y compris les dépendances de niveau OS). De plus, un conteneur s'appuie sur le noyau (kernel) du système d'exploitation hôte. Il est donc très léger et démarre presque aussi vite que le processus qu'il encapsule. Le nombre de conteneurs qu'un même hôte peut exécuter est donc nettement plus élevé que son équivalent en machines virtuelles.

## II. Docker networking

Les conteneurs ne sont que des machines virtuelles légères, mais ils ne sont pas si différents en termes de mise en réseau. Les conteneurs et les machines virtuelles ont **besoin d'un réseau** afin de communiquer avec l'extérieur et / ou entre eux.

Le réseau Docker est construit sur le modèle de réseau de conteneurs (CNM), qui permet à quiconque de créer son propre pilote de réseau. Ainsi, les conteneurs ont accès à différents types de réseaux et peuvent se connecter à plusieurs réseaux en même temps.

Pour afficher et obtenir plus de détails sur les réseaux, on exécute la commande suivante :

```
$ docker network ls
```

### 1. Différents types de réseau

Les réseaux peuvent être configurés pour fournir une isolation complète des conteneurs, ce

qui permet de créer des applications Web qui fonctionnent ensemble en toute sécurité.

Docker dispose de 4 pilotes intégrés :

### 1- Bridge :

Conteneurs exécutés par défaut sur ce réseau. La communication est établie via l'interface de pont sur l'hôte.

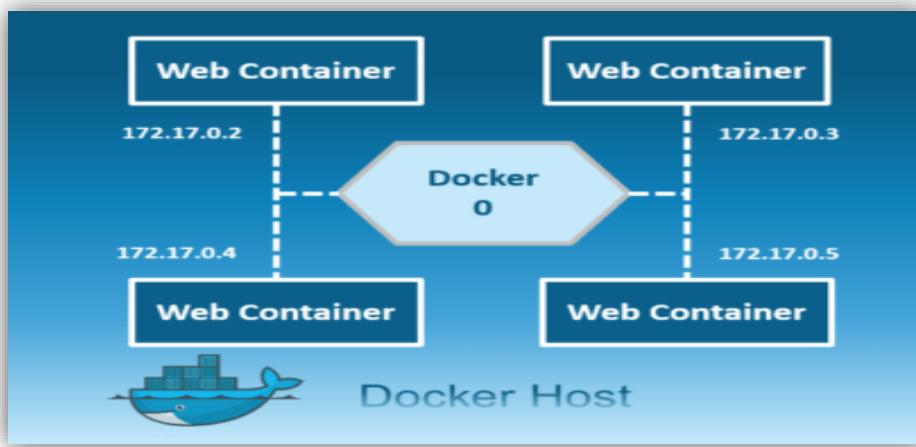


Figure5:Réseau Bridge

Les conteneurs qui utilisent le même réseau ont leur propre sous-réseau et ils peuvent se transmettre des données par défaut.

**Pour créer un réseau bridge (par défaut)** : on exécute la commande suivante

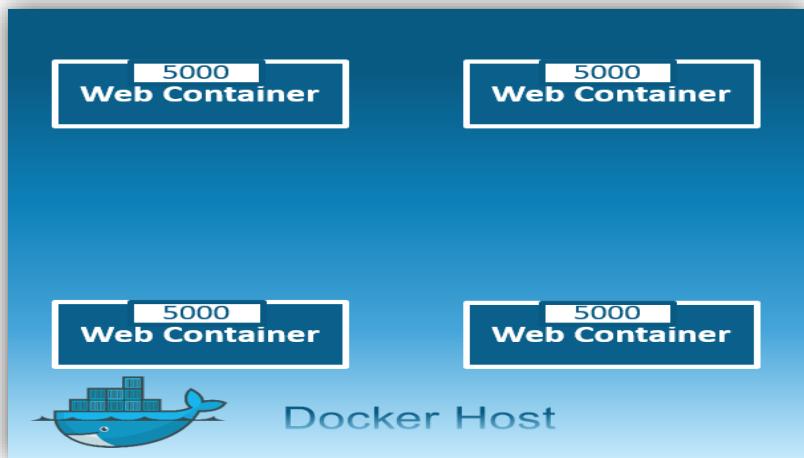
```
$docker network create [Network_name]
```

**Pour créer un réseau Bridge personnalisé** : on exécute la commande suivante

```
$docker network create --driver bridge bridge_personnalise
```

### 2- Host :

Ce pilote donne au conteneur l'accès à son propre espace hôte (le conteneur verra et utilisera la même interface que l'hôte).



*Figure6:Réseau Host*

Par exemple, si vous deviez exécuter un serveur Web sur le port 5000 dans un conteneur d'application Web attaché au réseau hôte, il est automatiquement accessible sur le même port en externe, sans nécessiter de publier le port à l'aide de l'option -p. En tant que conteneur Web utilisant le réseau host, cela signifierait que, contrairement à auparavant, vous ne pourrez plus exécuter plusieurs conteneurs Web sur le même hôte et le même port, car ceux-ci sont désormais communs à tous les conteneurs du réseau hôte.

**Nous ne pouvons pas créer un network host, car nous utilisons l'interface de notre machine hôte.**

### 3- Null :

Sur un réseau de ce type, les conteneurs ne sont connectés à aucun réseau et n'ont pas accès à un réseau externe ou à d'autres conteneurs.



*Figure7: Réseau Null*

Ainsi, ce réseau est utilisé lorsque vous souhaitez désactiver complètement la pile réseau dans le conteneur.

**Nous ne pouvons pas créer un network Null.**

#### 4- Macvlan :

Ce pilote donne aux conteneurs un accès direct à l'interface hôte et à la sous-interface (vlan). Il permet également la jonction.

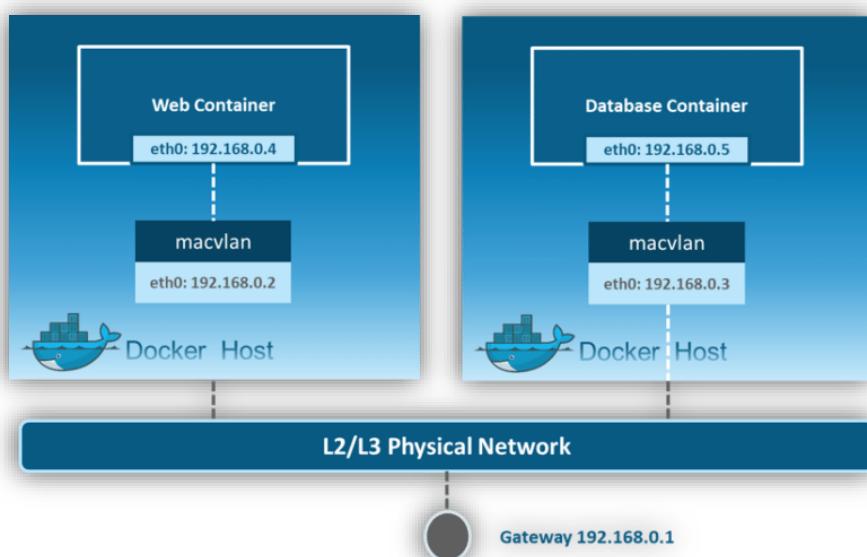


Figure8: Réseau Macvlan

Certaines applications, en particulier les applications héritées ou les applications qui surveillent le trafic réseau, s'attendent à être directement connectées au réseau physique. Dans ce type de situation, vous pouvez utiliser le macvlan pilote réseau pour attribuer une adresse MAC à l'interface réseau virtuelle de chaque conteneur, ce qui donne l'impression qu'il s'agit d'une interface réseau physique directement connectée au réseau physique. Dans ce cas, vous devez désigner une interface physique sur votre hôte Docker à utiliser pour le macvlan, ainsi que le sous-réseau et la passerelle du macvlan. Vous pouvez même isoler vos macvlan réseaux à l'aide de différentes interfaces réseau physiques.

**Pour créer un réseau Macvlan :** on exécute la commande suivante

```
$ docker network create --driver macvlan my_macvlan
```

#### 5- Overlay :

Ce pilote est un réseau qui fonctionne au niveau du cluster plutôt qu'au niveau du système

d'exploitation ce pilote nous permet de créer des réseaux sur plusieurs hôtes avec Docker (généralement sur un cluster Docker Swarm). Les conteneurs ont également leurs propres adresses de réseau et de sous-réseau, et ils peuvent communiquer directement, même s'ils sont physiquement situés sur des hôtes différents. Vous pouvez créer un nouveau réseau de type overlay, qui créera un réseau privé interne qui s'étend sur tous les nœuds participant au cluster swarm.

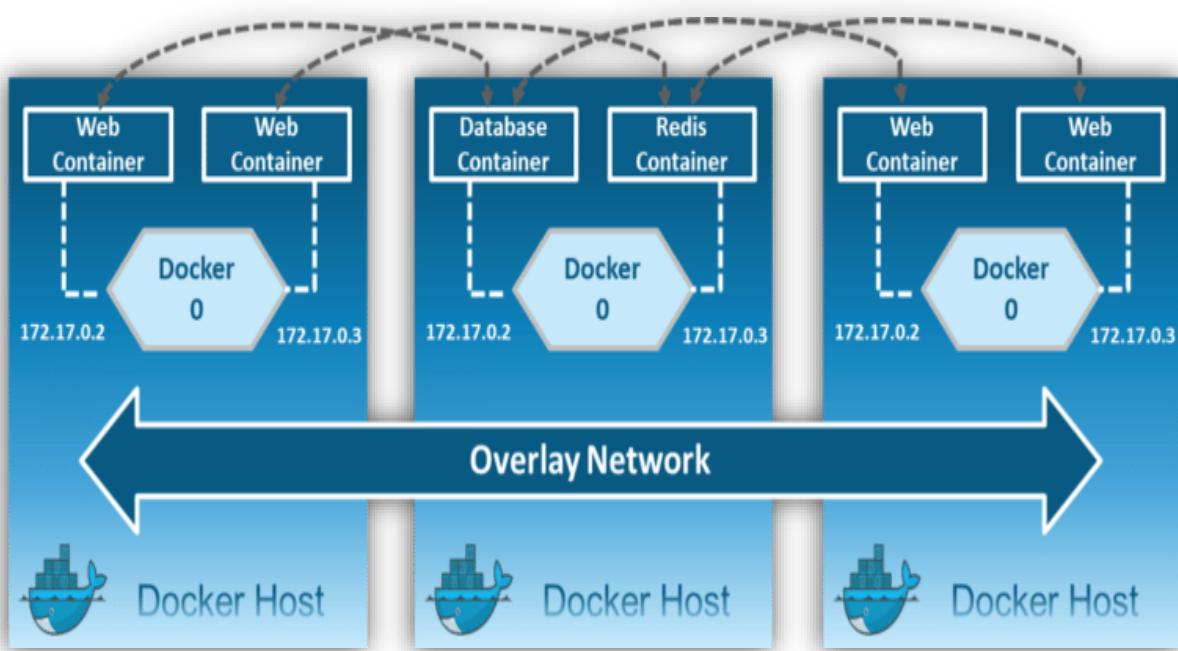


Figure9:Réseau Overlay

Vous pouvez créer des overlay réseaux définis par l'utilisateur à l'aide de la même manière que vous pouvez créer des bridges réseaux définis par l'utilisateur. Les services ou conteneurs peuvent être connectés à plusieurs réseaux à la fois. Les services ou conteneurs ne peuvent communiquer que sur les réseaux auxquels ils sont chacun connectés.

Bien que vous puissiez connecter à la fois des services Swarm et des conteneurs autonomes à un réseau de superposition, les comportements par défaut et les problèmes de configuration sont différents.

**Pour créer un réseau Overlay :** on exécute la commande suivante :

```
$ docker network create -d overlay my-overlay
```

Lorsque vous initialisez un swarm ou joignez un hôte Docker à un swarm existant, deux nouveaux réseaux sont créés sur cet hôte Docker:

Un réseau de superposition appelé ingress, qui gère le contrôle et le trafic de données liés aux services de swarm. Lorsque vous créez un service Swarm et ne le connectez pas à un réseau de superposition défini par l'utilisateur, il se connecte à l'ingress réseau par défaut.

Un réseau Bridge appelé **docker\_gwbridge**, qui connecte le démon Docker individuel aux autres démons participant à Swarm.

**Remarque :** Les pilotes de réseau Bridge et Overlay sont probablement les plus utilisés

## 2. Réseau Single-Host

Les hôtes Docker, par défaut, donnent à chaque conteneur une adresse IP sur une plage privée inutilisée, permettant aux conteneurs sur le même hôte de communiquer entre eux, étant donné la connaissance des adresses IP attribuées aux conteneurs et sur leurs ports exposés.

La fonctionnalité de conteneurs liés de Docker simplifie la communication entre les conteneurs s'exécutant sur le même hôte en permettant aux conteneurs de se référencer via leurs noms, plutôt que via des valeurs réseau qui peuvent changer à mesure que les conteneurs s'arrêtent et redémarrent.

Les conteneurs Docker peuvent également communiquer avec des conteneurs et des applications externes via la redirection de port qui connecte des ports de conteneur spécifiques à des ports attribués de manière statique ou dynamique sur la machine hôte.

Cependant, la liaison de conteneurs ne couvre pas plusieurs hôtes Docker et il est difficile pour les applications s'exécutant à l'intérieur de conteneurs de publier leur IP et leur port externes, car ces informations ne leur sont pas disponibles.

- **Communication entre conteneurs dans un même réseau**

### Pour le réseau bridge par défaut :

1- On lance deux conteneurs qu'ils vont être connectés par défaut au réseau bridge en utilisant la commande **docker run**

2- On connecte entre eux en utilisant l'adresse IP et le nom du conteneur :

- On se connecte à l'un des conteneurs en utilisant la commande docker exec
- On ping sur l'adresse IP de l'autre conteneur ou le nom de l'autre conteneur

On remarque donc que la connexion via le nom échoue !

**Conclusion :** Dans le réseau de pont Docker par défaut (qui s'installe avec le démon Docker), la résolution DNS automatique est désactivée pour maintenir l'isolement du conteneur.

### **Pour le bridge personnalisé :**

- 1- On crée un réseau bridge personnalisé
- 2- On lance deux conteneurs qu'ils vont être connectés à ce réseau
- 3- On connecte entre eux en utilisant l'adresse IP et le nom du conteneur :  
On remarque que la connexion cette fois via le nom réussie!

**Conclusion :** Il est donc toujours conseillé d'utiliser des réseaux personnalisés par l'utilisateur plutôt que d'utiliser des réseaux dockers par défaut

**Remarque :** si on essaye de connecter deux conteneurs existants dans deux réseaux différents la connexion échoue, il est nécessaire qu'ils seront dans le même réseau.

### **3. Réseau Multi-Host :**

Dans la partie précédente on a utilisé Docker sur un seul hôte. Cependant, si la capacité d'un hôte n'est pas suffisante pour supporter la charge de travail, on doit soit acheter une boîte plus grande ou ajouter plus de machines du même type.

Dans cette dernière solution nous nous retrouvons avec un réseau de machine (**Cluster**). Maintenant, un certain nombre de questions se posent :

## **III. Conclusion**

Afin d'automatiser les déploiements, la gestion, la mise à l'échelle et la mise en réseau des conteneurs, et pour pouvoir déployer et gérer des milliers des conteneurs on a besoin d'un outil, c'est l'orchestrateur des conteneurs cependant beaucoup de question se posent :

**Comment les conteneurs communiquent-ils entre eux sur différents hôtes ?**

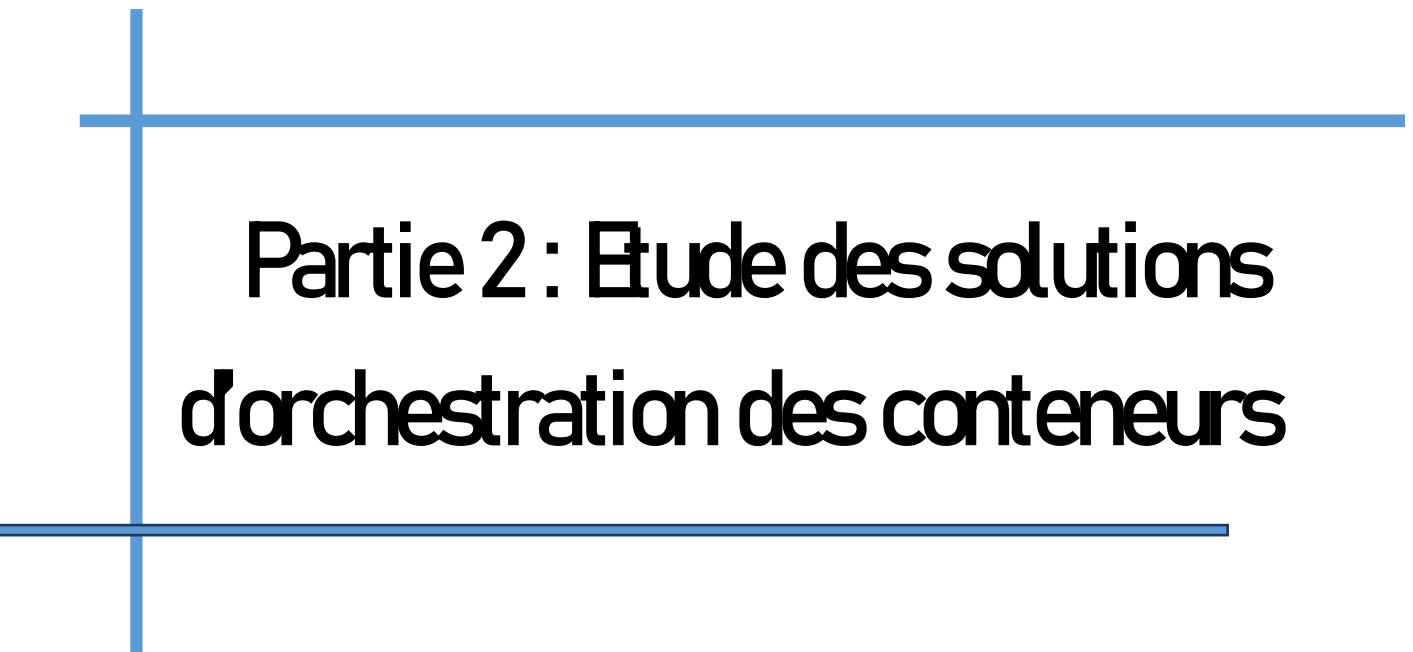
**Comment contrôler la communication entre les conteneurs et entre le monde extérieur ?**

**Comment garder l'état, comme les attributions d'adresses IP, cohérent dans un cluster ?**

**C'est quoi un orchestrateur ?**

**Quelles sont les solutions d'orchestration possible ? Quelle est la différence entre ces solutions ?**

**Afin de répondre à ces questions et d'autres, nous passerons à la deuxième partie.**



## **Partie 2 : Etude des solutions d'orchestration des conteneurs**

## I. Introduction :

Les conteneurs Docker constituent une méthode de virtualisation légère au niveau du système d'exploitation qui permet de lancer une application et ses dépendances à travers un ensemble de processus isolés du reste du système. Cette méthode permet d'assurer le déploiement rapide et stable des applications dans n'importe quel environnement informatique. En plein essor depuis quelques années, les conteneurs Docker ont modifié la façon dont nous développons, déployons et maintenons des logiciels. Leur légèreté et flexibilité ont favorisé l'apparition de nouvelles formes d'architectures qui organisent les applications au sein de conteneurs Docker, prêts à être déployés sur un cluster (groupe) de machines virtuelles ou physiques. Toutefois, cette nouvelle approche requiert des nouveaux outils d'orchestration de conteneurs Docker.

Manipuler quelques conteneurs Docker sur une seule machine est une tâche facile. Lorsqu'il s'agit de faire passer ces conteneurs en production (sur un ensemble d'hôtes distribués), de nombreuses questions se posent :

- Comment gérer les déploiements et leurs emplacements?
- Comment gérer l'équilibrage des charges?
- Comment gérer la communication entre les conteneurs ?
- Comment gérer la découverte de services?
- Comment gérer les mises à jour ?
- Comment gérer la montée en échelle ?
- Comment gérer le stockage nécessaire à la persistance des données ?
- Comment gérer la configuration et les secrets?

## II. Orchestration des conteneurs :

L'orchestration de conteneurs automatise le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs. Les entreprises qui ont besoin de déployer et de gérer des centaines ou des milliers de conteneurs et hôtes Linux peuvent bénéficier de l'orchestration de conteneurs.

L'orchestration de conteneurs peut être utilisée dans n'importe quel environnement où vous utilisez des conteneurs. Il peut vous aider à déployer la même application dans différents environnements sans avoir besoin de la repenser. Et les microservices dans des conteneurs facilitent l'orchestration des services, y compris le stockage, la mise en réseau et la sécurité.

## **1. Propriétés :**

L'orchestration de conteneurs est utilisée pour automatiser et gérer des tâches telles que:

- Provisionnement et déploiement
- Configuration et ordonnancement
- Affectation des ressources
- Disponibilité des conteneurs
- Mise à l'échelle ou suppression de conteneurs en fonction de l'équilibrage des charges de travail dans votre infrastructure
- Équilibrage de charge et routage du trafic
- Surveillance de la santé des conteneurs
- Configuration des applications en fonction du conteneur dans lequel elles s'exécuteront
- Protéger les interactions entre les conteneurs

## **2. Outils d'Orchestration des conteneurs :**

Les outils d'orchestration de conteneurs fournissent un cadre de gestion à grande échelle de l'architecture des conteneurs et des microservices. Il existe de nombreux outils d'orchestration de conteneurs qui peuvent être utilisés pour la gestion du cycle de vie des conteneurs. Certaines options populaires sont Kubernetes, Docker Swarm et Apache Mesos.

### **a- Kubernetes**

Kubernetes souvent abrégé k8s, "k + 8 caractères + s", écrit en Go (aussi appelé Golang est un langage de programmation open source relativement jeune, développé 2007 par Robert Griesemer, Rob Pike et Ken Thompson, travaillant aujourd'hui chez Google), est un projet initié par Google en 2014 quand il a perçu l'avantage des conteneurs Docker par rapport à la virtualisation traditionnelle. Il s'agit d'une innovation après plus d'une décennie d'expérience avec Borg son premier gestionnaire de conteneurs. L'orchestrateur Kubernetes automatise le déploiement et la gestion d'applications conteneurisées à grande échelle. La plateforme K8s permet d'exécuter et de coordonner des conteneurs sur un ensemble de machines physiques et/ou virtuelles. Elle est conçue pour gérer entièrement le cycle de vie des applications conteneurisées en utilisant des méthodes de prédictibilité, d'extensibilité et de haute disponibilité

De nos jours le projet n'appartient plus vraiment à Google, car Google a fait don du projet Kubernetes en 2015 à la toute récente Cloud Native Computing Foundation.

## b- Docker Swarm

Swarm est le premier gestionnaire de conteneurs Docker, lancé par la société Docker en 2014. Il est devenu une fonctionnalité native et intégré au démon Docker depuis la version 1.12 en 2016. L'activation du mode Swarm peut pallier les lacunes de Docker en termes de déploiement, d'exploitation et de gestion sur plusieurs hôtes (Naik, 2016).

Docker Swarm légèrement moins extensible et complexe que Kubernetes, c'est un bon choix pour les amateurs de Docker qui veulent un chemin plus facile et plus rapide vers les déploiements de conteneurs.

## c- Apache Mesos (et Marathon) :

Apache Mesos , légèrement plus âgé que Kubernetes, est un projet de logiciel open source développé à l'origine à l'Université de Californie à Berkeley , mais maintenant largement adopté dans des organisations comme Twitter, Uber et Paypal .

