

# Virtualisation & Cloud Computing

**Dr. Wael Sellami**  
[wael.sellami@gmail.com](mailto:wael.sellami@gmail.com)

2023-2024

# **Chapitre 4 : Conteneurs dans le Cloud Computing**

# Plan du chapitre

---

1. Evolution des serveurs
2. Conteneur
3. Docker
4. Orchestration des conteneurs

# Evolution des serveurs

## Serveurs physiques

---

- Historiquement, quand nous avons besoin de serveurs, nous achetions des serveurs physiques avec une quantité définie de CPU, de mémoire RAM ou de stockage sur le disque.



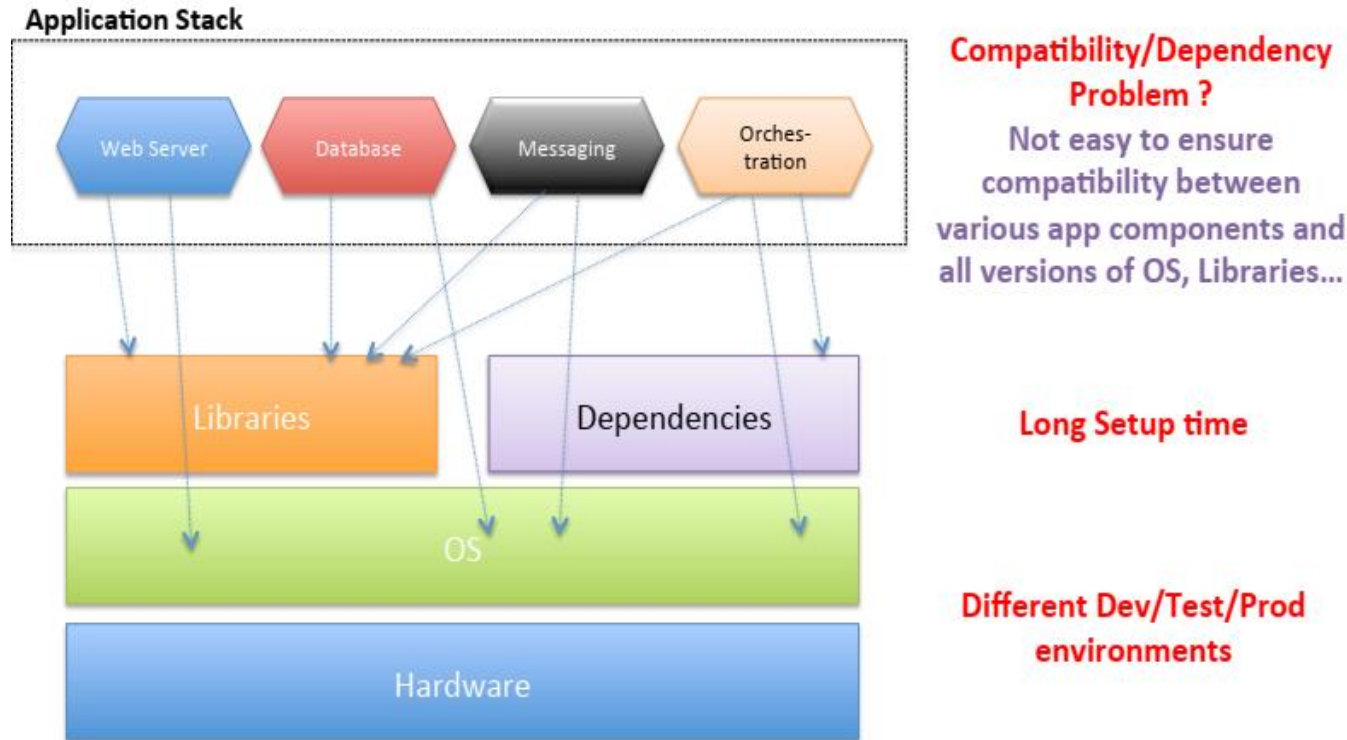
**Serveur physique**



Partage de l'espace logistique

# Evolution des serveurs

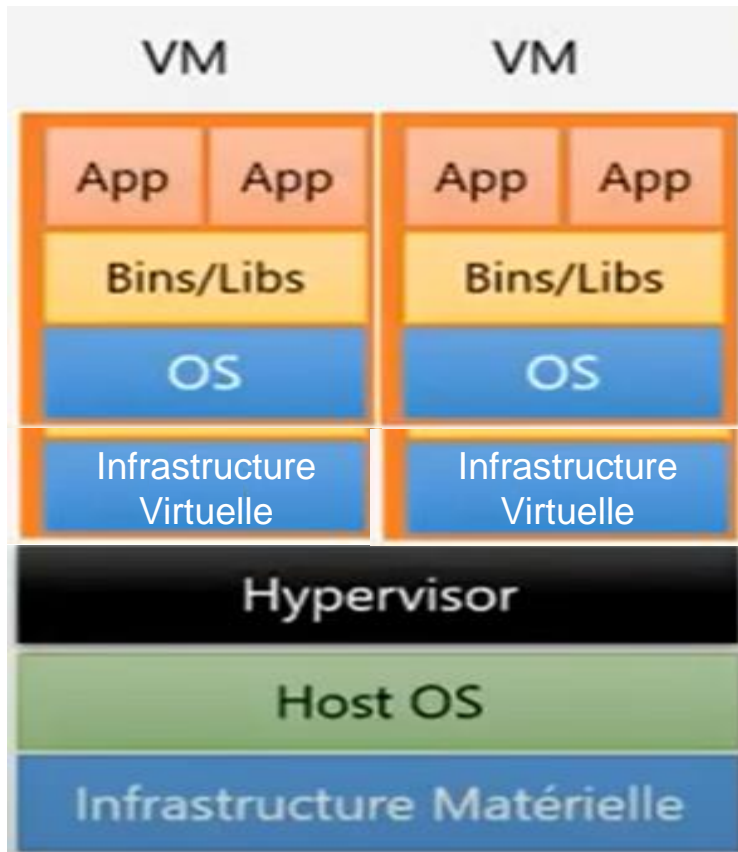
## Serveurs physiques



- Besoin d'avoir de la puissance supplémentaire pour des périodes de forte charge (fête de Noël, par exemple).
  - Acheter plus de serveurs pour répondre aux pics d'utilisation.
- ➔ Une solution a alors été créée : **la machine virtuelle.**

# Evolution des serveurs

## Virtualisation



### Les machines virtuelles



Partage des ressources d'une machine

# Evolution des serveurs

## Virtualisation

---

- Cette solution présente de nombreux **avantages** :
  - ✓ Une machine virtuelle est **totalelement isolée** du système hôte ;
  - ✓ Les **ressources** attribuées à une machine virtuelle lui sont **totalelement réservées** ;
  - ✓ Installer **différents OS** (Linux, Windows, BSD, etc.).
- Cependant, avec les machines virtuelles (VM), nous faisons ce qu'on appelle de la **virtualisation lourde**.

# Evolution des serveurs

## Virtualisation

---

- En effet, un système complet dans le système hôte est créé, pour qu'il ait ses propres ressources.
  - L'isolation avec le système hôte est donc totale; cependant, cela apporte plusieurs contraintes :
    - ✗ une machine virtuelle prend du temps à démarrer ;
    - ✗ une machine virtuelle réserve les ressources (CPU/RAM) sur le système hôte.
- ➔ Alors est né un nouveau système de virtualisation plus léger au niveau processus : **les isolateurs**.



# Virtualisation des processus

## Présentation de l'isolation

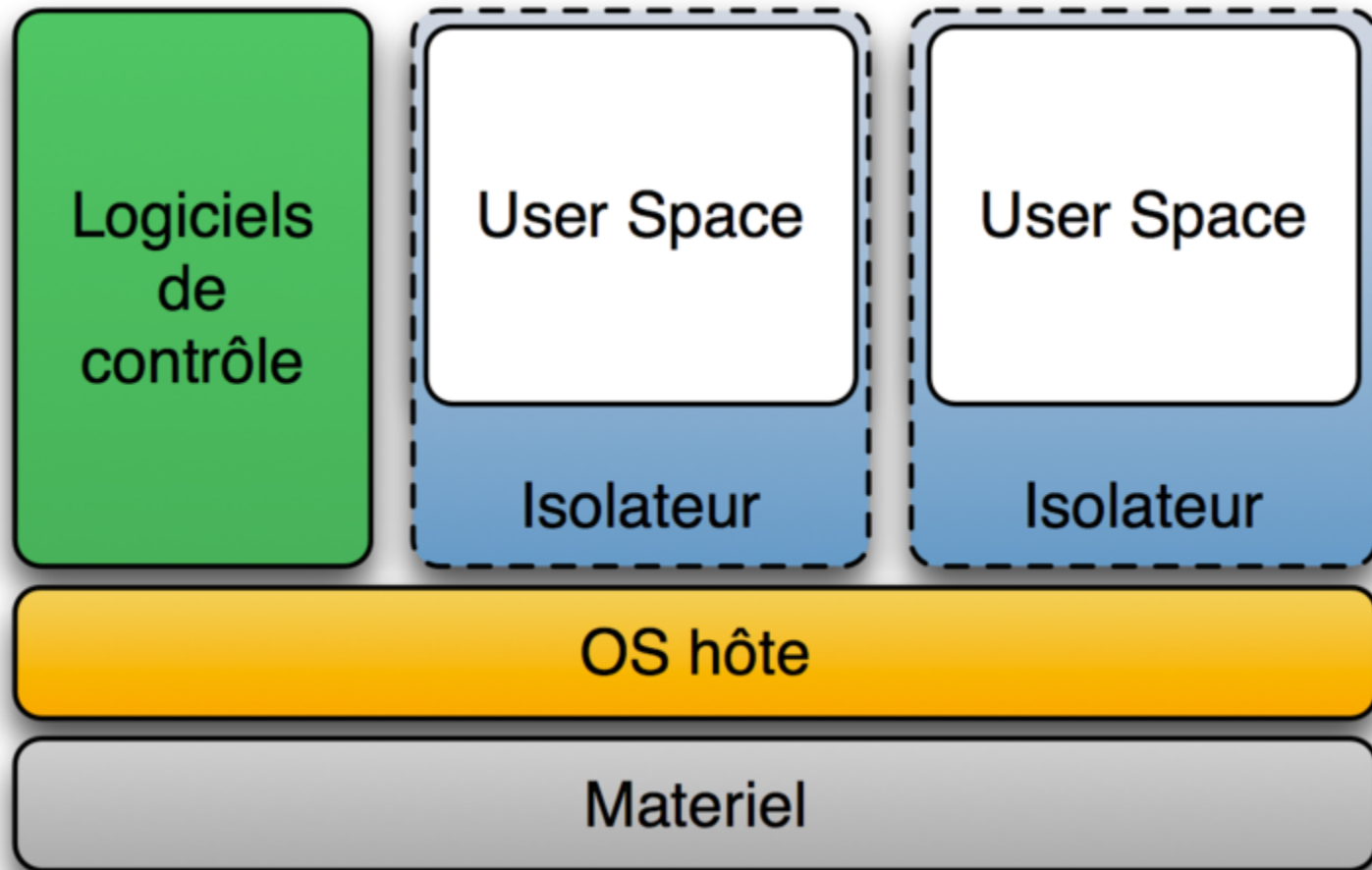
---

- **L'isolation** (aussi appelé cloisonnement) est une technique qui intervient au sein d'un même système d'exploitation.
- Elle permet de **séparer** un système en **plusieurs contextes ou environnements**. Chacun d'entre eux est régi par l'OS hôte, mais les programmes de chaque contexte ne peuvent communiquer qu'avec les processus et les ressources associées à leur propre contexte.
- Il est ainsi possible de **partitionner un serveur en plusieurs dizaines de contextes**, presque sans ralentissement.
- Il est possible également de lancer des programmes dans une autre distribution que celle du système principal.

# Virtualisation des processus

## **Présentation de l'isolation**

---



# Virtualisation des processus

## **Unix chroot / BSD Jail**

---

- Via des mécanismes comme Unix chroot (1982) ou BSD jail (1998), il est possible d'exécuter des applications dans un environnement qui n'est pas celui du système hôte, tout en étant légers. Il s'agit d'un « mini système » ne contenant que ce dont l'application a besoin, et n'ayant que des accès limités aux ressources.

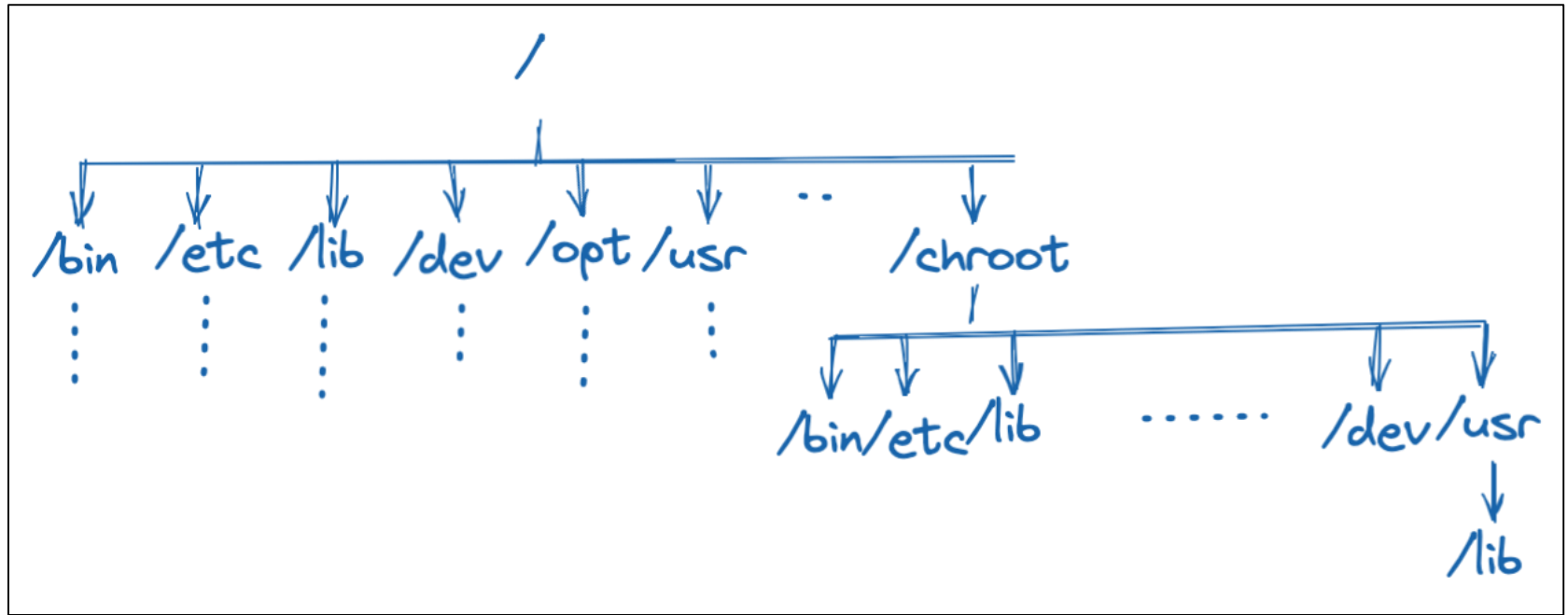
### **chroot**

- chroot signifie « **change root** », traduisez changement de racine
- Elle permet d'isoler la racine du système de fichier (le / de l'arborescence) pour une commande spécifique. La racine du système de fichier visible par la commande "chrootée" est une sous-arborescence du système de fichier complet. Ceci permet, par exemple, de sécuriser un serveur en lui donnant accès à un système de fichier restreint.

# Virtualisation des processus

## Unix chroot / BSD Jail

---



# Virtualisation des processus

## Conteneur Linux LXC



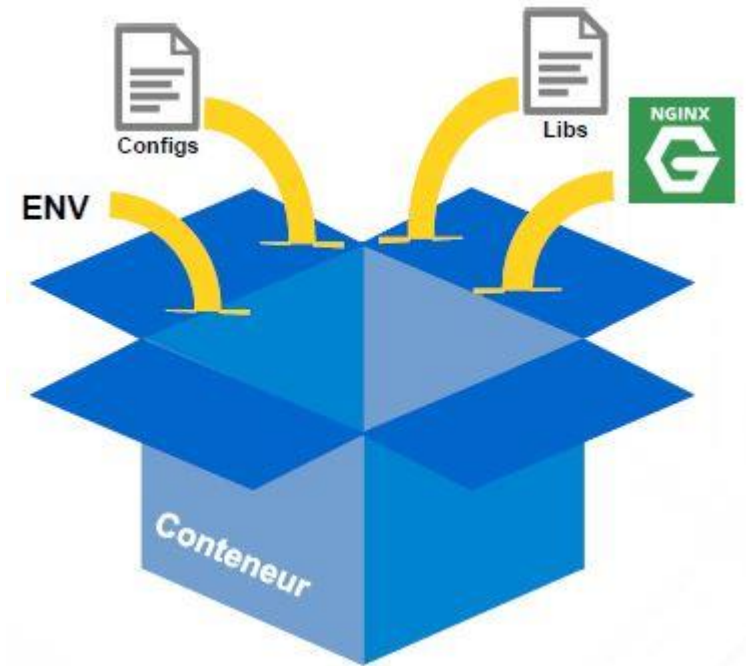
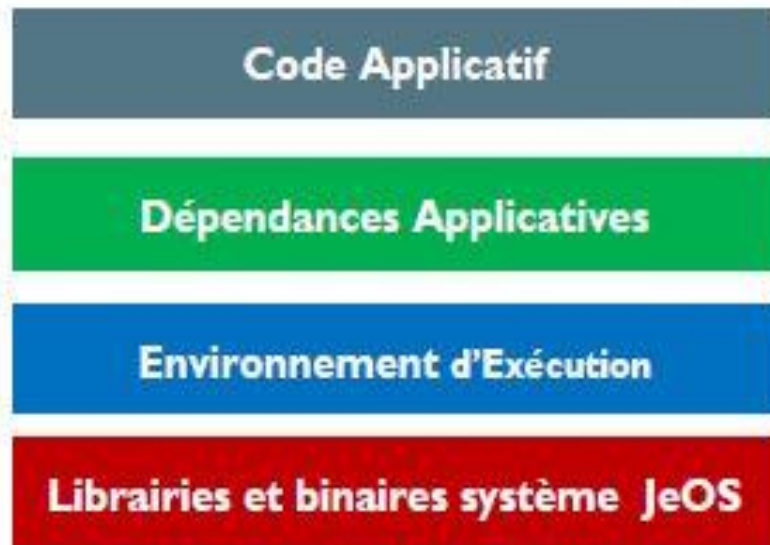
### LXC : C'est quoi?

