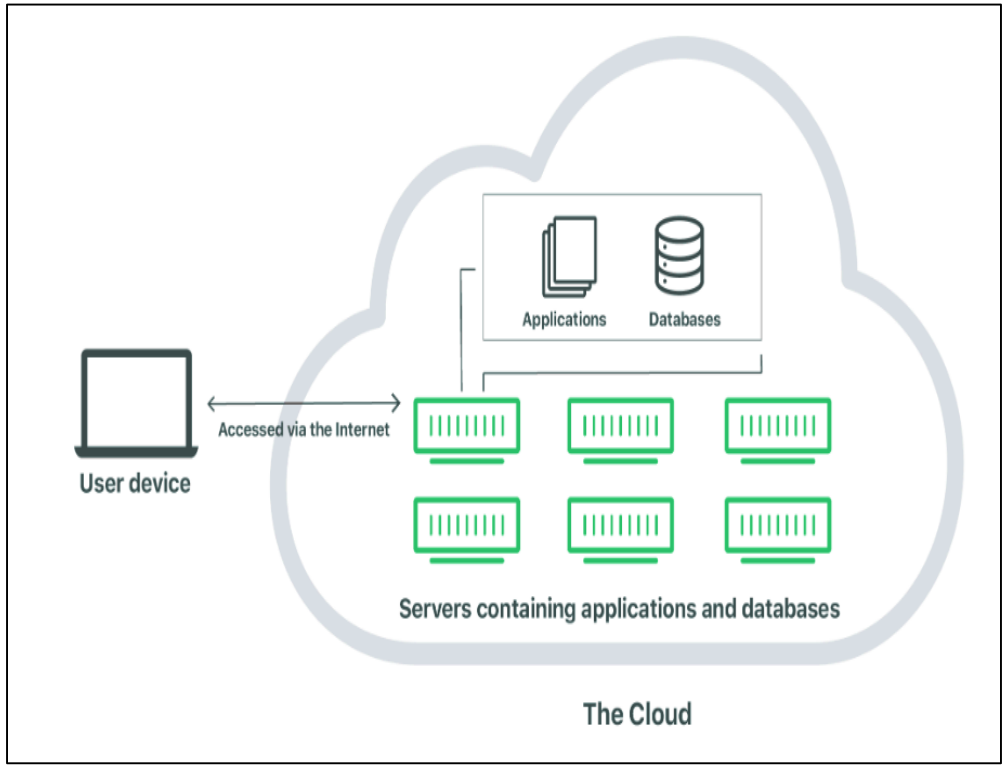
DOCKER

DÉFINITION DU CLOUD

Le terme « cloud » désigne les serveurs accessibles sur Internet, ainsi que les logiciels et bases de données qui fonctionnent sur ces serveurs. Les serveurs situés dans le cloud sont hébergés au sein de datacenters répartis dans le monde entier. L'utilisation du cloud computing permet aux utilisateurs et aux entreprises de s'affranchir de la nécessité de gérer des serveurs physiques eux-mêmes ou d'exécuter des applications logicielles sur leurs propres équipements.



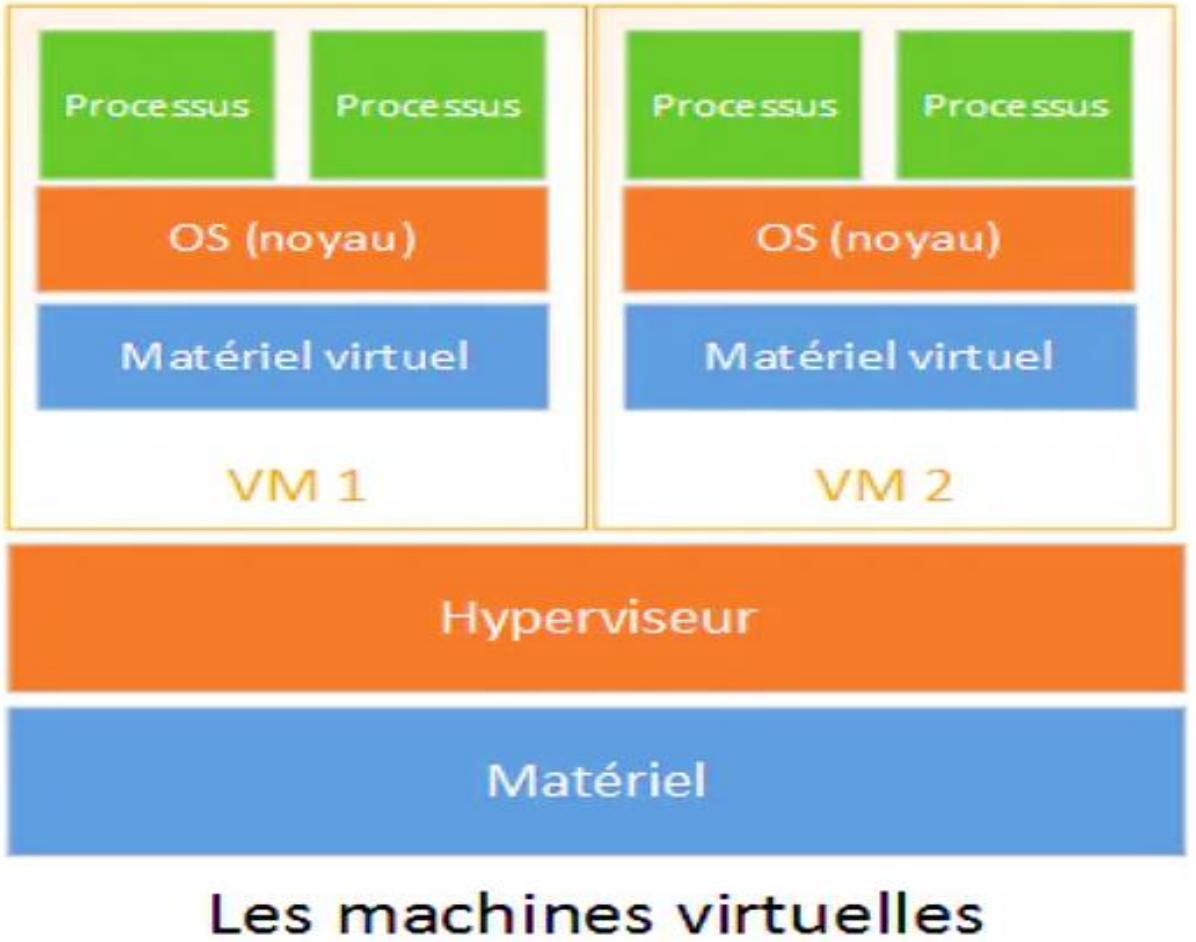
LA VIRTUALISATION

- La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique.
- La virtualisation est une technologie qui permet de créer et d’exécuter une version virtuelle des machines physiques, ainsi que des ressources associées.

LA VIRTUALISATION

La virtualisation est une technologie qui vous permet de ***créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique***.
Son logiciel, appelé ***hyperviseur***, est directement relié au matériel et vous permet de fragmenter ce système unique en plusieurs environnements sécurisés distincts. C'est ce que l'on appelle les machines virtuelles.

LA VIRTUALISATION



Les machines virtuelles

LA VIRTUALISATION ≠ CLOUD COMPUTING

Si le Cloud Computing s'appuie sur la virtualisation, les deux concepts diffèrent :

- la virtualisation permet de désolidariser les environnements informatiques de leurs machines. Comment? => *Chaque machine virtuelle dispose de ses propres ressources allouées, telles que la mémoire, le processeur, le stockage, etc*

- le Cloud Computing consiste à héberger et exploiter des données sur des serveurs distants, par le biais du réseau internet. Comment ? => *Cloud Computing vise à fournir des services informatiques à la demande sur Internet*

LE CLOUD COMPUTING

- Le cloud computing désigne un ensemble de principes et d'approches visant à ***mettre à la disposition d'utilisateurs qui en font la demande,*** quel que soit le réseau, des ressources de calcul, de réseau et de stockage, ainsi que des services, des plateformes et des applications.
- Ces ressources d'infrastructure, services et applications proviennent d'un cloud, c'est-à-dire ***d'un pool de ressources virtuelles orchestrées par des logiciels de gestion et d'automatisation et accessibles à la demande***.