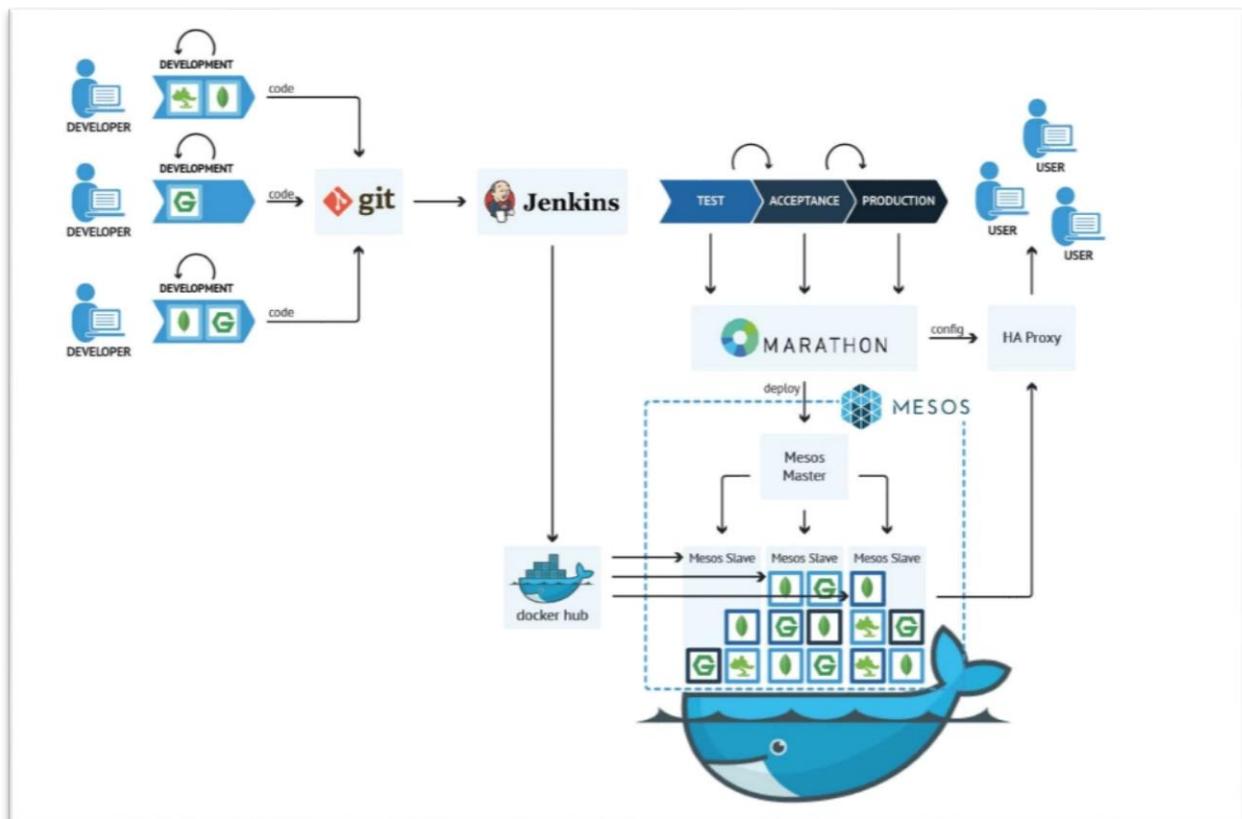


Figure10:Architecture Mesos

L'interface légère de Mesos lui permet d'évoluer facilement jusqu'à 10 000 nœuds (ou plus) et permet aux frameworks qui s'exécutent dessus d'évoluer indépendamment. Ses API prennent en charge les langages populaires comme Java, C ++ et Python, et il prend également en

charge la haute disponibilité prête à l'emploi. Contrairement à Swarm ou Kubernetes, cependant, Mesos ne fournit que la gestion du cluster, de sorte qu'un certain nombre de cadres ont été construits au-dessus de Mesos, y compris Marathon , une plate-forme d'orchestration de conteneurs de «production-grade».

Nous s'intéressons maintenant à Docker swarm et Kubernetes, et nous effectuons une étude comparative des deux solutions

### III. Docker Swarm

#### 1. Présentation :

##### 1- Swarm :

Swarm est un outil d'orchestration natif de Docker (2014). Il est autonome à partir du moteur Docker et sert à connecter les moteurs Docker ensemble pour former un cluster. Il est alors possible de se connecter à Swarm et d'exécuter des conteneurs sur le cluster. Swarm a quelques caractéristiques:

Jusqu'à présent, vous utilisiez Docker en mode hôte unique sur votre ordinateur local. Mais Docker peut également être basculé en mode swarm permettant ainsi l'utilisation des commandes liées au Swarm. L'activation du mode Swarm sur hôte Docker fait instantanément de la machine actuelle un manager Swarm. À partir de ce moment, Docker exécute les commandes que vous exécutez sur le Swarm que vous gérez, plutôt que sur la seule machine en cours. Voici quelques fonctionnalités :

- Permet de spécifier un service de découverte
- Un certain contrôle sur l'emplacement des conteneurs (à l'aide de filtres / contraintes / stratégies de distribution, etc...)
- Expose la même API que le moteur Docker lui-même, permettant aux outils tiers d'interagir de manière transparente

##### 2- Swarmkit :

Swarmkit est un nouvel outil développé en 2016 par l'équipe Docker qui fournit des fonctionnalités pour l'exécution d'un cluster et la distribution de tâches aux machines du cluster. Voici les principales caractéristiques:

- **Distributed**: SwarmKit utilise l'algorithme de consensus Raft pour coordonner et ne s'appuie pas sur un seul point de non-respect des décisions.

- **Secure** : la communication et l'appartenance aux nœuds d'un essaim sont sécurisés hors de la boîte. SwarmKit utilise tls mutuelle pour l'authentification des nœuds, l'autorisation de rôle et le chiffrement de transport, automatisant à la fois l'émission de certificats et la rotation.
- **Simple** : SwarmKit est simple sur le plan opérationnel et minimise les dépendances d'infrastructure. Il n'a pas besoin d'une base de données externe pour fonctionner.

Les machines exécutant SwarmKit peuvent être regroupées afin de former un Swarm, en coordonnant les tâches les unes avec les autres. Une fois qu'une machine se joint, elle devient un nœud de swarm. Les nœuds peuvent être des nœuds de travail ou des nœuds de gestionnaire.

- Les nœuds **Workers** sont responsables de l'exécution des tâches à l'aide d'un exécuteur. SwarmKit est livré avec un exécuteur de conteneur Docker par défaut qui peut être facilement remplacé.
- Les nœuds **Managers** acceptent les spécifications de l'utilisateur et sont responsables de la réconciliation de l'état souhaité avec l'état réel du cluster.

Un opérateur peut mettre à jour dynamiquement le rôle d'un nœud en passant du Worker en Manager ou l'inverse.

### 3- Swarm VS Swarm mode VS Swarmkit

- Le tableau suivant compare Swarm et Swarm Mode :

Swarmkit	Swarm Mode
Projet d'opensource	Swarmkit utilisé en mode swarm et étroitement intégré au moteur Docker
Swarmkit doit être construit et exécuté séparément	Docker 1.12 est livré avec le mode swarm
Pas de service discovery, pas d'équilibrage de charge et de maillage de routage	Service discovery, équilibrage de la charge et maillage de routage disponibles
Utiliser swarmctl CLI	Utiliser docker CLI régulier

Tableau 4:Swarm & Swarm Mode

- Le tableau suivant compare Swarmkit et Swarm Mode :

swarm	Swarm mode
Séparer de Docker Engine et peut fonctionner en tant que conteneur	Moteur Docker intégré à l'intérieur
Besoin d'un magasin KV externe comme Consul	Pas besoin de magasin KV externe séparé
Modèle de service non disponible	Le modèle de service est disponible. Cela fournit des fonctionnalités telles que la mise à l'échelle, la mise à jour continue, la découverte de services, l'équilibrage de charge et le maillage de routage
Communication non sécurisée	Le contrôle et le plan de données sont sécurisés
Intégré à la machine et à la composition	Pas encore intégré à la machine et composé à partir de la version 1.12. Sera intégré dans les prochaines versions

Tableau 5: Swarmkit & swarm mode

**Remarque :** Swarmkit a des primitives pour gérer des fonctionnalités d'orchestration comme la gestion des nœuds, la découverte, la sécurité et la planification.

## 2. Cluster Swarm :

### 1- Introduction :

Docker Swarm est un groupe de machines physiques ou virtuelles qui exécutent l'application Docker et qui ont été configurées pour se réunir dans un cluster.

L'outil Docker Swarm est utilisé pour gérer les clusters d'hôtes Docker ou l'orchestration des hôtes Docker.

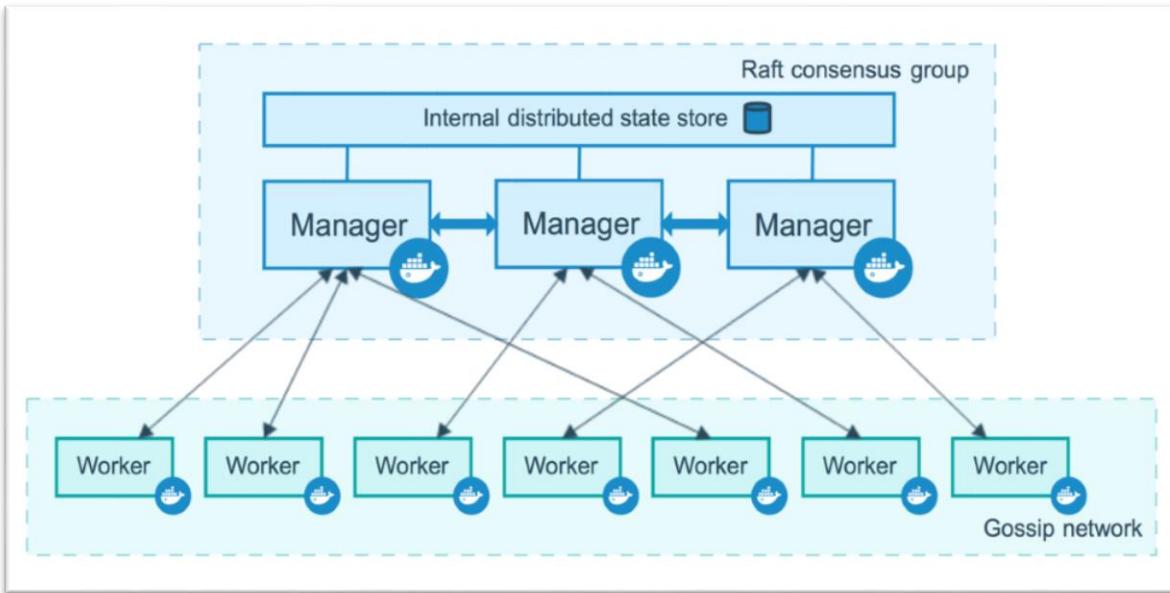


Figure11: Architecture Swarm

Les machines qui ont rejoint le cluster sont appelées nœuds. Il existe deux types de nœuds : gestionnaires (Managers) et travailleurs (Workers).

### a- Manager Node :

Les nœuds Manager distribuent et planifient les tâches entrantes sur les nœuds Worker, maintiennent l'état du cluster et effectuent des fonctions d'orchestration et de gestion de cluster. Les nœuds Manager peuvent également éventuellement exécuter des services pour les nœuds Worker.

Les tâches de gestion de cluster incluent:

- Maintenir l'état du cluster
- Services de planification
- Servir le mode Swarm aux points de terminaison de l'API HTTP

Il devrait toujours y avoir plusieurs nœuds Manager dans votre Swarm pour :

- Maintenir une haute disponibilité
- Récupérez facilement après une panne de nœud Manager sans temps d'arrêt

**Remarque:** Docker recommande un maximum de sept nœuds Manager pour un Swarm.

### b- Leader Node :

Lorsqu'un cluster est établi, l'algorithme de consensus Raft est utilisé pour attribuer l'un d'eux comme «nœud leader». Le nœud leader prend toutes les décisions de gestion et d'orchestration

des tâches pour le Swarm. Si le nœud leader devient indisponible en raison d'une panne, un nouveau nœud leader peut être choisi parmi les autres.

### c- Worker node :

Les nœuds Worker sont également des instances du Docker Engine dont le seul but est d'exécuter des conteneurs et des services conformément aux instructions des nœuds Manager. Pour déployer votre application sur un Swarm, vous avez besoin d'au moins un nœud Manager. Par défaut, chaque Manager est aussi un Worker mais il est possible d'empêcher un Manager d'exécuter des tâches des Workers, en ajustant sa configuration par défaut.

Nous verrons dans la partie suivante comment créer localement un docker swarm cluster à l'aide de "VirtualBox" et de docker-machine. Le docker-machine crée des nœuds virtuels hébergés par le docker qui sont beaucoup plus rapides que l'exécution de machines virtuelles en mode natif sur "VirtualBox".

## 2- docker-machine :

Docker Machine est un outil de provisioning et de gestion des hôtes Docker (hôtes virtuels exécutant le moteur Docker). Vous pouvez utiliser Docker Machine pour créer des hôtes Docker sur votre ordinateur personnel ou sur le datacenter de votre entreprise à l'aide d'un logiciel de virtualisation tel que VirtualBox ou VMWare, vous pouvez aussi déployer vos machines virtuelles chez des fournisseurs de cloud, tels qu'Azure, AWS, Google Compute Engine ...

**Remarque :** Docker machine est installé par défaut dans windows.

### - Implémentation :

Docker machine utilise le concept des drivers (en Fr : pilotes). Les drivers vous permettent depuis votre Docker machine de créer un ensemble complet de ressources sur vos machines virtuelles sur des services tiers tels qu'Azure, Amazon, VirtualBox, etc.

Utiliser docker-machine est la meilleure méthode pour installer Docker sur une machine. Il s'appliquera automatiquement les meilleurs paramètres de sécurité disponibles, y compris la génération d'une paire unique de certificats SSL pour authentification mutuelle et clés SSH.

### Création d'une machine locale :

On exécute la commande suivante :

```
$docker-machine create --drive <DRIVER NAME>< MACHINE NAME>
```

La commande « **docker-machine create** » télécharge une distribution Linux légère nommée boot2docker venant avec le moteur Docker installer et crée et démarre la machine virtuelle.

### 3- Création du cluster swarm:

#### a- Initialisation du swarm :

Dans cette partie, nous allons activer le mode Swarm sur notre Docker machine de la même manière elle deviendra le leader de notre Swarm. Pour cela, nous utiliserons la commande suivante :

```
$docker swarm init [OPTIONS]
```

#### b- Rejoignez le cluster en tant que master :

```
$docker swarm join --token <Token>
```

#### c- Rejoignez le cluster en tant que worker :

1/ dans le nœud manager on lance la commande

```
$docker swarm join-token worker
```

2/ on copie le token dans le nœud worker

#### d- Lister les nœuds :

```
$docker node ls
```

#### e- Commandes associées :

\$ docker swarm ca	Afficher et faire pivoter l'autorité de certification racine
\$ docker swarm init	Initialiser un Swarm

\$ docker swarm join	Rejoignez un swarm en tant que nœud et / ou Manager
\$ docker swarm join-token	Gérer les jetons de jointure
\$ docker swarm leave	Laissez le Swarm
\$ docker swarm unlock	Débloquer le Swarm
\$ docker swarm unlock-key	Gérer la clé de déverrouillage
\$ docker swarm update	Mettre à jour du Swarm

Tableau 6 : Quelques commandes docker swarm

## f- Visualisation du cluster :

Pour visualiser notre cluster on exécute la commande suivante :

```
$docker run -it -d - <port>:8080 -v
>/var/run/docker.sock:/var/run/docker.sockdockersamples/visualizer
```

### a. Services:

Outre les Nœuds, Swarm fonctionne grâce aux services, qui sont des descriptions de l'état qu'on souhaite garder pour les nœuds du cluster. Le concept de service est nouveau. Pour fonctionner, un service a besoin d'un conteneur et de commandes à exécuter sur celui-ci.

Les services exécutés sur Swarm peuvent avoir plusieurs caractéristiques, telles que :

- **Options des services :** lors de la création du service, vous pouvez configurer plusieurs paramètres selon les besoins de vos applications (limites mémoire, le nombre des répliques de l'image à exécuter sur Swarm, etc ...).
- **État désiré :** le déploiement du service permet de définir l'état désiré sur le Swarm. L'état désiré représente le comportement normal ou la configuration idéale de

l'application sur Swarm. Par exemple, lorsqu'un problème survient et met à défaut l'état désiré, les "Manager Node" interviennent pour corriger le problème en affectant plus de ressources au service.

Pour pouvoir communiquer entre les nœuds, les services dans un swarm doivent être sur le même réseau.

- 1- Crédation du network**
- 2- Crédation du service**
- 3- Affichage des informations détaillées**
- 4- Lister les services**
- 5- Suppression du service**
- 6- Scaling automatique & Load Balancing :**

Mise à l'échelle (Scaling) : Désigne la capacité d'un produit à s'adapter à un changement d'ordre de grandeur de la demande (montée en charge), en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande Équilibrage de charge (Load Balancing) : Lorsque des conteneurs sont déployés sur un cluster de serveurs, les équilibreurs de charge s'exécutant dans des conteneurs Docker permettent d'accéder à plusieurs conteneurs sur le même port hôte. Pour mettre à l'échelle le service exécuté ci-dessous, il évoluera dans tout le cluster sur tous les nœuds actifs disponibles.

**Remarque:** Si vous voyez que le service est automatiquement équilibré en charge entre les nœuds actifs, c'est la beauté de l'équilibrer de charge automatique du Swarm.

- 7- Lister les taches de service :** Répertorie les tâches en cours d'exécution dans le cadre des services spécifiés
- 8- Mise à jour des services**

## b. Configuration de DNS dans Docker Swarm:

Le **Domain Name System**, généralement abrégé **DNS**, qu'on peut traduire en « système de noms de domaine », est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements. En fournissant dès les premières années d'Internet, autour de 1985, un service distribué de résolution de noms, le DNS a été un composant essentiel du développement du réseau.

À la demande de la DARPA (Defense Advanced Research Projects Agency, « Agence pour les projets de recherche avancée de défense ») américaine, Jon Postel et Paul Mockapetris ont conçu le « *Domain Name System* » en 1983 et en ont rédigé la première implémentation.

Par défaut, un conteneur hérite des paramètres DNS de l'hôte, tels que définis dans le fichier de configuration/etc/resolv.confj Les conteneurs qui utilisent le bridge réseau par défaut obtiennent une copie de ce fichier, tandis que les conteneurs qui utilisent un réseau personnalisé utilisent le serveur DNS intégré de Docker, qui transfère les recherches DNS externes aux serveurs DNS configurés sur l'hôte.

Les hôtes personnalisés définis dans /etc/hosts ne sont pas hérités. Pour passer des hôtes supplémentaires dans votre conteneur, reportez-vous à l'ajout d'entrées au fichier d'hôtes du conteneur dans la commande docker run.

### c. Interfaces Graphiques:

L'interface graphique désigne la manière dont est présenté un logiciel à l'écran pour l'utilisateur. C'est le positionnement des éléments : menus, boutons, fonctionnalités dans la fenêtre. Une interface graphique bien conçue est ergonomique et intuitive afin que l'utilisateur la comprenne tout de suite.

L'interface graphique est le langage d'échange entre l'humain (vous) et la machine (votre ordinateur). Votre ordinateur affiche à l'écran des éléments que vous comprenez et que vous interprétez.

#### 1- Portainer :

Portainer est une interface de gestion légère qui vous permet de gérer facilement votre hôte Docker ou votre cluster Swarm. Portainer est censé être aussi simple à déployer qu'à utiliser. Il se compose d'un seul conteneur qui peut s'exécuter sur n'importe quel moteur Docker (Docker pour Linux et Docker pour Windows sont pris en charge).Et c'est totalement gratuit.

#### 1. Implémentation :

Portainer a été installé en tant que conteneur Docker.

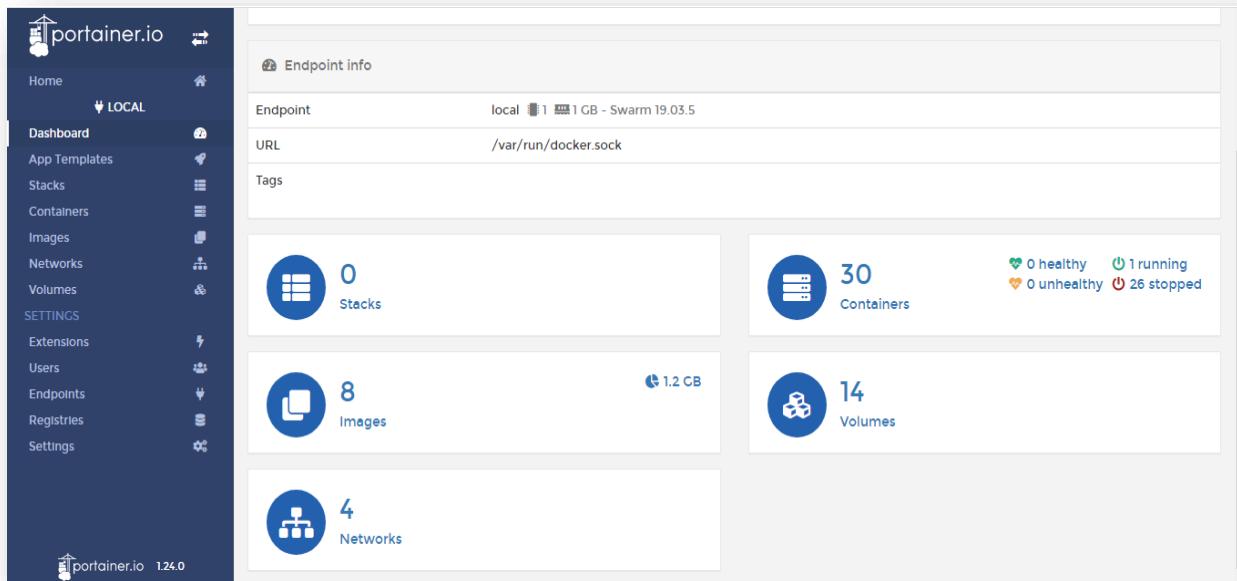


Figure 12:Page principale de Portainer

#### **g- Gestion d'environnement Docker à l'aide de Portainer**

##### **➤ Gestion des conteneurs :**

Portainer fournit un moyen simple et facile d'utiliser la gestion des Docker Containers.

Cliquez sur le menu 'Conteneurs' sur la gauche et vous obtiendrez la page comme ci-dessous.

The screenshot shows the Portainer interface for managing Docker containers. On the left, a sidebar menu includes Home, LOCAL (Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes), SETTINGS (Extensions, Users, Endpoints, Registries, Settings), and a footer for portainer.io 1.24.0. The main area is titled 'Container list' and shows a table of containers:

Name	State	Quick actions	Stack	Image	Created	Published Ports
gifted_agnesi	created		-	portainer/portainer	2020-06-29 15:07:35	-
charming_lichterman	created		-	portainer/portainer	2020-06-29 15:07:24	-
laughing_northcutt	created		-	portainer/portainer	2020-06-29 15:04:16	-
portainer	running		-	portainer/portainer	2020-06-29 15:51:46	8000:8000 9000:9000
pfe2	stopped		-	erichough/nfs-server.latest	2020-07-06 20:20:00	-
pfe	stopped		-	erichough/nfs-server.latest	2020-07-06 19:47:19	-
web.2.u5dcvmrr75iuoyouj9gre4g6c	stopped		-	xataz/nginx.mainline	2020-06-29 17:42:45	-
web.4.rleljakb7e8goubcvokj4fjn	stopped		-	xataz/nginx.mainline	2020-06-29 17:42:45	-
web.5.qpq5gyqdx56c7h7m2tkrkJ5b3	stopped		-	xataz/nginx.mainline	2020-06-29 17:42:45	-
infallible_wilbur	stopped		-	docker/getting-started.pwd	2020-06-29 15:24:21	80:80

Figure 13: Liste des conteneurs dans Portainer

Nous pouvons démarrer, arrêter, redémarrer, créer un nouveau conteneur, accéder à la coque du conteneur, voir le journal du conteneur et les statistiques du conteneur à partir de cette gestion de conteneur Portainer.