- La virtualisation par conteneurs se base sur la virtualisation Linux LXC, pour Linux Containers (2008).
- Un **conteneur Linux** est une **enveloppe virtuelle** qui permet de **packager une application avec tous les éléments** dont elle a besoin pour fonctionner (**fichiers source, run-time, librairies et dépendances**).
- Le conteneur permet de faire de la **virtualisation légère**, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une isolation des processus. Le conteneur **partage** donc **les ressources** avec le **système hôte**.

# Virtualisation des processus

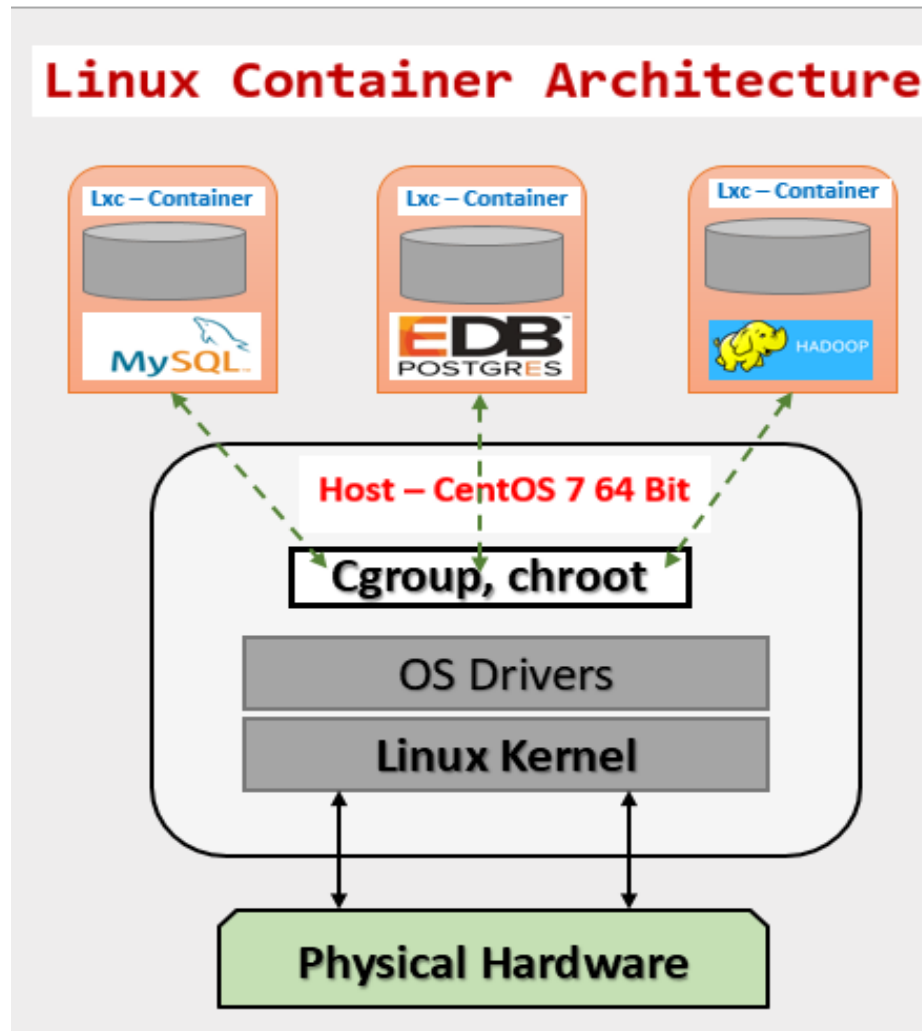
## Conteneur Linux LXC

---



# Virtualisation des processus

## Conteneur Linux LXC



# Virtualisation des processus

## Conteneur Linux LXC



### LXC : C'est quoi?

- Il s'agit d'une méthode de cloisonnement au **niveau du système d'exploitation**.
- **Ces conteneurs** sont **isolés** du reste du système. Ils sont packagés en un ensemble cohérent et prêt à être déployé sur un serveur et son OS.
  - ✓ portables et fonctionnent de la même manière dans les environnements de développement, de test et de production.
  - ✓ déplacer l'application jusqu'en production sans aucun effet secondaire.
- Les conteneurs partagent entre eux le **kernel Linux** ; ainsi, il n'est pas possible de faire fonctionner un système Windows ou BSD dans celui-ci.



# Virtualisation des processus

## Conteneur Linux LXC



### LXC : Technologies de base

- LXC repose sur la notion de groupes de contrôle Linux (**cgroups**) :
  - permet de limiter et d'isoler l'utilisation des ressources qu'un processus peut utiliser (processeur, mémoire, réseau, système de fichier, etc), et ce sans recourir à des machines virtuelles à part entière.
- LXC repose aussi sur une isolation des espaces de nommage du noyau (**namespace**) :
  - Permet d'empêcher qu'un groupe puisse « voir » les ressources des autres groupes (systèmes de fichiers, les ID réseau et les ID utilisateur)
- LXC repose sur les **bibliothèques Profils Apparmor** (Application Armor) et **SELinux** (Security-Enhanced Linux) pour la sécurité en termes des restrictions, permissions et droits utilisateur

# Virtualisation des processus

## Conteneur Linux LXC

---



- Intégré officiellement au noyau Linux :
  - ✓ Mount namespace (Linux 2.4.19)
  - ✓ PID namespace (Linux 2.6.24) - x PID/process
  - ✓ Net namespace (Linux 2.6.19-2.6.24)
  - ✓ User namespace (Linux 2.6.23-3.8)
  - ✓ cgroups (Linux 2.6.24) - gérer la limitation de ressource

# Avant / Après les conteneurs

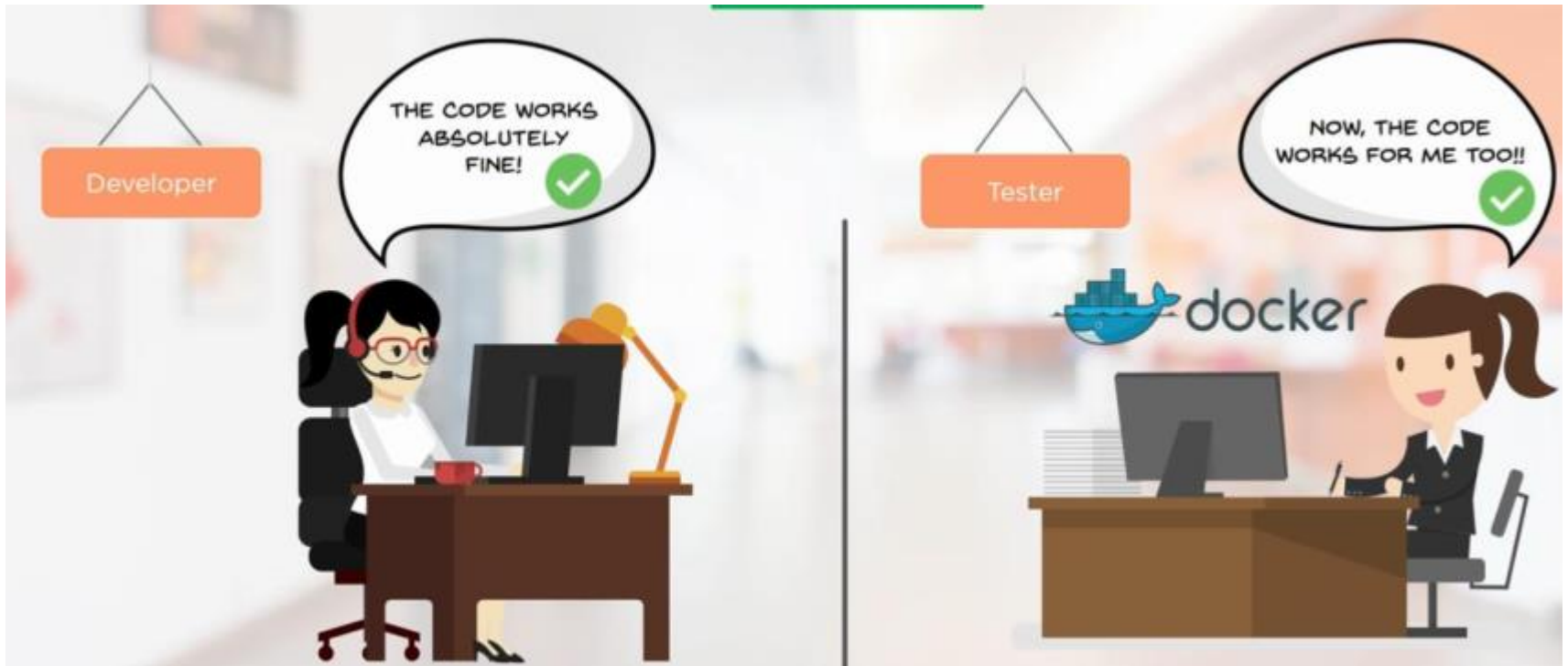
---

- **Dev** : Cela fonctionne bien dans mon système
- **Testeur** : Cela ne fonctionne pas dans mon système !!



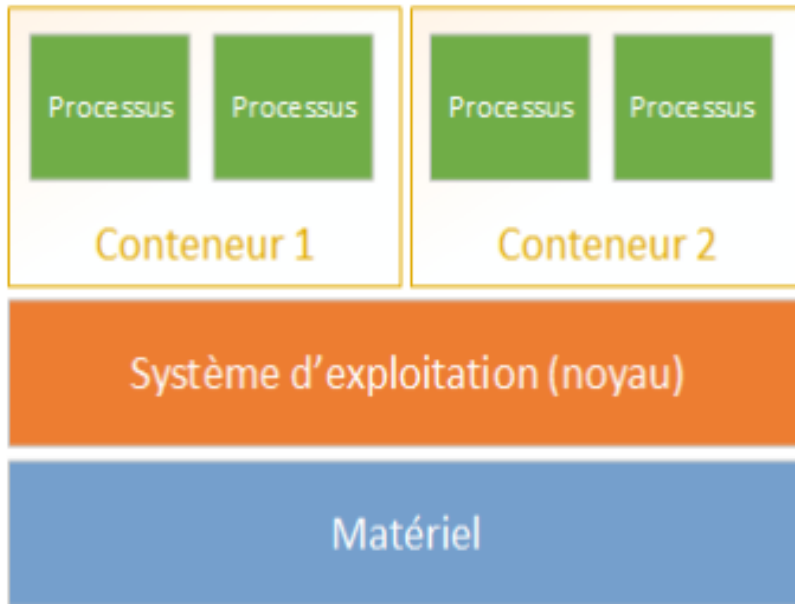
# Avant / Après les conteneurs

Le testeur et le développeur exécutent tous les deux la même application sans avoir à faire face aux différences de dépendances comme auparavant.



# Virtualisation des processus

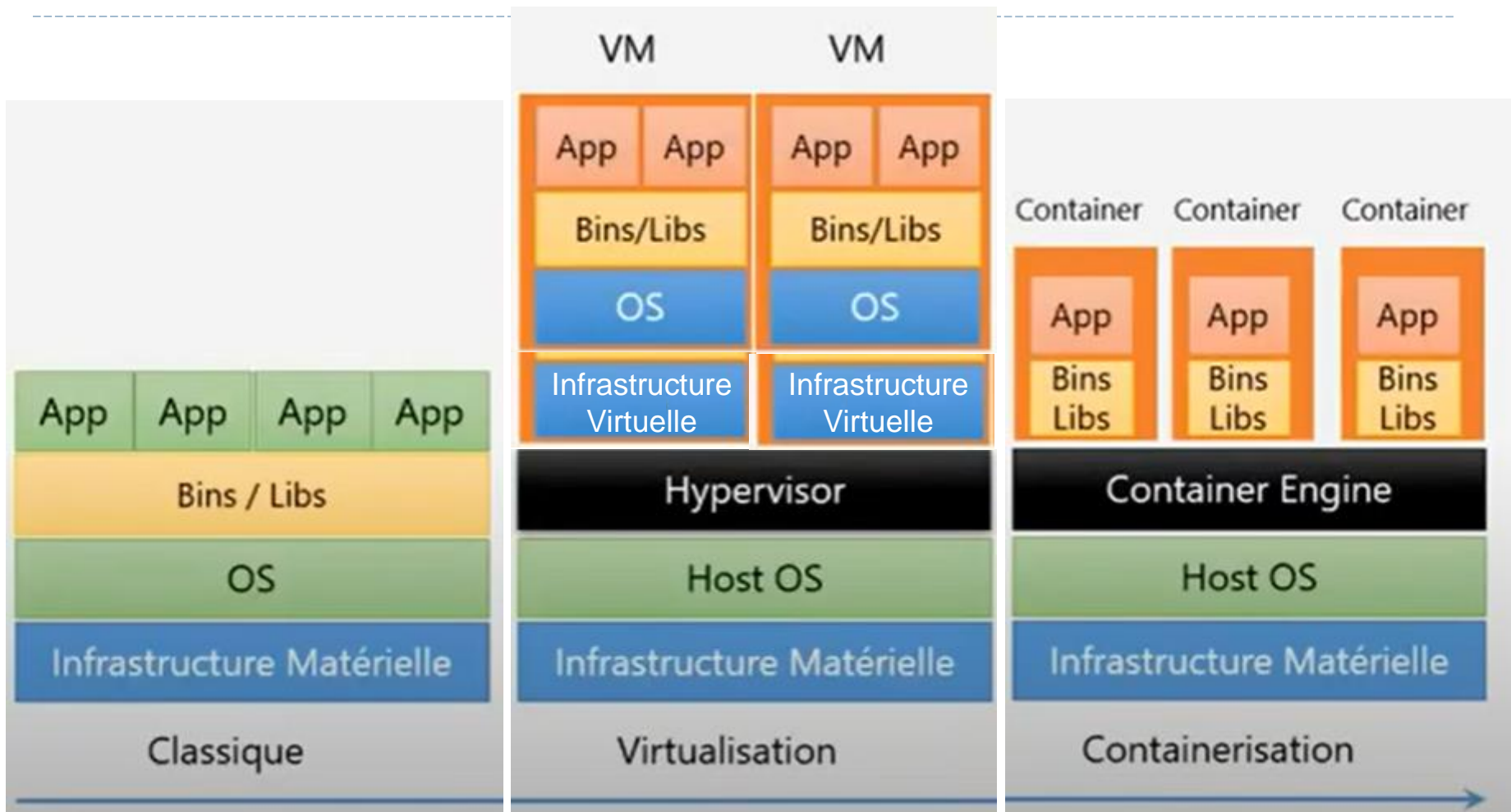
## Conteneur Linux LXC



Les conteneurs

Optimisation des ressources sans perte du cloisonnement

# Evolution des serveurs



# Un conteneur (jolie métaphore), c'est quoi ?

---

- Encapsule une application
- Fournit à l'application un système de fichiers complet (/ , /usr/, /bin, /opt, /etc)
- Fournit les binaires nécessaires à son exécution (bash, sh, python, etc.)
- Possède sa propre interface réseau
- Possède ses propres utilisateurs Linux (ex : root)
- Facile à créer et à supprimer
- Léger en terme de ressource demandée.

# Conteneur (jolie métaphore), c'est quoi ?

---

- Les mêmes idées que la virtualisation, mais sans virtualisation :
  - ✓ Isolation et automatisation
  - ✓ Agnostique sur le contenu et le transporteur
  - ✓ Principe d'infrastructure consistante et répétable
  - ✓ Peu de surcharge (overhead) par rapport à une VM.
- Les conteneurs permettent d'exécuter plusieurs parties d'une application dans des **micro-services**, indépendamment les uns des autres, sur le même matériel, avec un niveau de contrôle bien plus élevé sur leurs éléments et cycles de vie.



# Avantages des conteneurs

---

- **Réduire des coûts**

Les conteneurs permettent de réduire les coûts, d'augmenter la densité de l'infrastructure, tout en améliorant le cycle de déploiement.

- **Se limiter aux ressources nécessaires**

Conteneur ne réserve pas la quantité de CPU, RAM et disque attribuée auprès du système hôte. Ainsi, nous pouvons allouer 16 Go de RAM à notre conteneur, mais si celui-ci n'utilise que 2 Go, le reste ne sera pas verrouillé (à la différence de la VM).

# Avantages des conteneurs

---

- **Démarrer rapidement vos conteneurs**

Les conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation, ils peuvent démarrer beaucoup plus rapidement et plus fréquemment qu'une machine virtuelle sur nos serveurs hôtes, et ainsi réduire encore un peu les frais de l'infrastructure.

- **Donner plus d'autonomie à vos développeurs**

Possibilité de faire tourner des conteneurs sur le poste des développeurs, et ainsi de réduire les différences entre la "sainte" production, et l'environnement local sur le poste des développeurs.

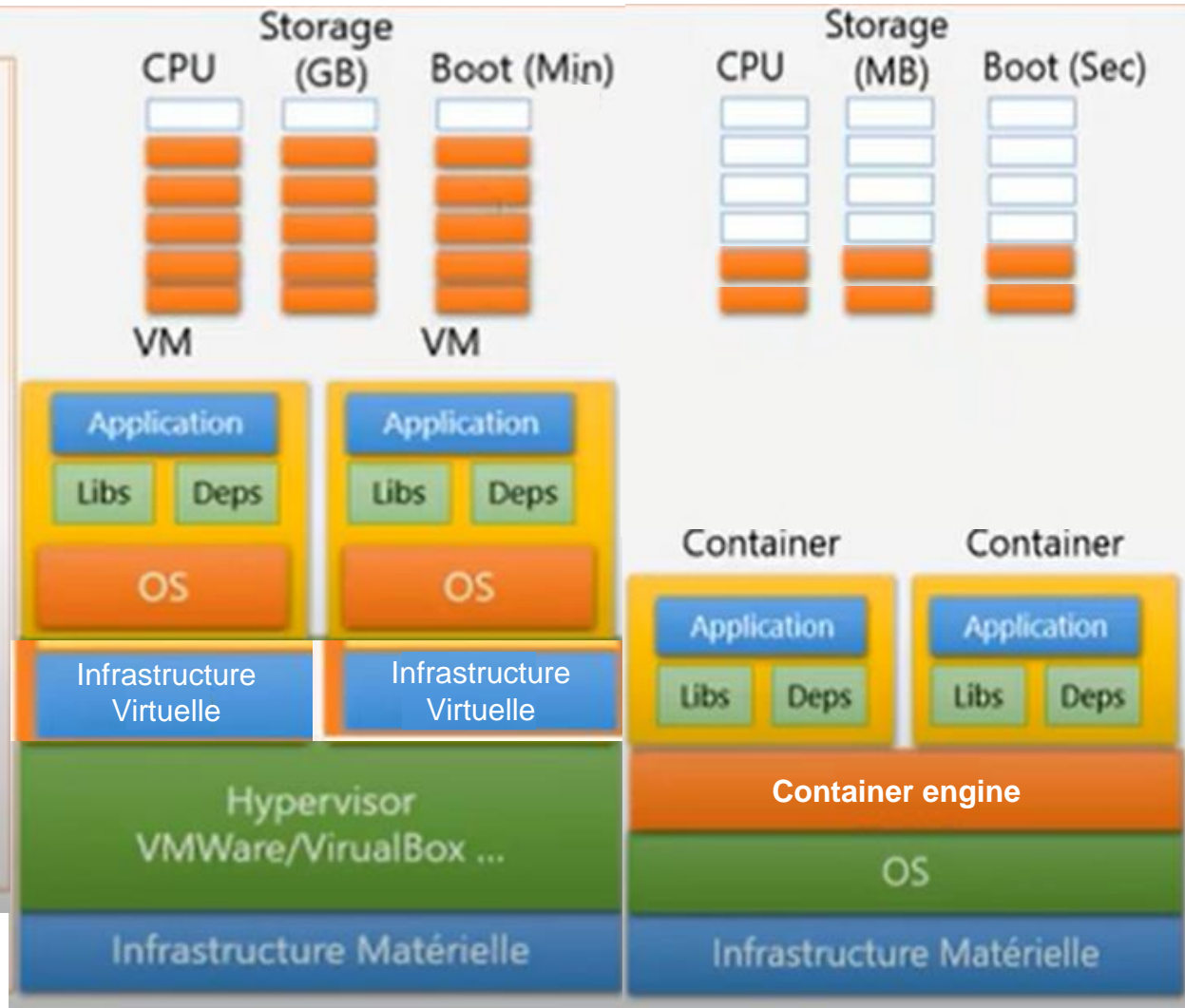
# Conteneurs VS machines virtuelles

- Machine Virtuelle :

- Permet de virtualiser une machine physique.
- Chaque VM a son propre OS
- Une VM consomme bcp de ressources (CPU, Stockage) et prend assez de temps pour booter (qq minutes).