LA VIRTUALISATION ≠ CLOUD COMPUTING

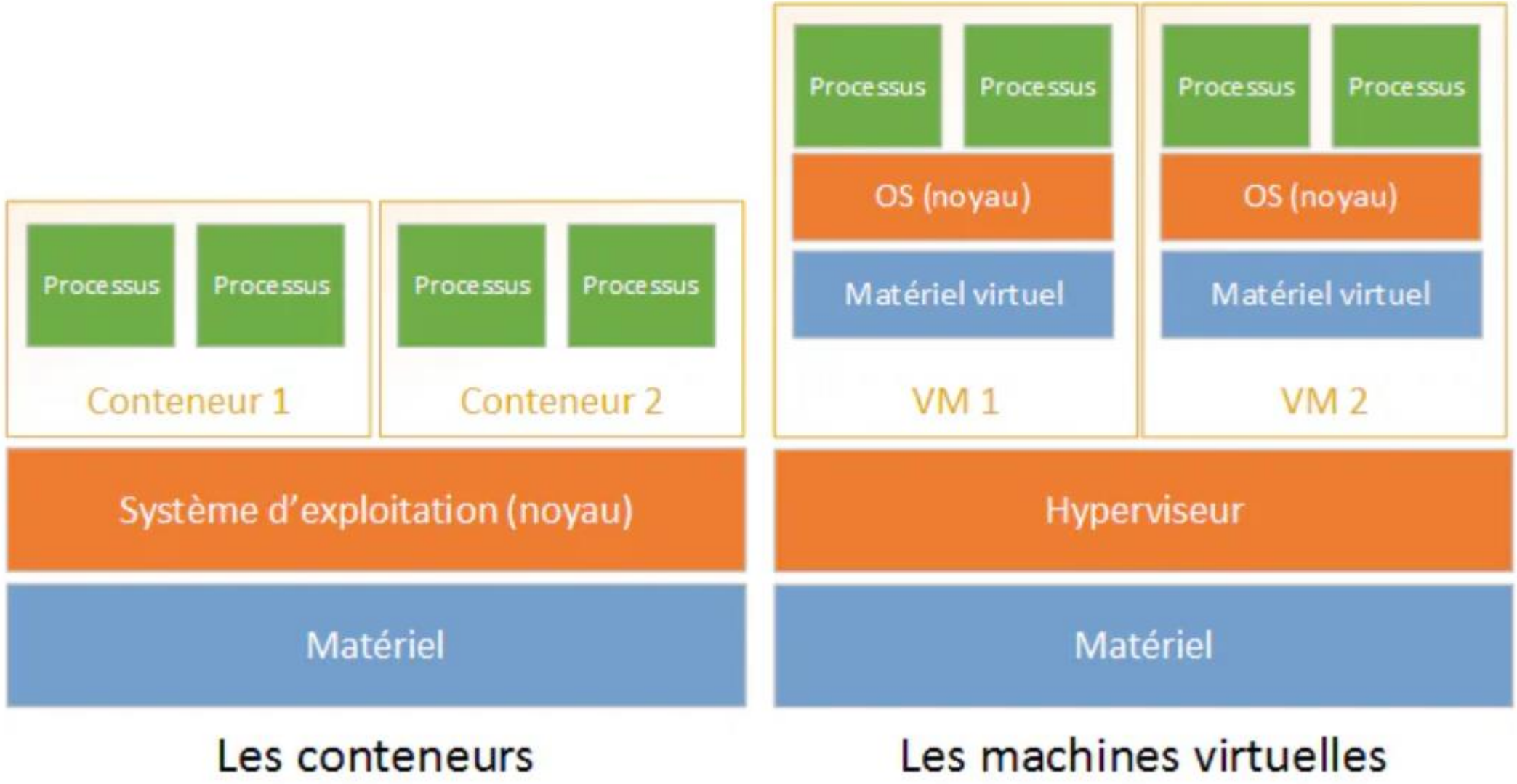
	Virtualisation	**Cloud Computing**
Objet	Créer plusieurs environnements simulés à partir d'un même système physique	Regrouper et automatiser des ressources virtuelles pour une utilisation à la demande
Utilisation	Fournir des ressources en paquets à des utilisateurs spécifiques pour une tâche spécifique	Fournir des ressources variables à des groupes d'utilisateurs pour diverses tâches
Configuration	À partir d'une image	A partir d'un modèle
Durée	Années (long terme)	Heure ou mois (court terme)

MACHINES VIRTUELLES ≠ CONTENEURS

- La notion de conteneur est aussi souvent rapprochée de celle de virtualisation. Mais une fois de plus, des différences existent.
- Comme son nom l'indique, ***une machine virtuelle est l'imitation virtuelle d'un appareil informatique créée, dans le cadre de la virtualisation, à l'aide d'un logiciel hyperviseur, et doté d'un système d'exploitation (ou OS) complet.***

- La virtualisation par conteneurisation, quant à elle, ***consiste à cloisonner directement au niveau du système d'exploitation.*** Ainsi, chaque conteneur exécute son environnement, ***mais partage le même OS hôte***. C'est pour cette raison que les conteneurs servent généralement à la virtualisation d'un programme, et non d'un serveur dans son intégralité

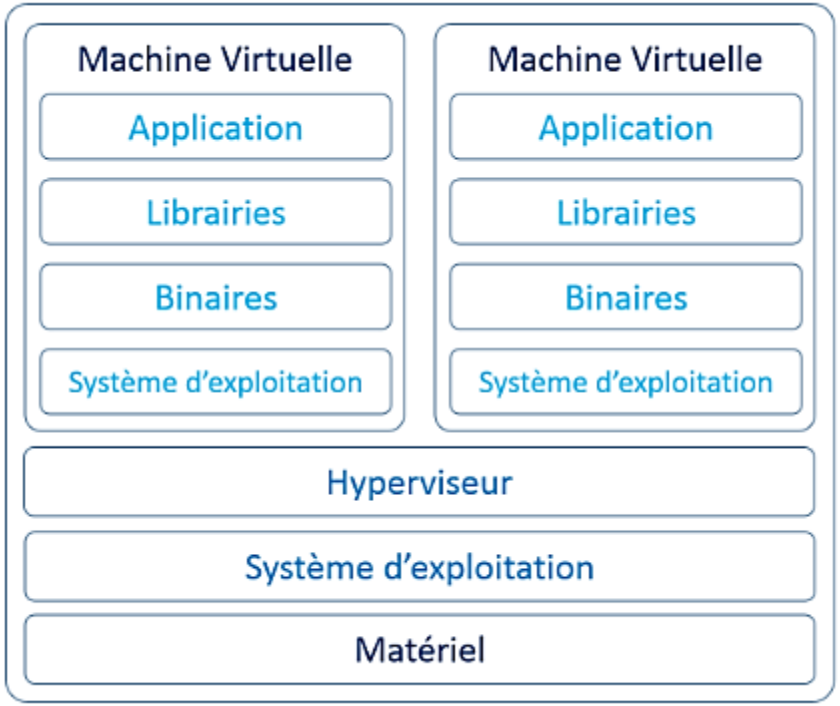
MACHINES VIRTUELLES ≠ CONTENEURS

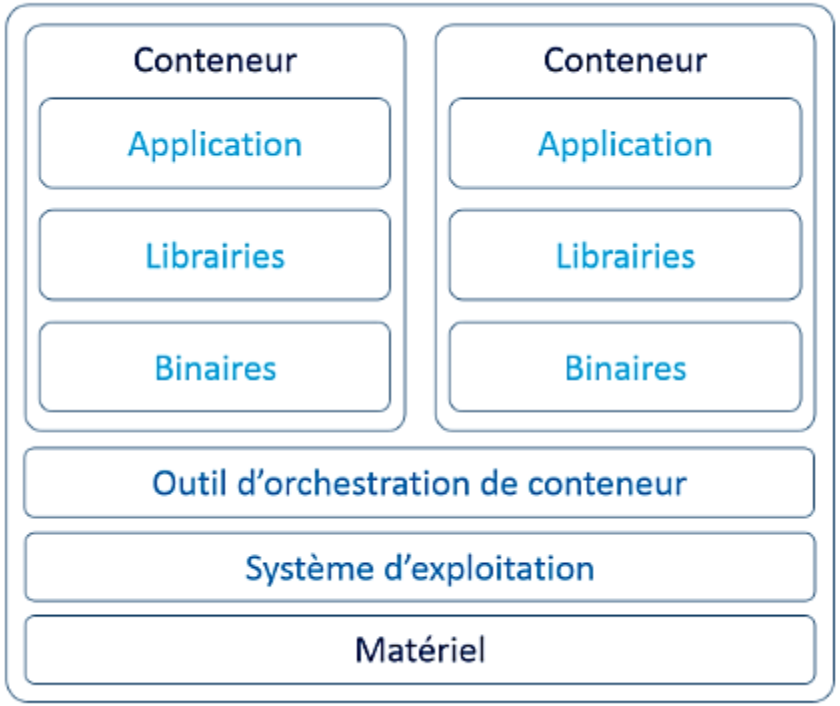


MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	Dans le cas de la virtualisation traditionnelle avec des machines virtuelles, chaque machine virtuelle dispose de son propre système d'exploitation. Ainsi, lors de l'exécution d'applications intégrées à des machines virtuelles, l'utilisation de la mémoire peut être supérieure à ce qui est nécessaire et les machines virtuelles peuvent commencer à utiliser les ressources requises par l'hôte.
Conteneur	Contrairement aux applications classiques, les applications conteneurisées partagent un environnement de système d'exploitation (noyau), elles utilisent donc moins de ressources que des machines virtuelles complètes et réduisent la pression sur la mémoire de l'hôte

MACHINES VIRTUELLES ≠ CONTENEURS

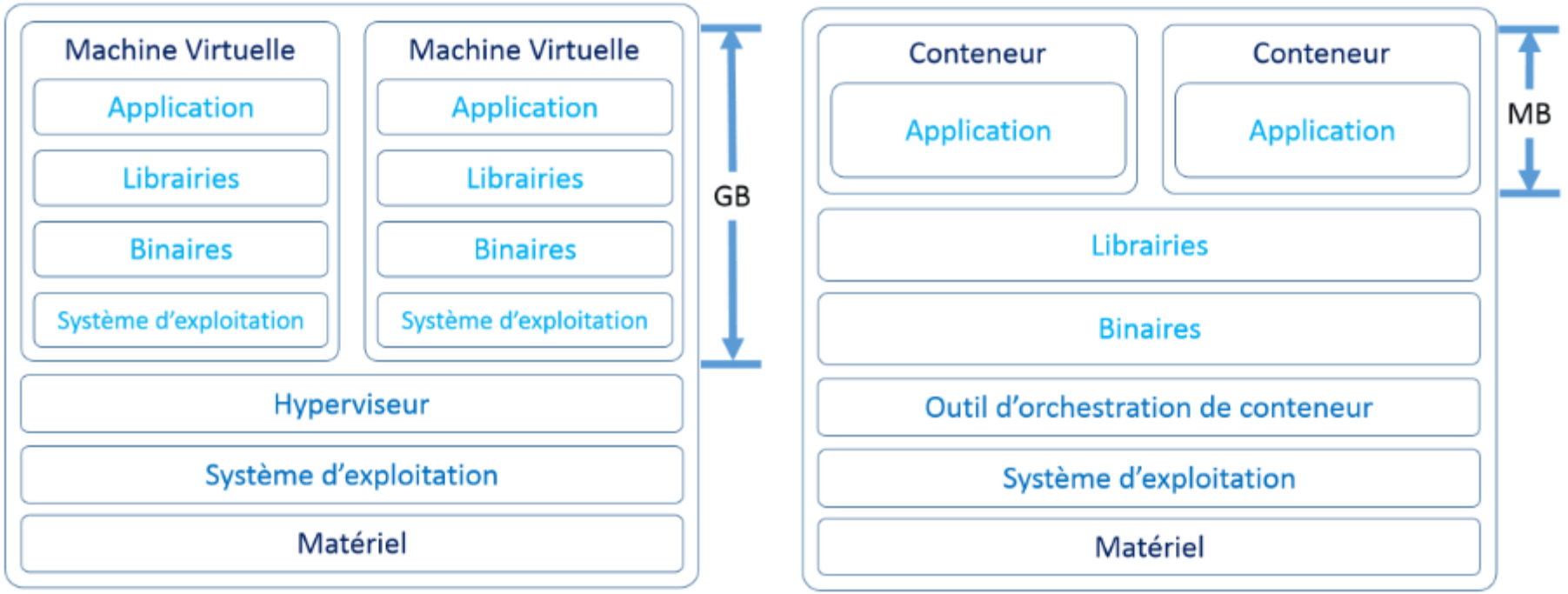




MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	Les machines virtuelles traditionnelles peuvent occuper beaucoup d'espace disque: elles contiennent un système d'exploitation complet et les outils associés, en plus de l'application hébergée par la machine virtuelle.
Conteneur	Les conteneurs sont relativement légers: ils ne contiennent que les bibliothèques et les outils nécessaires à l'exécution de l'application conteneurisée. Ils sont donc plus compacts que les machines virtuelles et démarrent plus rapidement.