Remarque : On peut spécifier « un volume – un network .... » lors de la création d'un conteneur.

### ➤ Gestion des images Docker :

The screenshot shows the Portainer interface for managing Docker images. On the left, a sidebar menu includes Home, LOCAL (Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes), SETTINGS (Extensions, Users, Endpoints, Registries, Settings), and a footer for portainer.io 1.24.0. The main area is titled 'Image list' and shows a table of images:

Id	Tags	Size	Created
sha256:20134a0795dcfc269c68308556fd13...	Unused	13.8 MB	2020-06-18 16:28:22
sha256:a24bb4013296f61e89ba57005a7b3e...	Unused	5.6 MB	2020-05-29 22:19:46
sha256:45328bd05eb96a5e01d481beda33...	alpine:latest	25.8 MB	2019-11-28 18:22:12
sha256:52e16db3eced77f66463d7f356946b...	erichough/nfs-server:latest	15.8 MB	2020-02-07 01:05:06
sha256:7278d88c86fe7ccbe22afa279d2f47...	tsoppela/swarmkit:latest	203.1 MB	2016-07-02 15:15:52

Cliquez sur 'Images' dans le menu et vous obtiendrez la page comme ci-dessous.

Figure 14: Gestion des images dans Portainer

Nous pouvons maintenant voir la liste des images Docker sur notre système, et nous pouvons créer manuellement une nouvelle image Docker, ou extraire / télécharger de nouvelles images à partir du référentiel Docker Hub ou, par créer un fichier Dockerfile.

#### ➤ Gestion des réseaux :

À partir de ce menu, nous créons de nouveaux réseaux personnalisés pour notre

The screenshot shows the Portainer interface for managing Docker networks. On the left is a sidebar with various icons and labels: Home, LOCAL, Dashboard, App Templates, Stacks, Containers, Images, Networks (which is selected and highlighted in blue), Volumes, SETTINGS, Extensions, Users, Endpoints, Registries, and Settings. The main content area is titled "Network list" and shows a table of networks. The table has columns for Name, Stack, Driver, Attachable, IPAM Driver, IPV4 IPAM Subnet, IPV4 IPAM Gateway, IPV6 IPAM Subnet, IPV6 IPAM Gateway, and Ownership. There are four entries:

Name	Stack	Driver	Attachable	IPAM Driver	IPV4 IPAM Subnet	IPV4 IPAM Gateway	IPV6 IPAM Subnet	IPV6 IPAM Gateway	Ownership
bridge	System	bridge	false	default	172.17.0.0/16	172.17.0.1	-	-	public
docker_gwbridge	-	bridge	false	default	172.18.0.0/16	172.18.0.1	-	-	administrators
host	System	host	false	default	-	-	-	-	public
none	System	null	false	default	-	-	-	-	public

At the bottom right of the table, there is a dropdown menu labeled "Items per page" with the value "10".

environnement Docker. Cliquez sur le menu "Réseaux"

Figure 15: Gestion des réseaux dans Portainer

Il y a toutes les informations concernant un réseau « le driver utilisé, l'adresse IP ... »

#### ➤ Gestion des volumes :

Nous avons juste besoin de créer de nouveaux volumes personnalisés, et lorsque nous voulons créer un nouveau conteneur, l'application, il suffit de le joindre au conteneur via le menu « Options avancées ».

Name	Stack	Driver	Mount point	Created	Owners
fae92b883eebe7451f7ee2a2c43db5f2403e0...	-	local	/mnt/sdal/var/lib/docker/[...]9d2172b6bc3b726de54/_data	2020-06-23 12:33:07	admil
mynfs	-	local	/mnt/sdal/var/lib/docker/volumes/mynfs/_data	2020-06-21 16:12:15	admil
pfe	-	local	/mnt/sdal/var/lib/docker/volumes/pfe/_data	2020-06-26 10:58:54	admil
portainer_data	-	local	/mnt/sdal/var/lib/docker/[...]umes/portainer_data/_data	2020-06-29 15:51:47	admil

Figure 16:Gestion des volumes

Dans ce menu on peut créer un volume avec des pilotes, on peut créer notre Point de montage etc....

## 2- Rancher :

Rancher, projet open source créé par la société Rancher Labs est un outil gratuit d'orchestration de conteneurs Docker. Il permet de facilement déployer des conteneurs Docker sur des machines possédant Docker. Grâce à une configuration simple et complète, il permet de lier ses conteneurs afin de composer des architectures de services aisément. Il peut déployer des containers sur des services cloud comme AWS, Azure, DigitalOcean, mais aussi sur des machines personnalisées possédant Docker tout en s'appuyant sur docker-machine.



## a- Contrôle de niveau entreprise :

La figure suivante illustre les principaux composants et fonctionnalités Rancher

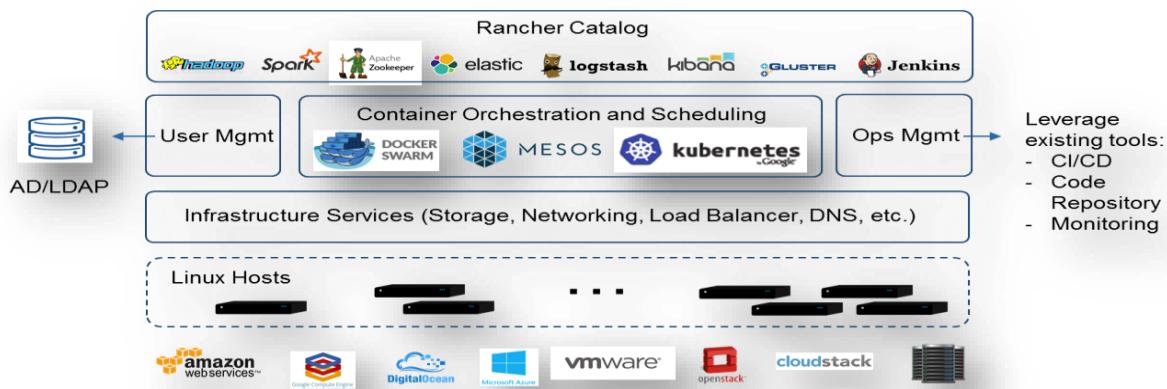


Figure 17:Fonctionnalité de Rancher

Rancher prend en charge les plug-ins d'authentification utilisateur flexibles et est livré avec une intégration d'authentification utilisateur prédéfinie avec Active Directory, LDAP et GitHub. Rancher prend en charge le contrôle d'accès basé sur les rôles (RBAC) au niveau des environnements, permettant aux utilisateurs et aux groupes de partager ou de refuser l'accès, par exemple, aux environnements de développement et de production.

## b- Implémentation :

Après l'installation de Rancher

La capture d'écran montre la interface web de Rancher. Le menu principal en haut comprend des options comme Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN et API. La page actuelle est "Hosts", où l'utilisateur peut ajouter de nouveaux hôtes. La liste des hôtes actifs affiche "default" avec l'IP 192.168.99.111 et l'alias 19.03.12. Des détails supplémentaires sont fournis pour Boot2Docker 19.03.12 (4.19.130), 2 GHz de processeur, 195 GiB de RAM et 17.8 GiB de stockage.

Figure 18:Page principale de Rancher

Rancher permet d'accéder à un shell et une vue des logs, ce qui est très pratique pour le débogage.

## 6. Sécurisation:

La sécurité de Docker est relative aux menaces et attaques qui remettraient en question la sécurité informatique basée sur les services de confidentialité, d'intégrité et de disponibilité. L'article traite des différents types d'attaques, des risques et des solutions apportées afin de sécuriser Docker.

Incontestablement, Docker s'adresse à trois grands profils de clients : les early adopter, les utilisateurs qui découvrent cet environnement et ceux qui ne l'ont pas encore utilisé. Pour autant, quel que soit son niveau de maturité, la sécurité reste toujours un obstacle à franchir

Concrètement, lorsque l'on utilise des « containers », la sécurité traditionnelle ne suffit pas. On ne sécurise pas une infrastructure « containérisée » comme on sécurise son infrastructure virtualisée. Il est donc important de bien comprendre les enjeux de la sécurité dans une infrastructure « containérisée », ainsi que les risques associés pour les intégrer efficacement dans sa politique de sécurité.

### **Pourquoi les containers nécessitent une sécurité différente ?**

#### **➤ Plus de composants :**

Dans une infrastructure virtualisée, les flux sont plus au moins maîtrisés. Dans une infrastructure « containérisée », cela se complique au regard des architectures complexes type « micro services », où le code est éclaté sur différents composants. Pour les équipes sécurité, il faut donc s'adapter à cette multitude de flux.

#### **➤ Automatisation à prendre en compte :**

Avec les containers, les notions d'automatisation, de déploiement massif et de portabilité sont fondamentales. Il sera donc important de contrôler et valider la chaîne d'intégration.

#### **➤ Durée de vie courte des containers :**

En comparaison avec les machines virtuelles, un container a une durée de vie plus courte. Les équipes de développement vont donc constamment recréer des containers. Suivre ces changements est alors un véritable challenge pour les équipes sécurité.

## **Quels changements au niveau des risques ?**

Les risques sont pratiquement identiques à ceux des infrastructures virtualisées ou classiques. Cependant, certains points sont intéressants à préciser :

➤ **Risques sur les configurations par défaut :**

Garder les configurations par défaut amène un risque important dans une infrastructure conteneurisée.

➤ **Le top ten owasp reste valable :**

Avec les containers, on utilise toujours les mêmes protocoles de communication : les attaques applicatives classiques sont donc courantes.

➤ **Brèches au niveau du code :**

C'est un risque classique. Avec un code mal développé ou une mauvaise librairie, les brèches existent dans les containers et les attaques sont possibles.

➤ **Attaque réseau :**

Si un pirate prend la main sur un container, il pourra lancer un scan réseau et faire des déplacements latéraux. Comme il y a un manque de visibilité, les équipements de sécurité ne vont pas détecter ce scan, et le pirate pourra facilement passer d'un container à l'autre.

## **Comment faire pour sécuriser ses opérations sur Docker ?**

Il faut commencer bien évidemment par former l'équipe de sécurité à la culture devops et aux architectures micro-services. Ensuite, voici quelques bonnes pratiques à mettre en place :

- Créer une partition physique séparée pour Docker
- Maintenir le système et les images à jour
- Éviter le stockage de secrets au sein d'une image
- Ne pas utiliser n'importe quel registre
- Contrôler les communications entre containers
- Définir des politiques sécurisées de communication au sein de l'infrastructure
- Gérer les droits d'accès au registre avec un mécanisme de gestion de rôle

- Installer des outils dédiés aux containers pour vérifier le niveau de sécurité global afin d'identifier les vulnérabilités.
- Vérifier les sources de construction des images utilisées par les conteneurs et assurer un suivi de leurs évolutions en appliquant une signature numérique interne

Enfin, l'aspect organisationnel est également incontournable : le passage en mode Devops et l'orchestration du déploiement d'un conteneur ont leur propre mode de fonctionnement et impliquent de nouvelles méthodologies : les équipes développement, opérationnelles et sécurité doivent alors savoir travailler ensemble.

#### **- Les vertus du shift-left :**

L'idée du shift-left est de rapprocher les considérations de qualité et de sécurité des applications du développeur pour que les problèmes potentiels soient résolus avant que le code ne soit déployé. Cela a en effet un impact important sur la sécurité : les équipes sécurité se concentrent alors sur la sécurité au niveau système et peu au niveau de l'application. Ainsi, la sécurité est optimisée dès la conception de l'application.

#### **Méthodes de sécurisation :**

Docker vit par « Sécurisé par défaut ». Avec Docker Enterprise (DE), la configuration et les politiques par défaut fournissent une base solide pour un environnement sécurisé. Cependant, ils peuvent facilement être modifiés pour répondre aux besoins spécifiques de toute organisation.

Il y a plusieurs méthodes pour sécuriser le cluster, parmi ces méthodes :

#### **a- Docker Bench for Security :**

Le Docker Bench for Security est un script qui recherche des dizaines de bonnes pratiques courantes concernant le déploiement de conteneurs Docker en production. Les tests sont tous automatisés et s'inspirent du Benchmark CIS Docker Community Edition v1.1.0.

Docker Bench for Security est un excellent outil de sécurité car il est fabriqué et maintenu par les créateurs de Docker, et il est gratuit.

## b- Accès à distance au Cluster Swarm :

L'administration de cluster à distance est une bonne méthode pour sécuriser votre environnement de travail, le concept ici est de créer un fichier OPENVPN pour accéder à notre cluster à distance avec un mot de passe. Alors on a besoin d'un est un service DNS dynamique et un logiciel ou une application VPN.

- **DuckDNS (dynamic DNS service) :**

DuckDNS est un service DNS dynamique gratuit qui pointera un sous-domaine de [duckdns.org](http://duckdns.org) vers une IP de votre choix. Cette IP peut ensuite être mise à jour à l'aide d'un processus scripté sur une machine cliente.

Nous devrons commencer par aller sur le site Web de DuckDNS et nous connecter en utilisant l'un des fournisseurs OAuth disponibles, Google, GitHub, etc.

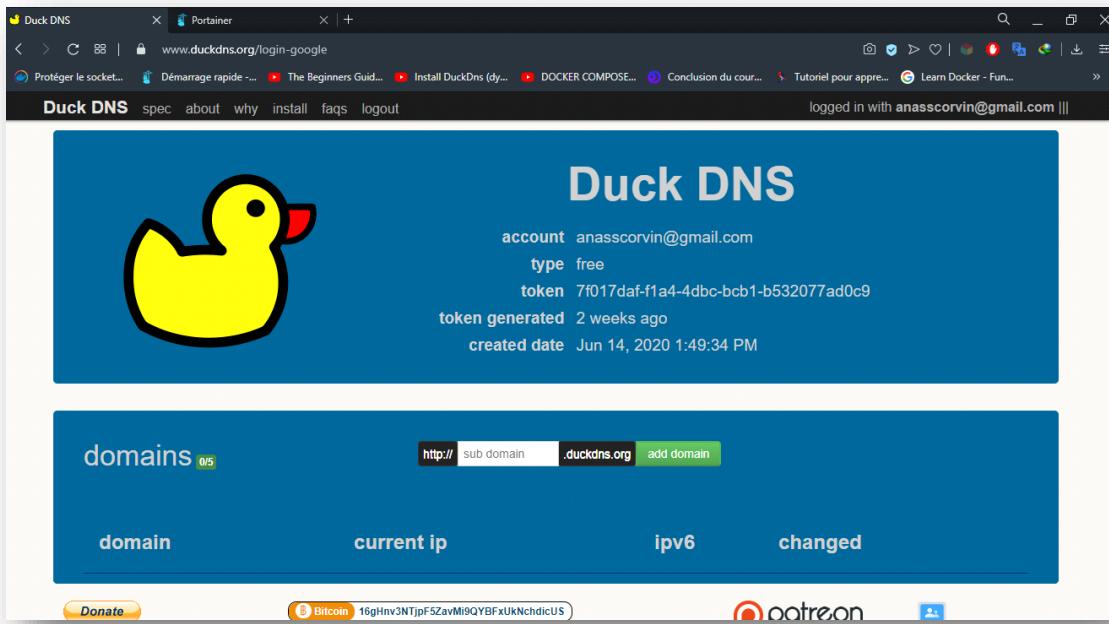


Figure 19:DuckDns

Nous utiliserons l'image du conteneur LinuxServer à nos fins ici. Il s'agit d'une image régulièrement mise à jour et disponible sur le [Docker Hub](#).

- **OpenVPN pour Docker :**

OpenVPN fournit un moyen de créer des réseaux privés virtuels (VPN) en utilisant le cryptage TLS (évolution de SSL). OpenVPN protège le trafic réseau contre les attaques d'espionnage et les attaques de type « homme au milieu » (MITM). Le réseau privé peut être utilisé pour connecter en toute sécurité un périphérique, tel qu'un ordinateur portable ou un téléphone portable fonctionnant sur un réseau Wifi non sécurisé, à un serveur distant qui relaie ensuite le trafic sur Internet. Les réseaux privés peuvent également être utilisés pour connecter en toute sécurité des périphériques entre eux via Internet.

Choisissez un nom pour le conteneur de volume de données \$OVN\_DATA. Il est recommandé d'utiliser le préfixe ovpn-data pour fonctionner de manière transparente avec le service systemd de référence. Les utilisateurs sont encouragés à remplacer l'exemple par un nom descriptif de leur choix.

```
Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ OVPN_DATA="ovpn-data-openvpn"
```

Figure 20:initialisation du nom pour conteneur

Initialisez le conteneur \$ OVN\_DATA qui contiendra les fichiers de configuration et les certificats. Le conteneur demandera une phrase secrète pour protéger la clé privée utilisée par l'autorité de certification nouvellement générée.

```
Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker volume create --name $OVN_DATA
ovpn-data-openvpn

Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker run -v $OVN_DATA:/etc/openvpn --log-driver=none --rm kylemanna/openvpn ovpn_genconfig -u udp://pfevpn.duckdns.org
```

Figure 21:creation du volume et démarrage du conteneur OpenVPN

```

Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm -it kylemanna/openvpn ovpn_initpki

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /etc/openvpn/pki

Using SSL: openssl OpenSSL 1.1.1d 10 Sep 2019

Enter New CA Key Passphrase:
Re-Enter New CA Key Passphrase:
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0xe010001)
Can't load /etc/openvpn/pki/.rnd into RNG
140365624761672:error:2406F079:random number generator:RAND_load_file:Cannot open file:crypto/rand/randfile.c:98:Filename=/etc/openvpn/pki/.rnd
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:SMIPFE

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/etc/openvpn/pki/ca.crt

Using SSL: openssl OpenSSL 1.1.1d 10 Sep 2019
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time

```

*Figure 22:générer un certificat privé:*

## Démarrer le processus du serveur OpenVPN

```

Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker run -v $OVPN_DATA:/etc/openvpn -d -p 1194:1194/udp --cap-add=NET_ADMIN kylemanna/openvpn
26e99743ced70bbc513e9ad74d834992486c171c9d486a1397d4423918d7f4af

```

*Figure 23:Démarrage du serveur OpenVPN*

## Générer un certificat client

```

Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm -it kylemanna/openvpn easyrsa build-client-full SMIPFE

Using SSL: openssl OpenSSL 1.1.1d 10 Sep 2019
Generating a RSA private key
.....+++++
.....+++++
writing new private key to '/etc/openvpn/pki/private/SMIPFE.key.XXXXFLOeIP'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
Using configuration from /etc/openvpn/pki/safessl-easyrsa.cnf
Enter pass phrase for /etc/openvpn/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName      :ASN.1 12:'SMIPFE'
Certificate is to be certified until Jun 20 17:43:27 2023 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated

```

*Figure 24:Générer un certificat client*

Récupérer la configuration client avec des certificats intégrés

```
Corvin@DESKTOP-DT6RC7D MINGW64 /c/Program Files/Docker Toolbox
$ docker run -v $OVPN_DATA:/etc/openvpn --log-driver=none --rm kylemanna/openvpn ovpn_getclient SMIPPE > SMIPPE.ovpn
```

Figure 25:Récupération de la configuration client

### c- Network File System NFS :

Le **NFS**, également connu sous le nom de *Network File Service* est un protocole développé par Sun Microsystems qui permet d'accéder aux fichiers via un réseau. Les fichiers ne sont pas comme B. avec FTP, mais les utilisateurs peuvent accéder aux fichiers qui se trouvent sur un ordinateur distant comme s'ils étaient stockés sur leur disque dur local.

Dans ce Unix protocole de réseau est un Internet - norme (RFC 1094 , RFC 1813 , RFC 3530 , RFC 7530 ), également connu sous le nom distribué système de fichiers (anglais système de fichiers distribué). L'équivalent de NFS est appelé Server Message Block (SMB) dans les environnements Windows et OS / 2 . Alors que l'utilisateur s'authentifie avec SMB, le NFSv3 le plus populaire authentifie l'ordinateur client, seul NFSv4 permet l'authentification de l'utilisateur. Les services NFS sont également sur Microsoft Windows - serveurs , rendant UNIX disponibles les postes de travail d' accès peuvent obtenir leurs fichiers, mais SMB est dans des environnements mixtes généralement avec Samba utilisé sur le côté Unix.

NFS a initialement travaillé sur le protocole réseau IP avec UDP sans état . En attendant, il existe également NFS sur TCP . NFSv4 ne fonctionne qu'avec TCP et n'a besoin que d'un seul port (2049), ce qui facilite l'utilisation des pare-feu . NFSv4 a été développé de manière significative par l'IETF après que Sun a abandonné le développement.

- **Configuration dans les systèmes Windows :**

La première chose que nous devons faire est d'installer le client NFS, ce qui peut être fait en suivant les étapes ci-dessous:

**Étape 1:** Ouvrez Programmes et fonctionnalités.

**Étape 2:** cliquez sur « Activer ou désactiver les fonctionnalités Windows »

**Étape 3:** Cochez l'option **Services pour NFS**, puis cliquez sur **OK**.

*Maintenant on peut travailler avec le service NFS dans Docker swarm*

- **NFS dans Docker swarm :**

On peut créer le volume avec le service NFS par ligne de commande ou bien dans PORTAINER

```
zahra@DESKTOP-IUBF00M MINGW64 /c/Program Files/Docker Toolbox
$ docker volume create --opt type=nfs --opt device=:/myNFSShare --name myNFSShare
myNFSShare

zahra@DESKTOP-IUBF00M MINGW64 /c/Program Files/Docker Toolbox
$ docker run -it --name myImage -v myNFSShare:/var/nfsmount -e LICENSE=accept --entrypoint=/bin/bash ubuntu -i
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
a4a2a29f9ba4: Pull complete
127c9761dcba: Pull complete
d13bf203e905: Pull complete
4039240d2e0b: Pull complete
Digest: sha256:35c4a2c15539c6c1e4e5fa4e554dac323ad0107d8eb5c582d6ff386b383b7dce
Status: Downloaded newer image for ubuntu:latest
```

Figure 26:Creation du volume avec NFS

The screenshot shows the Portainer.io interface with a sidebar on the left containing navigation links like Home, LOCAL, Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes, SETTINGS, Extensions, Users, Endpoints, Registries, and Settings. The main content area is titled 'Volume details' under 'Volumes > myNFSShare'. It contains the following information:

Volume details	
ID	myNFSShare <a href="#">Remove this volume</a>
Created	2020-07-09 19:43:08
Mount path	/mnt/sda1/var/lib/docker/volumes/myNFSShare/_data
Driver	local
<b>Access control</b>	
Ownership	<a href="#">administrators</a>
<a href="#">Change ownership</a>	
<b>Volume options</b>	
device	./myNFSShare
type	nfs

Figure 27:NFS dans Portainer

Maintenant on peut clairement voir que le volume NFS a été bien créé dans notre service.

## d- Protéger le socket du démon Docker

Par défaut, Docker s'exécute via un socket UNIX non en réseau. Il peut également éventuellement communiquer via un socket HTTP.

Si vous avez besoin que Docker soit accessible via le réseau de manière sûre, vous pouvez activer TLS en spécifiant `tlsverify`indicateur et en pointant `tlscacert`indicateur de Docker vers un certificat d'autorité de certification de confiance.