- Conteneur :

- Permet de créer un environnement d'exécution des applications
- Les conteneurs utilisent le même OS
- Tous les conteneurs utilisent le même kernel OS (Linux), Consomme peu de ressources, boot rapide (qq secondes)



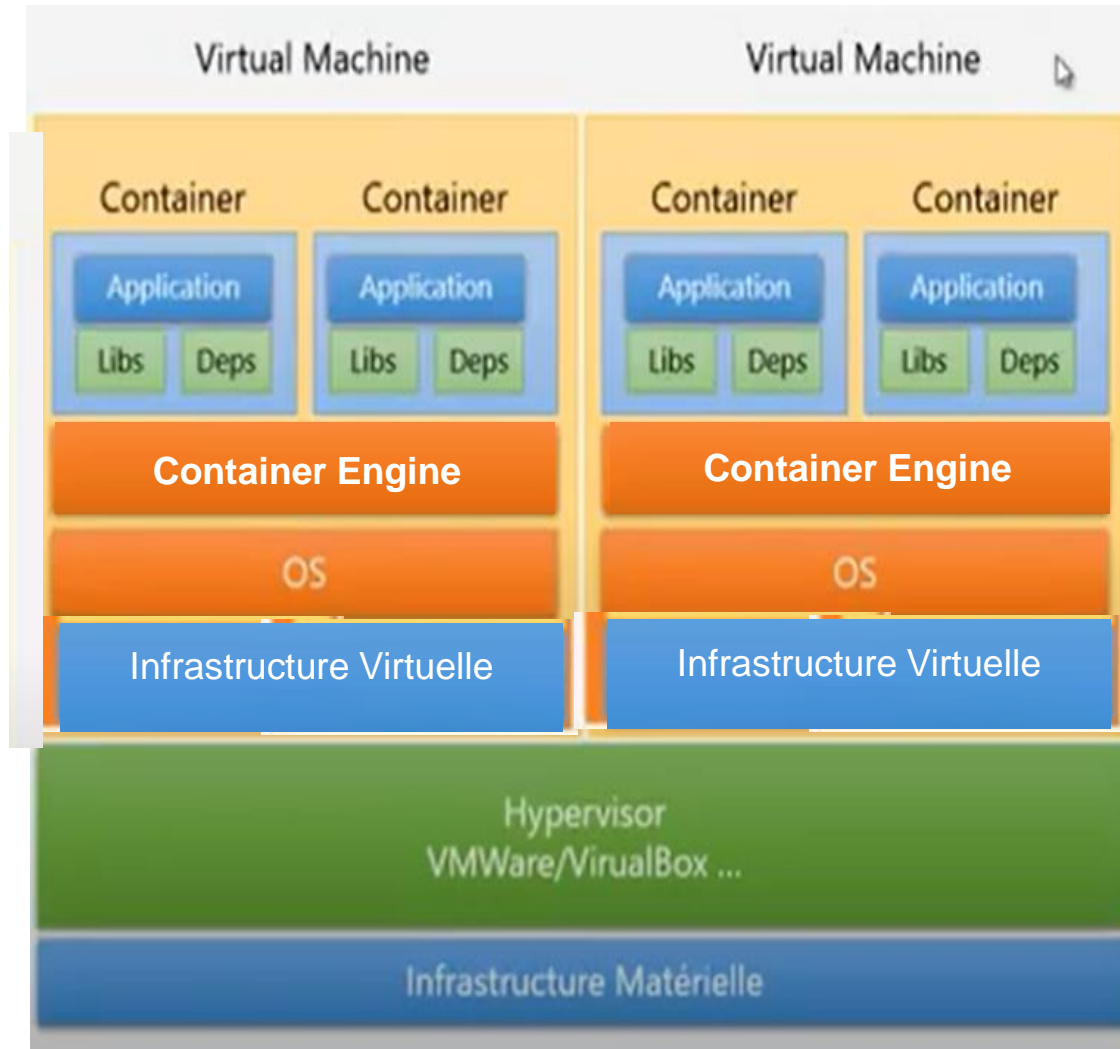
# Conteneurs VS machines virtuelles

Caractéristique	Conteneurs	Machines Virtuelles
Isolation	Isolation des processus, des systèmes de fichiers et des réseaux.	Isolation complète incluant le système d'exploitation.
Performance	Légers et partagent le même noyau avec l'hôte, offrant des performances élevées.	Plus lourdes en raison de l'émulation matérielle, ce qui peut entraîner une surcharge.
Taille	Plus petits en taille car ils ne contiennent que les dépendances de l'application.	Plus grands en taille en raison de l'inclusion du système d'exploitation complet.
Démarrage	Démarrage rapide grâce à la réutilisation du noyau de l'hôte.	Démarrage plus lent en raison de la nécessité de démarrer un système d'exploitation complet.
Flexibilité	Moins de flexibilité car ils partagent le même noyau avec l'hôte.	Plus de flexibilité car ils peuvent exécuter différents systèmes d'exploitation.
Sécurité	Moins sécurisé car une faille dans le noyau de l'hôte peut affecter tous les conteneurs.	Plus sécurisé car chaque VM est complètement isolée, réduisant ainsi la surface d'attaque.
Portabilité	Hautement portable car ils incluent toutes les dépendances nécessaires à l'application.	Moins portable car ils dépendent du système d'exploitation spécifique.

# Des conteneurs dans les machines virtuelles ?

- Conteneur ne vient pas pour remplacer la machine virtuelle.
- Dans la pratique on utilise les deux:
  - Les machines virtuelles pour virtualiser les machines
  - Utiliser le conteneur pour isoler les environnements d'exécution des applications dans des machines virtuelles.

➔ Ceci pour tirer les bénéfices des technologies



# Inconvénients des conteneurs

---

- Les conteneurs doivent être compatibles avec le système sous-jacent. → **Ils ne sont pas cross-platform.**
  - ✓ Les OS Linux ARM exécutent des conteneurs Linux ARM,
  - ✓ les OS Linux x86 exécutent des conteneurs Linux x86
  - ✓ les OS Windows x86 exécutent des conteneurs Windows x86.

# Environnements d'exécution

---

- Des exemples courants d'environnements d'exécution de conteneur sont :

- runC



- Containerd



- Docker



- Podman



- L'outil « Docker » est l'outil le plus populaire !!



docker.



# Arrivé du Docker

---



- Docker a été créé pour les besoins d'une société de Platform as a Service (PaaS) appelée **DotCloud**.
- Arrivé sur le marché en 2013.
- Open Source → gratuit.
- Permet de procurer une API de haut niveau pour encapsuler des applications.
- Basé initialement sur LXC.
- Repose maintenant sur leur propre technologie libcontainer.



# Docker

---



- Docker est un outil permettant d'**empaqueter une application et ses dépendances** dans un conteneur virtuel, qui pourra être exécuté sur n'importe quel **serveur Linux**.
- Docker **étend** le format de **conteneur** Linux standard, **LXC**, avec une **API de haut niveau** fournissant une solution de virtualisation qui exécute les processus de façon isolée.
- Cela permet de garantir la fiabilité d'exécution et la stabilité d'une application.

# Docker, pour quoi faire ?

---

- **Le développement** : cela permet de facilement avoir le même environnement de développement qu'en **production**. Cela permet également de pouvoir sur la même machine, tester avec plusieurs versions d'un même logiciel.
- **Le déploiement** : puisque Docker a pour vocation de conteneuriser des applications, il devient simple de créer un conteneur pour une application, et la dispatcher. Un conteneur qui fonctionne sur une machine avec une distribution X, fonctionnera sur une autre machine avec une distribution Y.
- **L'installation des applications** : étant donné que Docker propose une multitude d'outils, il est facile et rapide d'installer une application : bien souvent une seule ligne de commande suffit pour avoir par une application fonctionnelle.

# Docker

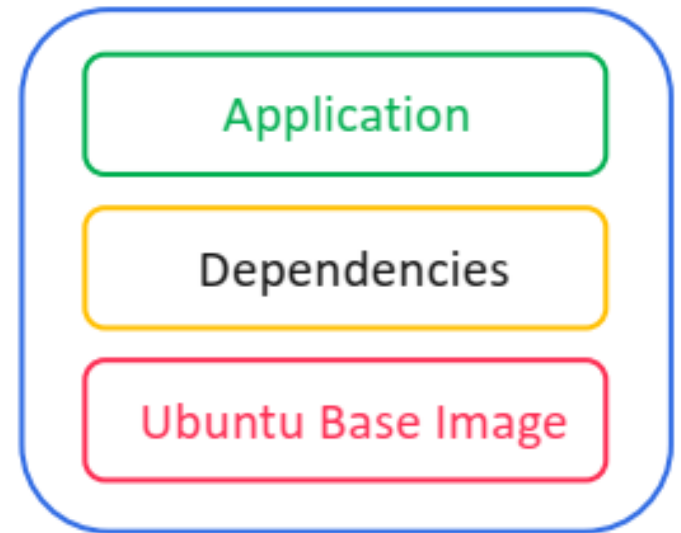
---

- Les points forts :
  - Installation simple (Linux, OSX, Windows)
  - Ligne de commande très sympathique (docker help)
  - Langage de description des images (avec notion de parent)
  - Communauté très active
  - API pour le pilotage:
  - GUI, Orchestration, hébergement cloud, intégration continue, OS, ...

# Exemple

---

Ubuntu + Python + Dependencies



**Conteneur**

# Qu'apporte la nouvelle version de docker?

---

Docker existe maintenant sous deux versions :

- Pour faciliter la gestion des architectures complexes, Docker a construit une plateforme de Containers-as-a-Service. Baptisée Docker Enterprise Edition (Docker EE)
- Docker possède également développé une version destinée aux développeurs et à tous ceux qui veulent apprendre Linux, Docker Community Edition baptisée (Docker CE)

# Evidement, il faut vivre : CE vs EE

---



# Qui utilise Docker?

---





# Installation Docker

---

- **Docker for Linux**
- **Docker for Windows** : Win10 Pro/Ent only Uses Hyper-V with tiny Linux VM for Linux Containers
- **Docker for MAC**
- **Online Emulator** : <https://labs.play-with-docker.com/>

# Docker

---

- Terminologie :
  - **Client/server** : outil utilisant l'API du serveur/Daemon
  - **Image** : conteneur en lecture seule
  - **Conteneur** : élément exécutable
  - **Docker hub** : répertoire (dépôts) public
- Analogie avec le développement objet :
  - **Images** équivalentes aux **classes**
  - Les **couches** sont équivalentes à l'**héritage**
  - Les **conteneurs** sont **des instances d'objets**



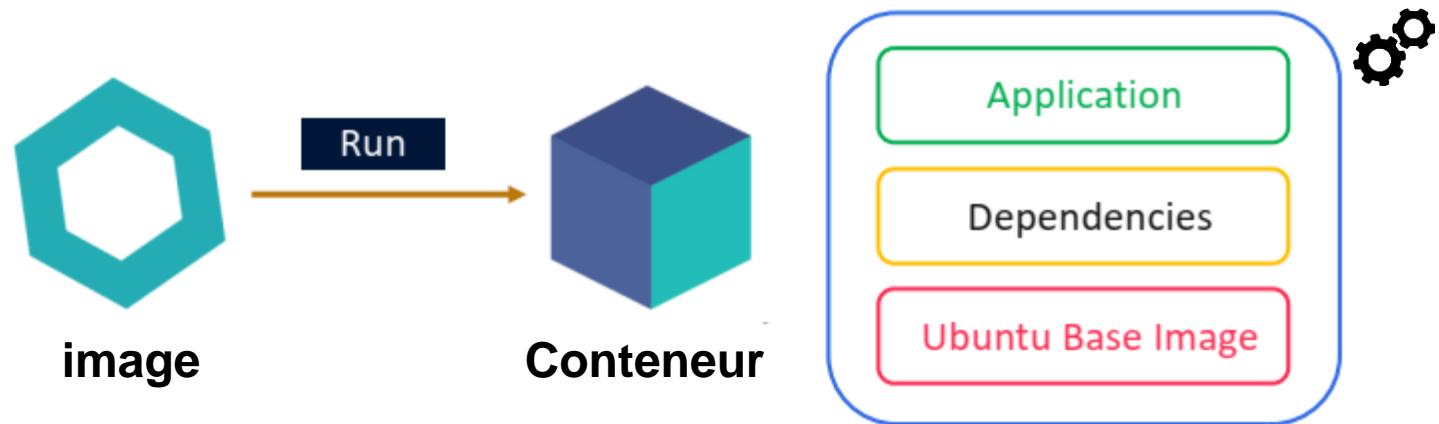
# Image Docker

---

- Tout comme les machines virtuelles, les conteneurs Docker sont basés sur des images.
- Une image est un modèle en **lecture seule** qui contient **toutes les instructions** dont le moteur Docker a besoin afin de **créer un conteneur Docker**.
- Une image portable d'un conteneur est décrite comme image Docker sous la forme d'un fichier texte, on parle alors d'un « **Dockerfile** ».
- Si un conteneur doit être démarré sur un système, un paquet avec l'image correspondante est **chargé en premier**, si elle n'existe pas localement. Donc, l'image chargée fournit le système de fichiers requis pour l'exécution, y compris tous les paramètres.

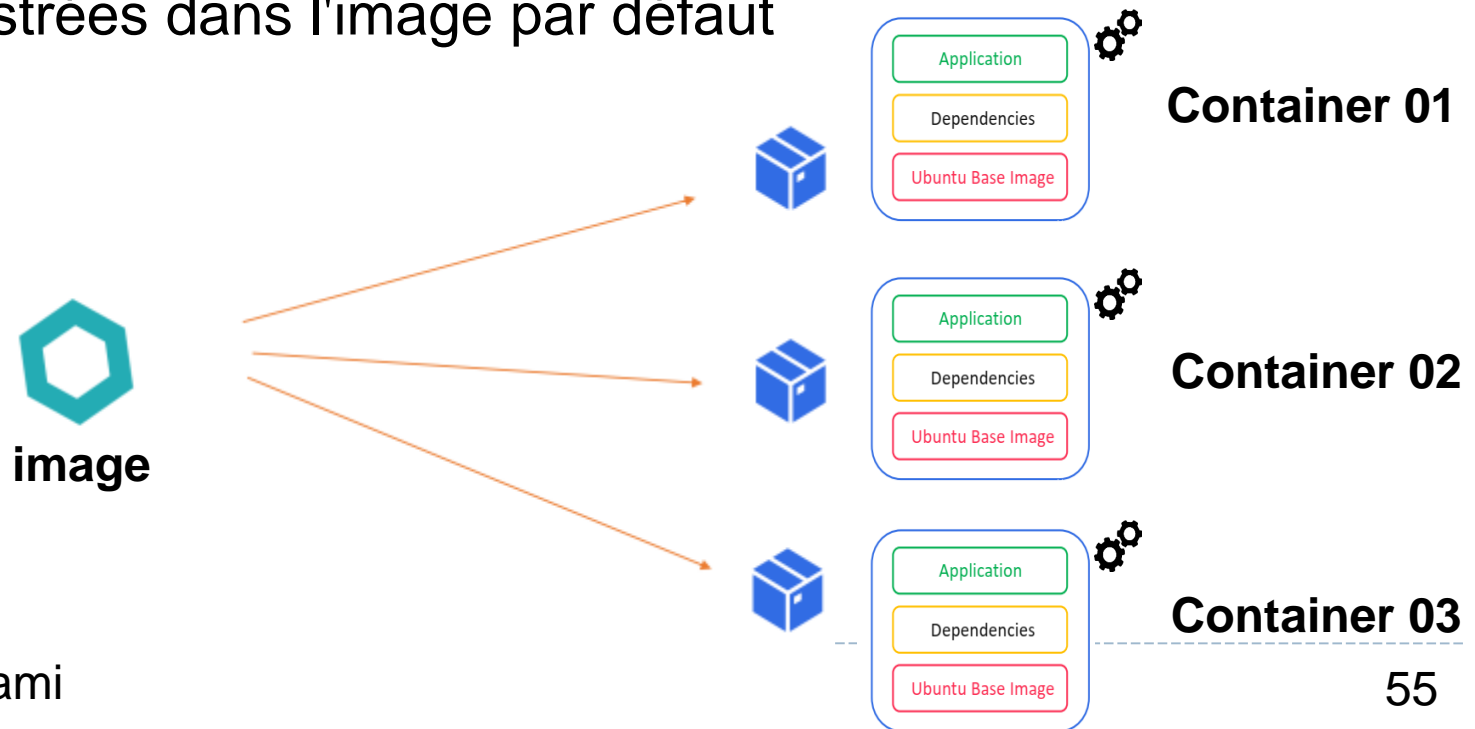
# Conteneur Docker

- Les conteneurs sont **une instance exécutable** d'images ou d'applications prêtes créées à partir d'images Docker. Grâce à l'API Docker ou à la CLI, nous pouvons créer ou supprimer un conteneur.
- Les conteneurs peuvent être connectés à un ou plusieurs réseaux, même créer une nouvelle image ou attacher un stockage à son état actuel. Les conteneurs sont par défaut isolés les uns des autres et de leur machine hôte.