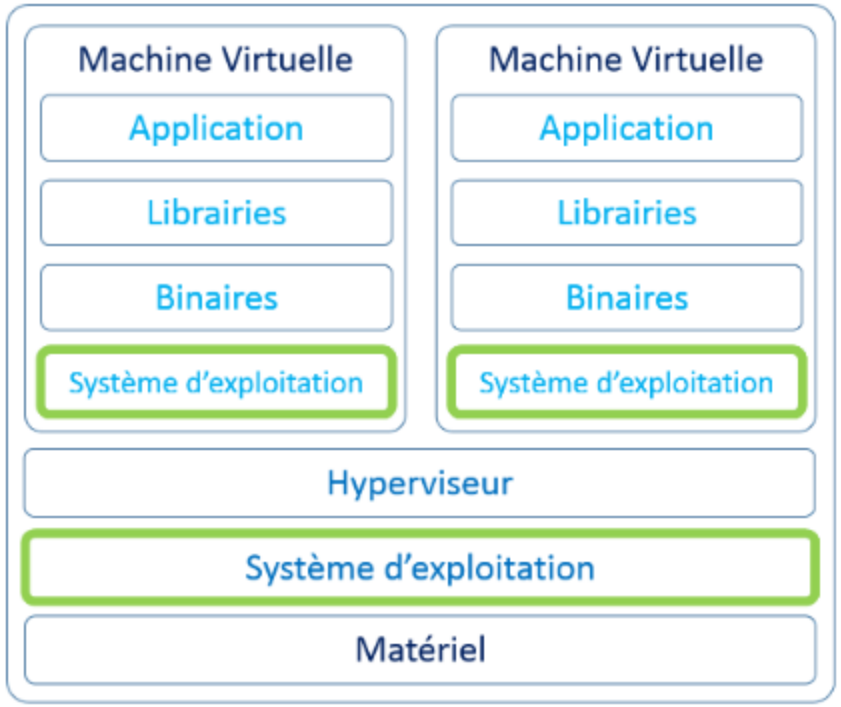
MACHINES VIRTUELLES ≠ CONTENEURS

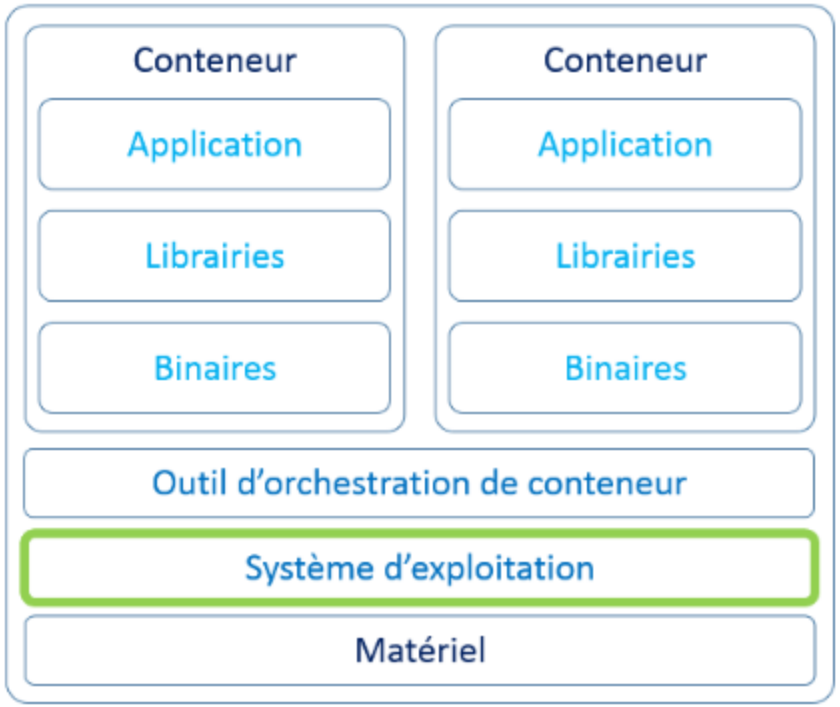


MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	En ce qui concerne la mise à jour ou la correction du système d'exploitation, les machines traditionnelles doivent être mises à jour une par une: chaque système d'exploitation invité doit être corrigé séparément.
Conteneur	Avec les conteneurs, seul le système d'exploitation de l'hôte du conteneur (la machine hébergeant les conteneurs) doit être mis à jour. Cela simplifie considérablement la maintenance.

MACHINES VIRTUELLES ≠ CONTENEURS





AVANTAGES DU CONTENEURS

- Meilleures performances
  - Accès direct au matériel
- Démarrage beaucoup plus rapide
  - Pas besoin de démarrer un système complet
- Images plus légères
  - Ne contient que les informations en lien avec l'application
  - Moins coûteux en terme d'espace de stockage
  - Plus rapide à transférer


docker

DEFINITION

Docker est une plateforme de conteneurs ayant largement contribué à la démocratisation de la conteneurisation.
Elle permet de créer facilement des conteneurs et des applications basées sur les conteneurs. Il en existe d'autres, mais celle-ci est la plus utilisée. Elle est par ailleurs plus facile à déployer et à utiliser que ses concurrentes.

Il s'agit d'une plateforme logicielle open source permettant de

- Créer,
- Déployer
- Gérer des containers d'applications virtualisées sur un système d'exploitation.

DEFINITION

Docker est composé de trois éléments :

- le daemon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (Containerd avec runC)
- une API de type REST qui permet de communiquer avec le daemon
- Le client en CLI (command line interface) : commande docker

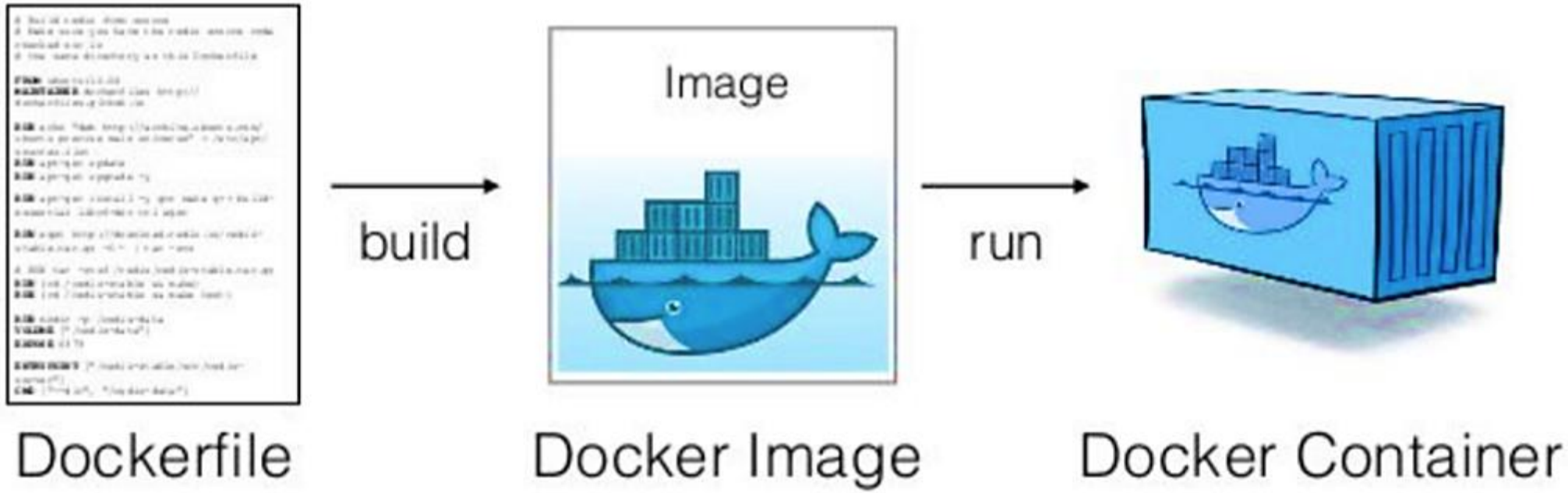


Image
build
run
Dockerfile
Docker Image
Docker Container

Fichier DOCKER

Il s'agit d'un fichier texte simple contenant une collection de commandes ou de procédures. Ces commandes et directives que nous exécutons agissent sur l'image de base configurée pour créer une nouvelle image Docker.

Un Dockerfile est le code source de l'image Docker. Un Dockerfile est un fichier texte contenant diverses instructions et configurations. La commande FROM dans un Dockerfile identifie l'image de base à partir de laquelle vous construisez.

Les images DOCKER

Les images sont des plans en lecture seule qui incluent des instructions de création de conteneurs. Une image Docker est un conteneur créé pour fonctionner sur le framework Docker. Considérez une image comme un plan ou une image de ce qu'il y aura dans un conteneur lorsqu'il sera opérationnel.