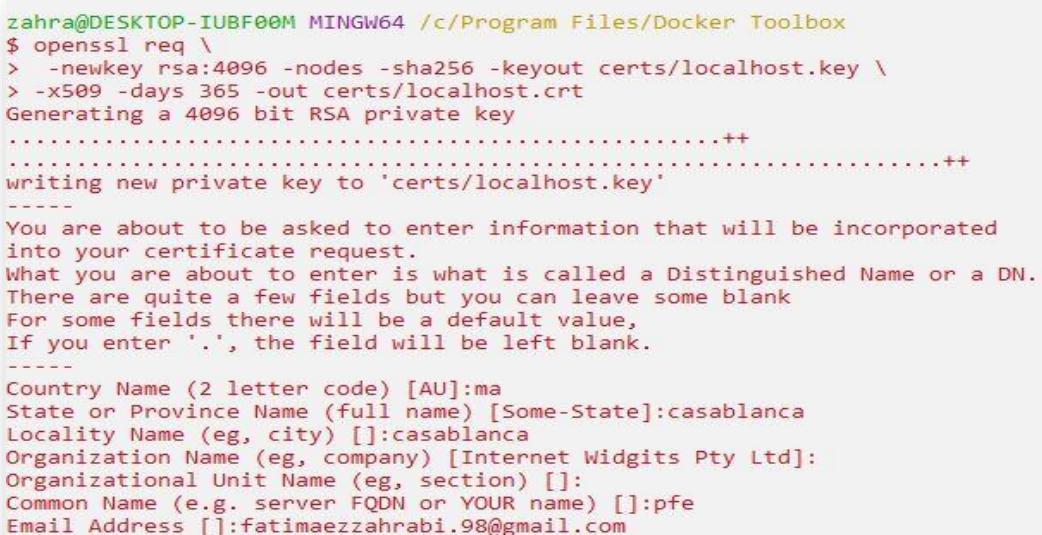
En mode démon, il autorise uniquement les connexions des clients authentifiés par un certificat signé par cette autorité de certification. En mode client, il se connecte uniquement aux serveurs avec un certificat signé par cette autorité de certification.

- **Implémentation :**

Maintenant que vous disposez d'une autorité de certification, vous pouvez créer une clé de serveur et une demande de signature de certificat (CSR). Assurez-vous que « Common Name » correspond au nom d'hôte que vous utilisez pour vous connecter à Docker.

Étant donné que les connexions TLS peuvent être établies via l'adresse IP ainsi que le nom DNS, les adresses IP doivent être spécifiées lors de la création du certificat. Docker sur TLS doit s'exécuter sur le port TCP 2376.

Un certificat personnalisé est configuré en créant un répertoire sous en /etc/docker/certs.dutilisant le même nom que le nom d'hôte du registre, tel que localhost. Tous les \*.crtfichiers sont ajoutés à ce répertoire en tant que racines de l'autorité de certification.



```
zahra@DESKTOP-IUBF00M MINGW64 /c/Program Files/Docker Toolbox
$ openssl req \
> -newkey rsa:4096 -nodes -sha256 -keyout certs/localhost.key \
> -x509 -days 365 -out certs/localhost.crt
Generating a 4096 bit RSA private key
.....+
.....+
writing new private key to 'certs/localhost.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ma
State or Province Name (full name) [Some-State]:casablanca
Locality Name (eg, city) []:casablanca
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:pfe
Email Address []:fatimaezzahrabi.98@gmail.com
```

Figure 28:Génère une clé RSA

Utilisez OpenSSL commandes pour générer d'abord une clé RSA, puis utilisez la clé pour créer le certificat.

## I. Kubernetes

### 1. Composants & Objets Kubernetes

Lorsque qu'on travaille avec Kubernetes (K8S), de nombreuses nouvelles terminologies sont à apprendre, à comprendre et à mémoriser. Cette partie contiendra des explications sur tous les objets et composants nécessaires à la compréhension de Kubernetes.

#### 1- Objets Kubernetes

Kubernetes a pour objectif principal de dissimuler la complexité de la gestion d'un parc de conteneurs, et pour cela, différents objets existent pour faciliter la vie :

Node	Un node ("nœud" en Fr) est une machine de travail du cluster Kubernetes. Cesont des unités de travail qui peuvent être physiques, virtuelles mais aussi des instances cloud.
Pod	Il s'agit de l'unité la plus petite de K8s, un pod encapsule le ou les conteneur(s) formant votre application conteneurisée partageant ainsi la même stack réseau (chaque pod se voit attribuer une adresse IP unique) et le même stockage, plus précisément un volume partagé
Replicas	c'est le nombre d'instances d'un Pod ("réplique" en Fr)
ReplicaSet	s'assure que les répliques spécifiés sont actifs
Deployment	défini l'état désiré et fournit des mises à jour déclaratives de vos Pods ReplicaSets.
Service	Un service peut être défini comme un ensemble logique de pods exposés en tant que service réseau. C'est un niveau d'abstraction au-dessus du pod, qui fournit une adresse IP et un nom DNS unique pour un ensemble de pods. Avec les Services, il est très facile de gérer la configuration de Load Balancing (équilibrage de charge) permettant ainsi aux pods de scaler plus facilement.
Endpoint	Représente l'adresse IP et le port d'un service, il est automatiquement créé lors de la création d'un service avec les pods correspondants.

Tableau 7:Objets de kubernetes

## 2- Composants Kubernetes

Kubernetes suit l'architecture **maître-esclave** :

Maître	plus communément appelé master existe principalement pour gérer votre cluster Kubernetes
Esclaves	sont quant à eux plus connus sous le nom de workers (on les appelle aussi minions) et ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre node ce qu'il peut ou ne peut pas faire.

Tableau 8:Composants de kubernetes

Les composants clés du master et worker (figure 15) :

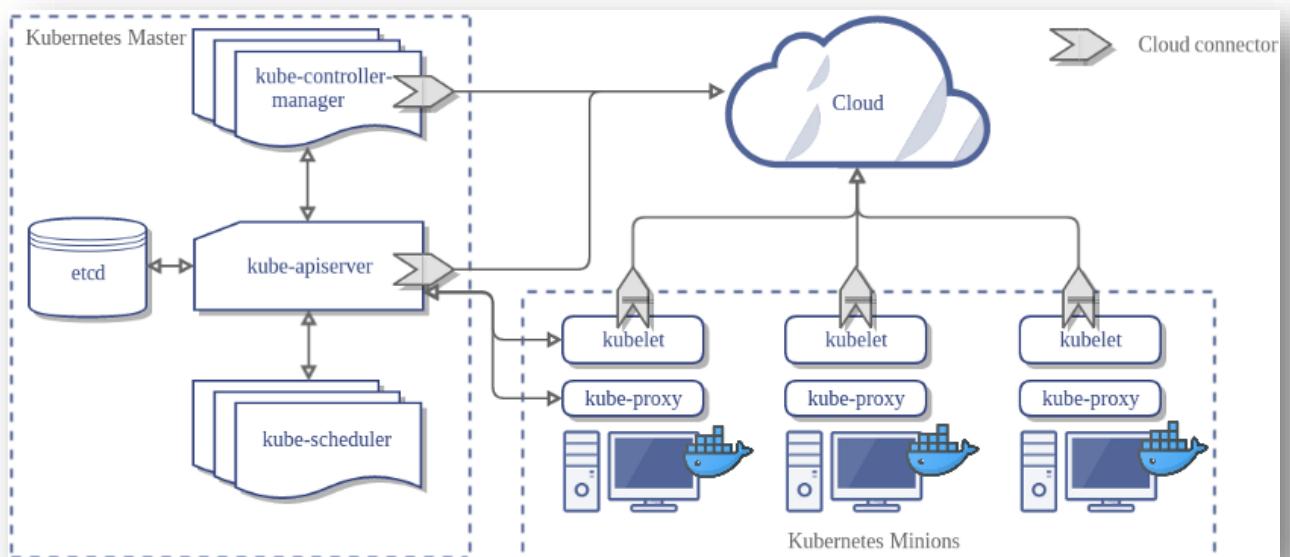


Figure 29:Architecture Kubernetes

### a- Composants de Master :

Sur un node de type master, nous aurons les composants suivants :

kube-apiserver	Point d'entrée exposant l'API HTTP Rest de k8s depuis le maître du cluster Kubernetes. Différents outils et bibliothèques peuvent facilement communiquer avec l'API.
----------------	--

<code>kube-scheduler</code>	Il est responsable de la répartition et l'utilisation de la charge de travail sur les nœuds du cluster selon les ressources nécessaires et celles disponibles.
<code>kube-controller-manager</code>	Ce composant est responsable de la plupart des collecteurs qui récupèrent des informations du cluster tout en effectuant des actions de correctives en cas de besoin, en apportant des modifications pour amener l'état actuel du serveur à l'état souhaité. Il est composé de plusieurs contrôleurs, on peut par exemple retrouver un contrôleur de réPLICATION qui va s'assurer que vous avez le nombre désiré de répliques sur vos pods, mais aussi d'autres contrôleurs clés comme , le contrôleur de Endpoint, le contrôleur d'espace de noms et le contrôleur de compte de service.
<code>cloud-controller-manager</code>	Effectue les mêmes actions que le kube-controller-manager mais pour des fournisseurs de cloud sous-jacents (AWS, Azure, Google Cloud Platform, etc....).
<code>etcd</code>	Il stocke les informations de configuration pouvant être utilisées par chacun des nœuds du cluster. Ces informations sont conservées sous forme de clé et valeurs à haute disponibilité

Tableau 9:Composants du Master

### b- Composants Worker :

Sur chaque worker, nous aurons les composants suivants :

<code>kubelet</code>	Il s'agit d'un agent qui s'exécute dans chaque nœud chargé de relayer les informations au Master. Il interagit avec la base de données etcd du Master pour récupérer des informations afin de connaître les tâches à effectuer. Il assume la responsabilité de maintenir en bon état de fonctionnement les conteneurs d'un pod et s'assure qu'ils tournent conformément à la spécification. Il ne gère pas les conteneurs qui n'ont pas été créés par Kubernetes. Il communique avec le Master et redémarre le conteneur défaillant en cas de crash.
<code>kube-proxy</code>	il active l'abstraction réseau du Service Kubernetes en maintenant les règles du réseau et permet l'exposition des services vers l'extérieur.

Container Runtime	L'environnement d'exécution de conteneurs est le logiciel responsable de l'exécution des conteneurs. Kubernetes est compatible avec plusieurs environnements d'exécution de conteneur: Docker, containerd, cri-o, rktlet ainsi que toute implémentation de Kubernetes CRI
-------------------	---

*Tableau 10: Composants du Worker*

## 1. Google Cloud Platform (GCP) :

Google Cloud Platform (GCP) est une plateforme de cloud computing fournie par Google, proposant un hébergement sur la même infrastructure que celle que Google utilise en interne pour des produits tels que son moteur de recherche1. Cloud Platform fournit aux développeurs des produits permettant de construire une gamme de programmes allant de simples sites web à des applications complexes.

Google Cloud Platform fait partie d'un ensemble de solutions pour les entreprises appelé Google Cloud, et fournit des services modulaires basés sur le cloud, tels que le stockage d'informations, le calcul, des applications de traduction et de prévision, etc....

La Google Cloud Platform est composée d'une famille de produits, chacun comportant une interface web, un outil de lignes de commande, et une interface de programmation applicative REST.

➔ Parmi les produits existants :

Cloud Translation	une API de traduction automatique qui utilise la même technologie que Google Traduction.
Vision API	une API de reconnaissance d'images.
AutoML	une série d'APIs qui permet de créer automatiquement des modèles d'apprentissage automatisé personnalisés.

*Tableau 11: Quelques produits de GCP*

➔ APIs de haut niveau:

Google App Engine	une plate-forme en tant que service pour tester des applications dans un bac à sable. App Engine offre du changement d'échelle automatique, augmentant les ressources pour faire face à la charge du serveur.
Google Compute Engine	le composant infrastructure en tant que service de la Google Cloud Platform permettant aux utilisateurs de lancer des machines virtuelles (VMs) à la demande.
Google Kubernetes Engine	le composant infrastructure en tant que service de la Google Cloud Platform permettant aux utilisateurs de lancer des machines virtuelles (VMs) à la demande.
Google Cloud Storage	un système de stockage en ligne de fichiers.

Tableau 12:Quelque API de GCP

## 2. Commande de configuration Kubectl

Kubectl contrôle le cluster Kubernetes. C'est l'un des composants clés de Kubernetes qui s'exécute sur le poste de travail sur n'importe quelle machine lorsque la configuration est terminée. Il a la capacité de gérer les nœuds du cluster. Elle est principalement utilisée pour communiquer avec les serveurs de l'API Kubernetes pour créer, mettre à jour et supprimer des charges de travail dans Kubernetes.

La majorité des kubectl commandes courantes fournissent une opération ou une action spécifique à effectuer, comme créer, supprimer, etc. Cette méthode implique généralement l'interprétation d'un fichier (YAML ou JSON) qui décrit l'objet dans Kubernetes (un POD, un service, une ressource, etc.).

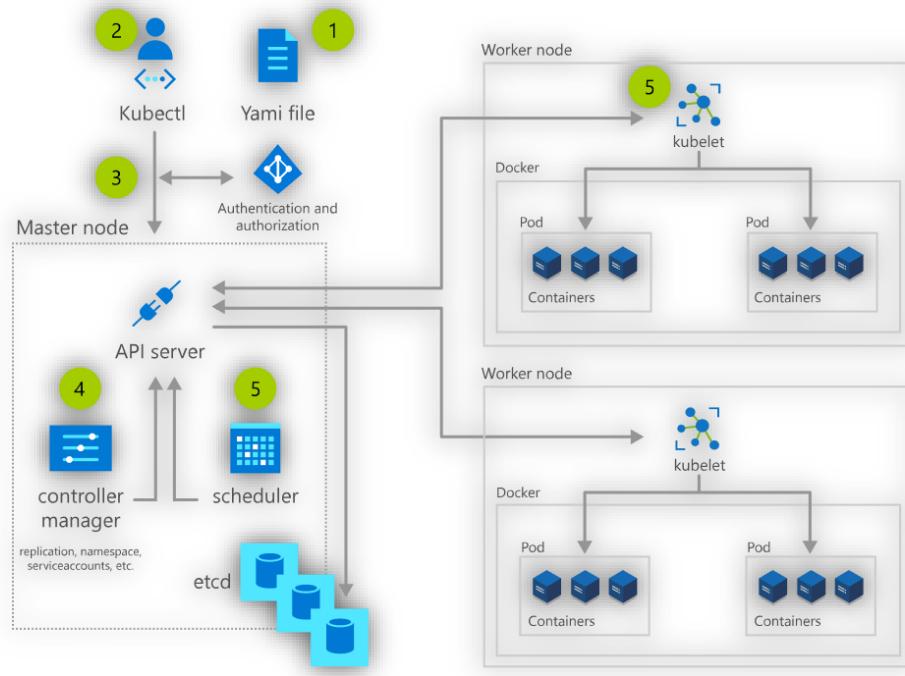


Figure30:Fonctionnement des déploiements de k8s

Les opérations indiquées sur la ligne de commande sont transmises au serveur API qui, à son tour, communique avec les services backend de Kubernetes si nécessaire. (Voir figure 39)

## 1- Syntaxe de kubectl :

**Kubectl** a une syntaxe à utiliser comme suit:

```
$kubectl [command] [TYPE] [NAME]
```

Command	fait référence à ce que vous souhaitez effectuer (créer, supprimer, etc.)
Type	fait référence au type de ressource contre lequel vous exécutez une commande (Pod, Service, etc.)
Name	nom sensible à la casse de l'objet. Si vous ne spécifiez pas de nom, il est possible d'obtenir des informations sur toutes les ressources auxquelles votre commande correspond (Pods, par exemple)

flags	ceux-ci sont facultatifs mais sont utiles lors de la recherche de ressources spécifiques. Par exemple, --namespace vous permet de spécifier un espace de noms particulier pour effectuer une opération dans.
-------	--

Tableau 13: Composants de la syntaxe de Kubectl

## 2- Fichier de configuration de Kubectl :

Les fichiers kubeconfig organisent des informations sur les clusters, les utilisateurs, les espaces de noms et les mécanismes d'authentification. La commande kubectl utilise ces fichiers pour trouver les informations dont elle a besoin pour choisir un cluster et communiquer avec lui.

**Remarque:** Un fichier utilisé pour configurer l'accès aux clusters est appelé **fichier kubeconfig**. Il s'agit d'une façon générique de faire référence aux fichiers de configuration. Cela ne signifie pas qu'il existe un fichier nommé kubeconfig.

Pour accéder à votre cluster Kubernetes, kubectl utilise un fichier de configuration. Le kubectl fichier de configuration par défaut se trouve dans ~/.kube/config et est appelé kubeconfig fichier

## 3. Crédit et gestion d'un cluster multi-nœud :

### 1- Crédit du cluster

- **En utilisant LINUX :**

Dans une machine virtuelle Ubuntu 20 TLS, on installe Kubernetes suivant les commandes suivantes :

- **Configurer Docker**

```
$Sudo su

#apt-get update //Mettre à jour la liste des packages

#apt-get install docker-io //installation de Docker

#docker ---version //Vérification de l'installation et la version

#systemctl enable docker //Démarrer docker
```

- **Installer Kubernetes**

```
#sudo apt install curl //installer curl  
  
#curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key  
add //ajouter la clé de signature kubernetes  
  
#apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"  
//ajouter kubernetes dans les référentiels logiciels
```

```
#apt-get install kubeadm kubelet kubectl //installer les outils kubernetes  
  
#apt-mark hold kubeadm kubelet kubectl  
  
#kubeadm version //vérification de l'installation de kubeadm
```

- **Deploiement de Kubernetes :**

```
#swapoff -a //désactiver la mémoire d'échange sur chaque serveur  
  
//toutes les commandes précédentes doivent se répéter dans chaque nœud de  
serveur  
  
(Assurez-vous d'installer la même version de chaque package sur chaque  
machine)  
  
#hostnamectl set-hostname <nom-machine> //attribuer un nom d'hôte unique à  
chaque nœud de serveur
```

Dans le nœud maître on initialise kubernetes en utilisant la commande suivante:

```
#kubeadm init --pod-network-cidr=10.244.0.0/16
```

Parfois la commande peut générer une erreur concernant le nombre de CPU donné à la machine car le nombre de CPU doit être 2, mais on peut ignorer l'erreur en ajoutant **--ignore-preflight-errors=NumCPU** à la commande **kubeadm init**

Une fois la commande est terminée, elle affichera les étapes à suivre pour utiliser le cluster :

**1- Crédit d'un répertoire pour le cluster avec les 3 commandes suivantes :**

- 2- Déploiement du Pod Network sur le cluster :**est un moyen qui permet la communication entre différents nœuds du cluster
- 3- Joindre les Workers au cluster :**en utilisant la commande kubeadm join indiquée dans la sortie de la commande kubeadm init ➤On entre la commande sur chaque nœud worker pour le connecter au cluster :
- 4- On vérifie que le nœud worker a bien été ajouté au cluster :**en exécutant la commande suivante

```
#kubectl get nodes
```

Après avoir suivi tous les étapes décrit dans cette partie, on a donc kubernetes installé et le cluster est créé et tous les nœuds sont près.

#### - En utilisant Google Cloud Platform (GCP) :

Pour créer un cluster, on accède d'abord à l'environnement Kubernetes Engine .

Google Kubernetes Engine (GKE) est un environnement géré grâce auquel vous pouvez déployer, gérer et faire évoluer vos applications en conteneurs à l'aide de l'infrastructure Google. Il comprend plusieurs machines (en particulier, les instances de Google Compute Engine) regroupées pour former un cluster de conteneurs.

Les types de clusters disponibles sont les suivants : zonal (zone unique ou multizones) et régional.

##### a. Clusters à zone unique :

Un cluster à zone unique possède un seul plan de contrôle (maître) s'exécutant dans une zone. Ce plan de contrôle gère les charges de travail sur les nœuds exécutés dans la même zone.

##### b. Clusters multizones :

Un cluster multizone comporte une seule instance dupliquée du plan de contrôle s'exécutant dans une seule zone et plusieurs nœuds s'exécutant dans plusieurs zones. Une mise à niveau du cluster ou une panne de la zone dans laquelle s'exécute le plan de contrôle n'empêchent pas les charges de travail de s'exécuter normalement. Toutefois, le cluster, ses nœuds et ses charges de travail ne peuvent pas être configurés tant que le plan de contrôle n'est pas disponible. Les clusters à zones multiples équilibrivent la disponibilité et les coûts afin d'assurer la cohérence des charges de travail.

### c. Clusters régionaux :

Un cluster régional comporte plusieurs instances dupliquées du plan de contrôle, qui s'exécutent dans plusieurs zones au sein d'une région donnée. Les nœuds s'exécutent également dans chaque zone dans laquelle s'exécute une instance dupliquée du plan de contrôle. Dans la mesure où un cluster régional réplique le plan de contrôle et les nœuds, il consomme davantage de ressources Compute Engine qu'un cluster analogue à zone unique ou multizones.

#### La création d'un cluster à zone unique :

Avant de commencer, effectuez les tâches suivantes :

- Assurez-vous d'avoir activé l'API Google Kubernetes Engine
- Assurez-vous d'avoir installé le [SDK Cloud](#).

Configurez les paramètres gcloud par défaut à l'aide de l'une des méthodes suivantes :

- Utilisez **gcloud init** pour suivre les instructions permettant de définir les paramètres par défaut.
- Utilisez **gcloud config** pour définir individuellement l'ID, la zone et la région de votre projet.
- Créez ou sélectionnez une configuration.
- Choisissez un projet Google Cloud.
- Choisissez une zone Compute Engine par défaut.

#### - Création du cluster :

Les instructions suivantes montrent comment créer un cluster à zone unique à l'aide de **ligne de commande gcloud** ou de **Google Cloud Console**.

##### • Ligne de commande Gcloud

Pour créer un cluster à zone unique avec l'outil de ligne de commande gcloud, utilisez la commande suivante.

```
$export my_zone=<nameZ>//définir une variable d'environnement contenant la zone
$export my_cluster=<nameC>//définir une variable d'environnement contenant le nom du cluster
$gcloud container clusters create $my_cluster --zone $my_zone -num-nodes <num>
```

- **Google Cloud Console :**

1. Accédez au menu Google Kubernetes Engine de Cloud Console.
2. Cliquez sur le bouton **Créer un cluster**.
3. Dans la section **Paramètres de base du cluster**
  - a. Saisissez le **nom** de votre cluster.
  - b. Pour le **type d'emplacement**, sélectionnez **Zonal**, puis choisissez **la zone** souhaitée pour votre cluster.
  - c. Choisissez une **version maître**. Nous vous recommandons de sélectionner une **version disponible**. Si vous devez spécifier une version statique, assurez-vous que la **mise à niveau automatique** est activée pour vos pools de nœuds.
4. Dans le volet de navigation, cliquez sur **default-pool** sous **Pools de nœuds**.
- Dans la section **Détails du pool de nœuds** :
  - a. Saisissez **un nom** pour le **pool de nœuds** par défaut.
  - b. Choisissez **la version de nœud** pour vos nœuds.
  - c. Saisissez **le nombre de nœuds** à créer dans le cluster. Vous devez disposer d'un **quota de ressources** disponible pour les nœuds et les ressources associées (telles que les routes de pare-feu).
5. Dans le volet de navigation, cliquez sur **Nœuds** sous **Pools de nœuds** :
- Dans la liste déroulante **Type d'image**, sélectionnez l'**image de nœud** souhaitée.
- Sélectionnez la **configuration de la machine** à utiliser par défaut pour les instances. Chaque type de machine est facturé différemment. Le type de machine par défaut est n1-standard-1. Pour plus d'informations sur les tarifs applicables aux différents types de machine, consultez la **grille tarifaire par type de machine**.
- Dans la liste déroulante **Type de disque de démarrage**, sélectionnez le **type de disque** souhaité.
- saisissez **la taille du disque de démarrage**.
6. Cliquez sur **Créer** :

Après avoir créé un cluster, vous devez configurer kubectl pour pouvoir interagir avec le cluster depuis la ligne de commande.