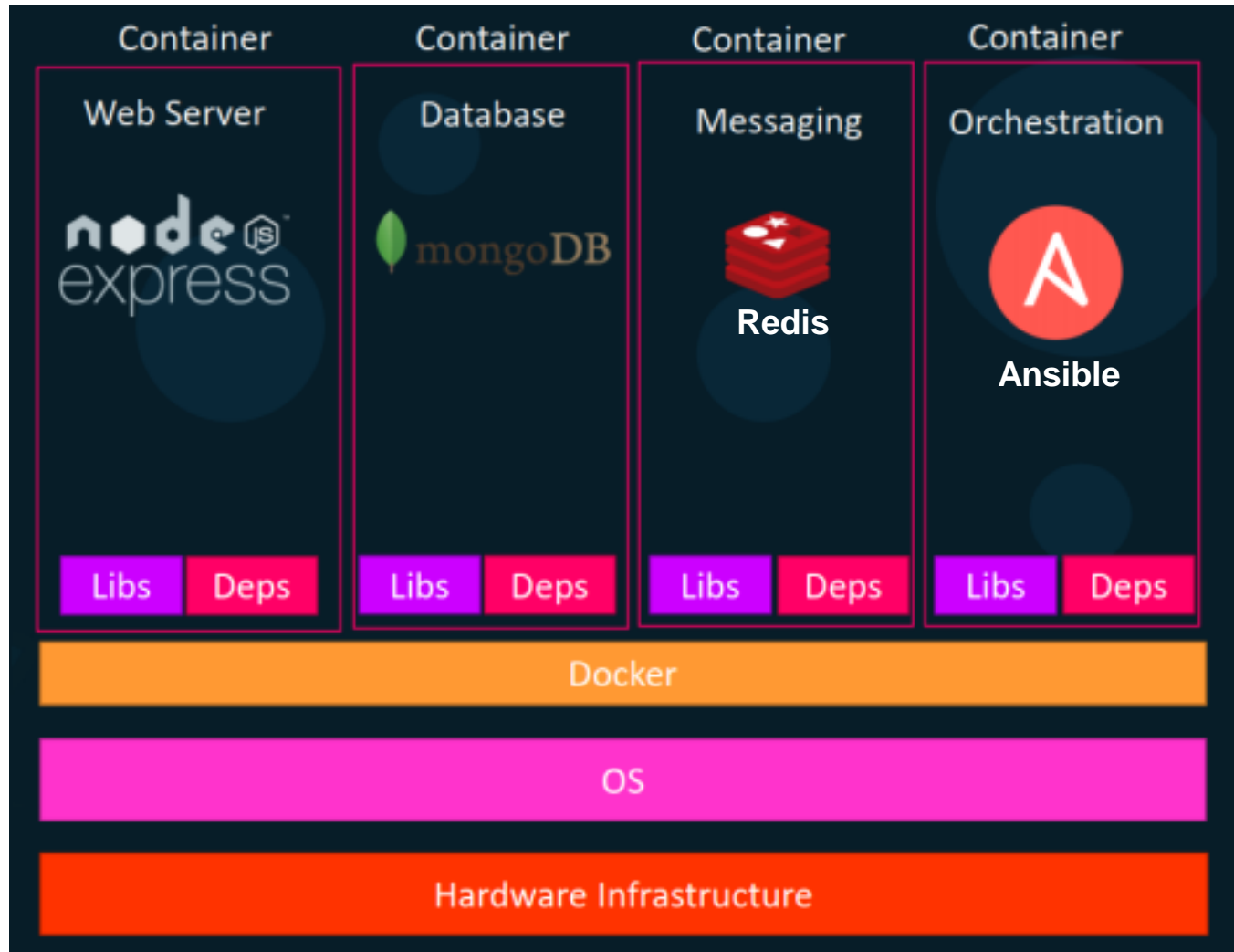


# Conteneur Docker

- Nous pouvons exécuter n'importe quel nombre de conteneurs basés sur une image et Docker s'assure que chaque conteneur créé a un nom unique dans l'espace de noms.
- L'image Docker est un modèle en lecture seule. Les modifications apportées aux conteneurs ne seront pas enregistrées dans l'image par défaut



# Conteneur Docker



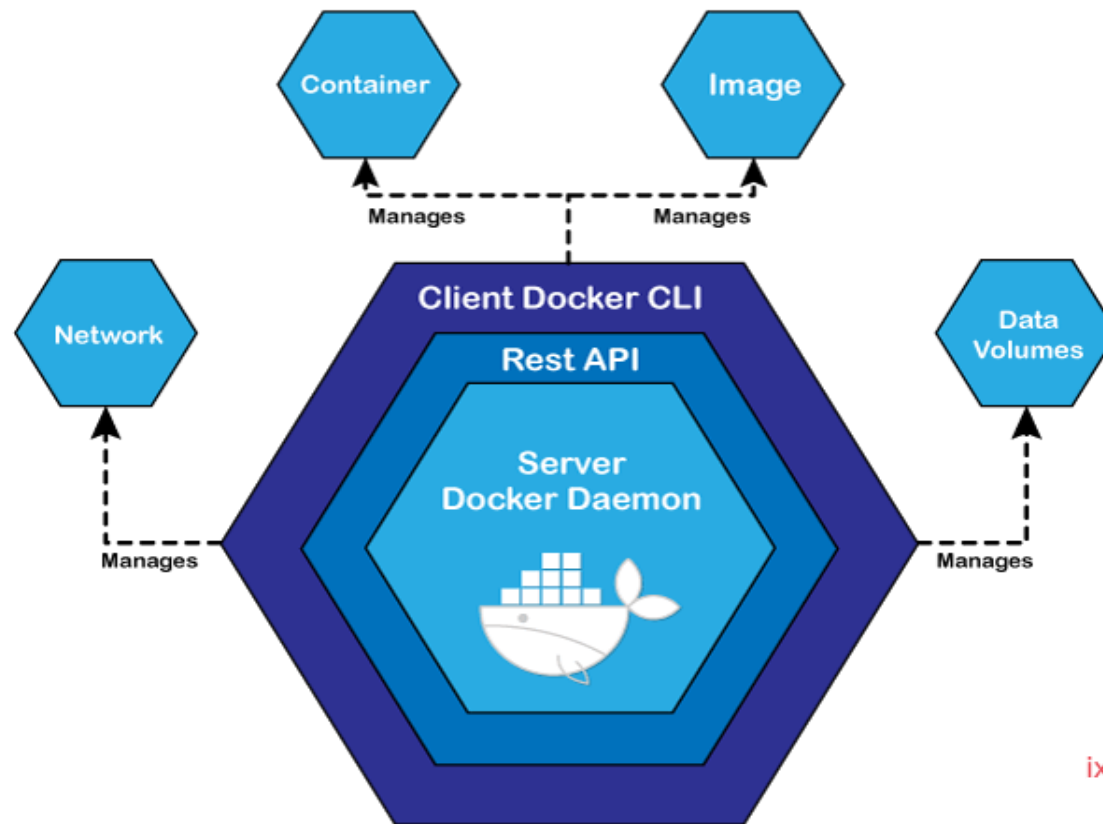
# Docker Architecture

---

- Docker est considéré comme une application **client-serveur** constituant la base de la plateforme de conteneur.
- Docker se compose de deux composants de base à savoir :
  - ✓ Moteur Docker (Docker-Engine)
  - ✓ Registre Docker (Docker hub)

# Le moteur Docker (Docker-Engine)

- L'architecture du « moteur Docker » est divisée en trois composants :



ix



# Le moteur Docker (Docker-Engine)

---

- Les trois composants sont :
  - ✓ Terminal du système d'exploitation (Command-Line Interface, CLI) qui interagit avec le daemon docker via l'API REST et permet aux utilisateurs de le contrôler grâce aux scripts ou aux entrées utilisateur.
  - ✓ Interface de programmation REST (API REST) basée sur le paradigme de programmation REST en spécifiant un ensemble d'interfaces qui permettent à d'autres programmes de communiquer et de donner des instructions au daemon Docker.
  - ✓ Le daemon docker : Il s'exécute en arrière-plan sur le système hôte et sert à contrôler le moteur Docker de manière centralisée. Dans cette fonction, il crée et gère toutes les images, conteneurs ou réseaux.
- Le Terminal du système d'exploitation (CLI) utilise l'API REST pour interagir ou contrôler le démon Docker via des commandes CLI directes ou des scripts.

# Registre Docker (Docker hub)


---



- Le registre Docker est basé sur le Cloud pour les référentiels (dépôts) de logiciels, en d'autres termes une sorte de **bibliothèque** pour stocker les images.
- Plusieurs images sont disponibles (des images de base et des images préconfigurées).
- Le service en ligne est divisé en un espace public et un espace privé.
  - **L'espace public** offre aux utilisateurs la possibilité de télécharger leurs propres images et de les partager avec la communauté.
  - **L'espace privé** présente une zone restreinte où les images téléchargées du registre ne sont pas accessibles au public et peuvent, par exemple, être partagées au sein de l'entreprise ou avec des amis et des connaissances.







# Registre Docker (Docker hub)

- Il est accessible via : <https://hub.docker.com> :

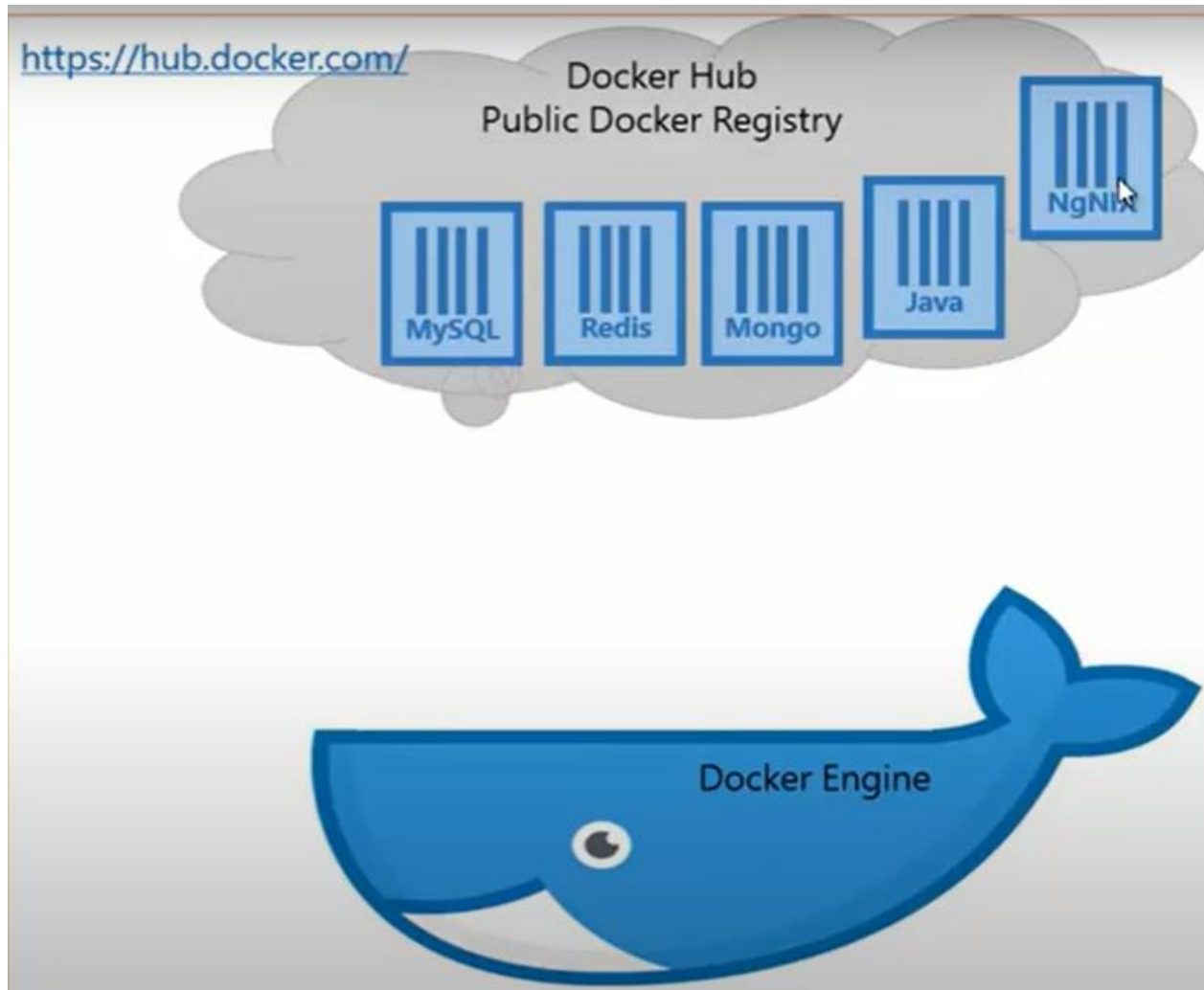


DashboardExploreOrganizationsCreate▼myfreedockerid▼

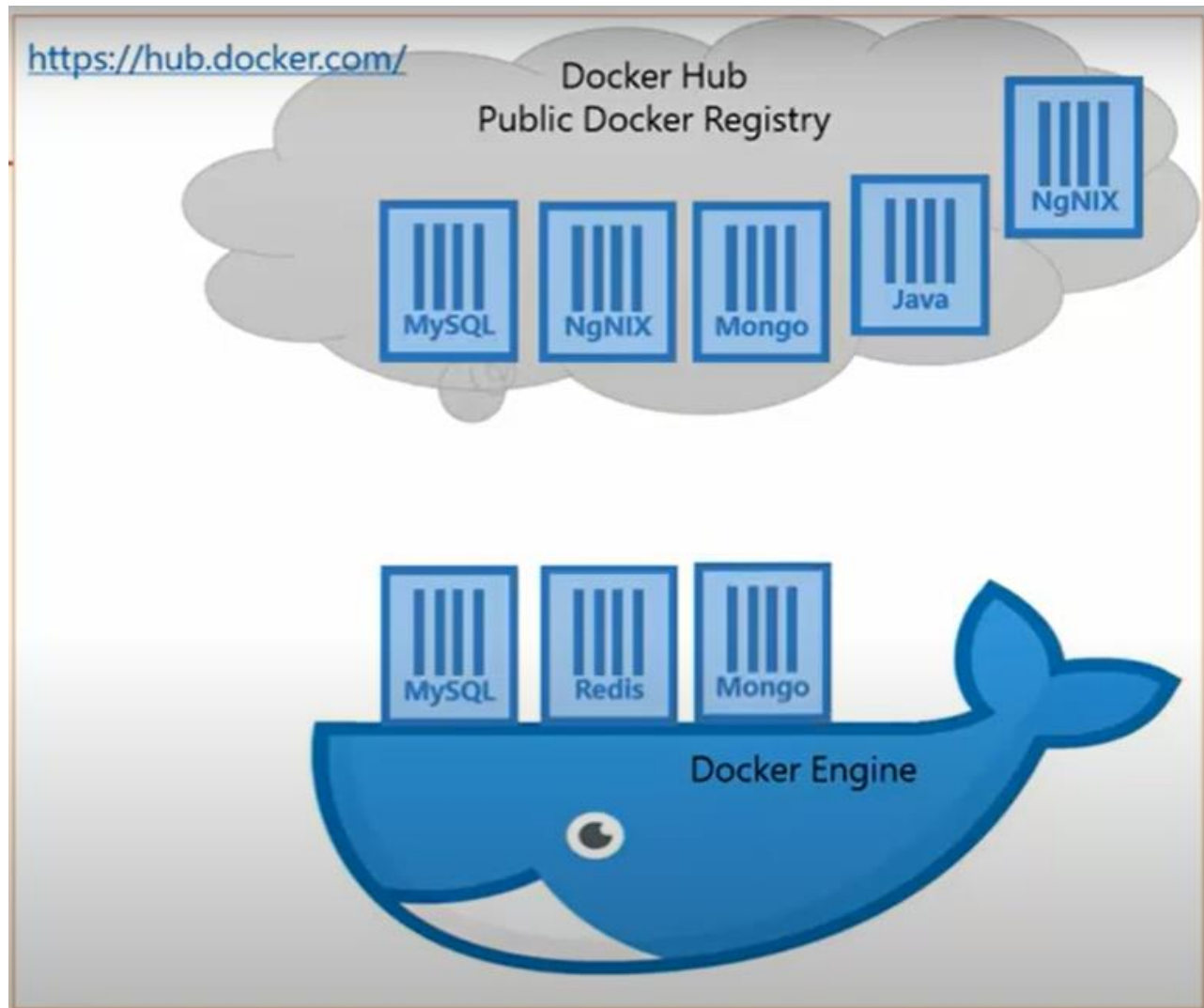
### Explore Official Repositories

 <b>nginx</b> official	5.5K STARS	10M+ PULLS	> DETAILS
 <b>redis</b> official	3.5K STARS	10M+ PULLS	> DETAILS
 <b>busybox</b> official	945 STARS	10M+ PULLS	> DETAILS
 <b>ubuntu</b> official	5.6K STARS	10M+ PULLS	> DETAILS
 <b>alpine</b> official	2.0K STARS	10M+ PULLS	> DETAILS
 <b>registry</b> official	1.4K STARS	10M+ PULLS	> DETAILS

# Registre Docker (Docker hub)

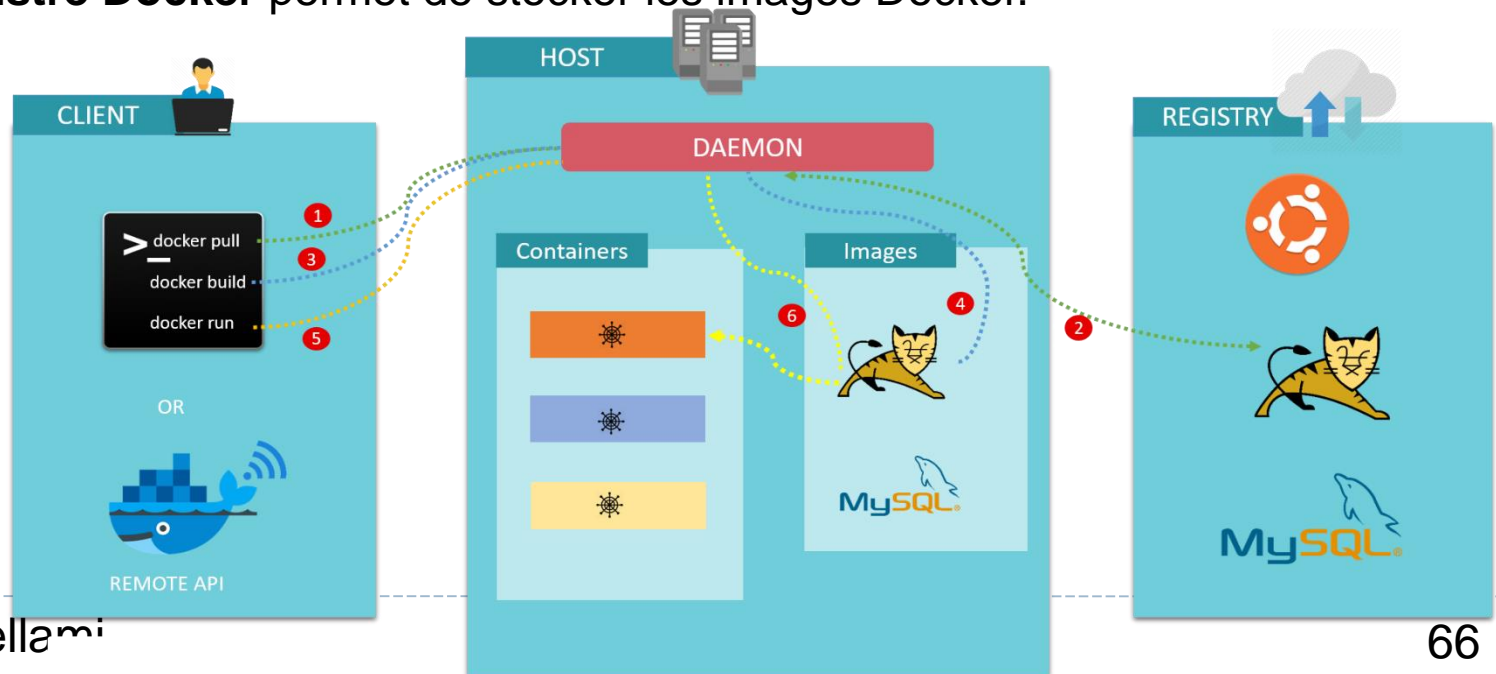


# Registre Docker (Docker hub)



# Docker : structure et fonctions

- Docker fonctionne sur une architecture **client-serveur**. Il comprend :
  - Le **client Docker** est utilisé pour envoyer des commandes au démon Docker afin de déclencher les commandes de création, gestion et surveillance des conteneurs, des images, des réseaux, etc.
  - **Démon Docker (ou Serveur Docker)** est responsable de l'exécution des commandes Docker et de la gestion des ressources système pour les conteneurs.
  - Le **registre Docker** permet de stocker les images Docker.