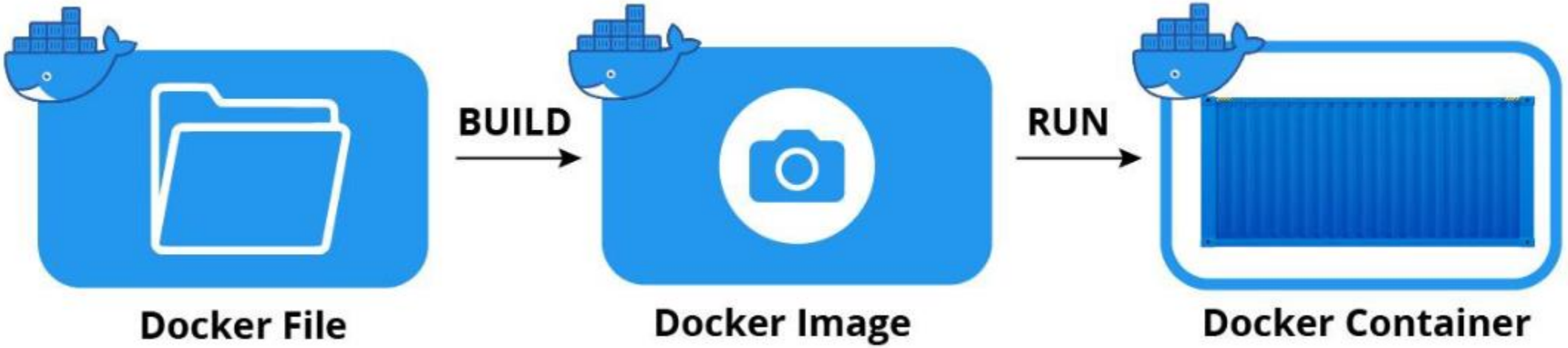
LES CONTENEURS

Les conteneurs sont des environnements d'exécution compacts et virtualisés utilisés pour exécuter des applications. Chaque conteneur est un progiciel* comprenant tous les fichiers de configuration, dépendances, outils système, bibliothèques et code source requis pour exécuter une certaine application. Ils sont distincts de l'hôte et de toute autre instance exécutée sur l'hôte.

*Ensemble de logiciels munis d'une documentation, conçus pour répondre à des besoins spécifiques et permettre une utilisation autonome.

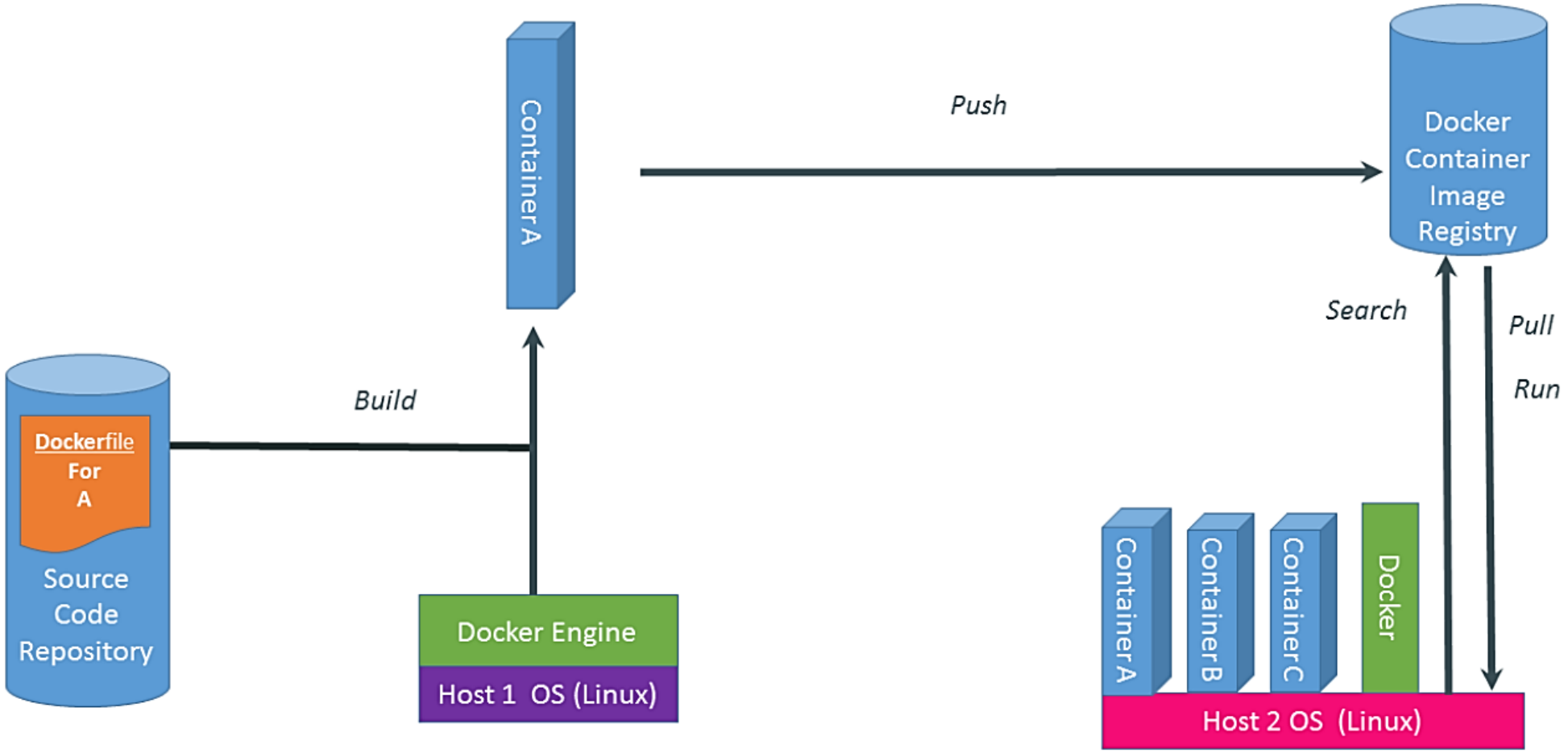
DOCKER HUB

Docker HUB : Dépôt public d’images mises à disposition par Docker



•Créez un fichier Docker et incluez les instructions pour créer votre image Docker.
•Exécutez la commande docker build pour créer votre image Docker.
•Utilisez la commande docker run pour créer des conteneurs maintenant que l'image docker est prête à être utilisée.

PRINCIPES DE FONCTIONNEMENT DE DOCKER



DOCKER: LES BRIQUES PRINCIPALES

Docker engine

Un environnement d'exécution et un ensemble de services pour manipuler des conteneurs docker sur une machine.

Une application client-serveur

Le serveur -- Un daemon (processus persistant) qui gère les conteneurs sur une machine

Le client -- Une interface en ligne de commande

Un/des registres d'images docker

Bibliothèque d'images disponibles

Docker Hub

PRINCIPES DES IMAGES DOCKER

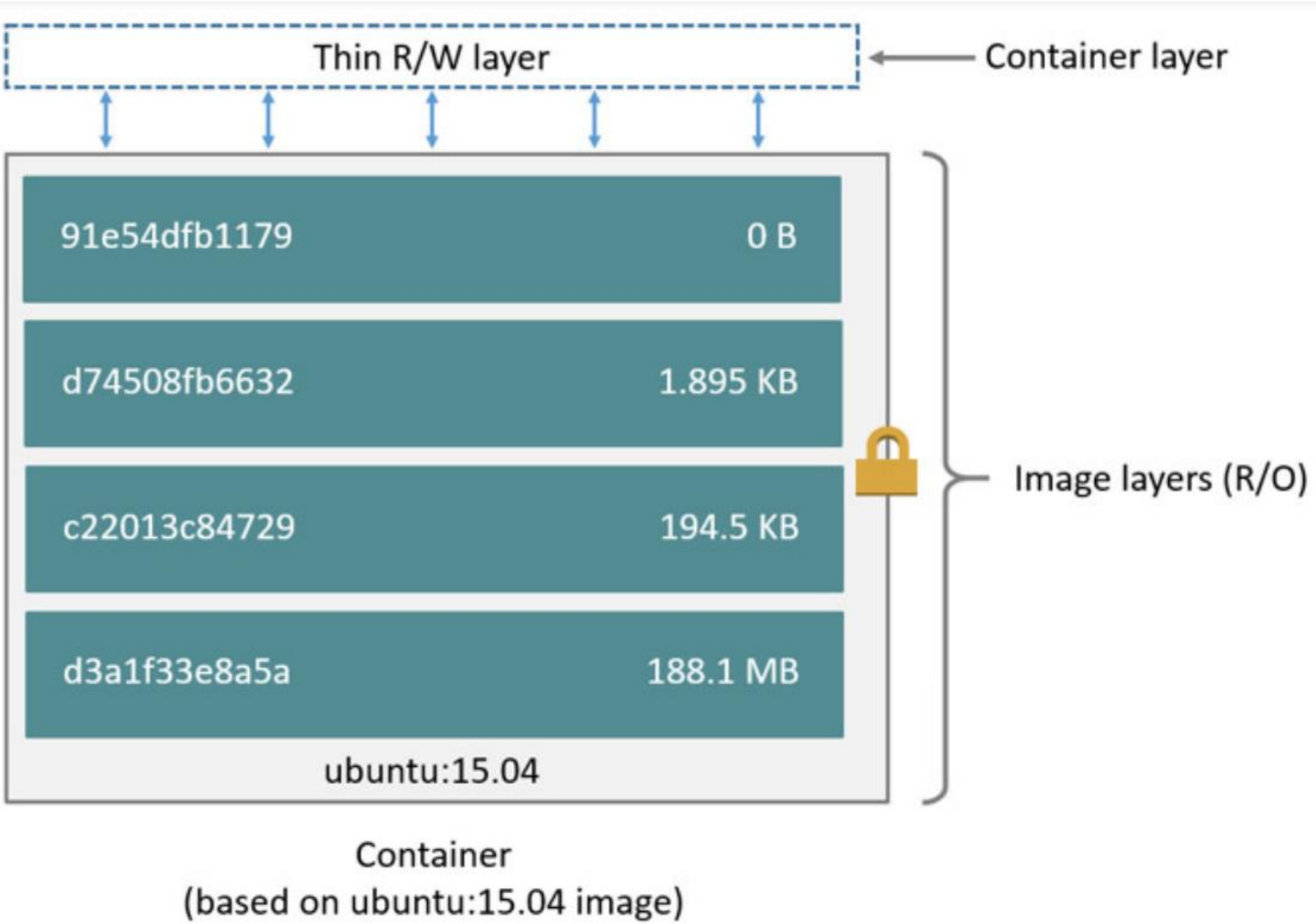
Fondé sur l'utilisation d'un Union File System

- Crée la vision d'un système de fichier cohérent à partir de fichiers/répertoires appartenant à des systèmes de fichiers différents

Un ensemble de couches

- Une image est composée d'un ensemble de couches (layers)
- L'Union File System est utilisé pour combiner ces couches
- Le file system utilisé par défaut s'appelle overlay2
- Chaque couche correspond à une instruction dans le fichier Dockerfile décrivant l'image.