## 2- Gestion du cluster :

Une fois que vous avez un cluster en cours d'exécution, vous pouvez le gérer avec la commande kubectl. La plupart des commandes vous pouvez l'obtenir avec la commande **kubectl –help**.

Liste des commandes de Kubectl :

<b>Introspecter le cluster</b>	# lister tous les services	\$kubectl get services
	# Décrire un nœud	\$kubectl describe node <name>
	# Connaitre la version	\$kubectl version
	# Récupérer les informations sur le cluster	\$kubectl cluster-info
	# Récupérer la configuration sur le cluster	\$kubectl config view
<b>Introspecter les pods</b>	# Lister les pods :	kubectl get pods
	# Décrire un pod particulier :	kubectl describe pod <name>
	# Lister les réplications controllers	kubectl get rc
	# Décrire la réPLICATION controller <name>	kubectl describe rc <name>

Interagir avec les pods	# Lancer un pod	<code>kubectl run &lt;name&gt; --image=&lt;image-name&gt;</code>
	# Créer un service	<code>kubectl create -f &lt;manifest.yaml&gt;</code>
	# Augmenter les replicas	<code>kubectl scale --replicas=&lt;count&gt; rc &lt;name&gt;</code>
	# Mapper un port	<code>kubectl expose rc &lt;name&gt; --port=&lt;external&gt;--target-port=&lt;internal&gt;</code>
Suppression des ressources	# Supprimer un pod	<code>kubectl delete pod &lt;name&gt;</code>
	# Supprimer un réPLICATION controller	<code>kubectl delete rc &lt;name&gt;</code>
	# Supprimer un service	<code>kubectl delete svc &lt;name&gt;</code>
	# Retirer le nœud <node> du cluster	<code>kubectl delete node &lt;name&gt;</code>
	# Supprimer tous les pods d'un nœud	<code>kubectl drain &lt;n&gt; --delete-local-data --force --ignore-daemonsets</code>

Tableau 14:Quelque commande de kubectl

## 4. Gestion et manipulation d'un pod

Un Pod est un groupe d'un ou plusieurs conteneurs, avec un **stockage** et un **réseau** partagé (**voir figure 40**). Par conséquence, ces conteneurs communiquent plus efficacement entre eux et assurent une localisation de la donnée :

- **Réseau** : les pods reçoivent automatiquement des adresses IP uniques. Les conteneurs d'un pod partagent le même espace de noms réseau, y compris l'adresse IP et les ports réseau. Les conteneurs d'un pod communiquent entre eux à l'intérieur du pod sur localhost.
- **Stockage** : les pods peuvent spécifier un ensemble de volumes de stockage partagés pouvant être communs aux différents conteneurs.



Figure31: composants des Pods

Vous pouvez considérer un pod comme un "hôte logique" autonome et isolé, dépositaire des besoins systémiques de l'application qu'il gère.

Un pod est conçu pour exécuter une seule instance de votre application sur votre cluster. Toutefois, il n'est pas recommandé de créer directement des pods individuels. Au lieu de cela, vous créez généralement un ensemble de pods identiques, constituant des *instances dupliquées*, pour exécuter votre application. Un tel ensemble de pods répliqués est créé et géré par un *contrôleur*, par exemple un objet Déploiement. Les contrôleurs gèrent le cycle de vie de leurs pods constitutifs et peuvent également gérer le *Scaling horizontal* en modifiant le nombre de pods suivant les besoins.

## 1- Cycle de vie d'un pod :

Les pods sont éphémères. Ils ne sont pas conçus pour fonctionner à l'infini et, lorsqu'un pod est arrêté, il ne peut pas être restauré. En général, les pods ne disparaissent pas tant qu'ils ne sont pas supprimés par un utilisateur ou par un contrôleur.

Les pods ne peuvent pas se "corriger" ou se réparer. Ainsi, si un pod est planifié sur un nœud qui subit ultérieurement une défaillance, le pod est supprimé. De même, si un pod est évincé d'un nœud pour une raison quelconque, le pod ne peut pas se remplacer lui-même.

Chaque pod possède un objet d'API **PodStatus**, qui est représenté par le champ status. Les pods renseignent leur *phase* dans le champ status: phase. La phase d'un pod est une synthèse brève du pod dans son état actuel.

Lorsque vous utilisez la commande **kubectl get pod** pour inspecter un pod en cours d'exécution sur un cluster, ce pod peut se trouver dans l'une des status suivantes :

Pending	Le Pod a été accepté par Kubernetes, mais une ou plusieurs images de conteneurs n'ont pas encore été créées. Ceci inclut le temps avant d'être affecté ainsi que le temps à télécharger les images à travers le réseau, ce qui peut prendre un certain temps.
Running	Le pod a été affecté à un nœud et tous les conteneurs ont été créés. Au moins un conteneur est toujours en cours d'exécution, ou est en train de démarrer ou redémarrer.
Succeeded	Tous les conteneurs du pod ont terminé avec succès et ne seront pas redémarrés.
Failed	Tous les conteneurs d'un pod ont terminé, et au moins un conteneur a terminé en échec : soit le conteneur a terminé avec un status non zéro, soit il a été arrêté par le système.
Unknown	Pour quelque raison l'état du pod ne peut pas être obtenu, en général en cas d'erreur de communication avec l'hôte du Pod.

Tableau 15:Differents status d'un pod

De plus, PodStatus contient un tableau appelé **PodConditions**, représenté dans le fichier manifeste du pod sous l'intitulé conditions.

De ce fait, il existe deux types de Pods :

- **Pods qui exécutent un seul conteneur** : Le modèle «un conteneur par module» est le cas d'utilisation le plus courant de Kubernetes; dans ce cas, vous pouvez considérer un pod comme un wrapper autour d'un seul conteneur, et Kubernetes gère directement les pods plutôt que les conteneurs.

- **Pods qui exécutent plusieurs conteneurs qui doivent fonctionner ensemble :** Un pod peut encapsuler une application composée de plusieurs conteneurs colocalisés qui sont étroitement couplés et doivent partager des ressources. Ces conteneurs colocalisés peuvent former une seule unité de service cohérente - un conteneur servant les fichiers d'un volume partagé au public, tandis qu'un conteneur «sidecar» distinct actualise ou met à jour ces fichiers. Le pod encapsule ces conteneurs et ressources de stockage en une seule entité gérable.

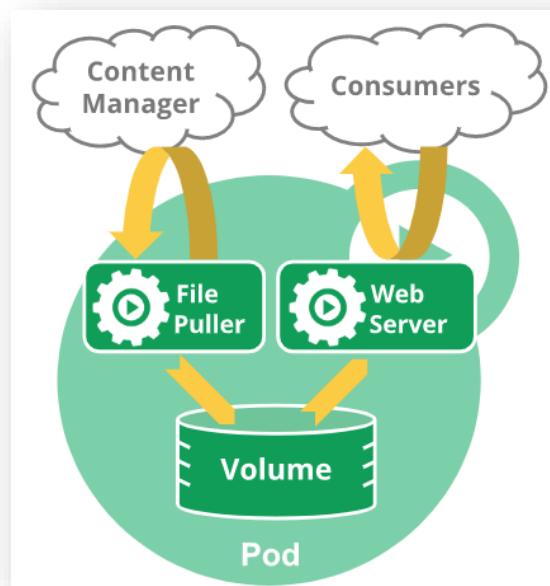


Figure 32:Diagramme de pod

Un pod multi-conteneurs contient un extracteur de fichiers et un serveur Web qui utilise un volume persistant pour le stockage partagé entre les conteneurs

## 2- Fonctionnement des pods :

Comme les conteneurs d'application individuels, les pods sont considérés comme des entités relativement éphémères (plutôt que durables). Comme indiqué dans le cycle de vie des pods, les pods sont créés, attribués un ID unique (UID) et planifiés sur les nœuds où ils restent jusqu'à la fin (selon la stratégie de redémarrage) ou la suppression.

Si un Nœud meurt, les pods planifiés sur ce nœud doivent être supprimés après un délai d'expiration. Un pod donné (tel que défini par un UID) n'est pas "replanifié" vers un nouveau

nœud; au lieu de cela, il peut être remplacé par un Pod identique, avec même le même nom si vous le souhaitez, mais avec un nouvel UID. Quand on dit que quelque chose a la même durée de vie qu'un pod, comme un volume, cela signifie qu'il existe tant que ce pod (avec cet UID) existe. Si ce pod est supprimé pour une raison quelconque, même si un remplacement identique est créé, l'élément connexe (par exemple le volume) est également détruit et recréé.

### 3- Requête des pods

Lorsqu'un pod démarre, il demande une quantité de processeurs et de mémoire. Cela permet à Kubernetes de planifier le pod sur un nœud approprié afin d'exécuter la charge de travail. Un pod ne sera pas programmé sur un nœud qui ne dispose pas des ressources nécessaires pour répondre à la requête du pod. Une requête correspond à la quantité *minimale* de processeurs ou de mémoire garantie par Kubernetes à un pod.

Vous pouvez configurer les requêtes de processeurs et de mémoire pour un pod, en fonction des ressources dont vos applications ont besoin. Vous pouvez également spécifier les requêtes pour les conteneurs individuels qui s'exécutent dans le pod. Tenez bien compte des éléments suivants :

- La requête de processeurs par défaut est de 100 millions. Ce nombre est trop faible pour de nombreuses applications. En outre, il est probablement bien plus faible que la quantité de processeurs disponible sur le nœud.
- Il n'existe pas de requête de mémoire par défaut. Un pod sans requête de mémoire par défaut peut être planifié sur un nœud ne disposant pas d'une quantité de mémoire suffisante pour exécuter les charges de travail du pod.
- Si vous définissez une valeur trop faible pour les requêtes de processeurs ou de mémoire, un nombre trop élevé de pods ou une combinaison sous-optimale de pods sont planifiés sur un nœud donné, ce qui entraîne une baisse de performance.
- Si vous définissez une valeur trop élevée pour les requêtes de processeurs ou de mémoire, le pod risque de devenir non planifiable et d'augmenter le coût des ressources du cluster.
- En plus ou au lieu de définir les ressources d'un pod, vous pouvez spécifier des ressources pour les conteneurs individuels qui s'exécutent dans le pod. Si vous ne spécifiez que des ressources pour les conteneurs, les requêtes du pod correspondent à la somme des requêtes spécifiées pour les conteneurs. Si vous exécutez ces deux

méthodes, la somme des requêtes pour tous les conteneurs ne doit pas dépasser les requêtes du pod.

#### 4- Limite des pods

Par défaut, la limite d'un pod est inférieure à la quantité maximale de processeurs ou de mémoire qu'il peut utiliser sur un nœud. Vous pouvez définir des limites pour contrôler la quantité de processeurs ou de mémoire que votre Pod peut utiliser sur un nœud. Une limite correspond à la quantité *maximale* de processeurs ou de mémoire garantie par Kubernetes à un pod.

En plus ou au lieu de définir les limites d'un pod, vous pouvez spécifier des limites pour les conteneurs individuels qui s'exécutent dans le pod. Si vous ne spécifiez que des limites pour les conteneurs, les limites du pod correspondent à la somme des limites spécifiées pour les conteneurs. Toutefois, chaque conteneur est limité dans son accès aux ressources. Par conséquent, si vous décidez de ne spécifier des limites que sur les conteneurs, vous devez spécifier des limites pour chacun d'entre eux. Si vous spécifiez les deux, la somme des limites pour tous les conteneurs ne doit pas dépasser la limite du pod.

#### 5- Contrôler les nœuds sur lesquels un pod s'exécute

D'office, les pods s'exécutent sur des nœuds du pool de nœuds par défaut du cluster. Vous pouvez configurer explicitement ou implicitement le pool de nœuds sélectionné par un pod :

- Il est possible de forcer explicitement un pod à se déployer sur un pool de nœuds spécifique en définissant un sélecteur de nœuds dans le fichier manifeste du pod. Cela oblige un pod à ne s'exécuter que sur les nœuds de ce pool de nœuds.
- Vous pouvez spécifier des requêtes de ressources pour les conteneurs que vous exécutez. Le pod ne s'exécutera que sur des nœuds qui répondent aux requêtes de ressources. Par exemple, si la définition de pod inclut un conteneur qui nécessite quatre processeurs, le service ne sélectionnera pas les pods s'exécutant sur des nœuds avec deux processeurs.

#### 6- Arrêt d'un pod

Les pods s'arrêtent normalement lorsque leurs processus sont terminés. Kubernetes impose par défaut un délai de grâce de 30 secondes. Lorsque vous supprimez un pod, vous pouvez remplacer ce délai de grâce en définissant l'option **--grace-period** sur le temps d'attente souhaité (exprimé en secondes) avant l'arrêt forcé du pod.

## 7- Création du pod :

- Pod de conteneur unique :

Pour créer ce pod il existe deux manières :

- 1- Ils peuvent être simplement créés avec la commande **kubectl run**, où vous avez une image définie dans le registre Docker que nous allons extraire lors de la création d'un pod.

```
$kubectl run <pod name> --image=<image>
```

- 2- On peut également utiliser un fichier YAML, en exécutant la commande **kubectl create** après la création de ce fichier

Dans la création du fichier YAML: on doit préciser tous les caractéristiques de notre pod après on crée une commande en utilisant la commande suivante :

```
$kubectl create -f <name>.yaml
```

- Pod de conteneur multiple :

Ce type de pod pour le créer on utilise un fichier YAML contenant le nombre de conteneurs voulu dans le pod, en exécutant la commande **kubectl create**.

Dans la création du fichier YAML: on spécifie toutes les caractéristiques des conteneurs voulu dans le pod puis on crée le pod on utilisant la commande kubectl create

**Conclusion :** Les pods étant éphémères, il n'est pas nécessaire de les créer directement. De même, puisque les pods ne peuvent ni se réparer ni se remplacer, il n'est pas recommandé de créer des pods directement. À la place, vous pouvez utiliser un *contrôleur* tel qu'un objet Déploiement, qui crée et gère les pods pour vous. Les contrôleurs sont également utiles pour déployer des mises à jour, par exemple pour changer la version d'une application s'exécutant dans un conteneur, car le contrôleur gère l'intégralité du processus de mise à jour.

## 5. Gestion et manipulation des Deployments

Les déploiements représentent un ensemble de pods identiques sans identités uniques. Un déploiement exécute plusieurs instances dupliquées de votre application et remplace automatiquement les instances qui échouent ou ne répondent plus. De cette manière, les déploiements permettent de s'assurer qu'une ou plusieurs instances de votre application sont disponibles pour répondre aux demandes des utilisateurs. Les déploiements sont gérés par le contrôleur de déploiement Kubernetes.

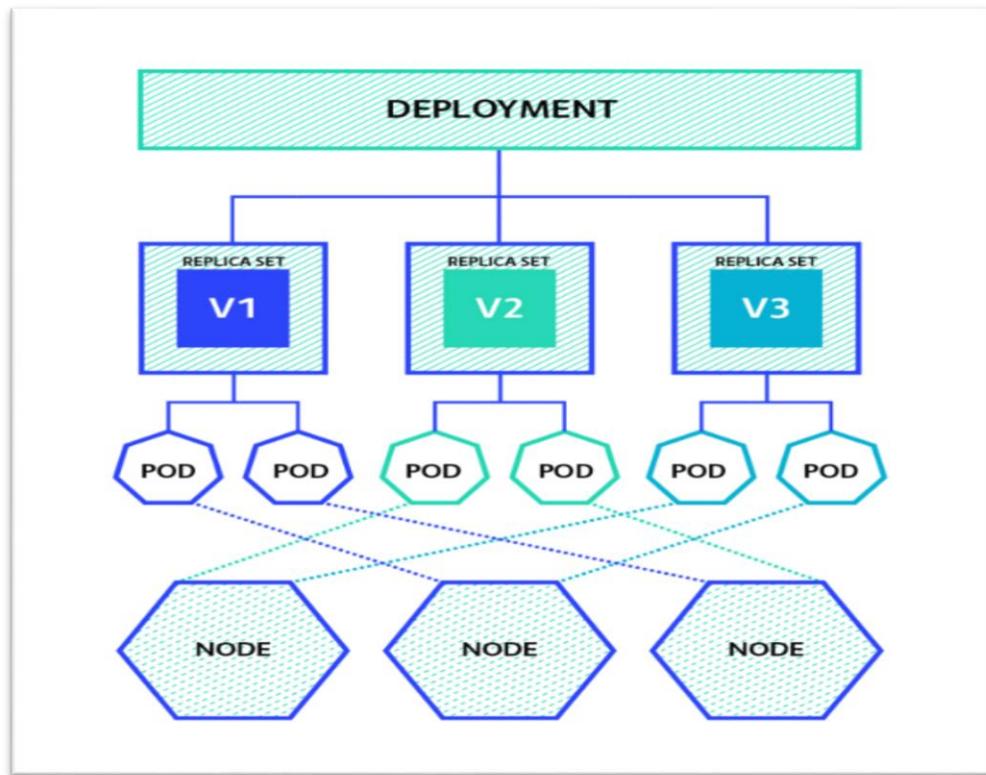


Figure 33:Fonctionnement du déploiement

Les déploiements utilisent un modèle de pod qui contient une spécification pour ses pods. La spécification du pod détermine l'apparence de chaque pod : les applications à exécuter dans ses conteneurs, les volumes que les pods doivent monter, ses libellés, etc. Lorsqu'un modèle de pod de déploiement est modifié, de nouveaux pods sont automatiquement créés un à un.

**Remarque :** la console GCP ne dispose pas d'une fonction de rollback directe

## 6. Jobs & Cronjobs

### 1- Jobs :

Un Job est un objet contrôleur qui représente un travail fini. Il crée un ou plusieurs pods pour exécuter un processus spécifique. Dans sa forme la plus simple, un job crée un pod et suit

l'avancement du processus dans ce pod. Une fois terminé, le processus arrête le pod et indique que le job a réussi, ainsi vous pouvez vous servir d'un job pour exécuter en parallèle des éléments de travail indépendants, mais liés : envoi d'e-mails, affichage d'images, transcodage de fichiers, analyse de clés de base de données, etc. Toutefois, les jobs ne sont pas conçues pour les processus parallèles en communication proche, tels que les flux continus de processus en arrière-plan.

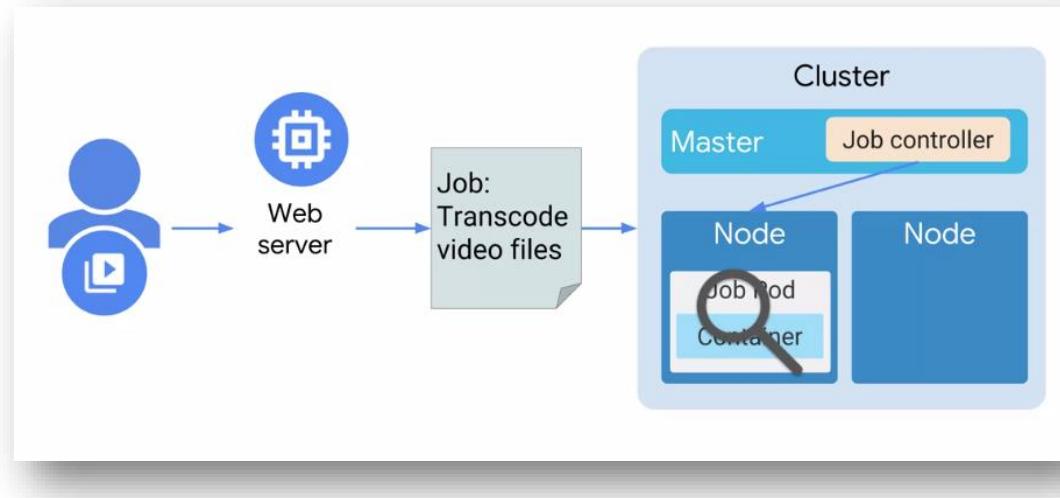


Figure34:Exemple fonctionnement d'un Job

Dans ce scénario, un utilisateur importe un fichier vidéo sur un serveur Web pour conversion ou transcodage. Le serveur Web crée un manifeste d'objet de job pour ce transcodage. Un job est créé sur le cluster et le contrôleur planifie un pod pour ce job sur le nœud et surveille le pod.

En cas de défaillance du nœud entraînant la perte du pod, le contrôleur sait que le processus n'est pas terminé. Il replanifie le pod du job pour une exécution sur un autre nœud. Le contrôleur continue de surveiller le pod jusqu'à la fin du processus. Une fois celui-ci terminé, le contrôleur supprime le job et tous les pods associés.

### **Deux types de Jobs sont disponibles :**

Jobs non parallèles	créent un seul pod (qui est recréé en cas d'échec) et s'arrêtent lorsque le pod se termine avec succès
---------------------	--

Jobs parallèles avec nombre d'achèvements	se terminent lorsqu'un certain nombre de pods se terminent avec succès.
---	---

*Tableau 16:Types des Jobs*

Les jobs sont représentées par les objets jobs Kubernetes. Lorsqu'un job est créé, le contrôleur de job crée un ou plusieurs pods et s'assure qu'ils se terminent correctement. Une fois les pods terminés, un job analyse le nombre de pods ayant terminé leur travail avec succès. Lorsque le nombre souhaité d'achèvements est atteint, le job est terminé. Comme pour les autres contrôleur, un contrôleur de job crée un pod si l'un de ses pods échoue ou est supprimé.

### a- Crédation d'un job :

Vous pouvez créer un job en utilisant la commande **kubectl apply** avec un fichier YAML.

Ensuite on vérifie que les jobs sont créés en exécutant la commande :

```
$kubectl get jobs
```

Et pour voir la sortie du Job via le pod qu'il a supervisé, on exécute la commande suivante :

```
$kubectl logs <pod name>
```

### b- Suppression du job :

Lorsqu'un job est terminé, elle arrête de créer des pods. L'objet de l'API Job n'est pas supprimé lorsqu'il est terminé, ce qui vous permet d'afficher son état. Les pods créés par le job ne sont pas supprimés, mais terminés. La conservation des pods vous permet de consulter leurs journaux et d'interagir avec eux.

Pour supprimer un job, exécutez la commande suivante :

```
$kubectl delete job <jobName>
```

Lorsque vous supprimez un job, tous ses pods sont également supprimés.

Pour supprimer un job tout en conservant ses pods, spécifiez l'option **--cascade false** :

```
$kubectl delete job <jobName> --cascade false
```

## 2- Cronjobs :

Les Jobs, une tâche ou un lot de tâches que nous voulons exécuter sur Kubernetes une seule fois. Job va créer un pod qui a toujours un état de terminaison. Après l'état de terminaison arrive, le pod ira dans l'état 'completed' et il n'y aura plus de conteneurs en cours d'exécution. Les jobs attendent toujours que le processus génère du code de sortie 0 dans le conteneur. Le code de sortie 0 indique toujours l'achèvement du processus et c'est pourquoi le pod sera automatiquement supprimé.

Maintenant, il y a toujours un cas d'utilisation où les administrateurs de cluster veulent exécuter des tâches spécifiques à une heure spécifique. Et pour cela Kubernetes venir avec CronJobs, Emplois qui seront programmés pour être exécutés. CronJobs sont très pratiques lorsqu'il existe des cas comme la sauvegarde de base de données planifiée ou la planification de courrier électronique ou le processus de nettoyage des ressources. Cronjobs définissent toujours la planification en format Cron (\* \* \* \*). CronJob dans kubernetes exécute toujours un pod dans un cluster. Le pod détient toujours l'image et après l'exécution de cette image, il sera effacé. Donc, quelle que soit l'image que nous allons déployer, elle devrait être immuable. Pod ne doit contenir aucune donnée sensible au fur et à mesure qu'elle est détruite.