# Les commandes de base

---

- **Connaitre la version du docker**

\$ docker --version

- **Accéder au menu d'aide du docker**

\$ docker --help

- **Avoir l'aide sur une commande spécifique « pull »**

\$ docker pull --help

- **Rechercher une image dans le hub de docker**

\$ docker search ubuntu

---

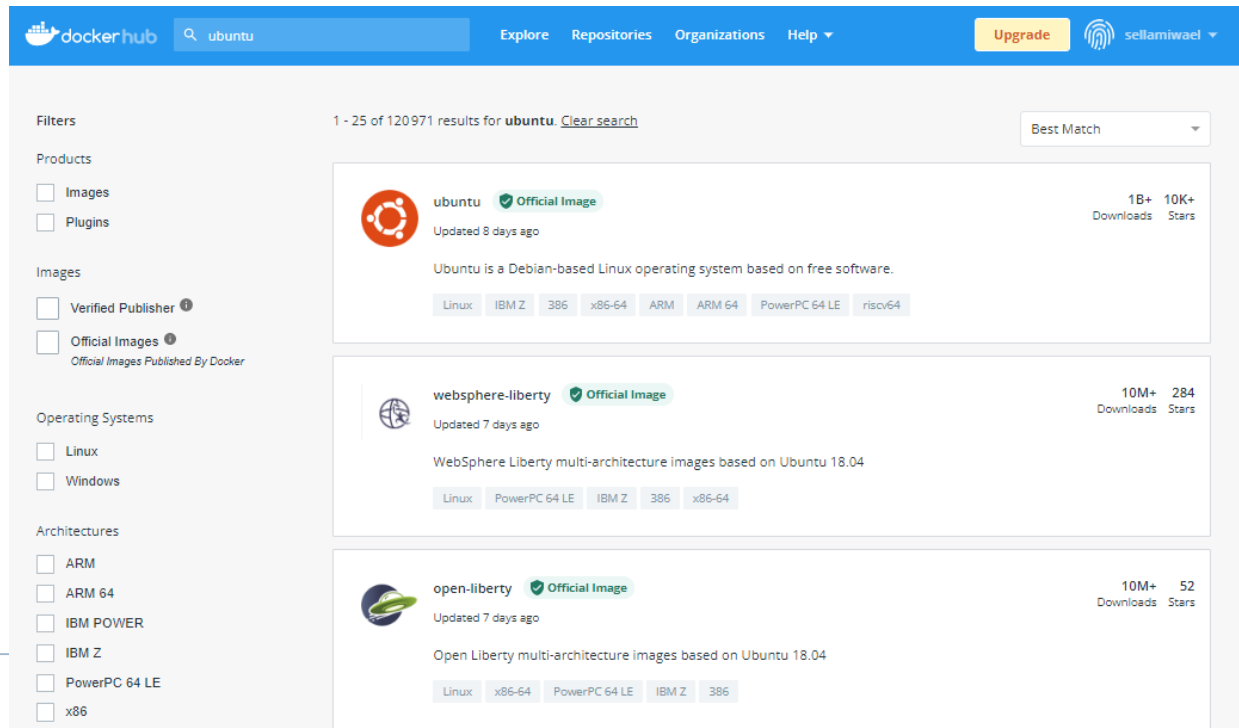
# Gestion des images et des conteneurs





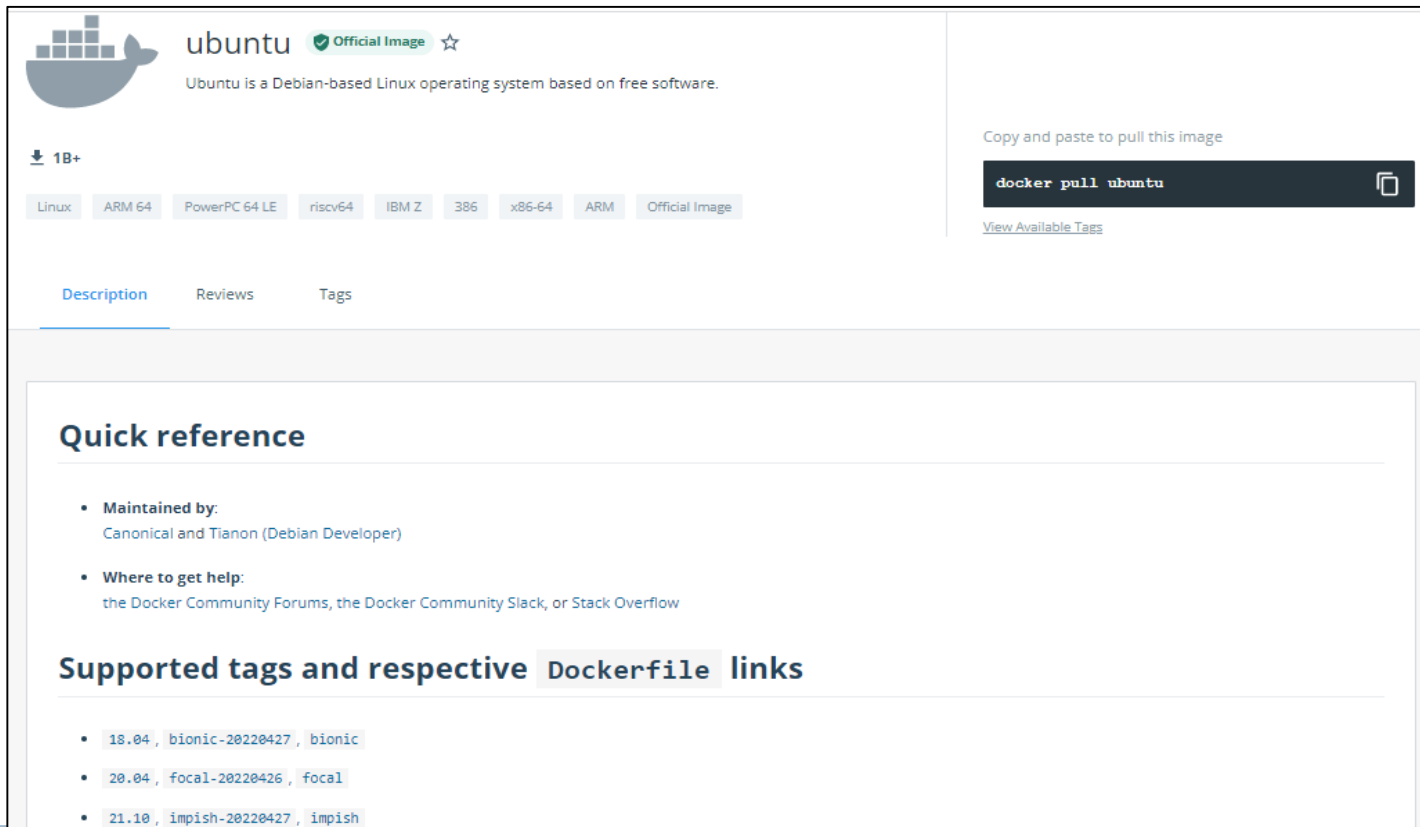
# Télécharger les images Docker

- **Exemple :** afin de trouver l'image «ubuntu», il faut accéder à la page d'accueil de «Docker-hub» et en tapant «ubuntu» dans la barre de recherche à droite du logo de Docker.
- Dans les résultats de la recherche, cliquer sur la ressource pour accéder au dépôt public de cette image.



# Télécharger les images Docker

- Dans la zone d'en-tête de la page, nous trouvons le nom de l'image, la catégorie du dépôt et l'heure du dernier téléchargement (last pushed).



# Télécharger les images Docker

---

- Structure d'une commande en Docker :

**\$ docker <command> <options> <image>**

- Pour télécharger une image à partir d'un dépôt:

**\$ docker pull [OPTIONS] NAME [:TAG|@DIGEST]**

- Déterminer l'image en spécifiant le nom de l'image (NAME).
- Spécifier les tags (:TAG) et les numéros d'identification uniques (@DIGEST) afin de télécharger une version spécifique d'une image.

- **Exemple** : \$ docker pull hello-world

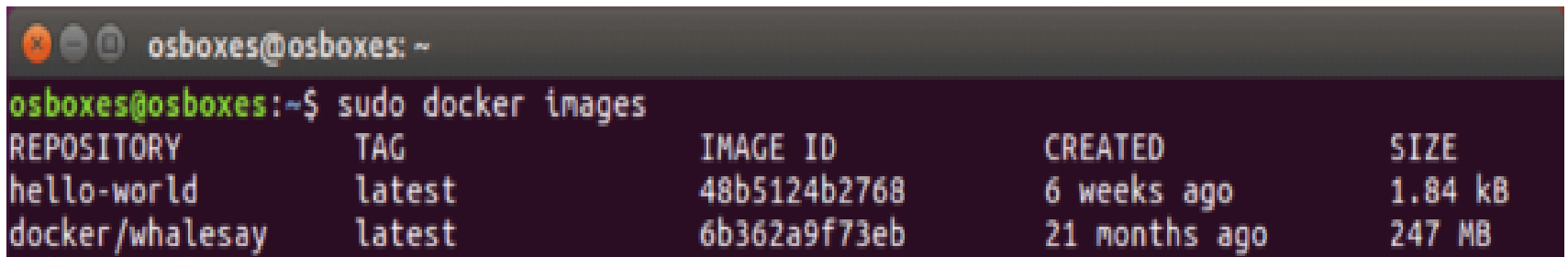
# Télécharger les images Docker

---

- Afficher une vue d'ensemble de toutes les images sur votre système :

**\$ docker pull hello-world**

**\$ docker images**



```
osboxes@osboxes: ~  
osboxes@osboxes:~$ sudo docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
hello-world         latest             48b5124b2768       6 weeks ago        1.84 kB  
docker/whalesay     latest             6b362a9f73eb       21 months ago      247 MB
```

- Lorsqu'on démarre un conteneur, l'image sous-jacente est téléchargée sous forme de copie à partir du dépôt et stockée de façon permanente sur votre ordinateur.
- Un nouveau téléchargement n'est lancé que si la source de l'image change, par exemple si une version plus récente est disponible dans le dépôt.

# Supprimer les images Docker

---

Pour supprimer une image :

**\$ docker image rm [OPTIONS] IMAGE [IMAGE...]**

**ou**

**\$ docker rmi [OPTIONS] IMAGE [IMAGE...]**

**Exemple : \$ docker rmi hello-world**

# Démarrer un conteneur

---

- Pour démarrer une image du docker, utiliser la commande :

```
$ docker run [OPTIONS] IMAGE [:TAG|@DIGEST] [CMD] [ARG...]
```

## N.B :

- ✓ Des configurations optionnelles peuvent être définies par des arguments supplémentaires (--name, -network, -d, -it, ....).
- ✓ La seule partie obligatoire est le **nom de l'image** du docker désiré.
- ✓ Si l'image n'existe pas, elle sera d'abord téléchargée à partir du « docker hub ».

# Démarrer un conteneur

---

- **\$ docker run image\_name:tag**
  - ✓ Pour spécifier la version (:tag)
  - ✓ **Exemple :**

```
$ docker run --name ContWeb nginx
```

```
$ docker run --name ContWeb nginx:latest
```

```
$ docker run --name ContWeb nginx:1.21.3
```

# Démarrer un conteneur

---

- **\$ docker run -d image\_name**
  - ✓ -d : (**détacher**) continuer à utiliser la console pendant que votre conteneur tourne sur un autre processus;
  - ✓ La valeur par défaut de l'image est la dernière version

Exemple :

```
$ docker run -d --name ContWeb2 nginx
```



# Démarrer un conteneur

---

- **\$ docker ps**

- ✓ Permet de **lister** les conteneur qui sont **en cours d'exécution**
- ✓ Chaque conteneur créé dispose d'un identifiant unique et d'un nom du conteneur.

```
PS C:\Users\sellami> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
9816b85020a0   nginx    "/docker-entrypoint...."  5 minutes ago  Up 5 minutes  80/tcp       nostalgic_allen
```

# Démarrer un conteneur

- Pour afficher une vue d'ensemble de **tous les conteneurs actifs et non actifs**

**\$ docker ps -a**

```
osboxes@osboxes: ~  
osboxes@osboxes:~$ sudo docker ps -a  
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES  
f4b2a3131480       docker/whalesay    "cowsay What did t..." 27 minutes ago     Exited (0) 27 minutes ago              kickass_bartik  
07e4a970f186       docker/whalesay    "cowsay What did t..." 28 minutes ago     Exited (0) 28 minutes ago              serene_bohr  
a5fd4f05a980       docker/whalesay    "cowsay boo"          40 minutes ago     Exited (0) 40 minutes ago              zen_fermat  
98344a7a2a9a       docker/whalesay    "cowsay"              17 hours ago       Exited (0) 17 hours ago              wizardly_payne  
277225cdcf03       docker/whalesay    "cowsay boo"          17 hours ago       Exited (0) 17 hours ago              distracted_euclid  
74ffb508094b       docker/whalesay    "/bin/bash"           17 hours ago       Exited (0) 17 hours ago              sad_sammet  
6f88a421cd40       docker/whalesay    "cowsay boo"          18 hours ago       Exited (0) 18 hours ago              quirky_visvesvaraya  
fdccb23913a        hello-world         "/hello"              4 days ago         Exited (0) 4 days ago              quirky_turing  
1b75142bdc01       hello-world         "/hello"              5 days ago         Exited (0) 5 days ago              heuristic_montalcini  
d22ed3918a8a       hello-world         "/hello"              5 days ago         Exited (0) 5 days ago              silly_khorana  
osboxes@osboxes:~$
```

- La sortie du terminal comprend des informations telles que l'**ID** du conteneur respectif, l'**image** sous-jacente, la **commande exécutée** lorsque le conteneur a été démarré, l'**heure** à laquelle le conteneur respectif a été démarré et l'**état** actuel.