Un système de fichiers Union, dans le contexte de Docker et de la conteneurisation, est une technologie qui permet de combiner plusieurs couches de systèmes de fichiers en une seule vue unifiée.


Thin R/W layer
Container layer
91e54dfb1179
0 B
d74508fb6632
1.895 KB
c22013c84729
194.5 KB
d3a1f33e8a5a
188.1 MB
ubuntu:15.04
Image layers (R/O)
Container
(based on ubuntu:15.04 image)

DOCKER IMAGE LAYERS

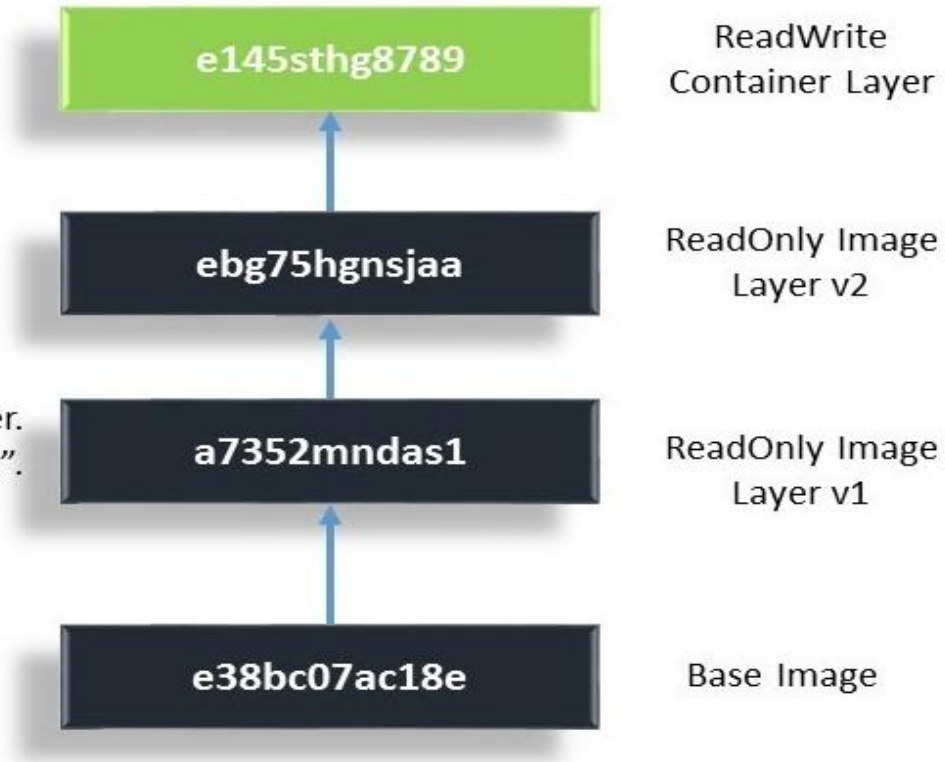
www.learnitguide.net

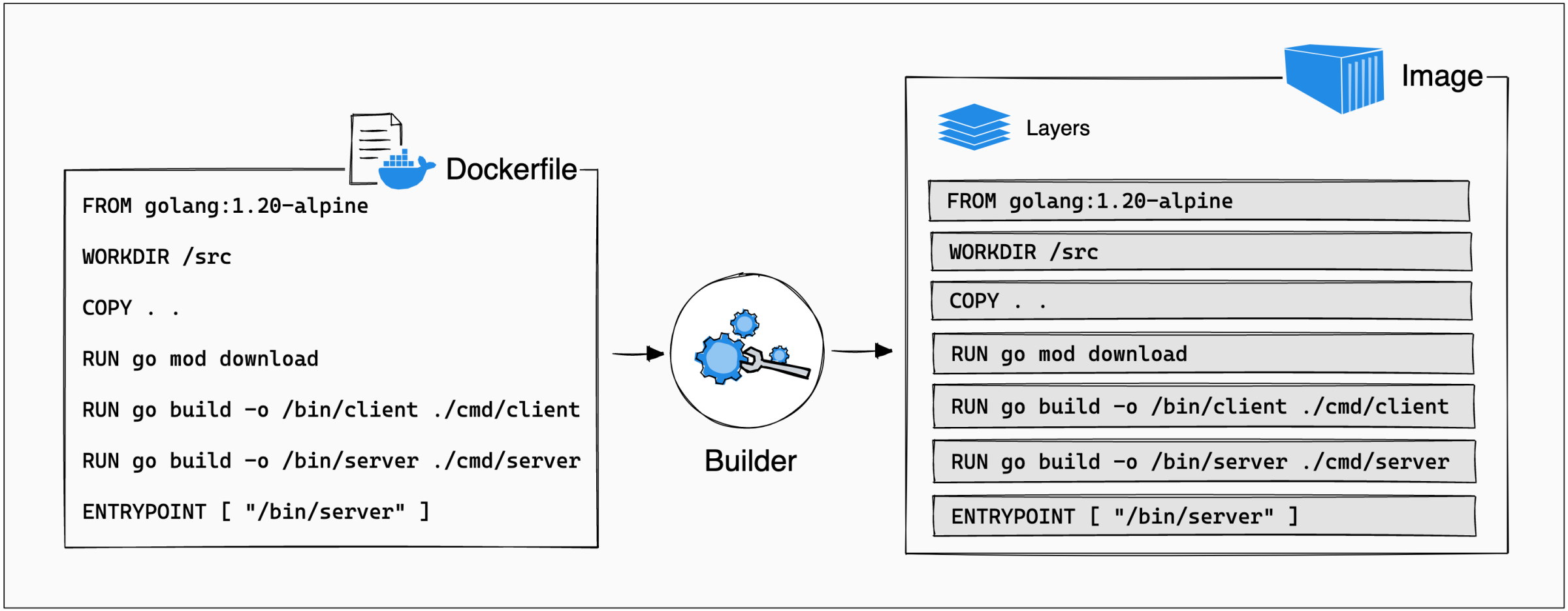
1. Started Image Layer 3 as a container and accessible by users

1. Started Image Layer v1 as a container.
2. Installed and Configured https web server.
3. Committed new layer v2

1. Started Base Image (**docker.io/centos**) as a container.
2. Package Updated on Base Image using “yum update”.
3. Committed new layer v1

Pulled CentOS image from Docker Hub using docker pull command. **Repo: docker.io/centos**




Dockerfile
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT ["/bin/server"]
Builder
Image
Layers
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT ["/bin/server"]

Exemple d'un Dockerfile

Les autres couches

- Les couches correspondent aux différentes modifications qui sont faites pour construire l'image à partir de l'image de base.
- Pour sauvegarder une nouvelle image, il suffit de sauvegarder les nouvelles couches qui ont été créées au dessus de l'image de base.
- Dans un conteneur en cours d'exécution, il existe une couche supplémentaire accessible en écriture
- Toutes les écritures vers le système de fichier faites à l'exécution du conteneur sont stockées dans cette couche.
- Les autres couches, définies au sein de l'image utilisée pour instancier le conteneur, ne sont accessibles qu'en lecture.
- Cette couche est supprimée à la suppression du conteneur

Optimisation des performances

- Partage des couches identiques entre conteneurs diminue l'espace de stockage utilisé par les conteneurs.

Affichage de l'ensemble des couches d'une image

```
$ docker history image_name
```

Avantages liés aux mécanismes de couches

Chaque couche est stockée une seule fois localement

Si certaines couches nécessaires pour une image à télécharger sont déjà présentes, pas besoin de les télécharger à nouveau.

Réduction de l'espace de stockage

Optimisations à l'exécution

Démarrage rapide

Démarrer un conteneur nécessite simplement de créer la couche accessible en écriture

Faible utilisation de l'espace disque

Si plusieurs containers sont instanciés à partir de la même image, ils partagent les couches en lecture seule.

Identification des couches/images

Dans la version courante de Docker :

- Chaque couche est identifiée par un **digest** qui est un hash du contenu de la couche
- Une image est définie par un ensemble de metadonnées (principalement la liste ordonnée des identifiants des couches constituant l'image)
- L'identifiant de l'image est créé en calculant un hash de ces metadonnées

CRÉER DES IMAGES

2 manières

- **Interactivement**
- **En utilisant un Dockerfile**

Avant de commencer: Premiers pas avec docker

L'outil à la ligne de commande pour exécuter des commandes Docker :

```
$ docker
```

```
docker run hello-world
```

Docker : Nous voulons exécuter une commande docker
Run : Commande pour créer et exécuter un conteneur docker
hello-world: Nom de l'image à partir de laquelle est construit le conteneur

docker pull: télécharge une image explicitement
docker run: Commande servant à créer un conteneur à partir d'une image. Télécharge l'image si elle n'est pas présente localement.

Que va-t-il se passer?