## 7. Mise en réseau de cluster Kubernetes

Le modèle de réseau kubernetes se base généralement et en grande partie sur les adresses IP , les services , les pods, les conteneurs et les nœuds communiquent via ces adresses et des ports , Kubernetes propose des différents types d'équilibrage de charges (Load Balancing) pour diriger le trafic vers le bon pod .

## 3- Interface réseau de conteneurs (CNI)

Un CNI est simplement un lien entre le temps d'exécution du conteneur (comme Docker) et le plugin réseau. Le plugin réseau n'est rien d'autre que l'exécutable qui gère la connexion réelle du conteneur vers ou depuis le réseau, selon un ensemble de règles définies par le CNI. Donc un CNI est un ensemble de règles et de bibliothèques Go qui aident à l'intégration conteneur/réseau-plugin. Toutes les CNI peuvent être déployées en exécutant simplement un

pod ou un DaemonSet qui lance et gère leurs daemons. Parmi les solutions de réseautage Kubernetes les plus connues :

- Flannel
- Calico
- Weave Net
- Cisco ACI,
- Cilium
- Contiv ...

#### 4- Pods :

Un pod représente un ensemble des conteneurs avec un réseau et un stockage partagés, voici comment fonctionne le modèle de réseau kubernetes :

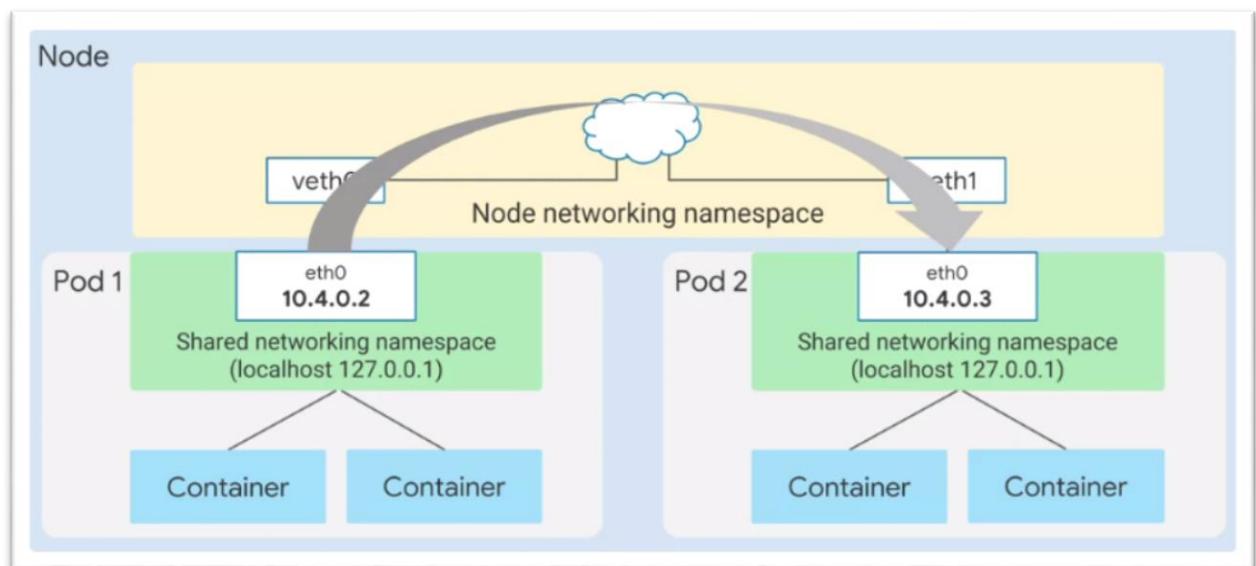


Figure 35:Fonctionnement de réseau kubernetes

Chaque pod reçoit une adresse IP unique et ses conteneurs partagent le même namespace et la même adresse IP

Étant donné que chaque pod dispose d'une adresse IP unique la communication directe pod-to-pod est possible sans nécessiter aucun type de proxy ou de traduction d'adresse. Cela

permet également d'utiliser des ports standards pour la plupart des applications car il n'est pas nécessaire d'acheminer le trafic d'un port hôte à un port à conteneurs, comme dans Docker. Notez que comme tous les conteneurs d'un pod partagent la même adresse IP, les ports conteneurs privés ne sont pas possibles (les conteneurs peuvent accéder aux ports de l'autre via **localhost :<port>**)

**Remarque :** les pods sont éphémères, si un pod est reprogrammé, il aura une nouvelle adresse IP et un nouveau namespace

Alors on a besoin d'une solution plus fiable pour localiser les applications exécutées dans notre cluster kubernetes, et pour avoir des adresses IP et des namespaces inchangables même en cas de reprogrammation ou mise à jour d'un pod.

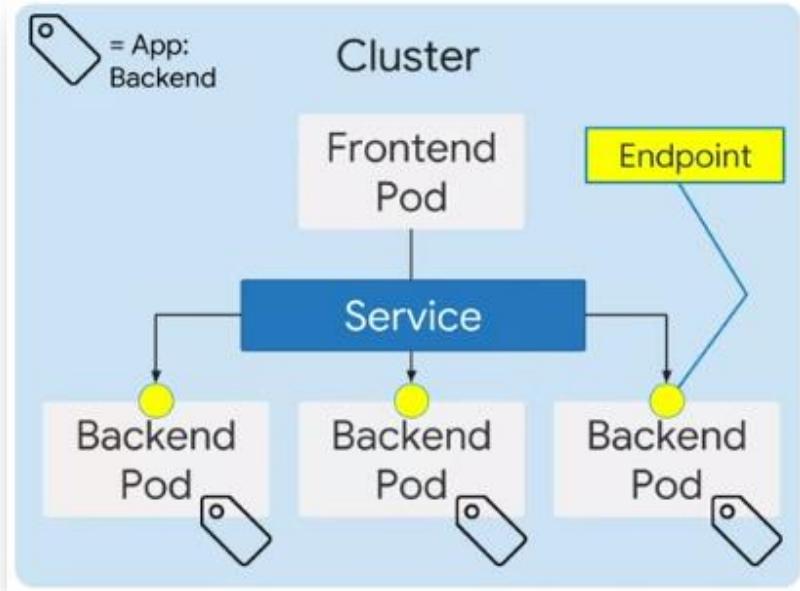
Cette solution est donc **les services**.

## 5- Services :

### a- Définition :

Dans un environnement de conteneurs changeant, les services affectent aux pods une adresse IP et un nom fixes qui restent identiques en cas de mise à jour, de mise à niveau, d'évolution ou de défaillance d'un pod. Au lieu de se connecter à un pod, les applications Kubernetes utilisent des services pour localiser le pod approprié et transférer le trafic.

L'idée d'un Service est de regrouper un ensemble de Pod en une seule ressource. Il existe des différentes façons d'accéder au regroupement. Par défaut, vous obtenez une adresse IP de cluster stable que les clients à l'intérieur du cluster peuvent utiliser pour contacter les composants Pods du Service. Un client envoie une demande à l'adresse IP stable, et la demande est dirigée vers l'un des pods du Service.



*Figure36:Fonctionnement du service*

Un service identifie ses pods membres avec un sélecteur. Pour qu'un Pod soit membre du Service, le Pod doit avoir toutes les étiquettes spécifiées dans le sélecteur. Une étiquette est une paire de clés/valeurs arbitraire qui est attachée à un objet.

### b- Types de services :

Il existe trois principaux types de services : **ClusterIP**, **NodePort** et **LoadBalancer**.

- **ClusterIP** :

Le type par défaut d'un service Kubernetes, il rend le service accessible uniquement à partir du cluster et nous ne pouvons pas atteindre ce service à partir de l'extérieur du cluster.

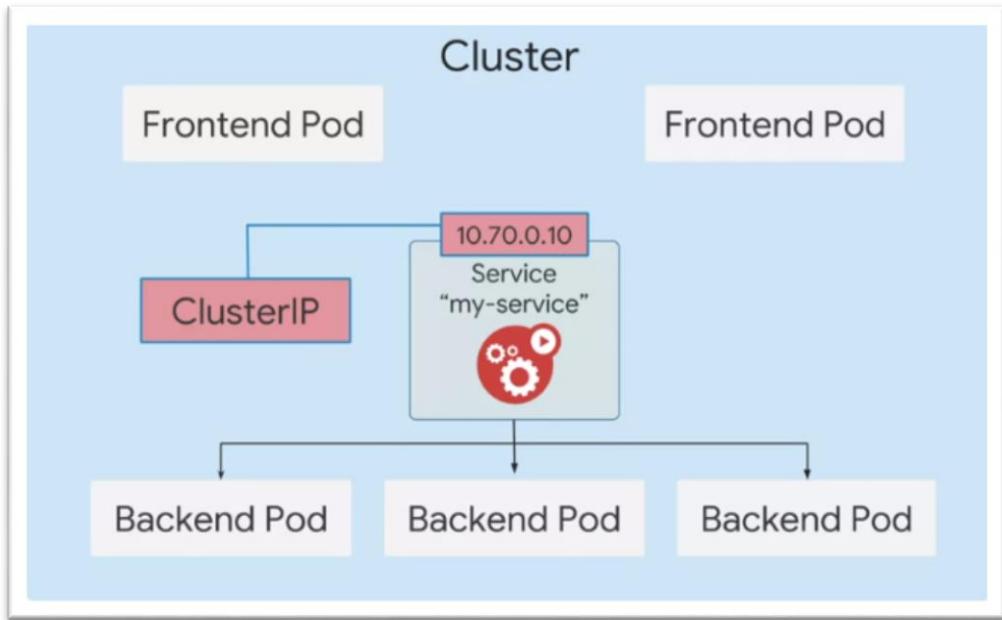


Figure37:Service ClusterIP

L'inconvénient de l'utilisation de 'ClusterIP' est que vous ne pouvez pas appeler les services de l'extérieur du cluster sans utiliser un proxy. Pour régler ce problème, on peut utiliser un contrôleur d'entrée Kubernetes, et nous pouvons exposer les services ClusterIP à l'extérieur du réseau.

- Création du service :

Pour créer un service d'abord nous avons besoin des pods, après nous pouvons exposer le service avec des pods étiquetées particulières, en exécutant la commande suivante :

```
$kubectl run [PodName] --image=[ImageName] --port=80 --labels= "[key]=[value]"
```

Ensuite on crée notre service à l'aide d'un fichier YAML et une commande kubectl apply.

Nous pouvons voir l'IP et les ports du cluster, en accédant à l'application à partir de <clusterip>: <port du cluster>

- Vérification du service

Pour vérifier qu'un service a bien été créé on se connecte au pod utilisant ce service en exécutant la commande suivant :

```
$kubectl exec -it <pod name> curl <clusterip>:<clusterPort>
```

Si la commande curl n'est pas disponible, on peut l'installer :

```
$apt update -y  
$apt install curl -y
```

Après avoir installé curl dans le pod, exécutez **curl <clusterip>: <port du cluster>** ou bien **curl <ServiceName>**. Après on se déconnecte du pod en tapant EXIT.

On remarque que lorsque se déconnecte du pod, en tapant EXIT, on peut se connecter au service en utilisant l'adresse IP et le port mais on ne peut pas se connecter avec le nom du service car le nom du service n'est pas accessible en dehors du pod

- **NodePort :**

Le service NodePort est le moyen le plus primitif de diriger le trafic externe vers le service. NodePort, comme son nom l'indique, ouvre le port spécifié pour tous les nœuds (machines virtuelles) et le trafic vers ce port est redirigé vers le service.

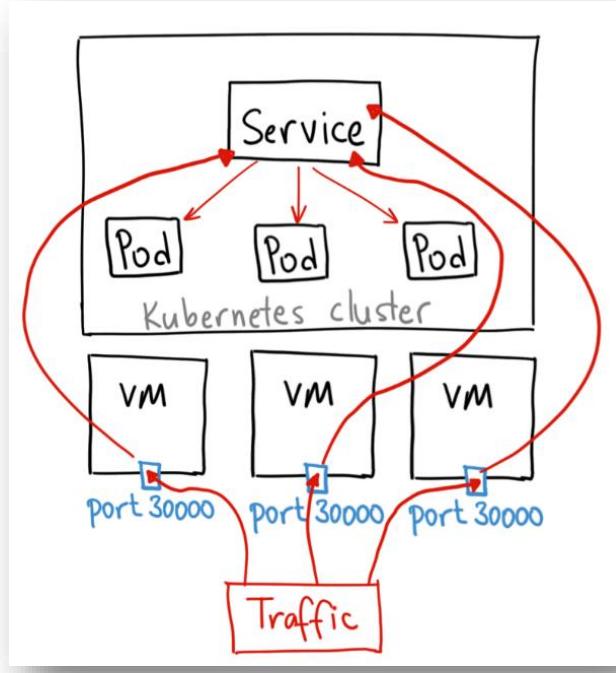


Figure 38:Service NodePort

Le contrôleur Kubernetes alloue un port à partir d'une plage spécifiée par (généralement 30000–32767).

Nous pouvons accéder à l'application ou au service depuis l'extérieur du cluster, en demandant <NodeIP>: <NodePort>

Si vous ne spécifiez pas ce port, il choisira une plage de ports aléatoire (généralement 30000–32767). Si vous souhaitez un numéro de port spécifique, vous pouvez spécifier une valeur dans le champ nodePort.

Les services NodePort sont faciles à créer mais difficiles à sécuriser car ils ouvrent le même port dans tous les nœuds au public, et les ports standards tels que 80, 443 ou 8443 ne peuvent pas être utilisés.

- Création du service :

Pour créer un service d'abord nous avons besoin des pods, après nous pouvons exposer le service avec des pods étiquetées particulières, en exécutant la commande suivante :

```
$kubectl run [PodName] --image=[ImageName] --port=80 --labels= "[key]=[value]"
```

Ensuite on crée notre service à l'aide d'un fichier YAML et une commande kubectl apply après on liste nos services

```
root@meryemelhaffad-virtual-machine:~# kubectl get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP       36h
mynginxsvc   ClusterIP   10.106.60.79  <none>        80/TCP,443/TCP 29m
mynginxsvc2  NodePort    10.110.23.4   <none>        80:30180/TCP,443:31443/TCP 36s
root@meryemelhaffad-virtual-machine:~#
```

Figure 39: service nodePort créé

Ici, nous pouvons voir que le port de service est mappé au port 30180 du nœud.

Si le cluster se dispose de plusieurs nœuds, on exécute la commande suivante pour savoir notre application dans quel nœud est exécuté :

```
$kubectl get services -o wide
```

Ensuite on peut accéder à l'application depuis un navigateur, en tapant

<AdressIP du nœud exécutant l'application> :<Port>

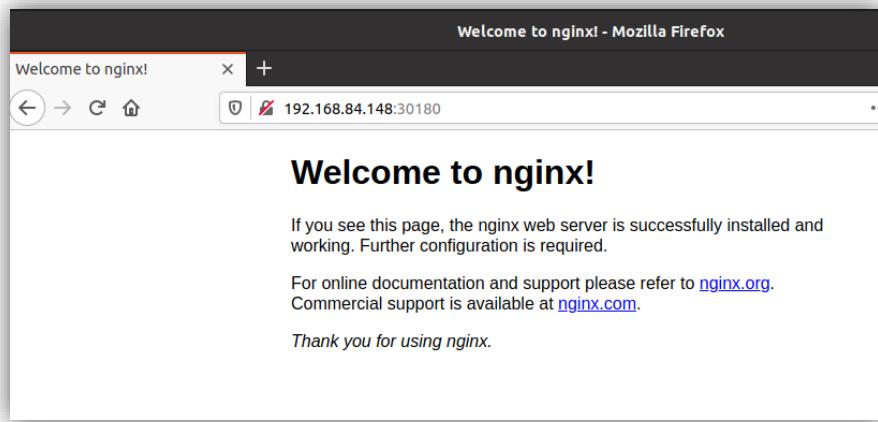


Figure 40: Connexion au service NodePort de l'extérieur

Nous ne pouvons utiliser les ports que dans une plage comprise entre 3000 et 32767 et si l'IP des VMs / nœuds change, nous devons résoudre ce problème, nous ne recommandons donc pas ce service de nodePort.

#### - LoadBalancer :

Les services en général ont leurs propres adresses IP qui sont relativement stables; ainsi, une demande d'une ressource externe est adressée à un service plutôt qu'à un pod, et le service envoie ensuite la requête à un pod disponible.

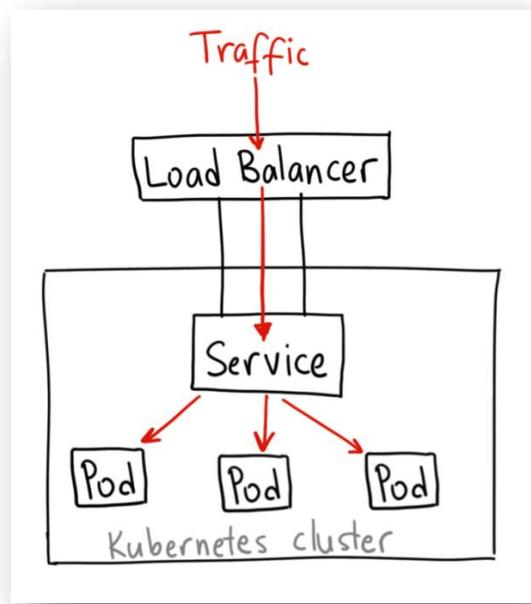


Figure 41: Architecture LoadBalancer

Un équilibrer de charge externe appliquera une logique qui garantit la distribution optimale de ces demandes.

- Création du service :

Pour créer un service Load Balancer il suffit de créer un fichier YAMLet exécuter la commande **kubectl apply** après on liste nos services

```
root@meryemelhaffad-virtual-machine:~# kubectl apply -f loadbalancer.yaml
service/sample-load-balancer created
root@meryemelhaffad-virtual-machine:~# kubectl get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         36h
mynginxsvc     ClusterIP  10.106.60.79  <none>        80/TCP,443/TCP  40m
mynginxsvc2    NodePort   10.110.23.4   <none>        80:30180/TCP,443:31443/TCP 11m
sample-load-balancer LoadBalancer  10.104.250.85  <pending>    80:32410/TCP   3s
```

Figure 42:liste des services créés

On remarque que l'état d'**EXTERNAL-IP** est **<pending>** car pour créer un service load balancer, le cluster doit être hébergés par un fournisseur de cloud ou un environnement qui prend en charge des équilibreurs de charge externes et est configuré avec le package de fournisseur d'équilibreur de charge de cloud correct (**voir figure 57**) Parmi les principaux fournisseurs de cloud tels qu'AWS, Azure et GCP.

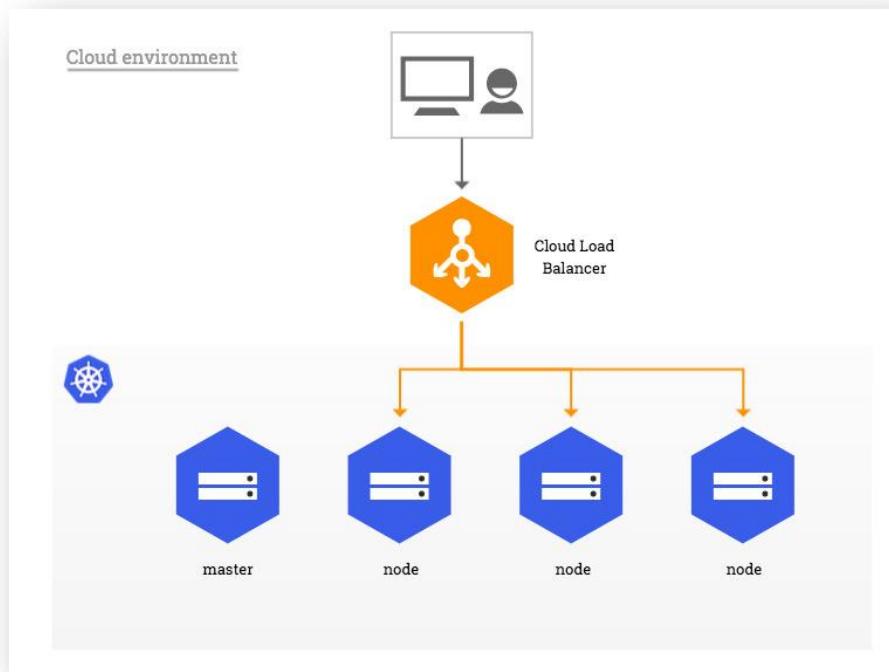


Figure43:Environnement Cloud

Dans notre cas les machines virtuelle ou par exemple les environnements Bare Metal ne sont pas hébergées par un fournisseur de cloud (voir figure 58) :

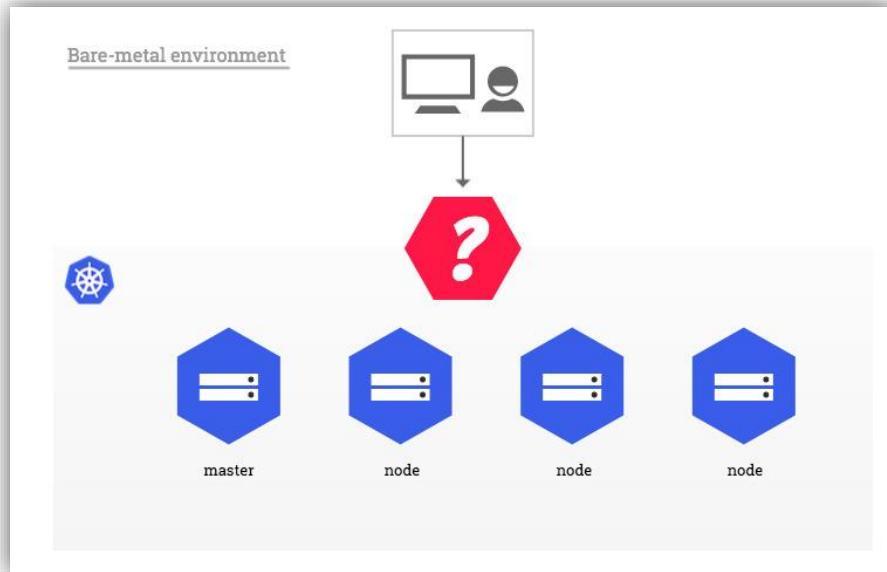


Figure44:Environnement Bare-metal

Ce problème qui nécessite une configuration légèrement différente pour offrir le même type d'accès aux consommateurs externes.

### Solution : MetalLB

MetalLB est une implémentation de Load Balancer pour les clusters Kubernetes Bare Metal, utilisant des protocoles de routage standard. Il fournit une implémentation d'équilibrage de charge réseau pour les clusters Kubernetes qui ne s'exécutent pas sur un fournisseur de cloud pris en charge, permettant efficacement l'utilisation des services LoadBalancer dans n'importe quel cluster. Pour cela, un pool d'adresses IP doit être réservé à MetalLB. Une fois que MetalLB a attribué une adresse IP externe à un service, il doit rediriger le trafic de l'IP externe vers le cluster. MetalLB utilise des protocoles standards tels qu'ARP et NDP, ou BGP.