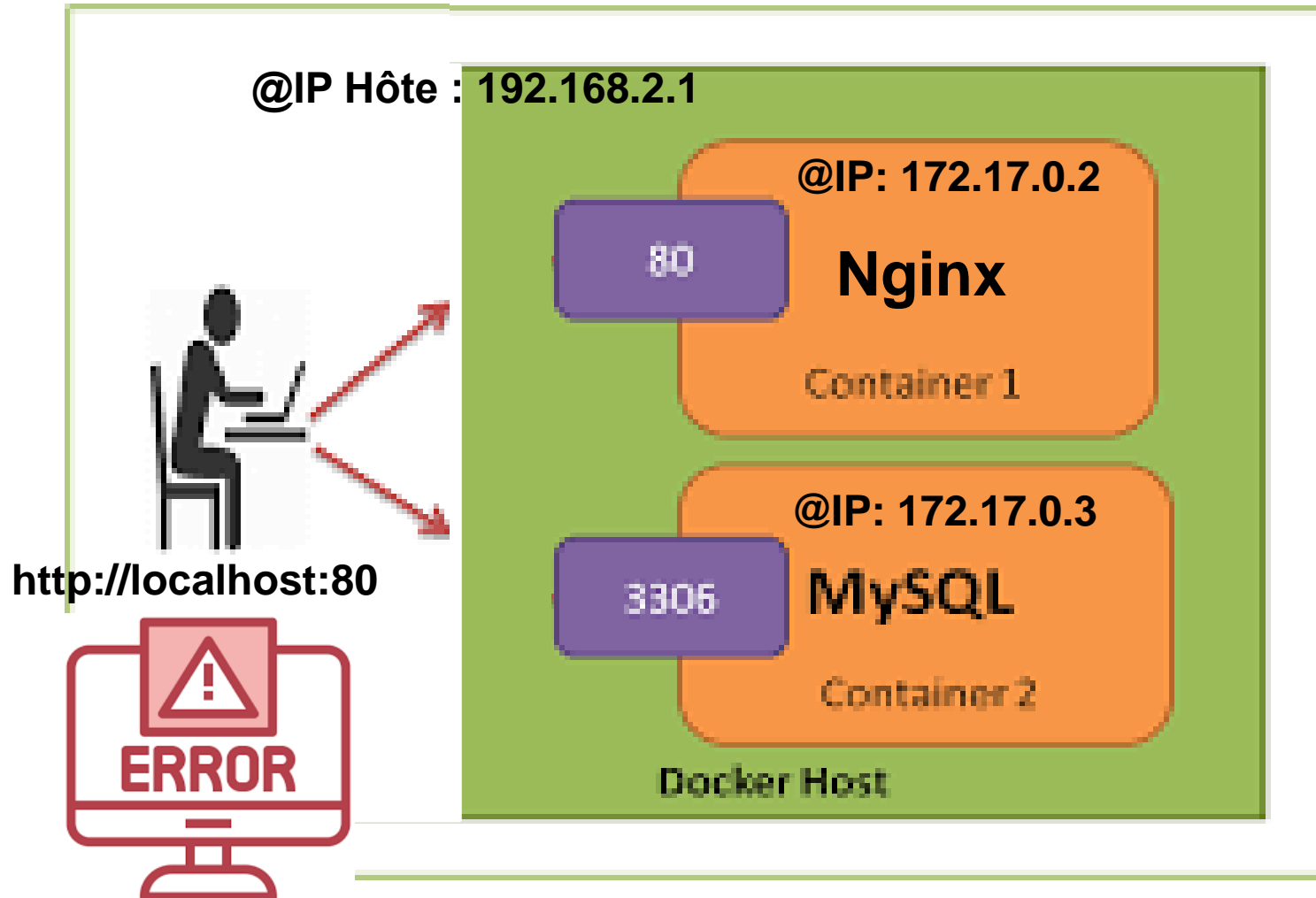
# Démarrage d'un conteneur

---

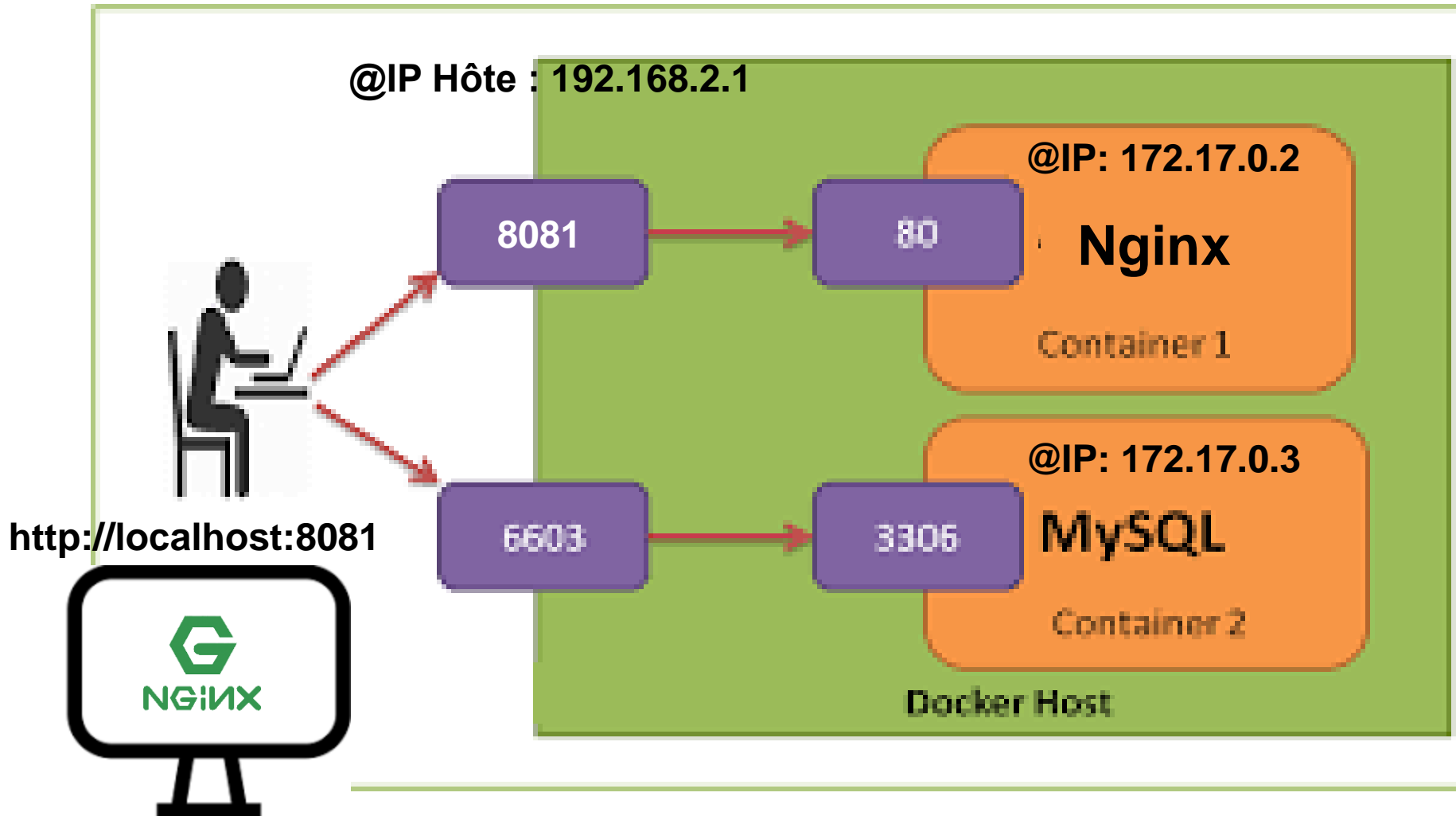
En quoi consiste le démarrage du container ?

1. Rechercher l'image ➔ Si l'image n'existe pas en local, alors téléchargement via le hub.
2. Créer un nouveau conteneur basé sur cette image et se préparer à démarrer
3. Attacher le conteneur au réseau privé et obtenir d'une adresse IP.
4. Ouvrir les ports pour répondre aux requêtes
5. Capturer des messages entrées-sorties

# RUN – Port Mapping



# RUN – Port Mapping

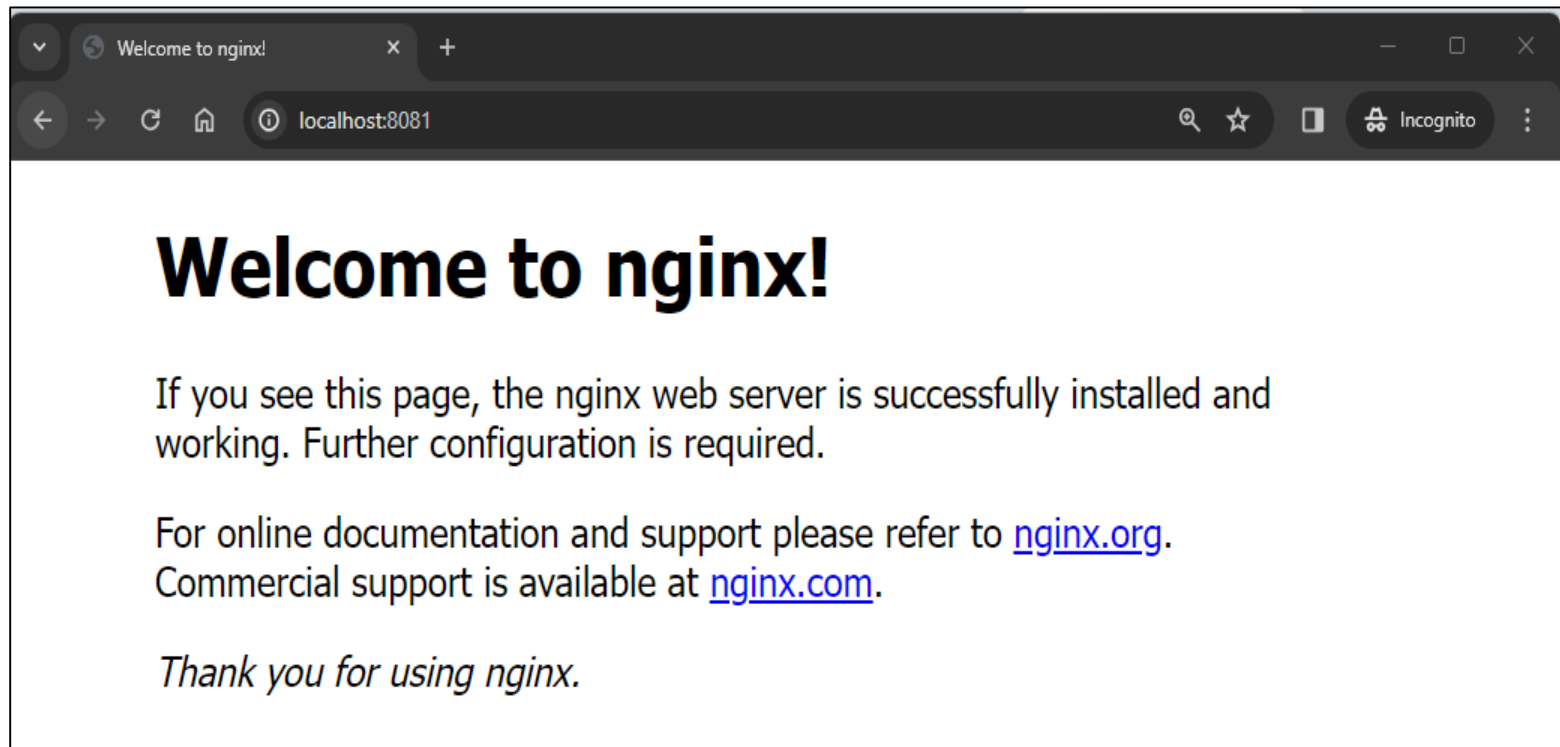


# Manipulation des conteneurs

---

Démarrons **le serveur Nginx** en container et exposons le port 80 sur le port 80 de notre machine :

```
$ docker run --name monServeurWeb -p 8081:80 -d nginx
```



# Démarrer un conteneur

---

Exemple :

**\$ docker run -it --name contdebian debian:stable-slim /bin/bash**

- run : on veut lancer le conteneur
- -it : on veut un terminal et être interactif avec lui
- Debian: l'image à utiliser pour ce conteneur
- /bin/bash : on lance « bash »

```
$ docker run -it --name contdebian debian:stable-slim /bin/bash
root@vm :/# cat /etc/issue
Debian GNU/Linux 12
root@vm :/# exit
```

- le daemon recherche d'abord l'image désirée dans le répertoire du fichier local.
- Comme aucun paquet du même nom n'est trouvé ici, une extraction du dépôt docker est initiée. Ensuite, le démon démarre le programme de « bash ».

# Arrêter/Démarrer

---

- Pour arrêter/démarrer un conteneur, il faut taper :
  - **\$ docker start contdebian**
  - **\$ docker stop contdebian**
- Pour supprimer un conteneur arrêté
  - **\$ docker rm <container-name/ID>**
- Pour supprimer tous les conteneurs arrêtés
  - **\$ docker container prune**



# Attach et Exec dans les conteneurs

---

- **Attach (Attacher) :**

Se connecte au sein du conteneur, permettant une interaction en temps réel avec ses processus.

**Exemple :**

```
$ docker start contdebian
```

```
$ docker attach contdebian
```

# Attach et Exec dans les conteneurs

---

- **Exec (Exécuter) :**

Exécuter de commandes spécifiques à l'intérieur du conteneur **sans entrer** dans son environnement interactif.

**Exemple :**

```
$ docker start contdebian
```

```
$ docker exec -it contdebian mkdir cloud2
```

```
$ docker exec -it contdebian ls
```

# EXEC Command

---

- Permet d'exécuter une commande dans un conteneur démarré.

**Exemple** : Créer un conteneur MySQL en définissant le mot de passe du root et créant un nouveau utilisateur « **wael** »

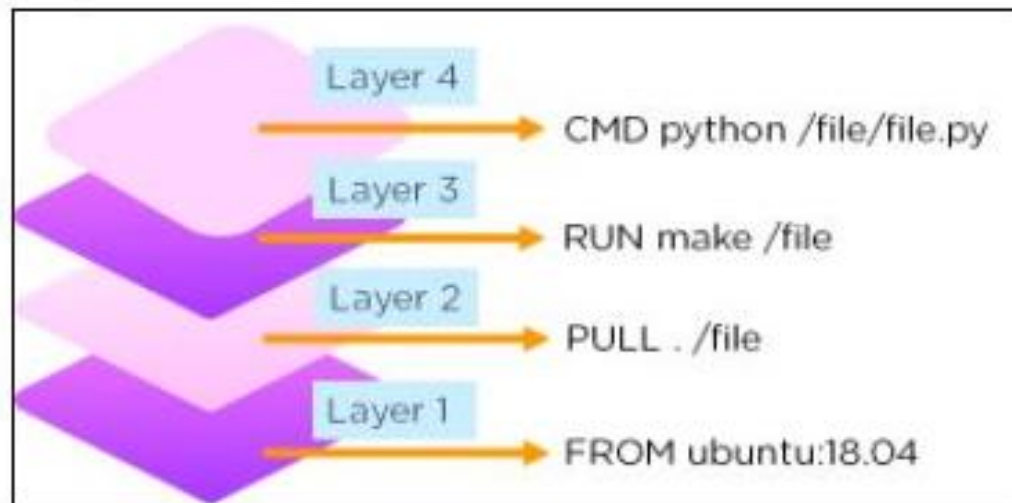
```
$ docker run --name mysqlContainer -e  
MYSQL_ROOT_PASSWORD=1234 -e MYSQL_USER=wael  
-e MYSQL_PASSWORD=1234 -d -p 3306:3306 mysql:5.7
```

- Exécuter une commande dans un conteneur démarré.

```
$ docker exec -it mysqlContainer mysql -u root -p
```

# Qu'est-ce qu'une image Docker ?

- Une image Docker est un paquet léger et autonome qui **comprend tout le nécessaire pour exécuter une application, y compris le code, le runtime, les bibliothèques, les variables d'environnement et les fichiers de configuration.**
- Il s'agit simplement d'un modèle en lecture seule . Il est utilisé pour stocker et expédier des applications.

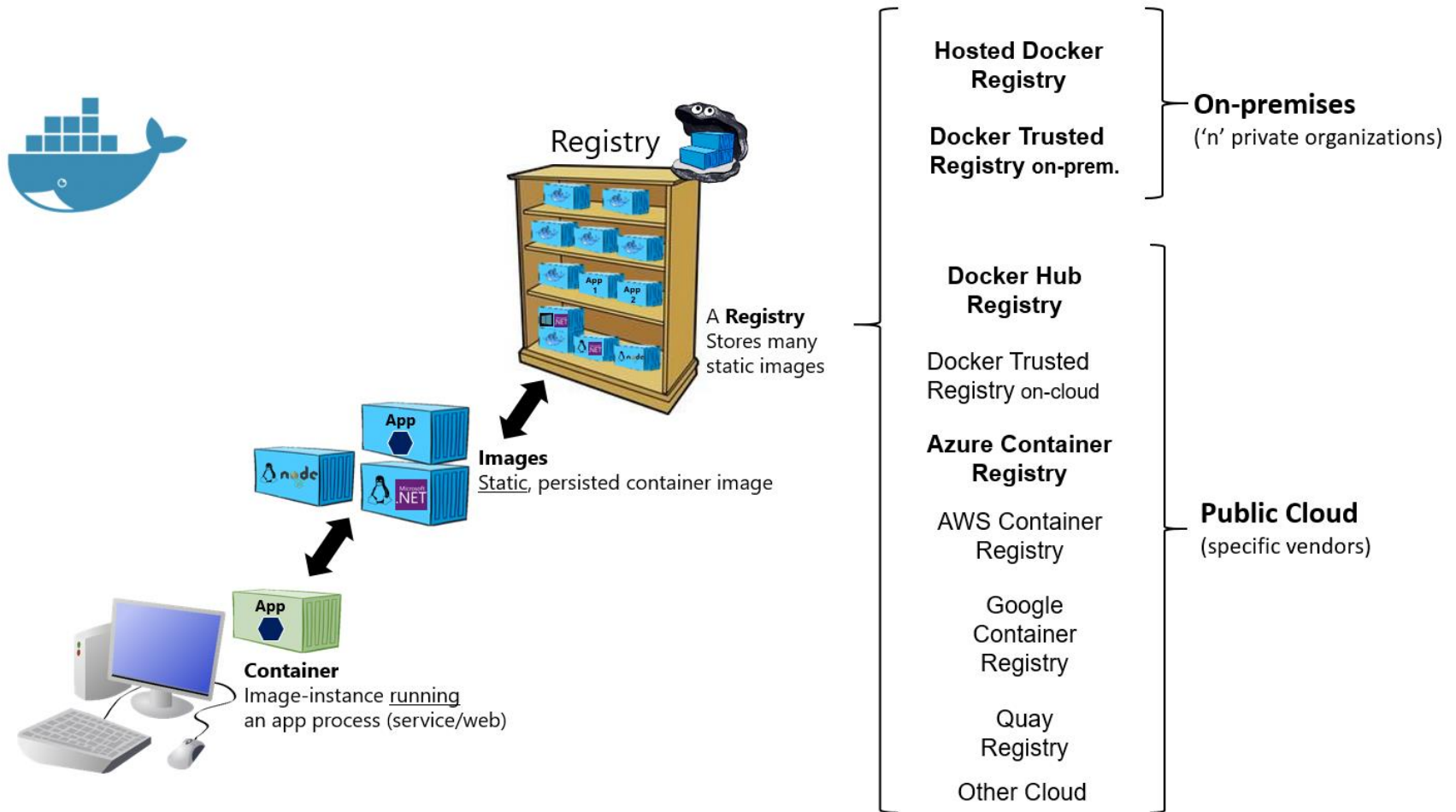


# Avantages des images Docker

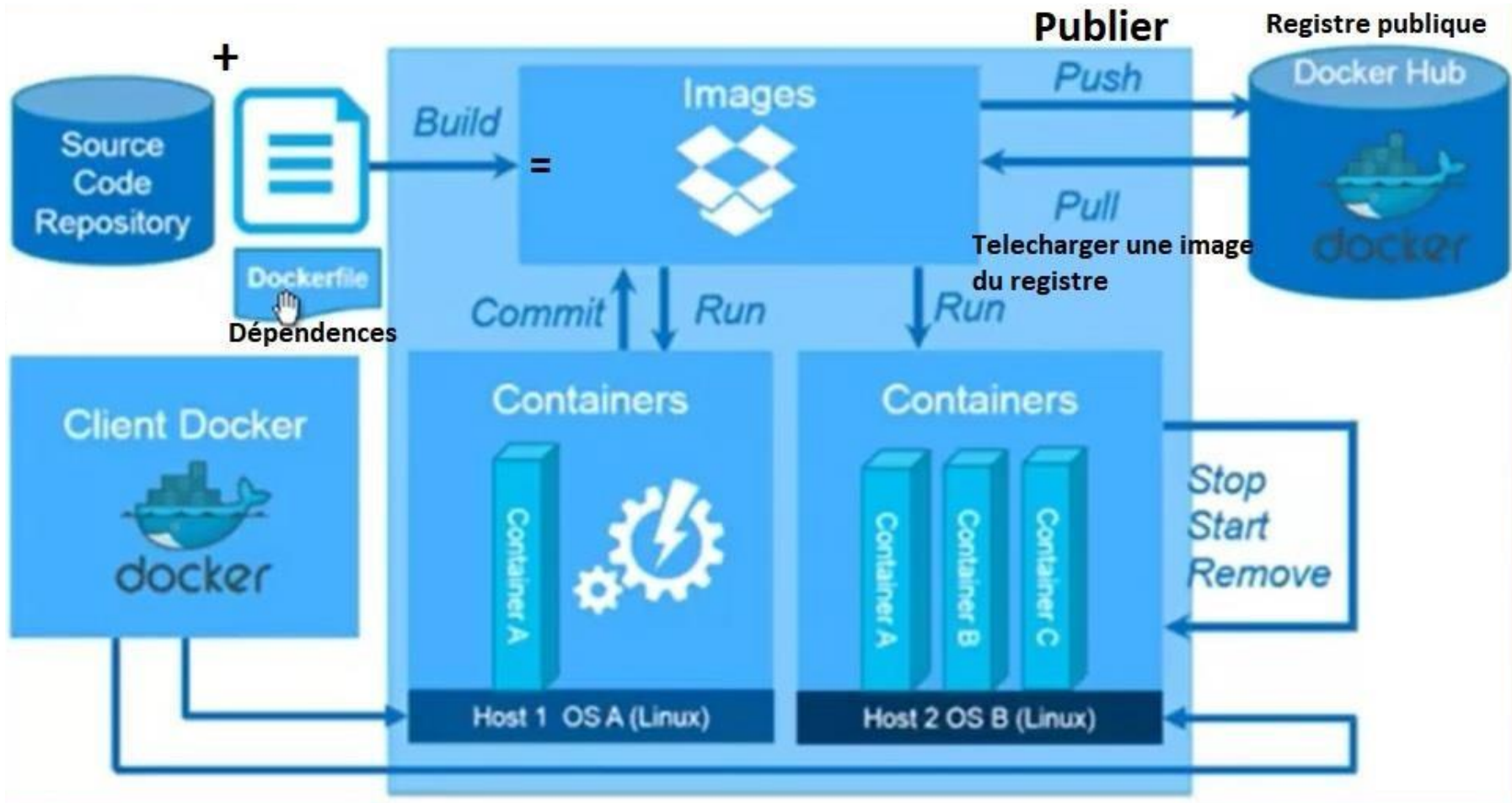
---

- **Portabilité** : Les images sont indépendantes du système d'exploitation et de l'infrastructure.
- **Rapidité** : Le déploiement d'une image est rapide et efficace.
- **Cohérence** : Assure une cohérence entre les environnements de développement, de test et de production.