L'image à charger est hello-world:latest
Vérification: est ce que l'image est présente localement?
Sinon, télécharger l'image depuis Docker Hub
Charger l'image dans le conteneur et exécuter la commande par défaut définie pour ce conteneur

Construire une image interactivement

Objectifs

Créer une image à partir d'une image de base dans laquelle nous allons installer cowsay

Les étapes

1. Créer un conteneur à partir de l'image de base
2. Installer le logiciel manuellement dans le conteneur et en faire une nouvelle image
3. Jouer avec:
    docker commit
    docker tag
    docker diff

Configuration du conteneur

Démarrer un conteneur Ubuntu

```
$ docker run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
ea362f368469: Pull complete
Digest: sha256:b5a61709a9a44284d88fb12e5c48db0409cfad5b69d4ff8224077c57302df9cf
Status: Downloaded newer image for ubuntu:latest
root@f461e2e7afff:/#
```

f461e2e7afff est l'id du conteneur créé

Installer le programme dans le conteneur

```
root@f461e2e7afff:/# apt-get update
root@f461e2e7afff:/# apt-get install cowsay
```

Configuration du conteneur

Quitter la session interactive

```
root@f461e2e7afff:/# exit
```

Inspecter les changements

```
$ docker diff f461e2e7afff
C /var
C /var/log
C /var/log/apt
C /var/log/apt/history.log
A /var/log/apt/term.log
C /var/log/apt/eipp.log.xz
C /var/log/dpkg.log
...
```

C: fichier ou répertoire modifié
A: fichier ou répertoire ajouté

Sauvegarder les changements dans une nouvelle image

```
$ docker commit <yourContainerId>
<newImageId>
```

Exécution de la nouvelle image

```
$ docker run -it <newImageId>
root@7267696dc8c6:/# /usr/games/cowsay bonjour
... ca marche
```

Tagger une image

On peut tagger une image pour lui associer un nom (plus facile à manipuler qu'un identifiant)

```
$ docker tag <newImageId> cowsay
```

Il faut maintenant automatiser le processus!

Les Dockerfile

- Un Dockerfile est une recette décrivant comment construire une image
- Contient une suite d'instructions
- La commande docker build permet de créer une image à partir d'un Dockerfile

Éléments de syntaxe

- FROM: Définit l'image à partir de laquelle la nouvelle image est créée
- LABEL: Associe des meta-données à la nouvelle image (par exemple, l'auteur de l'image)
- RUN: Définie une commande exécutée dans la couche au dessus de l'image courante lors de la construction de l'image
- CMD: Définie la commande exécutée au démarrage du conteneur
- EXPOSE: Informe docker que le conteneur va écouter sur le port réseau défini
- COPY: Copier un fichier/répertoire depuis le contexte de construction de l'image vers la nouvelle couche

Les commandes de Dockerfile ne sont pas sensibles à la casse. On les note en majuscule par convention (facilite la lecture)

Notre premier Dockerfile

1- La création d'un Dockerfile doit se faire dans un nouveau répertoire vide

```
mkdir my_image
```

2- Créer le Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install cowsay
```

3- Construire notre image

```
$ docker build -t cowsay .
```

-t permet de tagger avec un nom l'image qui va être créée
. indique le contexte de construction de l'image (où se trouve le Dockerfile)

4- Que se passe-t-il?

```
$ docker build -t cowsay .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu
 ---> d13c942271d6
Step 2/3 : RUN apt-get update
 ---> Running in 80f5510281d9
Removing intermediate container 80f5510281d9
 ---> cb1643c4393c
Step 3/3 : RUN apt-get -y install cowsay
 ---> Running in 01834650511b
Removing intermediate container 01834650511b
 ---> 9ca55c5ccc54
Successfully built 9ca55c5ccc54
Successfully tagged cowsay:latest
```

Le build context est envoyé vers le démon docker (contenu du répertoir .)
A chaque étape:

- Un conteneur est créé pour exécuter l'étape (Running in ...)
- Les modifications sont committées dans une nouvelle image (---> ...)
- Le conteneur est supprimé
- La nouvelle image est utilisée pour la prochaine étape

5- Construction de l'image

docker build
Crée une image à partir d'un ficher Dockerfile

```
docker build -t docker-whale .
```

Crée l'image docker-whale avec le contexte correspondant au répertoire courant (le '.' est nécessaire)
Par défaut, le Dockerfile est cherché à la racine du contexte

DOCKER-COMPOSE

- On sait créer des images
  - de manière manuelle
  - de manière automatisée
- On sait lancer des conteneurs

PROBLÈMES :

- On veut coordonner des conteneurs
- On veut simplifier la gestion multi-conteneurs

DOCKER-COMPOSE

COMMENT FAIRE POUR

- Gérer les deux conteneurs à la volée?
- Gérer des volumes?
- Gérer des ports différents?
- Me souvenir de ces commandes?

DOCKER-COMPOSE

SOLUTION

- Compose vous permet d'éviter de gérer individuellement des conteneurs qui forment les différents services de votre application.
- Outil qui définit et exécute des applications multi-conteneurs
- Utilise un fichier de configuration dans lequel vous définissez les services de l'application
- A l'aide d'une simple commande, vous contrôlez le cycle de vie de tous les conteneurs qui exécutent les différents services de l'application.

DOCKER-COMPOSE

UTILISATION

- Vous définissez l'environnement de votre application pour qu'il soit possible de la générer de n'importe où
    - à l'aide de Dockerfile
    - à l'aide d'image officielle
- Vous définissez vos services dans un fichier *dockercompose.yml* pour les exécuter et les isoler.
- Exécutez docker-compose qui se chargera d'exécuter l'ensemble de votre application

DOCKER-COMPOSE

Qu'est ce que Docker Compose?

- Un outil complémentaire du Docker Engine
  - présenté dans les versions récentes comme un plugin de Docker Engine
  - Ancienne syntaxe: docker-compose command
  - Nouvelle syntaxe: docker compose command
- Permet de décrire une application construite à base de conteneurs dans un fichier YAML
- Docker permet d'implémenter des micro-services, cela signifie que les conteneurs se limitent souvent à une tâche simple.

DOCKER-COMPOSE

Les principes du Docker-compose

- L'utilisateur décrit son application (multi) conteneurs dans un fichier YAML appelé *dockercompose.yml*
- Exécuter *docker-compose up* pour démarrer l'application

  Compose télécharge automatique les images, les build (si nécessaire), et démarre les conteneurs

  Compose peut configurer des volumes, le réseau, et toutes autres options liées à Docker

Exemple du fichier docker-compose.yml

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    depends_on:
      - redis
  redis:
    image: redis
volumes:
  logvolume01:
```

Exemple du fichier docker-compose.yml

- 2 services: web et redis
  - ✓ Par défaut, Compose créé un réseau bridge indépendant du réseau par défaut
  - ✓ La découverte de service fonctionne sur ce réseau

    web peut contacter redis en utilisant son nom
- Un conteneur pour chaque service est créé
  - ✓ Pour web, une image est d'abord recréé à partir du Dockerfile présent dans le répertoire courant
  - ✓ Pour redis, l'image redis est récupérée depuis Docker Hub
- Configuration du réseau
  - ✓ Le port 5000 de la machine hote est associé au port 5000 du conteneur web

Exemple 2 du fichier docker-compose.yml

```
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - "./.data/db:/var/lib/mysql"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    links:
```

Exemple 2 du fichier docker-compose.yml :

Démarrage d'une application

```
$ docker-compose up -d
Creating wordpress_db_1
Creating wordpress_wordpress_1
```

Les différents services qui composent mon application ont été démarrés, avec la configuration et l'environnement qui va bien.

Information de mon application

On utilise la commande ps de Compose:

```
$ docker-compose ps
       Name                     Command               State          Ports
-------------------------------------------------------------------------------
wordpress_db_1          docker-entrypoint.sh mysqld      Up      3306/tcp
wordpress_wordpress_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:8000->80/tcp

$ docker-compose ps wordpress
       Name                     Command               State          Ports
-------------------------------------------------------------------------------
wordpress_wordpress_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:8000->80/tcp
```

Exemple 2 du fichier docker-compose.yml :

Conteneurs Classiques

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
8d086aeba614        wordpress:latest    "/entrypoint.sh apach"   7 minutes ago       Up 7 minutes
689405bb755d        mysql:5.7           "docker-entrypoint.sh"   7 minutes ago       Up 7 minutes
```

Les services s'exécutent via des conteneurs sur l'hôte. Les commandes docker classiques sont toujours fonctionnelles.

Passage à l'échelle

On peut passer à l'échelle un service. Autrement dit, on peut augmenter/diminuer le nombre de conteneurs exécutant un service. Par défaut, Compose exécute chaque service avec un conteneur.
On utilise la commande scale pour changer le nombre de réplicas d'un service:

```
$ docker-compose scale db=3
$ docker-compose ps db
    Name                  Command            State    Ports
-------------------------------------------------------------
wordpress_db_1   docker-entrypoint.sh mysqld   Up      3306/tcp
wordpress_db_2   docker-entrypoint.sh mysqld   Up      3306/tcp
wordpress_db_3   docker-entrypoint.sh mysqld   Up      3306/tcp
$ docker-compose scale db=2
```

A RETENIR

- Compose est un outil pour définir, lancer et gérer des services qui sont définis comme une ou plusieurs instances d'un conteneur,
- Compose utilise un fichier de configuration YAML comme définition de l'environnement,
- Avec docker-compose on peut générer des images, lancer et gérer des services, …
- Certaines commandes de docker-compose sont équivalentes à l'outil docker, mais s'appliquent seulement aux conteneurs de la configuration de compose.

LES KUBERNETES

Problématique

Comment gérer des containers qui sont dispersés dans un grand nombre de machines ?
Dans le cas où il y a un container qui pose un problème, comment le détecter ?
Lequel aurait besoin de plus de réplications pour subvenir à une subite augmentation de la charge ?

Solution

L'équipe opérationnelle monitore les serveurs et si un de ces derniers tombe en panne, il devra créer de nouveaux containers avec les services déchus dans un autre serveur.

Solution automatisée

Kubernetes permet justement d'automatiser le déploiement, la gestion de demande de puissance et la gestion des applications containerisées.

LES KUBERNETES

- Kubernetes est un orchestrateur de multiples conteneurs notamment des conteneurs dockers.
- Sa responsabilité est de surveiller les différentes instances de micro containers qu'il supervise et d'en réguler la réplication sur différentes machines en fonction de la montée en charge des requêtes.
- Maintenir la haute disponibilité (maintenir les conteneurs up et assurer l'état des services qui ont été décrits).

LES KUBERNETES

- **Docker** est une technologie d'exécution de conteneurs qui vous permet de ***créer, tester et déployer des applications*** plus rapidement qu'avec des méthodes traditionnelles.
- **Kubernetes** est un outil d'orchestration de conteneurs qui vous permet ***de mettre à l'échelle vos systèmes de conteneurs*** afin que vous puissiez gérer, coordonner et planifier les conteneurs à grande échelle.

Architecture maître-esclave du Kubernetes

- Kubernetes suit l’architecture maître-esclave:
  - Le maître plus communément appelé master existe principalement pour gérer votre cluster Kubernetes.
  - Les esclaves sont quant à eux plus connus sous le nom de workers (on les appellent aussi minions) et ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre nœud ce qu'il peut ou ne peut pas faire.

L’architecture de Kubernetes

- **Volumes (persistent ou non persistent) :**
  - Ils constituent les lieux d’échange entre les pods
  - Persistent dans ce cas on va les stocker à l’extérieur des pods
  - Non persistent à l’intérieur des pods.
- **Déploiements :**
  - Gestion de création/suppression des pods
  - Gestion des nom de réplicas (les réplicas sets)
  - Assurer du respect des descriptions des relations entre conteneurs
- **Namespaces :**
  - Cluster virtuel (ensemble de services) à l’intérieur de KBS
  - Cloisonner à l’intérieur de notre cluster pour des services qui ne travaillent pas ensemble et avoir une cohérence des droits des utilisateurs pour accès aux services qui travaillent ensemble et gérer le cloisonnement si on a des services totalement différents.
  - Segmenter les pods

Architecture maître-esclave du Kubernetes

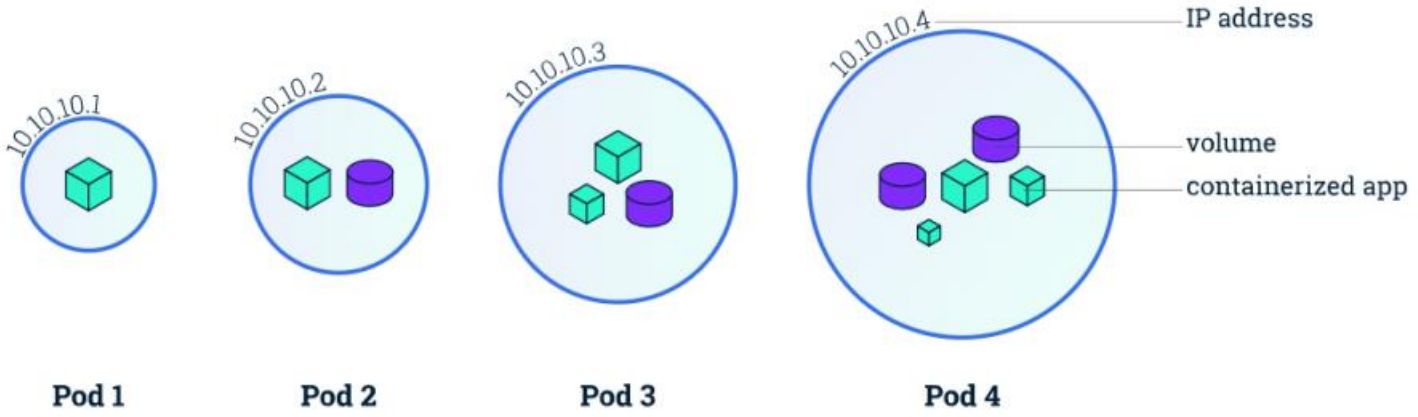
- Un cluster Kubernetes est une forme d'architecture de déploiement Kubernetes. L'architecture de base de Kubernetes existe en deux parties :

  Le plan de contrôle et les nœuds ou machines de calcul.

- Chaque nœud peut être une machine physique ou virtuelle et constitue son propre environnement Linux. Chaque nœud exécute également des pods, composés de conteneurs.