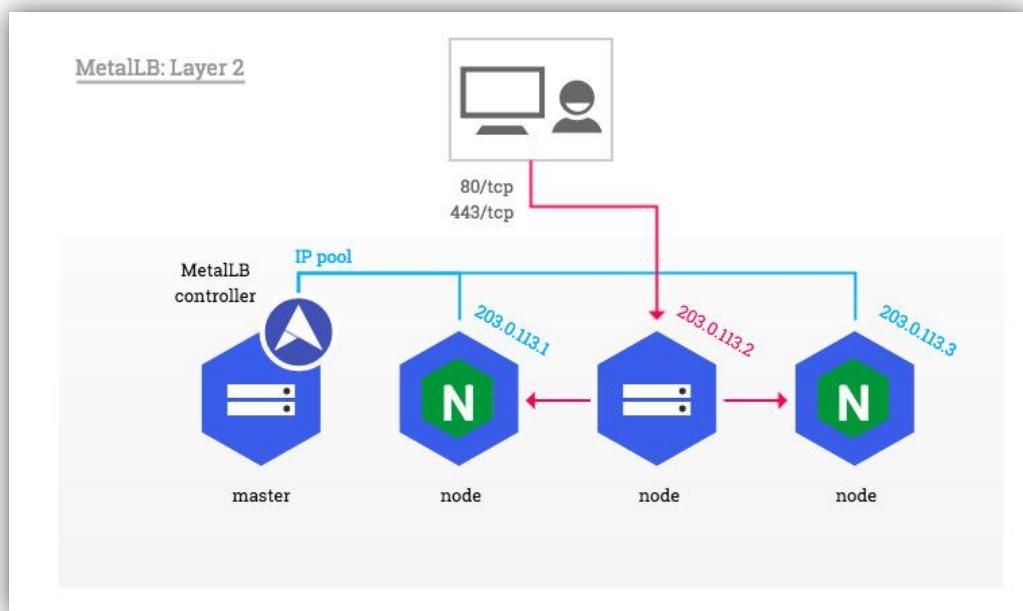


Figure45:Architecture de metallB

- Installation de metallB :

MetallB nécessite les éléments suivants pour fonctionner:

- UNE Kubernetes cluster, exécutant Kubernetes 1.13.0 ou version ultérieure, qui ne dispose pas déjà d'une fonctionnalité d'équilibrage de la charge réseau.
- UNE configuration du réseau en cluster qui peut coexister avec MetalLB.
- Quelques adresses IPv4 à distribuer par MetalLB.
- Selon le mode de fonctionnement, vous devrez peut-être un ou plusieurs routeurs capables de parler BGP.

MetalLB peut être déployé avec un simple manifeste Kubernetes ou avec Helm. Pour installer MetalLB avec un simple manifeste on suit les commandes suivantes :

```
$kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/namespace.yaml
$kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.9.3/manifests/metallb.yaml
```

Et pour utiliser d'autre méthode d'installation, on accède à la page web de MetalLB :<https://metallb.universe.tf/installation/>

- Configuration de MetalLB :

La configuration spécifique dépend du ou des protocoles que vous souhaitez utiliser pour annoncer les adresses IP des services. Sauter à:

- Configuration de la couche 2
- Configuration BGP
- Configuration avancée

Dans notre cas on a utilisé la configuration de la couche 2 car c'est la plus facile à utiliser :

Le mode de couche 2 ne nécessite que les adresses IP soient liées aux interfaces réseau de vos nœuds. Il fonctionne en répondant directement aux demandes ARP sur votre réseau local, pour donner l'adresse MAC de la machine aux clients.

Par exemple, la configuration suivante permet à MetalLB de contrôler les adresses IP de 192.168.84.150 à 192.168.84.155 et l'adresse de notre nœud est 192.168.84.148 :

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   namespace: metallb-system
5   name: config
6 data:
7   config: |
8     address-pools:
9       - name: default
10      protocol: layer2
11      addresses:
12        - 192.168.84.150-192.168.84.155
```

Figure 46:fichier configmap de MetalLB

Après avoir créé ce fichier et exécuter la commande kubectl apply , MetalLB est donc configuré, on peut maintenant créer un nouveau déploiement et lui associe un service LoadBalancer en utilisant un fichier YAML

**Remarque :** on peut créer plusieurs programmes dans un seul fichier en les séparant par trois tirets.

Après avoir créé le déploiement et le service, on exécute la commande **kubectl get services** pour lister les services créés :

```
root@meryemelhaffad-virtual-machine:~# kubectl apply -f deploymentlb.yaml
deployment.apps/nginx-deployment created
service/nginx created
root@meryemelhaffad-virtual-machine:~# kubectl get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         37h
mynginxsvc    ClusterIP  10.106.60.79  <none>        80/TCP,443/TCP  85m
mynginxsvc2   NodePort    10.110.23.4   <none>        80:30180/TCP,443:31443/TCP  56m
nginx          LoadBalancer 10.105.137.221 192.168.84.151  80:31092/TCP   12s
sample-load-balancer LoadBalancer 10.104.250.85 192.168.84.150  80:32410/TCP   44m
```

Figure 47: Service LoadBalancer créé

On remarque donc qu'une adresse IP externe est attribuée aux services de type LoadBalancer

Essayons d'exécuter cette adresse dans un navigateur web :

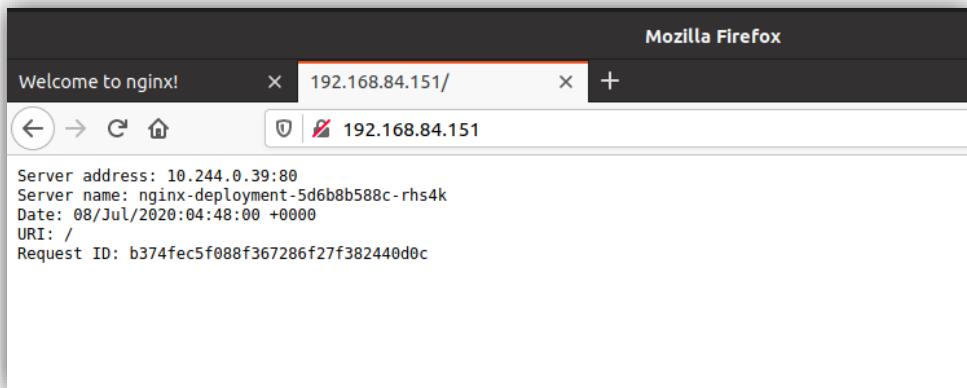


Figure 48 : Connexion au service LoadBalancer de l'extérieur

On peut également se connecter via la ligne de commande, en exécutant la commande curl et adresse IP externe

- Résolution DNS :

Après la création d'un service Load Balancer, on peut accéder à ce service à l'extérieur du cluster via l'adresse IP Externe attribué lors de la création, mais parfois on a besoin d'accéder plus facilement à ce service via un nom de domaine.

Pour effectuer un nom de domaine à ce service on modifie le fichier `/etc/hosts`, en lui ajoutant l'adresse suivie du nom du domaine souhaité. Puis on essaie maintenant d'accéder depuis le navigateur à ce nom de domaine et la connexion réussie.

**Conclusion :** En utilisant des clusters hébergés par des fournisseurs Cloud, on peut simplement créer un service LoadBalancer en utilisant un simple fichier YAML et le service sera créé ainsi que la connexion avec ce service de l'extérieur réussie, par contre en utilisant des clusters bare-metal on aura besoin d'une implémentation MetalLB.

- Ingress :

Un Ingress est un objet Kubernetes qui gère l'accès externe aux services dans un cluster, généralement du trafic HTTP. Il peut fournir un équilibrage de charge, une terminaison TLS et un hébergement virtuel **basé sur un nom**.

Ingress (ou une entrée réseau), ajouté à Kubernetes v1.1, expose les routes HTTP et HTTPS de l'extérieur du cluster à des services au sein du cluster. Le routage du trafic est contrôlé par des règles définies sur la ressource Ingress.

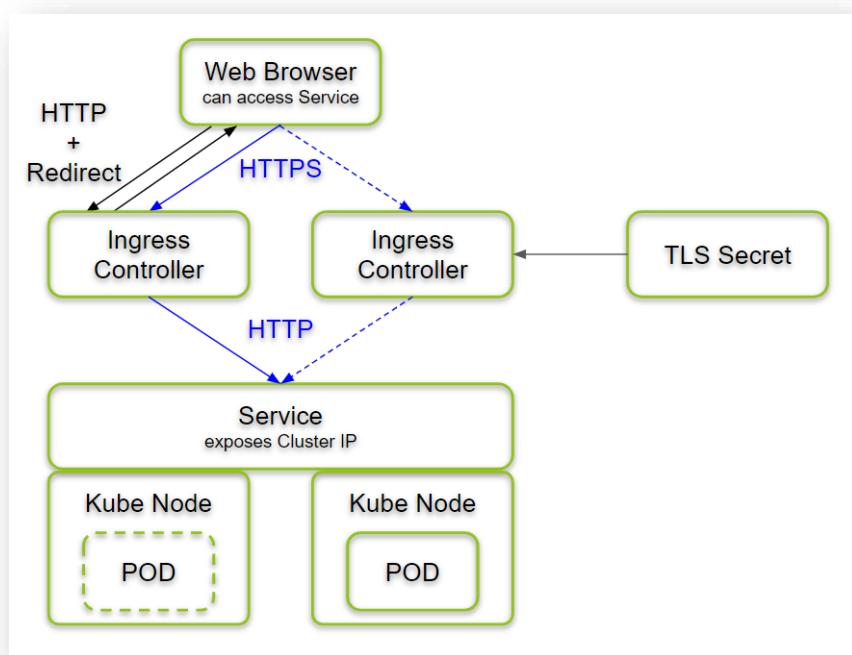


Figure 49: Architecture d'ingress

Un Ingress peut être configuré pour donner aux services des URLs accessibles de l'extérieur, du trafic de charge équilibrée, la terminaison SSL/TLS et un hébergement virtuel basé sur le nom. Un contrôleur d'Ingress est responsable de l'exécution de l'Ingress, généralement avec un LoadBalancer (équilibrEUR de charge), bien qu'il puisse également configurer votre routeur périphérique ou des interfaces supplémentaires pour aider à gérer le trafic.

## 8. Gestion des volumes :

### 1- Présentation :

Les fichiers sur disque dans un conteneur sont l'endroit le plus simple où une application peut écrire des données, mais cette approche présente des inconvénients. Les fichiers sont perdus lorsque le conteneur se bloque ou s'arrête pour toute autre raison. De plus, les fichiers d'un conteneur sont inaccessibles pour les autres conteneurs s'exécutant dans le même pod. L'abstraction Volume de Kubernetes répond à ces deux problèmes.

Conceptuellement, un volume est un répertoire accessible à tous les conteneurs d'un pod. La source du volume déclaré dans la spécification du pod détermine le mode de création du répertoire, le support de stockage utilisé et le contenu initial du répertoire. Un pod spécifie les volumes qu'il contient et le chemin d'accès où les conteneurs installent le volume.

Les types de volumes éphémères ont la même durée de vie que les pods qui les entourent. Ces volumes sont créés lors de la création du pod et ils persistent lors des redémarrages du conteneur. Lorsque le pod s'arrête ou s'il est supprimé, il en va de même pour ses volumes.

Les autres types de volumes sont des interfaces conçues pour le stockage durable qui existent indépendamment d'un pod. Contrairement aux volumes éphémères, les données d'un volume sauvegardé par un stockage durable sont préservées lorsque le pod est supprimé. Le volume est simplement démonté et les données peuvent être transférées à un autre pod. Vous devez utiliser les ressources PersistentVolume pour gérer le cycle de vie des types de stockage durable, plutôt que de les spécifier directement.

## 2- Types de volumes :

Les volumes diffèrent par leur mise en œuvre de stockage et leur contenu initial. Vous pouvez choisir la source de volume qui convient le mieux à votre cas d'utilisation. Plusieurs sources de volume courantes sont décrites ci-dessous :

- **Emptydir** : Un type de volume éphémère qui fournit un répertoire vide depuis lequel les conteneurs du pod peuvent lire et écrire. Lorsque le pod est supprimé d'un nœud pour une raison quelconque, les données d'emptyDir sont définitivement supprimées. Les volumes emptyDir sont stockés sur le support qui assure la sauvegarde du nœud. Selon votre environnement, il peut s'agir d'un disque, d'un disque dur SSD ou d'un stockage réseau. Les volumes emptyDir sont utiles pour l'espace de travail et le partage de données entre plusieurs conteneurs d'un pod.
- **ConfigMap** : La ressource ConfigMap permet d'injecter des données de configuration dans des pods. Les données stockées dans un objet ConfigMap peuvent être référencées dans un volume du même type, puis consommées via des fichiers exécutés dans un pod. Les fichiers d'un volume ConfigMap sont spécifiés par une ressource ConfigMap.
- **Secrets** : Un secret est un objet qui contient une petite quantité de données sensibles telles qu'un mot de passe, un jeton ou une clé. De telles informations pourraient autrement être placées dans une spécification de pod ou dans une image; le placer dans un objet secret permet de mieux contrôler la façon dont il est utilisé et réduit le risque d'exposition accidentelle. Les utilisateurs peuvent créer des secrets et le système crée également des secrets.
- **Persistent volume** : Les PersistentVolume sont des ressources de cluster qui existent indépendamment des pods. Cela signifie que le disque et les données représentés par un PersistentVolume continuent d'exister à mesure que le cluster change et que les pods sont supprimés et recréés. Les ressources PersistentVolume peuvent être provisionnées de manière dynamique via des ressources PersistentVolumeClaim ou peuvent être créées explicitement par un administrateur de cluster.

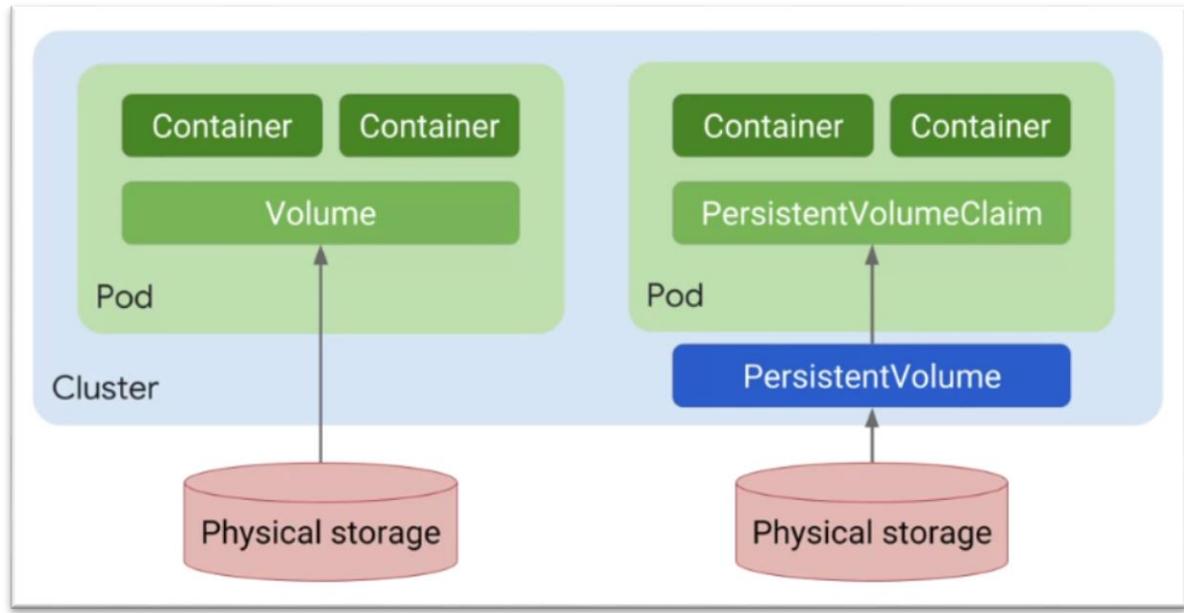


Figure 50: PersistentVolumeClaim

**PersistentVolumeClaim** est une requête et une demande d'une ressource **PersistentVolume**. Les objets **PersistentVolumeClaim** demandent une taille, un mode d'accès et une **StorageClass** spécifiques pour le **PersistentVolume**. Si un **PersistentVolume** qui répond aux conditions de la requête existe ou peut être provisionné, la **PersistentVolumeClaim** est associée à ce **PersistentVolume**.

Les pods utilisent les demandes sous forme de volumes. Le cluster inspecte la demande pour rechercher le volume associé et l'installe pour le pod.

**StorageClass** par défaut est utilisée lorsqu'une **PersistentVolumeClaim** ne spécifie pas de **StorageClassName**. Vous pouvez remplacer la **StorageClass** fournie par défaut par celle que vous avez définie.

Nous avons déjà terminé les deux premières étapes. Créons donc des pods à l'aide du déploiement. En spécifiant dans la section des volumes, le nom du claim qui est créé dans les étapes ci-dessus. De cette façon, nous pouvons ajouter un claim dans n'importe quel fichier manifeste. En utilisant la section des montages de volume, nous montons le répertoire / var / log / nginx dans pvc. Ici, nous avons monté / var / log / nginx sur pvc appelé ‘mypvc’. Donc, quelles que soient les données générées dans / var / log / nginx qui seront disponibles dans le volume persistant du nœud, i.e / devops / data.

Après avoir créé un volume persistant dans le répertoire / devops / data du nœud. Allons donc vérifiant les fichiers / var / log / nginx du conteneur, dans le répertoire / devops / data du nœud.

Si ce répertoire contient les fichiers access.log et error.log alors nous avons donc créé avec succès un volume persistant, un volume persistant claim et utilise ce claim dans le pod.

## 9. Sécurité

Nous avons appris à accéder aux services depuis l'extérieur du cluster. Intéressons-nous maintenant à leur sécurité. Les pods peuvent communiquer entre eux par défaut. Mais comment faire si nous devons limiter l'accès à certains pods ? En mettant donc en place **une règle de réseau (Network Policy)**.

Un ensemble de règles de fire-wall au niveau des pods qui restreignent l'accès aux autres pods et services du cluster. Les règles de réseau vous permettent de limiter les connexions entre les pods. Par conséquent, l'utilisation de règles de réseau offre une sécurité supérieure en réduisant le risque de problème de sécurité.

La majeure partie de la sécurisation du trafic réseau tourne généralement autour de la définition des règles Egress et Ingress. Du point de vue d'un pod Kubernetes, Ingress est le trafic entrant vers le pod et Egress est le trafic sortant du pod. Dans la stratégie réseau de Kubernetes, vous créez des règles d'entrée et de sortie «autoriser» indépendamment (Ingress, Egress ou les deux).

### a- Limiter le trafic entrant vers les pods :

Commencez par exécuter une application simple de serveur Web associée à l'étiquette **app=hello**, puis exposez-la en interne dans le cluster :

```
yashine@master-node:~$ kubectl run hello-web --labels app=hello \
>   --image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose
service/hello-web created
pod/hello-web created
```

Figure 51:exécution d'une application simple de serveur web

Ensuite, vous devez configurer une ressource **Networkpolicy** pour autoriser le trafic vers les pods hello-web provenant seulement des pods **app=foo**. Le reste du trafic entrant provenant de pods qui ne portent pas ce libellé, le trafic externe et le trafic provenant de pods situés dans d'autres espaces de noms sont bloqués.

Cette règle sélectionne les pods portant l'étiquette **app=hello** et spécifie une règle d'entrée pour autoriser le trafic provenant seulement des pods portant l'étiquette **app=foo**.

## b- Limiter le trafic sortant des pods:

Pour appliquer les règles de réseau de sortie :

Déployez une ressource **Networkpolicy** contrôlant le trafic sortant des pods portant l'étiquette **app=foo**, tout en n'autorisant le trafic que vers les pods portant l'étiquette **app=hello** ainsi que le trafic DNS :