# Avantages des images Docker



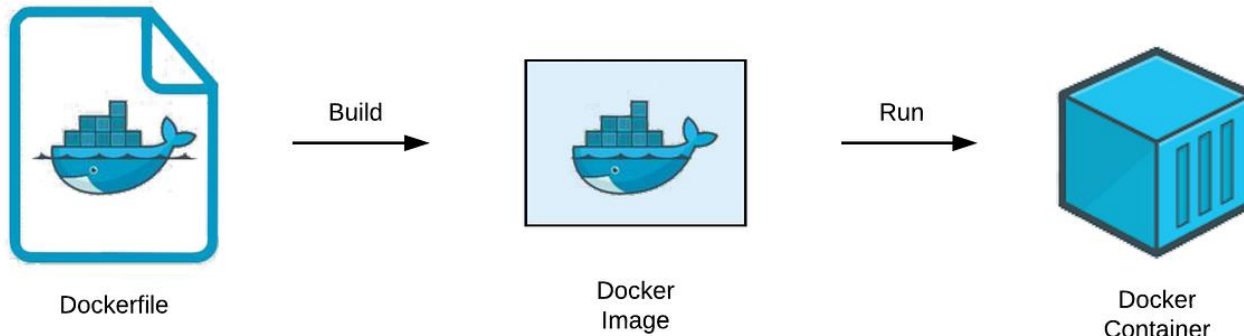
# Architecture Docker



# Dockerfile

---

- Un fichier Dockerfile est un fichier, «sans extension», qui contient une série d'instructions utilisées par Docker pour construire une image de conteneur.
- Ces instructions décrivent les étapes nécessaires pour configurer et préparer l'environnement dans lequel votre application s'exécutera à l'intérieur du conteneur Docker.





# Instructions du Dockerfile

---

Les instructions de Dockerfile :

- **FROM** : définir l'image **source**
- **CMD** : définir la commande qui doit être exécutée lors de démarrage du conteneur
- **RUN** : exécuter des **commandes** dans le conteneur ;
- **WORKDIR** : définir votre **répertoire de travail** ;
- **ADD** : ajouter des fichiers dans le conteneur;
- **COPY** : copier des fichiers dans le conteneur ;
- **EXPOSE** : définir les **ports d'écoute** par défaut ;
- **VOLUME** : définir le répertoire à partager avec la machine hôte.

# Dockerfile

---

## Structure de la commande :

INSTRUCTION

ARGUMENT

## Dockerfile

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

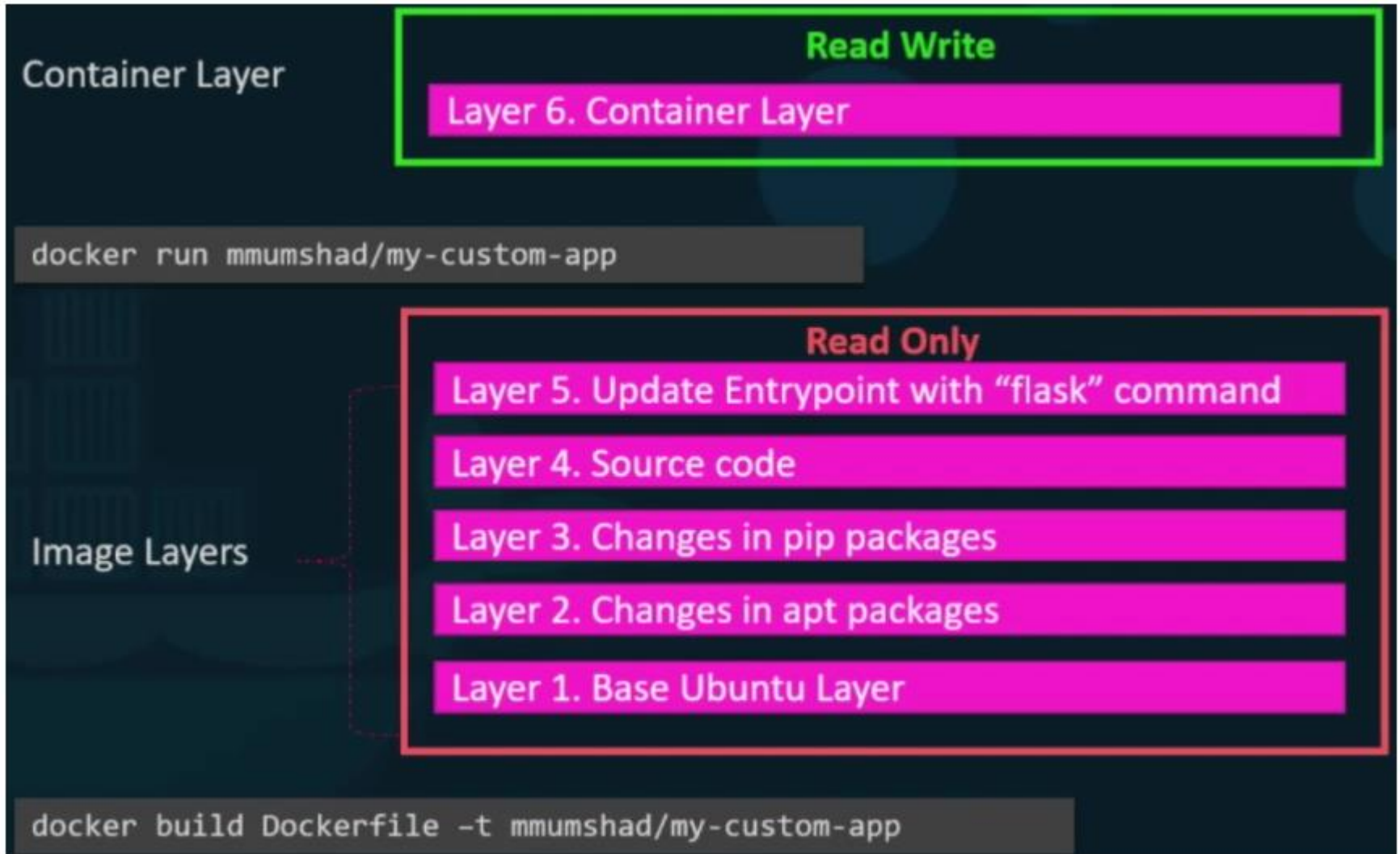
RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

1. OS - Ubuntu
2. Update apt repo
3. Install dependencies using apt
4. Install Python dependencies using pip
5. Copy source code to /opt folder
6. Run the web server using "flask" command

# Dockerfile : Architecture en couche



# Création d'une image

---

- Construire **une image** docker depuis un **Dockerfile**. L'image docker contient **tout ce dont l'application a besoin** pour s'exécuter correctement.

- **\$ docker build**

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

- Créer une image nommée « image-test » à partir du fichier Dockerfile (**y compris le point à la fin**):

**\$ docker build -t imagetest .**

# Création d'une image

---

- **Exemple** : un Dockerfile pour un serveur Apache avec une page HTML de bienvenue.

```
Site PFE/  
|-- Dockerfile  
|-- index.php  
|-- styles.css
```

# Création d'une image

---

## Index.php

```
<!DOCTYPE html>
<html>
<head>
  <title>Bienvenue sur mon site</title>
</head>
<body>
  <h1>Hello, Docker!</h1>
  <p>Ceci est une page de bienvenue.</p>
</body>
</html>
```

# Création d'une image

---

## **styles.css**

```
body {  
  background-color: #f0f0f0;  
  font-family: Arial, sans-serif;  
}  
  
h1 {  
  color: #333;  
}
```

# Création d'une image

---

## Dockerfile

```
# Utilisation d'une image de base avec Apache préinstallé
FROM php:7.4-apache

# Copie d'une page PHP de bienvenue
COPY . /var/www/html/projetPFE

# Exposition du port 80 (par défaut pour Apache)
EXPOSE 80
```



# Création d'une image

---

- Pour construire l'image, utiliser la commande build (**y compris le point à la fin**):

```
$ docker build -t sitepfe:1.0 .
```

- Exécuter un conteneur basé sur cette image

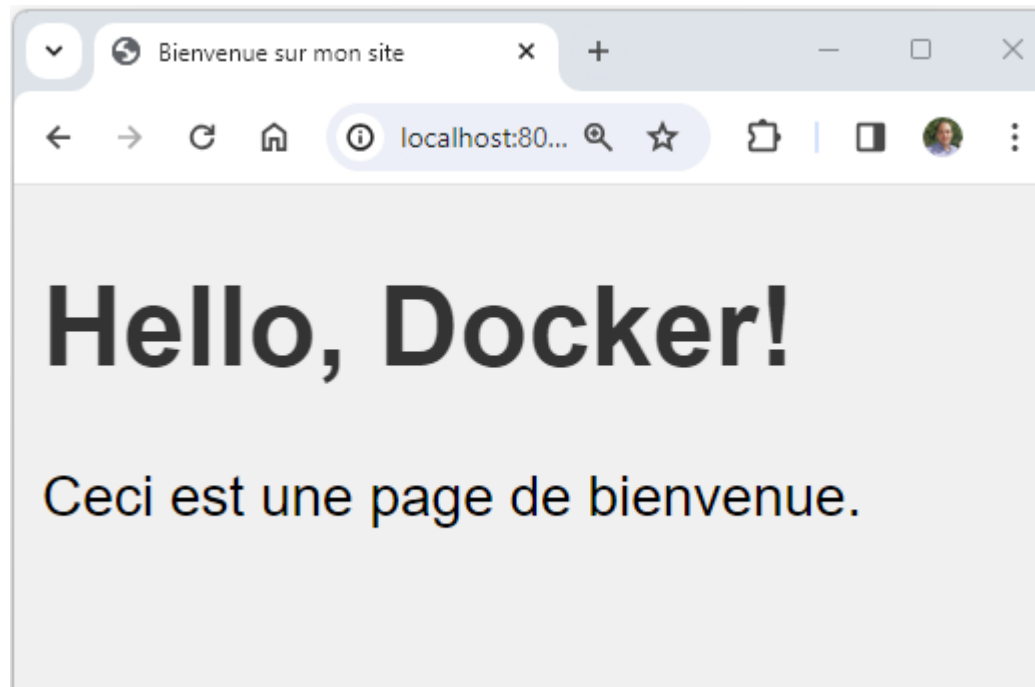
```
$ docker run --name contPFE -p 8081:80 -d sitepfe:1.0
```

# Création d'une image

---

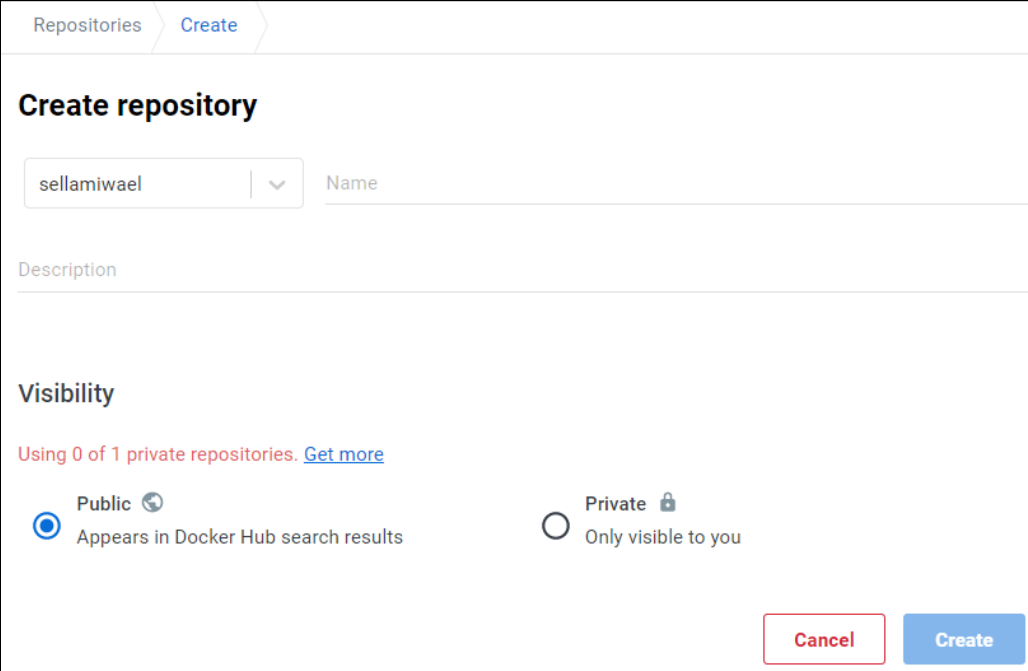
- Accéder à la page d'accueil en ouvrant votre navigateur et en visitant

<http://localhost:8081/projetPFE>



# Publier l'image sur le Docker Hub

1. Afin de rendre l'image docker accessible de façon publique, il faut se rendre sur le site « **<https://hub.docker.com/>** » et se connecter avec les paramètres du compte crée
2. Créer un **repository** « **sitepfe** » en saisissant le nom de l'image, ainsi qu'une description.



The screenshot shows the 'Create repository' form on Docker Hub. At the top, there are tabs for 'Repositories' and 'Create'. The form title is 'Create repository'. Below the title, there is a dropdown menu with 'sellamiwael' selected and a 'Name' label. Below that is a 'Description' label and a text input field. Further down is the 'Visibility' section, which states 'Using 0 of 1 private repositories. [Get more](#)'. There are two radio button options: 'Public' (selected) with a globe icon and the text 'Appears in Docker Hub search results', and 'Private' with a lock icon and the text 'Only visible to you'. At the bottom right, there are two buttons: 'Cancel' and 'Create'.

# Tag et espace de nom des images

---

- Les images peuvent avoir plusieurs versions (tags) :
  - Les tags symbolisent **des différences de version** d'une image
  - C'est le tag:latest qui est utilisé par défaut

## Syntaxe :

**\$ docker tag <local-image>:<tag> <username>/<hub-repo-name>:<tag>**

- Exemple :

**\$ docker tag sitepfe:1.0 sellamiwael/sitepfe:1.0**

# Publication d'un image en « docker hub »

---

- Connectez-vous à Docker Hub en utilisant la commande "docker login".