Les notions et concepts de Kubernetes

- **Pods: une instance de KBS : entité de référence de k8s :**
  - Chaque pod contient un fichier « descriptor » qui indique les paramètres d'installation du pod
  - Kubernetes ne fonctionne pas en interagissant directement avec les containers mais avec des pods.
  - Fournir un ensemble cohérent de conteneurs qui peut être un ou plusieurs conteneurs docker ou autre.



Les notions et concepts de Kubernetes

- **Services:**

  Le service se place au dessus des pods ce qui permet de faire une abstraction des pods.

  Le service évite la communication par IP comme dans le cas des docker (IP peut changer puisqu'on travaille avec les conteneurs).

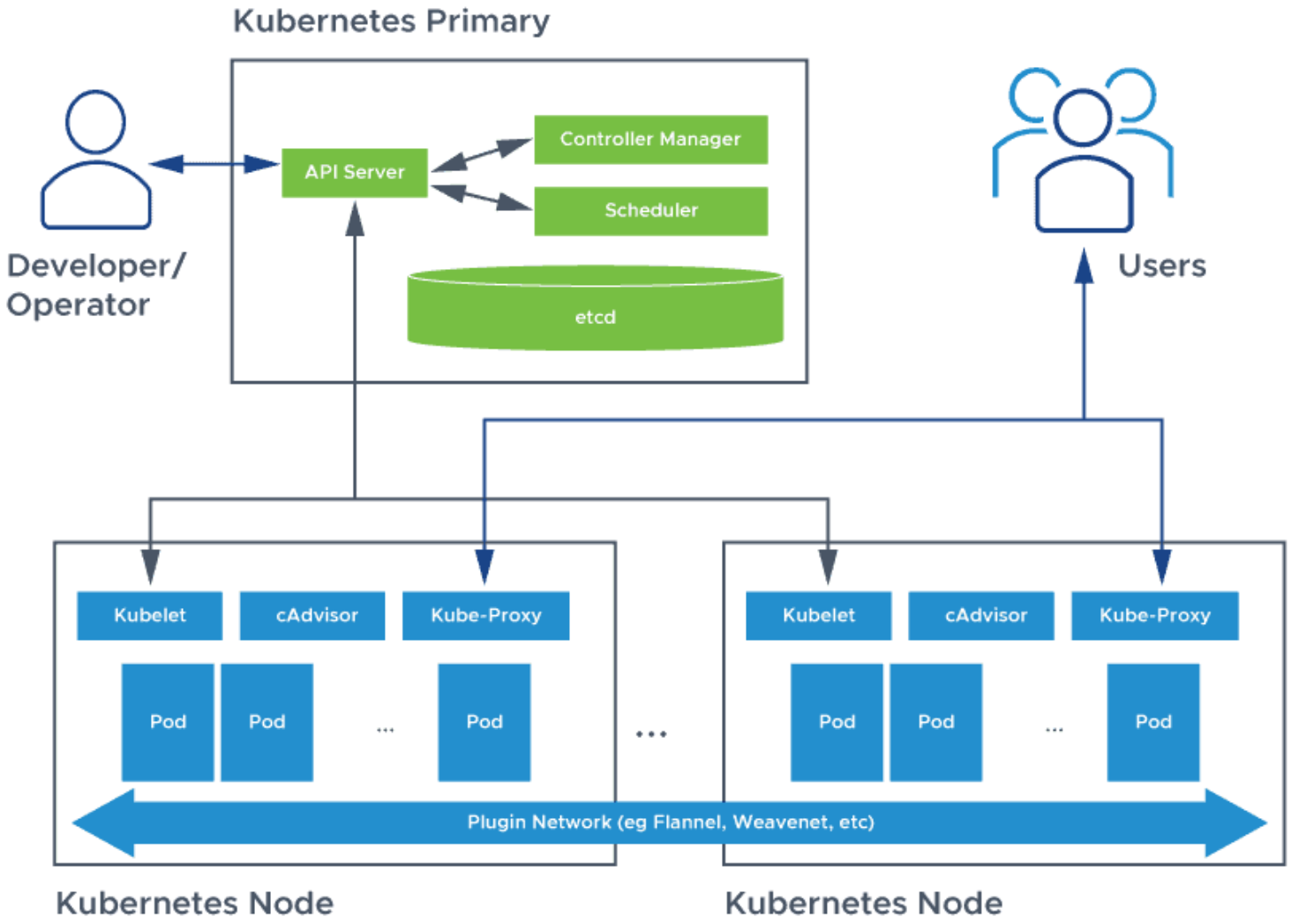
  Un service est compose d'un couple IP + Port fixe , qui permet de communiquer avec des conteneurs

Architecture de kubernetes

Kubernetes obéit à l'architecture maitre/esclave.

- Les composants de kubernetes sont divisés en composants de kubernetes master et ceux de kubenetes worker.
- Kubernetes master est une unité de contrôle qui charge la répartition de la charge de travail sur les pods, et dirige les communications dans le système et contrôle la santé des nœuds.
- Dès qu'il y a une panne dans pod il va arrêter et démarrer dans un autre nœud.
- Le kubernetes master comporte plusieurs composants et dans ce cas on a le choix d'installer ses composants sur une seul machine du cluster ou d'un ensemble de machines du cluster pour permettre la haute disponibilité.

Architecture du Kubernetes



Master Node

Le master node a la responsabilité d'administrer le cluster. Il coordonne les activités telles que la mise en échelle des applications, la maintenance des applications à l'état désiré et la propagation des mises à jour.

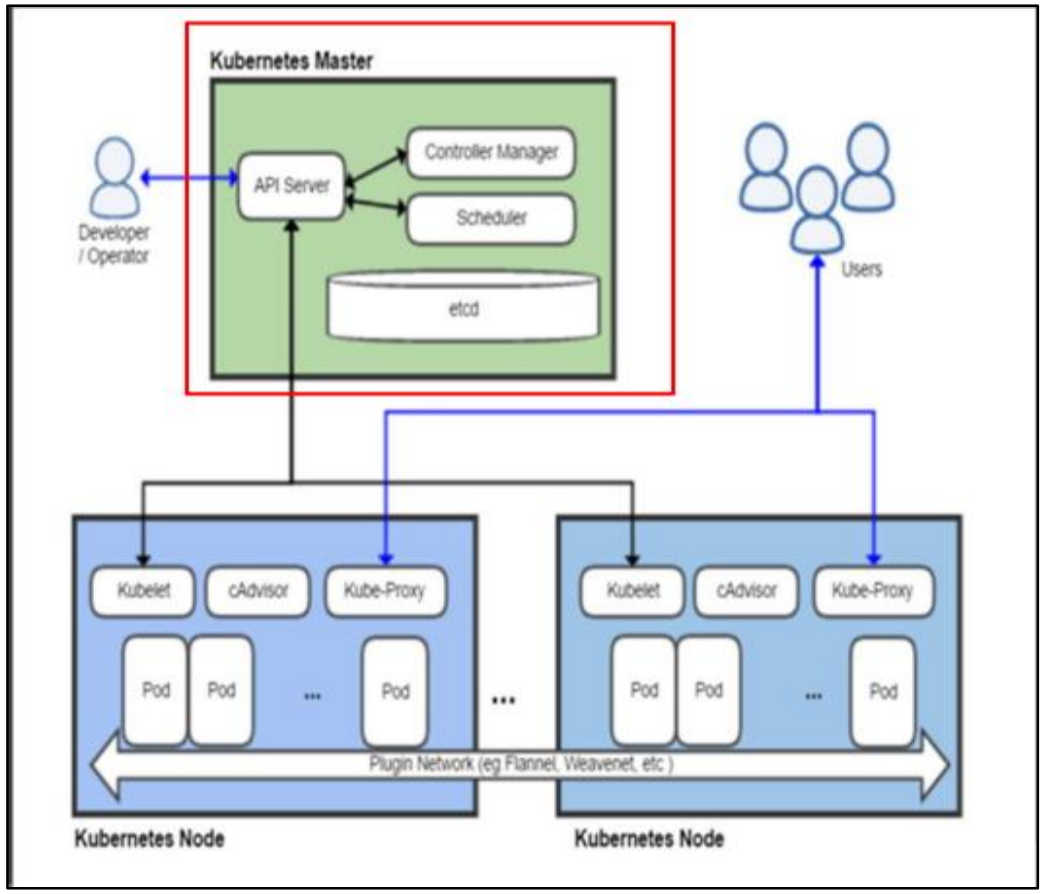
Kubernetes Master

Les composants déployés du plan de contrôle de kubernetes:

• Etcd: SGBD interne distribué et persistant qui stocke l'état du cluster (tous les évènement sui se produisent)

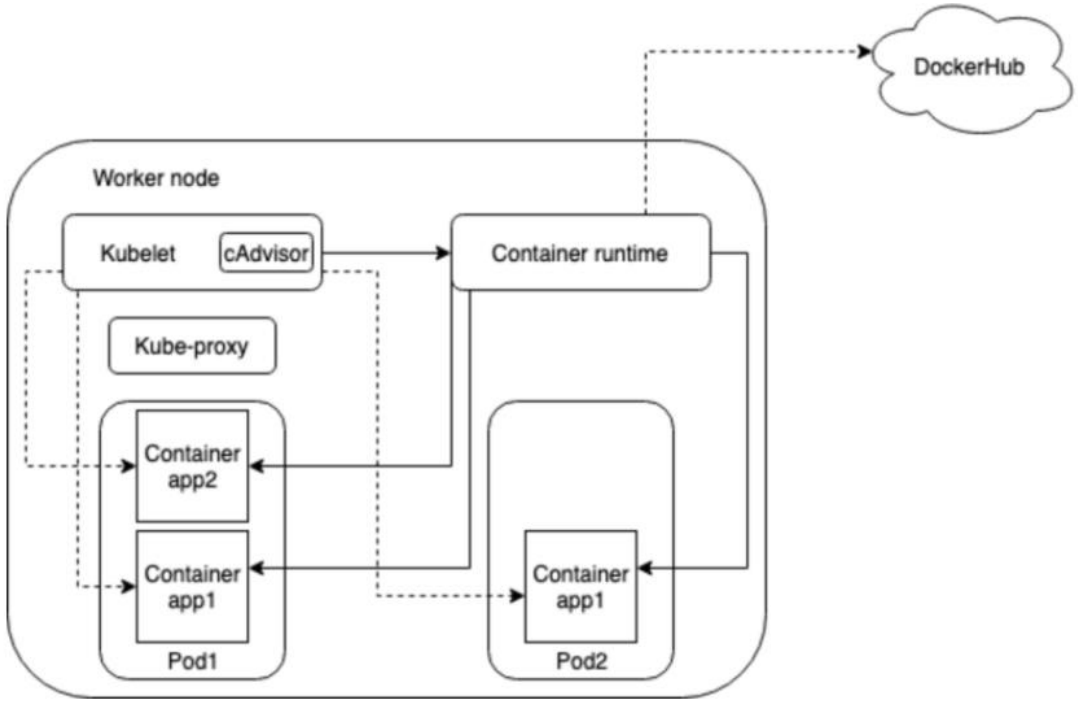
• API Server: conteneur web avec une API REST qui gère la communication avec les composants internes et externes

• Scheduler: l'ordonnanceur qui perme de déterminer la machine la moins chargé pour déployer le pod, donc il doit envoyer toutes les informations sur les nœuds (processus, mémoire…)



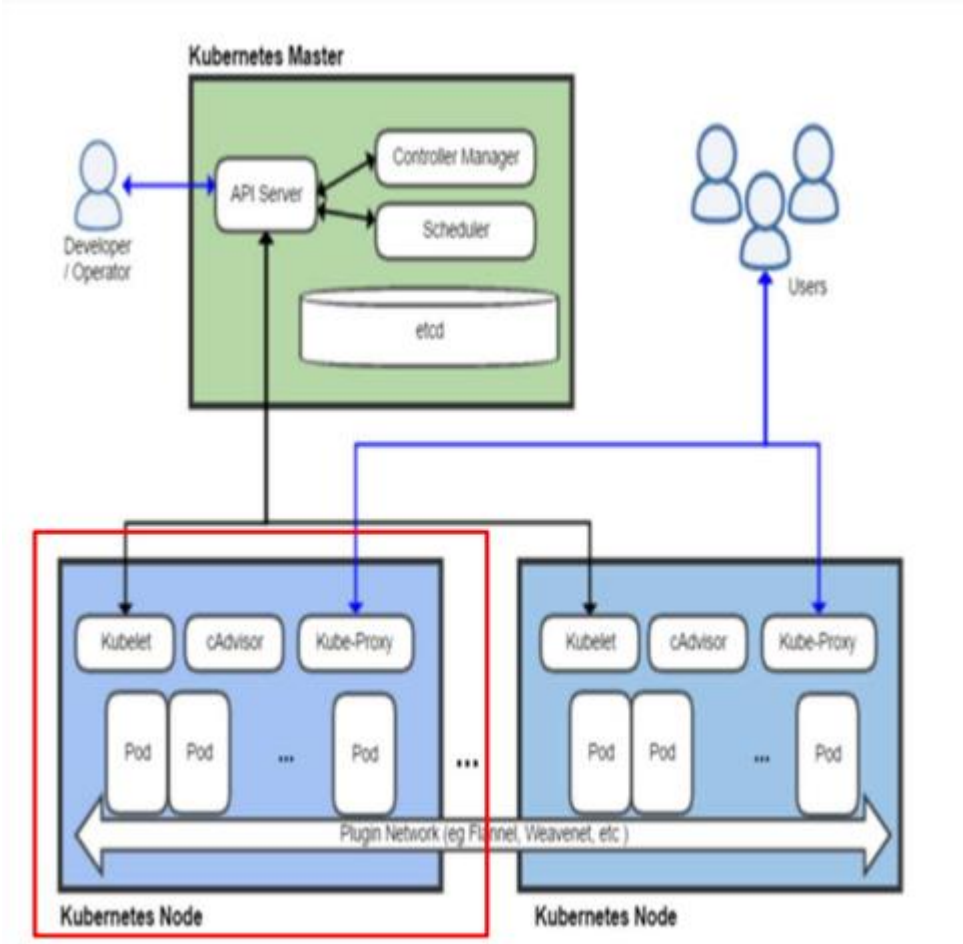
Worker Node

Un worker node (WN) est une machine physique ou une VM qui détient toutes les ressources nécessaires afin de garantir l'exécution d'un ou plusieurs pods. Cette entité va héberger tous les services qu'un développeur aura décidé de déployer.



Kubernetes Node

- Le Node appelé aussi Worker est une machine unique (ou une machine virtuelle) ou des conteneurs (les services sous forme de charges de travail) sont deployés sous forme de pod (les pods regroupent des conteneurs).
- Puisque les nodes contiennent des pods, donc chaque node du cluster doit exécuter le programme de conteneurisation de docker (Docker Engine)



Les Composants d'un Worker Node

Les composants d'un worker node sont responsables du déploiement des pods, et conteneurs, en plus, disposent des composants par default qui sont:

- Kubelet: Responsable de l'état d'exécution du nœud (c'est-à-dire, d'assurer que tous les conteneurs sur un nœud sont bien organisés en Pods et envoyer des informations en permanence au kubernetes master (plan de contrôle).
- Kube-proxy: Responsable d'effectuer le routage du trafic vers le conteneur approprié (sélectionné pour exécuter cette tâche) en se basant sur l'adresse IP et le numéro de port de la requête entrante (service demandé par l'utilisateur)
- cAdvisor: Agent qui surveille et récupère les données de consommation des ressources locales des performances comme le processeur, la mémoire, ainsi que l'utilisation disque et reseau des conteneurs du Node et envoyer ses données au composant scheduler (ordonnanceur) dans le composant master qui va se charger de repartir les charges selon l'état des ressources des nœuds.