```
yashine@master-node:~$ kubectl apply -f foo-allow-to-hello.yaml
networkpolicy.networking.k8s.io/foo-allow-to-hello created
```

Figure 52:Création du pod pour le trafic entrant

Ce fichier manifeste spécifie une règle de réseau contrôlant le trafic de sortie des pods portant l'étiquette **app=foo** avec deux destinations autorisées :

- Les pods situés dans le même espace de noms et portant l'étiquette **app=hello**
- Les pods du cluster ou les points de terminaison externes sur le port 53 (UDP et TCP).

**Conclusion :** Dans cette partie on a montré comment kubernetes pris en charge les stratégies réseau (NetworkPolicy) pour préciser comment les groupes de pods sont autorisés à communiquer entre eux et avec d'autres points de terminaison réseau.

## 10. Interfaces Graphiques :

### 1- Dashboard

Le tableau de bord (Dashboard) est une interface web pour Kubernetes. Vous pouvez utiliser ce tableau de bord pour déployer des applications conteneurisées dans un cluster Kubernetes, dépanner votre application conteneurisée et gérer les ressources du cluster. Vous pouvez utiliser le tableau de bord pour obtenir une vue d'ensemble des applications en cours d'exécution dans votre cluster, ainsi que pour créer ou modifier des ressources Kubernetes individuelles. (Comme des Deployments, Jobs, Daemonsets, etc). Par exemple, vous pouvez

redimensionner un Deployment, lancer une mise à jour progressive, recréer un pod ou déployez de nouvelles applications à l'aide d'un assistant de déploiement.

Le tableau de bord fournit également des informations sur l'état des ressources Kubernetes de votre cluster et sur les erreurs éventuelles.

- **Visualisation du cluster :**

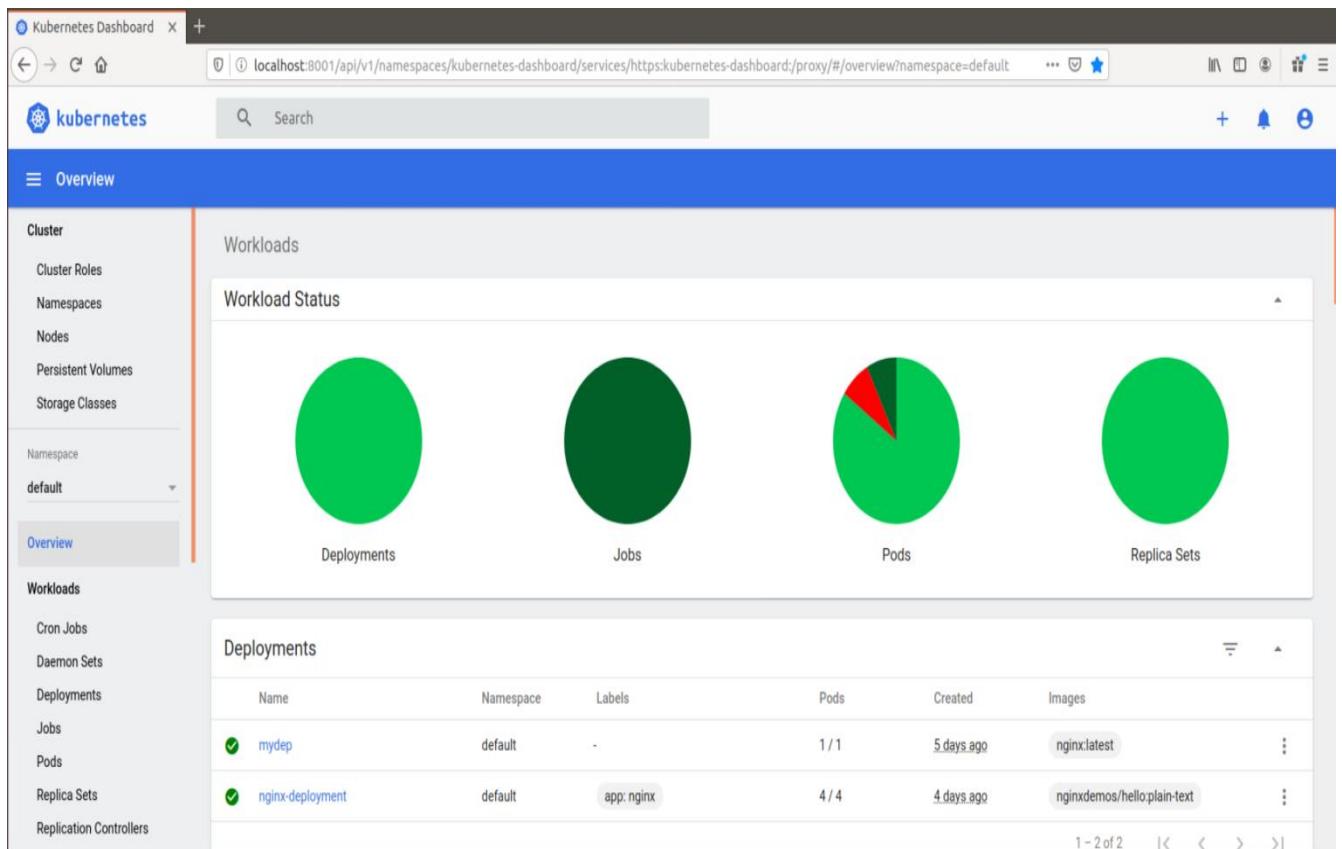


Figure 53:Page d'entrée du Dashboard

## 2- Kentora Lens

Lens est le seul IDE dont vous aurez besoin pour prendre le contrôle de vos clusters Kubernetes. Il s'agit d'une application autonome pour les systèmes d'exploitation MacOS, Windows et Linux. Il est open source et gratuit.

Parmi ses caractéristiques :

- Prise en charge de centaines de clusters
- Application autonome; Pas besoin d'installer quoi que ce soit en cluster

- Visualisation de l'état du cluster en temps réel
- Graphiques et tendances d'utilisation des ressources avec l'histoire alimentée par Prométhée intégré
- Accès terminal aux nœuds et conteneurs
- Performances optimisées pour gérer des clusters massifs (testés avec un cluster exécutant des pods de 25k)
- Soutien total à Kubernetes RBAC

### **3- Octant :**

Octant est une plate-forme web très extensible pour les développeurs afin de mieux comprendre la complexité des clusters Kubernetes.

Octant est un outil permettant aux développeurs de comprendre comment les applications s'exécutent sur un cluster Kubernetes. Il vise à faire partie de la boîte à outils du développeur pour obtenir un aperçu et l'approche de la complexité trouvée dans Kubernetes. Octant offre une combinaison d'outillage introspectif, de navigation en cluster et de gestion d'objets ainsi qu'un système de plugin pour étendre davantage ses capacités.

#### **▪ Fonctionnalité :**

<b>Visionneuse de ressources</b>	Visualiser graphiquement les relations entre les objets d'un cluster Kubernetes. L'état des objets individuels est représenté par la couleur pour afficher les performances de la charge de travail.
<b>Affichage récapitulatif</b>	Informations d'état et de configuration consolidées dans une seule page agrégée à partir de la sortie généralement trouvée à l'aide de plusieurs commandes kubectl.
<b>Port Forward</b>	Transférer un port local vers une nacelle en cours d'exécution avec un seul bouton pour les applications de débogage et même transférer plusieurs gousses dans les espaces de noms.
<b>Flux de journaux</b>	Affichez les flux de journaux de l'activité des gousses et des conteneurs pour le dépannage ou la surveillance sans tenir plusieurs terminaux ouverts.

<b>Filtre d'étiquette</b>	Organisez les charges de travail avec le filtrage d'étiquettes pour l'inspection des clusters avec un volume élevé d'objets dans un espace de noms.
<b>Cluster Navigation</b>	Modifiez facilement entre les espaces de noms ou les contextes entre différents clusters. Plusieurs fichiers kubeconfig sont également pris en charge.
<b>Plugin System</b>	Système de plugin très extensible pour les utilisateurs de fournir des fonctionnalités supplémentaires via gRPC. Les auteurs de plugins peuvent ajouter des composants au-dessus des vues existantes.

Tableau 17:Fonctionnalités d'Octant

## II. Kubernetes VS Docker Swarm

### 1. Différence entre Docker swarm et Kubernetes :

L'orchestration de conteneurs évolue rapidement et Kubernetes et Docker Swarm sont les deux principaux acteurs dans ce domaine. Kubernetes et Docker Swarm sont des outils importants qui sont utilisés pour déployer des conteneurs à l'intérieur d'un cluster. Ils ont de nombreux USP et pros de niche dans le domaine et ils sont là pour rester. Bien que les deux aient une manière tout à fait différente et unique d'atteindre les objectifs, à la fin de la journée, leur point final reste assez proche.

Bien que les deux outils d'orchestration soient assez similaires, il existe des variations et des différences techniques et fonctionnelles qui les rendent différents et également bons ou mauvais dans leurs propres niches. Certains de ces points sont énumérés ci-dessous.

Fonctionnalités	Kubernetes	Docker Swarm
Installation	Installation complexe mais un cluster résultant solide une fois mis en place	Installation simple mais le cluster résultant n'est pas relativement fort
GUI	Kubernetes dispose de tableaux de bord détaillés pour permettre même aux utilisateurs non techniques de contrôler efficacement les clusters	Docker Swarm, en variante, nécessite un outil tiers tel que Portainer.io pour gérer facilement l'interface utilisateur.
Scalability	Kubernetes agit comme un framework tout-en-un plus lorsque vous travaillez avec des systèmes distribués. Il est donc beaucoup plus compliqué car il offre de fortes garanties en termes d'état de cluster et un ensemble unifié d'API. La mise à l'échelle et le déploiement des conteneurs sont donc ralentis.	Docker Swarm peut déployer des conteneurs beaucoup plus rapidement que Kubernetes, ce qui permet des temps de réaction plus rapides pour une mise à l'échelle à la demande.
L'équilibrage de charge (Load Balancing)	Dans Kubernetes, les pods sont exposés via le service, ce qui leur permet d'être implémentés en tant qu'équilibrEUR de charge dans un cluster. Ingress est généralement utilisée pour l'équilibrage de charge.	Le mode Swarm est livré avec un élément DNS qui peut être utilisé pour distribuer les demandes entrantes à un nom de service. Ainsi, les services peuvent être attribués automatiquement ou fonctionner sur des ports prédéfinis par l'utilisateur.
Rollback	Rollback automatique avec possibilité de déployer des mises à jour continues	La fonction de restauration automatique n'est disponible que dans Docker 17.04 et versions supérieures si les mises à jour de service ne se déplient pas
Enregistrement et surveillance	Outils intégrés disponibles pour la journalisation et la surveillance	Manque d'outils intégrés. Nécessite des outils tiers à cet effet
Prise en charge des nœuds	Prends en charge jusqu'à 5000 nœuds	Prends en charge +2000 nœuds
Cible d'optimisation	Optimisé pour un seul grand cluster	Optimisé pour plusieurs petits clusters
Mises à jour	Les mises à jour de cluster en place mûrisent constamment	Le cluster peut être mis à niveau sur place
La mise en réseau	Kubernetes a un modèle de réseau plat, permettant à tous les pods de communiquer entre eux. Des stratégies réseau sont en place pour définir la façon dont les pods interagissent les uns avec les autres. Le réseau est généralement implanté en overlay, nécessitant deux CIDRS pour les services et les pods.	Lorsqu'un nœud rejoint un cluster swarm, il crée un réseau overlay pour les services pour chaque hôte dans Docker swarm. Il crée également un réseau pont Docker bridge uniquement pour les conteneurs. Cela donne aux utilisateurs un choix tout en chiffrant le trafic de données du conteneur pour créer son propre réseau overlay.
Disponibilité	Haute disponibilité. Les contrôleurs de santé sont effectués directement sur les pods	Haute disponibilité Les conteneurs sont redémarrés sur un nouvel hôte en cas de défaillance d'un hôte

Tableau 18:Kubernetes Vs Swarm

La figure ci-dessous, représente les fonctionnalités existant dans docker swarm et kubernetes :

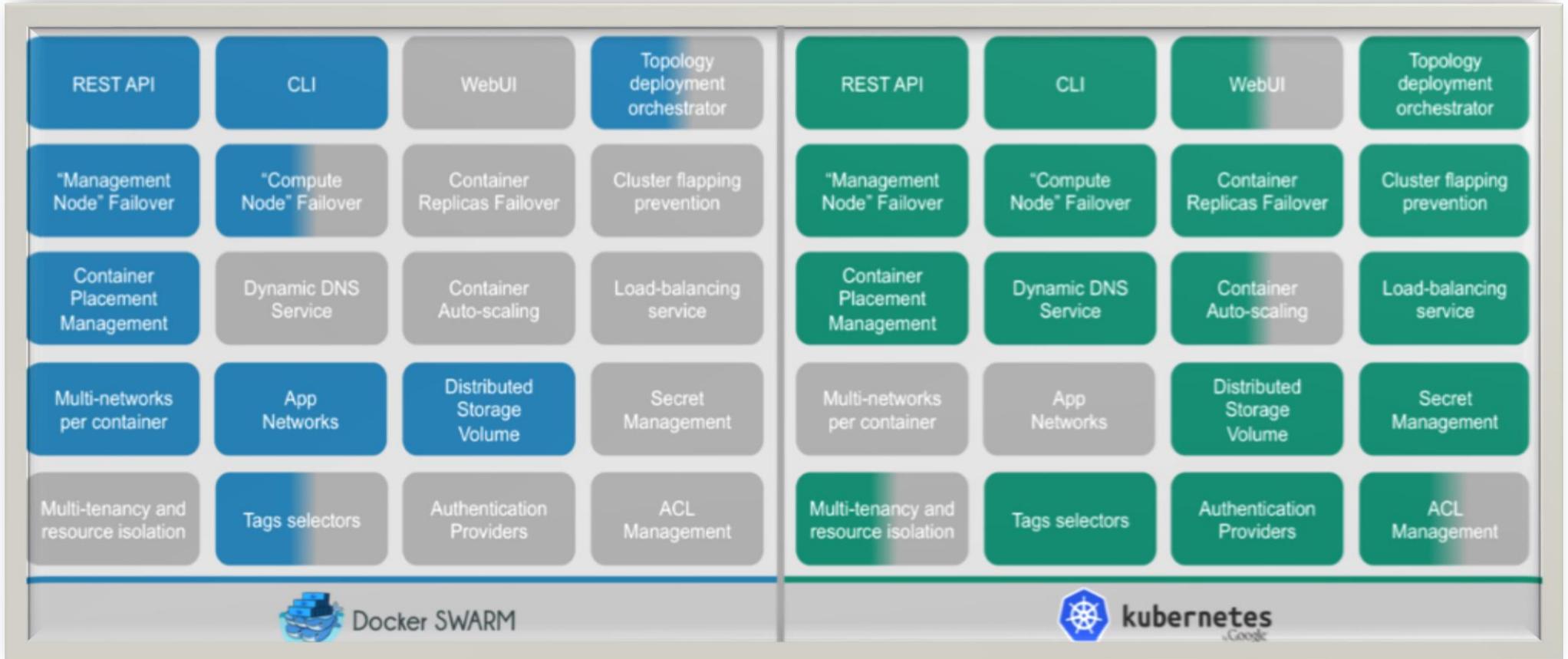


Figure 54:Kubernetes vs Swarm

## **Maintenant lequel devriez-vous utiliser: Kubernetes ou Docker Swarm?**

Docker Swarm est un bon choix si vous avez besoin d'une configuration rapide et n'avez pas d'exigences de configuration exhaustives. Il fournit efficacement des logiciels et des applications avec une architecture basée sur des microservices. Ses principaux points positifs sont la simplicité d'installation et une courbe d'apprentissage progressive. En tant qu'application autonome, elle est parfaite pour le développement et les tests de logiciels. Ainsi, avec un déploiement facile et une configuration automatisée, il utilise également moins de ressources matérielles, ce pourrait être la première solution à envisager.

L'inconvénient est que les outils de surveillance natifs font défaut et que l'API Docker limite les fonctionnalités. Mais il offre toujours un réseau overlay, un équilibrage de charge, une haute disponibilité et plusieurs fonctionnalités d'évolutivité.

### **Verdict final :**

Docker Swarm est idéal pour les utilisateurs qui souhaitent configurer une application conteneurisée et la mettre en service beaucoup de retard.

Kubernetes serait la meilleure plate-forme de conteneurisation à utiliser si l'application que vous développez est complexe et utilise des centaines de milliers de conteneurs dans le processus. Il dispose de politiques de haute disponibilité et de capacités de mise à l'échelle automatique. Malheureusement, la courbe d'apprentissage est abrupte et peut gêner certains utilisateurs. Le processus de configuration et d'installation est également long.

Kubernetes est destiné aux utilisateurs qui sont à l'aise avec la personnalisation de leurs options et qui ont besoin de fonctionnalités étendues.

## **Maintenant la question qui se pose est-ce qu'on peut utiliser les deux orchestrateurs ensemble ?**

### **2. Interopérabilité des deux orchestrateurs :**

Docker a annoncé en Mars 2017, l'intégration de Kubernetes à sa plateforme de pilotage d'architectures containérées. Swarm, son orchestrateur maison, ne serait alors plus le seul à être proposé avec Docker Enterprise Edition (Docker EE).

Les outils de développement de Docker deviennent utilisables pour construire des architectures Kubernetes, sans prise en charge des serveurs Windows. La jonction permet par ailleurs de renforcer la sécurité avec des pods comme Content Trust (certification de la qualité

des conteneurs) ou Security Scanning (déttection des vulnérabilités dans les images des conteneurs). Comparé à ce qu'offre Kubernetes, Docker EE permet une gestion des politiques d'accès plus granulaires, avec la possibilité de définir des droits en lecture et/ou écriture jusqu'à chaque objet de l'API Kubernetes. Vous pourrez même réaliser un partitionnement logique et physique du cluster, et ainsi assigner par exemple des équipes de développeurs à telle ou telle machine. Ce qui n'est pas réalisable nativement via le « role based access control » de Kubernetes.

**Remarque :** Docker Community Edition (CE) est le nouveau nom des produits Docker gratuits.

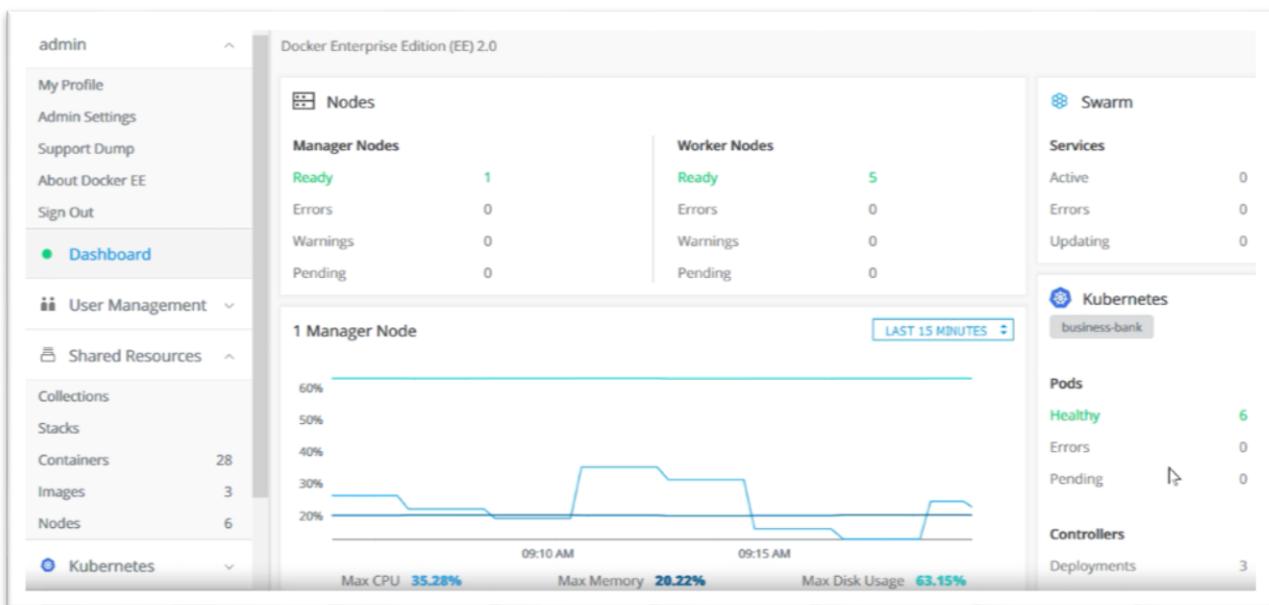


Figure 55:Docker EE Dashboard avec des conteneurs déployés avec Swarm et kubernetes

Par ailleurs, il faut noter que dans le cadre de Docker EE 2.0, les organisations disposent de ces fonctionnalités qui vont leur fournir plus de choix et leur éviter d'être verrouillés sur une architecture ou une technologie :

<b>Prise en charge de plusieurs systèmes d'exploitation</b>	disponibilité sur des plateformes d'infrastructure certifiées, y compris plusieurs distributions Linux (SLES, CentOS, RHEL, Ubuntu, Oracle Linux) et Windows Server.
<b>Multi-Cloud</b>	les entreprises ne sont pas confinées à une infrastructure sous-jacente et bénéficient de la plus grande flexibilité dans les déploiements de cloud hybrides sur tous les principaux clouds, notamment AWS et Azure.

<b>Choix d'orchestration</b>	Docker EE exécute à la fois Swarm et Kubernetes simultanément sur le même cluster - les développeurs n'ont donc pas besoin de faire un choix d'orchestration. Les équipes d'exploitation ont la possibilité de choisir les orchestrateurs de façon interchangeable.
<b>Réseautage</b>	Docker EE offre une mise en réseau sécurisée intégrée via Project Calico par et en collaboration avec Tigera, partenaire d'intégration de Docker pour Calico. Avec cette intégration CNI, les entreprises bénéficient d'une solution Kubernetes entièrement supportée avec Project Calico - la seule qui fonctionne de manière uniforme sur les principaux systèmes d'exploitation Linux et les principaux fournisseurs de cloud. Les sociétés disposant de plug-ins de réseau certifiés ou certifiés sur Docker Enterprise Edition 2.0 incluent: Cisco Contiv, Infoblox et Weaveworks.
<b>Stockage</b>	Les entreprises avec des plugins de volume certifiés ou certifiés sur Docker Enterprise Edition 2.0 incluent: Blockbridge, Dell EMC, Hedvig, HPE / Nimble, NetApp, Nexenta, Portworx, Pure Storage, StorageOS, Veritas, Virtuozzo.

Tableau 19:Fonctionnalité de docker EE



Figure 56: Architecture de docker EE

Docker Enterprise Edition est disponible en trois versions : basique, standard et avancée. L'édition basic est livré avec la plate-forme Docker, le support et la certification, tandis que la version standard et avancée ajoutent des fonctionnalités supplémentaires telles que la gestion des conteneurs (Docker Datacenter) et Docker Security Scanning.

**Remarque :** Toutes les versions de docker EE sont payantes c'est ce qui nous a empêché de réaliser la partie pratique de cette partie

## Conclusion générale :

Dans ce rapport, nous avons présenté le concept de la plate-forme opensource Docker.

Vu que ce projet a requis un temps important de recherche, on a pu approfondir nos connaissances dans ce sujet.

Au début on n'avait absolument aucune idée comment fonctionne docker et à quoi sert mais on était et on reste toujours intéressé par le sujet.

On a commencé par une étude générale et théorique sur cette plate-forme et ses composantes afin de comprendre le concept, ensuite on est passé à la partie pratique pour comprendre beaucoup mieux le principe qui se base sur la conteneurisation au lieu de virtualisation, ceci nous a amené à faire une étude comparative des deux.

Après, on s'est intéressé à la partie réseau de docker, plus précisément la partie d'orchestration des conteneurs, Nous avons fait une étude comparative des deux orchestrateurs Docker Swarm et Kubernetes, cette étude était pour nous une bonne occasion d'acquérir des nouvelles connaissances sur les orchestrateurs de conteneurs et de bien comprendre la différence profonde entre les deux, c'est ce qui est illustré dans la deuxième partie.

Enfin, on a pu affronter des différentes difficultés que nous avons rencontrées, mais certaines parmi les difficultés sont difficiles à résoudre et c'étaient essentiellement à propos des ressources insuffisantes dans nos machines qui nous ont empêché à réaliser quelques parties.

## Liste des abréviations :

<b>API</b>	Application Programming Interface
<b>CLI</b>	Command line Interface
<b>RAM</b>	Random Access Memory
<b>CPU</b>	Central Processing Unit
<b>VM</b>	Virtual Machine
<b>OS</b>	Operating System
<b>CNM</b>	Container Network Model
<b>K8S</b>	Kubernetes
<b>SSL</b>	Secure Socket Layer
<b>TLS</b>	Transport Layer Security
<b>CSR</b>	Certificate Signing Request
<b>HTTP</b>	HyperText Transfer Protocol
<b>HTTPS</b>	HyperText Transfer Protocol Over Secure Socket Layer
<b>IP</b>	Internet Protocol
<b>ID</b>	IDentifier
<b>RBAC</b>	Role Based Access Control
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>NFS</b>	Network File Service
<b>FTP</b>	File Transfer Protocol
<b>RFC</b>	Request for comments
<b>SMP</b>	Server Message Block
<b>UDP</b>	User Datagram Protocol
<b>IETF</b>	Internet Engineering Task Force
<b>DNS</b>	Domain Name System
<b>CRI</b>	Container Runtime Interface
<b>GCP</b>	Google Cloud Platform
<b>SDK</b>	Software Développement kit
<b>UTC</b>	Universel Temps Coordonné
<b>CNI</b>	Container Network interface
<b>ARP</b>	Adresse Resolution Protocol
<b>NDP</b>	Neighbor Discovery Protocol
<b>BGP</b>	Border Gateway Protocol

<b>MAC</b>	Media Access Control
<b>GCE</b>	Google Cloud Engine
<b>GKE</b>	Google Kubernetes Engine
<b>SSD</b>	Solid-state-drive
<b>PV</b>	PersistentVolume
<b>PVC</b>	PersistentVolumeClaim
<b>USP</b>	Unique Selling Proposition
<b>Docker EE</b>	Docker Enterprise Edition
<b>Docker CE</b>	Docker Community Edition

# Références:

Docker : Pratique des architectures à base de conteneurs Ed. 2 Auteur: Cloux, Pierre-Yves, Garlot, Thomas, Kohler, Johann Editeur: Dunod Année de Publication: 2019

Kubernetes : Maîtrisez l'orchestrateur des infrastructures du futur Auteur: Hightower, Kelsey, Burns, Brendan, Beda, Joe Editeur: Dunod Année de Publication: 2019

<https://techrocks.ru/2019/04/19/docker-guide-for-beginners/>

<https://timeweb.com/ru/community/articles/docker-konteynery-eto-prosto-1>

<https://www.docker.com/resources/what-container>

<https://www.supinfo.com/articles/single/6400-installation-utilisation-docker>

<https://actu.alfa-safety.fr/devops/conteneur-docker-fonctionnement-et-avantages-pour-heberger-ses-applications/>

<https://eternalhost.net/blog/razrabortka/chto-takoe-docker>

<https://medium.com/codingthesmartway-com-blog/docker-beginners-guide-part-1-images-containers-6f3507fffc98>

<https://linux-notes.org/rabota-s-tomami-volumes-v-docker/>

<https://www.tune-it.ru/web/adpashnin/blog/-/blogs/docker-volumes>

<https://www.lemagit.fr/conseil/Containers-Docker-quelles-sont-les-options-pour-le-stockage-persistant>

<https://habr.com/en/company/ruvds/blog/450312/>

<https://runnable.com/docker/basic-docker-networking>

<https://habr.com/en/post/333874/#obzor-setey-docker>

<https://www.oreilly.com/content/docker-networking-service-discovery/>

<https://codeburst.io/networking-docker-containers-c4f48339a0df>

<https://dev.to/mozartted/docker-networking--how-to-connect-multiple-containers-7fl>

<https://kerneltalks.com/networking/how-docker-container-dns-works/>

<https://blog.newrelic.com/engineering/container-orchestration-explained/>

<https://avinetworks.com/glossary/container-orchestration/>

<https://www.redhat.com/en/topics/containers/what-is-container-orchestration>

<https://kubernetes.io/docs/concepts/workloads/pods/pod/>

[https://www.tutorialspoint.com/kubernetes/kubernetes\\_pod.htm](https://www.tutorialspoint.com/kubernetes/kubernetes_pod.htm)