**\$ docker login**

- Les images docker peuvent être publiées dans un registre publique ou privé à travers la commande « **docker push** ».
- **Syntaxe :**

**\$ docker push <hub-user>/<repo-name>:<tag>**

- **Exemple :**

**\$ docker push sellamiwael/sitepfe:1.0**

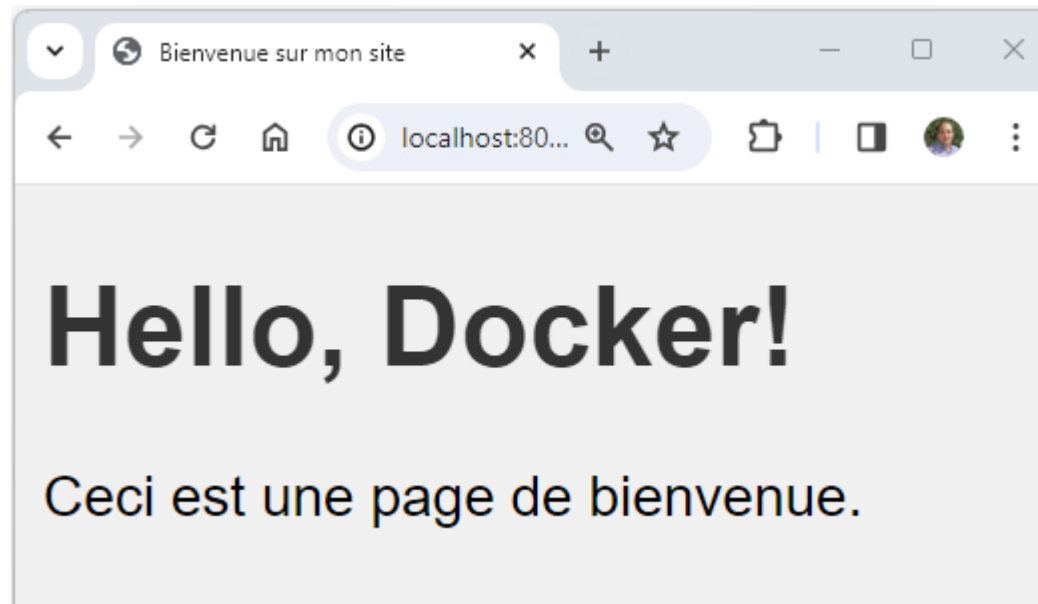
# Publication d'un image en « docker hub »

---

## Exécuter le conteneur

```
$ docker run --name contPFE -p 8081:80 -d sellamiwael/sitepfe:1.0
```

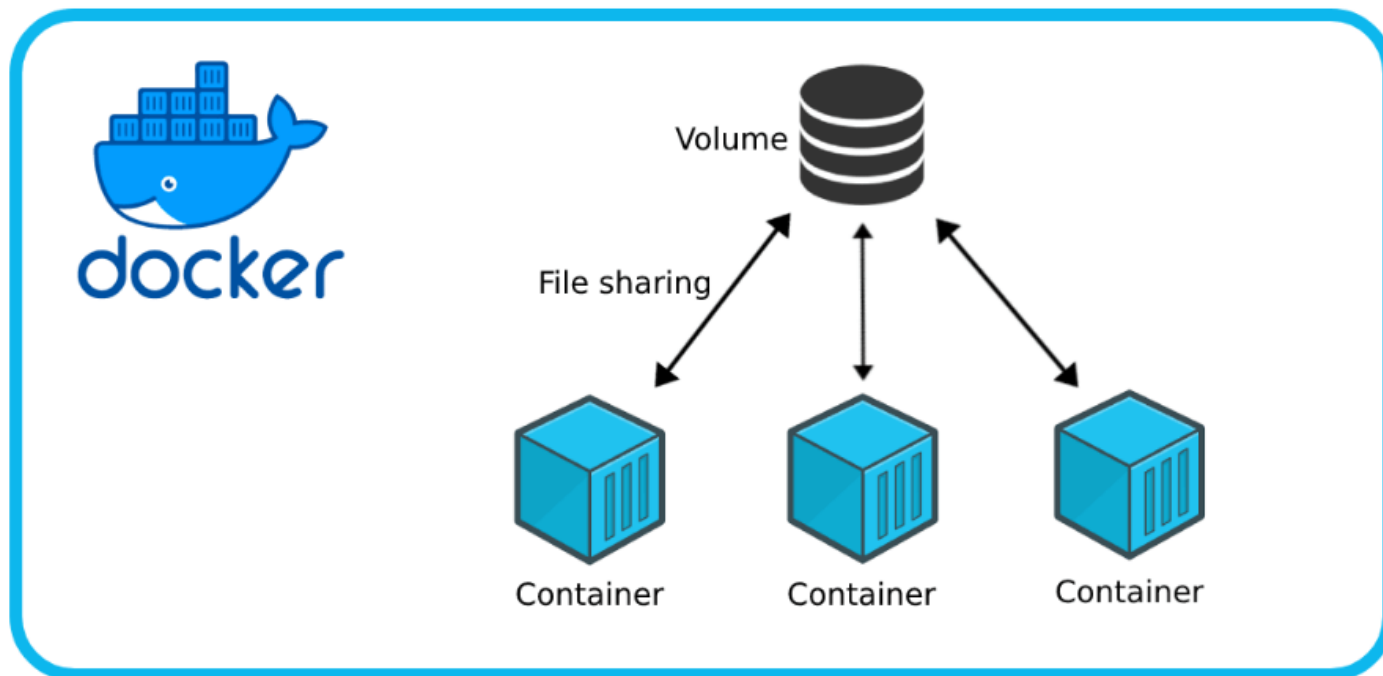
<http://localhost:8081/projetPFE>



# La persistance des données

## Volume

- Un volume est un moyen de **stocker et de partager** des données entre plusieurs conteneurs, ou même entre un conteneur et l'hôte sur lequel le conteneur est en cours d'exécution.



# La persistance des données

## Volume

---

Les volumes offrent plusieurs avantages :

- **Isolation** : les données stockées dans un volume sont isolées des modifications apportées au système de fichiers du conteneur.
- **Partage de données entre conteneurs** : plusieurs conteneurs peuvent partager le même volume, ce qui permet de centraliser les données partagées et de garantir la cohérence entre les applications.
- **Sauvegardes et migrations facilitées** : ils facilitent la sauvegarde et la migration des données, car elles sont stockées en dehors du conteneur lui-même.
- **Performances** : ils offrent de meilleures performances que le stockage de données directement dans le conteneur, car ils utilisent souvent des mécanismes optimisés pour la gestion des données.



# La persistance des données

## Volume

---

- La gestion des volumes dans Docker peut se faire de deux manières :
  - ✓ **localement à l'intérieur de Docker** : les données sont stockées dans des emplacements gérés par Docker au sein de son propre système de fichiers et isolés du système hôte.
  - ✓ **montés depuis la machine hôte** : les données sont partagées entre la machine hôte et le conteneur. Cela permet une plus grande personnalisation des chemins d'accès et le partage de données entre le système hôte et les conteneurs.

# La persistance des données

## Volume

---

- Créer un volume local :

**\$ docker volume create <VOLUME-NAME>**

Exemple :

**\$ docker volume create volume-test**

- Pour récolter des informations sur un volume, il faut utiliser la commande suivante :

**\$ docker volume inspect volume-test**

Résultat sous format JSON:

```
[
  {
    "CreatedAt": "2019-07-03T10:03:20+02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/volume-test/_data",
    "Name": "volume-test",
    "Options": {},
    "Scope": "local"
  }
]
```

# La persistance des données

## Volume

---

- Pour lister les volumes :

**\$ docker volume ls**

- Pour supprimer un volume :

**\$ docker volume rm volume-test**

# La persistance des données

## Attacher un conteneur à un volume

---

- Pour démarrer un conteneur avec un volume, utiliser l'option « **-v** » de la commande « **docker run** ».
- Si le conteneur démarre avec un volume qui n'existe pas encore, alors Docker le créera automatiquement.
- **Cas 1 : volume local** : monter le volume local « **volume-test** » dans le dossier **/chemin/dans/conteneur** :

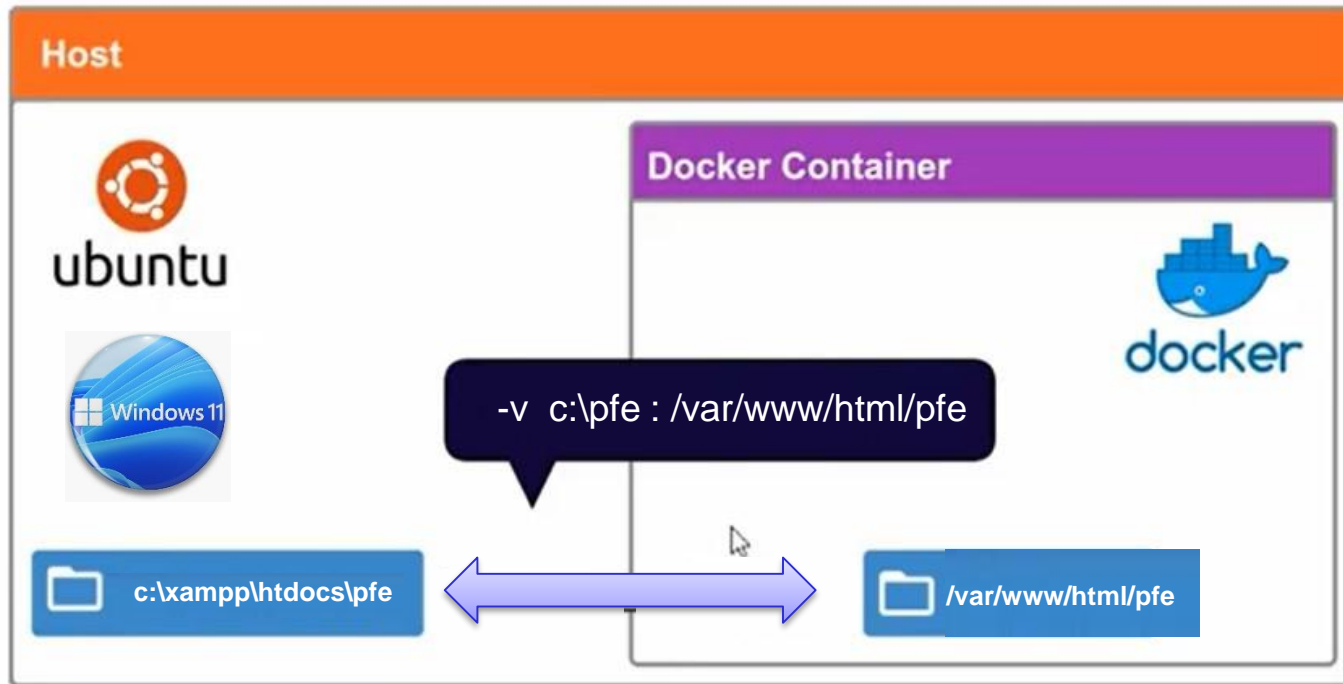
### Syntaxe:

```
$ docker run -v volume-test:/chemin/dans/conteneur image_docker
```

# La persistance des données

## Volume

### Cas 2 : la machine hôte : monter le répertoire



# La persistance des données

## Volume

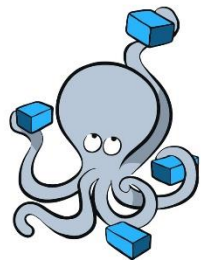
---

- **Cas 2 : la machine hôte : monter le répertoire `c:\xampp\htdocs\pfe` dans le dossier `/var/www/html/pfe` du conteneur.**

```
$ docker run -it --name cont4  
-v c:\xampp\htdocs\pfe:/var/www/html/pfe  
-w /var/www/html/pfe  
-p 8585:80  
ubuntu /bin/bash
```

---

# Orchestration de conteneurs



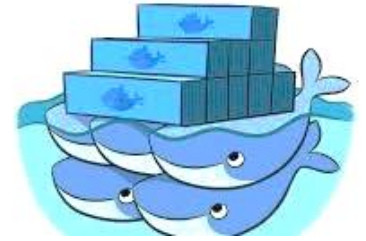
**docker**  
Compose



**kubernetes**

# CaaS (Container-as-a-Service)

---



- Le service de conteneur (CaaS) est un modèle métier dans lequel les fournisseurs de Cloud Computing proposent des services liés à la virtualisation basée sur les conteneurs en tant que services évolutifs en ligne.
- Ceci permet aux utilisateurs finaux d'utiliser des services de conteneurs sans avoir à fournir l'infrastructure dont ils auraient besoin.
- Il s'agit d'un terme marketing qui se réfère à des modèles de services Cloud existants tels que les services IaaS (Infrastructure as a Service), PaaS (Platform as a Service) et SaaS (Software as a Service).



# CaaS (Container-as-a-Service)

---

- Le service de conteneur permet aux utilisateurs de charger, d'organiser, d'exécuter, de faire évoluer, d'administrer et d'arrêter des conteneurs à l'aide des appels API ou de l'interface de portail Web d'un fournisseur.
- Comme pour la plupart des services de cloud, les utilisateurs ne payent que les ressources CaaS qu'ils utilisent (instances de traitement, équilibrage et planification de charge).
- Les trois plateformes CaaS les plus populaires avec Google Container Engine (GKE), Amazon EC2 Container Service (ECS) et Microsoft Azure Container Service (ACS).

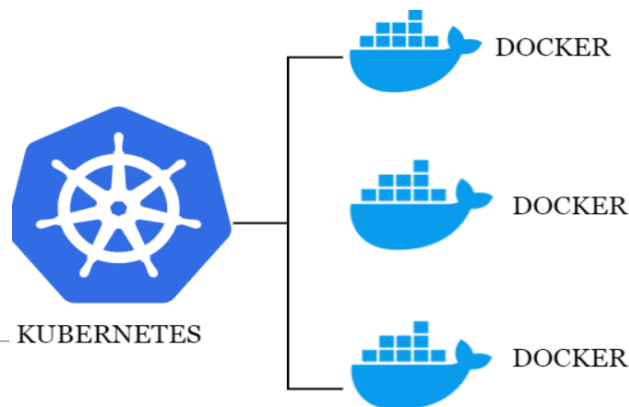
# CaaS (Conteneurs as-a-Service)

---

- Imaginons une application Web de vente en ligne organisée en une architecture de micro-services, où la propriété du domaine d'activité structure le système de services.
- Les domaines des services pourraient être : l'authentification, le panier d'achat, le paiement, etc. Chaque solution a sa propre base de code et demeure conteneurisée.
- Grâce au CaaS, ces services de conteneurs peuvent se déployer instantanément dans un système.

# Comment fonctionne CaaS ?

- L'interaction avec l'environnement de conteneur basé sur le Cloud se fait via une interface utilisateur graphique (GUI) ou sous forme d'appels d'API.
- Les technologies de conteneurs disponibles pour les utilisateurs diffèrent selon le fournisseur. Cependant, le noyau de chaque plateforme CaaS est un outil d'orchestration (aussi appelé orchestrator), qui permet la gestion d'architectures de conteneurs complexes.
- Le marché de la virtualisation en conteneur est dominé par trois outils d'orchestration (Docker Swarm, Kubernetes, etc)



# Orchestration des conteneurs

---

- L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs.
- Applications multi-conteneurs : une application Docker complexe qui inclut plusieurs conteneurs (par exemple, un serveur Web et une base de données s'exécutant dans des conteneurs distincts),
- La construction, l'exécution et la connexion des conteneurs à partir de fichiers Docker distincts sont fastidieuses et chronophages.

# À quoi sert l'orchestration des conteneurs ?

---

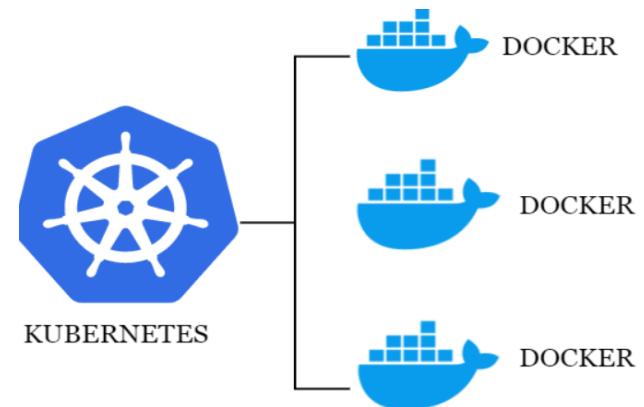
L'orchestration des conteneurs pour automatiser et gérer les tâches suivantes :

- Provisionnement et déploiement
- Configuration et planification
- Disponibilité des conteneurs
- Mise à l'échelle ou suppression de conteneurs en fonction des charges de travail dans l'infrastructure
- Équilibrage de la charge et routage du trafic
- Surveillance de l'intégrité des conteneurs
- Configuration des applications en fonction du conteneur sur lequel elles vont s'exécuter
- Sécurisation des interactions entre les conteneurs

# Outils d'orchestration des conteneurs

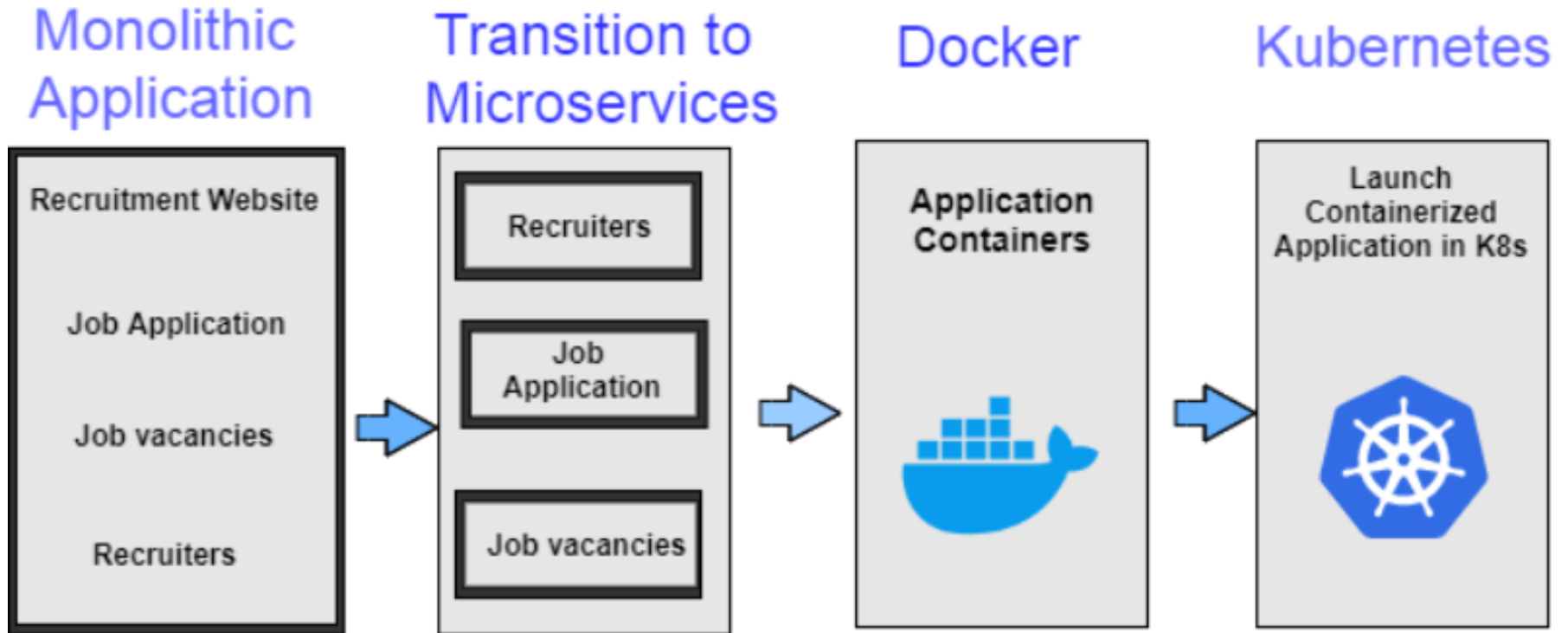
---

- Les outils d'orchestration des conteneurs fournissent un cadre pour la gestion à grande échelle de l'architecture de conteneurs et de micro services.
- Beaucoup de solutions sont disponibles sur le marché pour aider à gérer le cycle de vie des conteneurs. Parmi les plus connues, on peut citer :
  - ✓Kubernetes,
  - ✓Docker Compose
  - ✓Docker Swarm
  - ✓Apache Mesos.



# Outils d'orchestration des conteneurs

---





# Fin de ce chapitre

---

## Avez – vous des questions ?

