

# Docker



Khalid EL BAAMRANI

ENSA de Marrakech

1

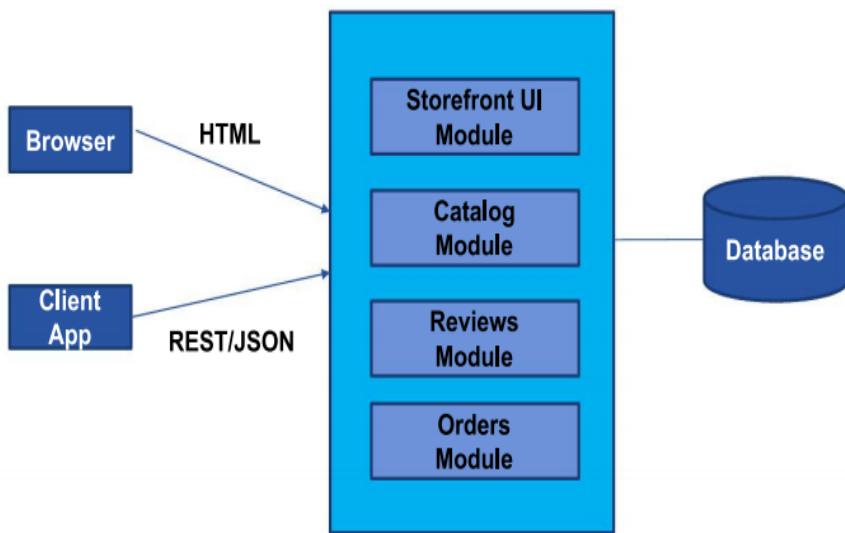
## Références

Les ressources suivantes ont été utilisées pour préparer  
ces slides:

- Slides Docker, Hervé Meftah, Ambient-IT
- Docker, dbskill.com
- ntroduction to Docker & Docker Swarm, Vikram, IoT Application Dev
- DEEP DIVE INTO DOCKER From DevOps4Beginners

# Architecture Monolithique

Dans une architecture monolithique, toute l'application est conçue comme un seul bloc, où toutes les fonctionnalités sont regroupées en une seule unité de déploiement.



3

## Avantages du monolithique

Simple à développer, tester, déployer:

- Simple à développer
- Facile à déployer car tous les composants sont regroupés dans un seul pack
- Mise à l'échelle facile de l'ensemble de l'application

4

# Inconvénients du Monolithique

- Très difficile à maintenir
- La défaillance d'un composant entraînera la défaillance de l'ensemble du système
- Très difficile de créer les correctifs pour une architecture monolithique
- Les modifications apportées à une section du code peuvent avoir un impact inattendu sur le reste du code.
- Difficile d'adopter une technologie nouvelle et avancée : étant donné que les changements de framework ou de langage affecteront l'ensemble d'une application
- Le démarrage prend beaucoup de temps car tous les composants doivent démarrer

5

# Applications ont radicalement changé

## Il y a une décennie

- Les applications étaient monolithiques
- Construit sur une seule pile (par exemple .NET ou Java)
- Longue vie
- Déployé sur un seul serveur

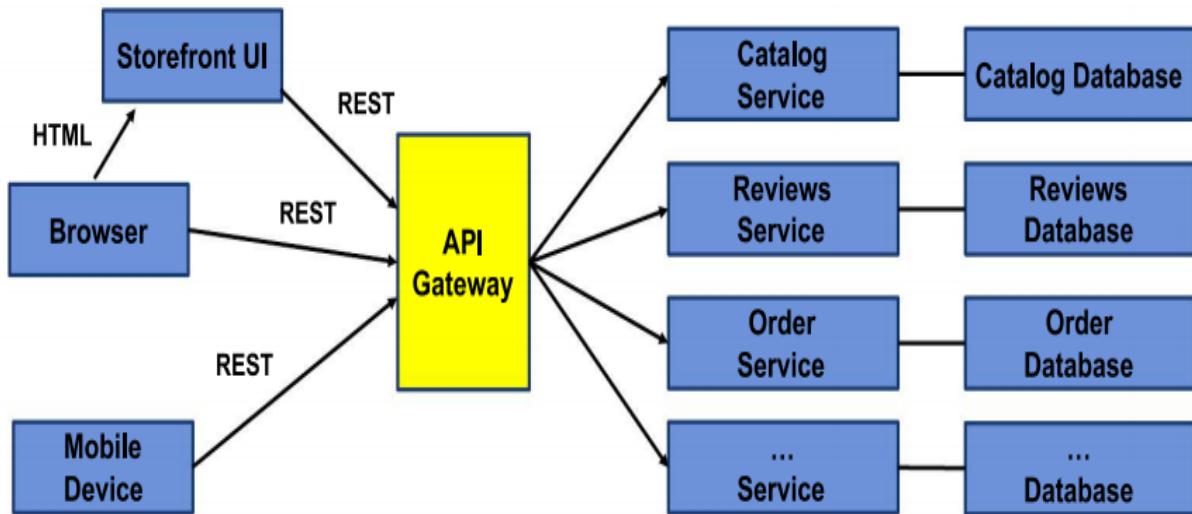
## Aujourd'hui

- Les applications sont constamment développées
- Construire à partir de composants faiblement couplés
- Les versions plus récentes sont souvent déployées
- Déployé sur des **conteneurs** sur un cluster de serveurs

6

# Microservice Architecture

L'architecture microservices consiste à diviser une solution en plusieurs services plus petits, individuels et indépendants les uns des autres et qui communiquent entre eux grâce à des API.



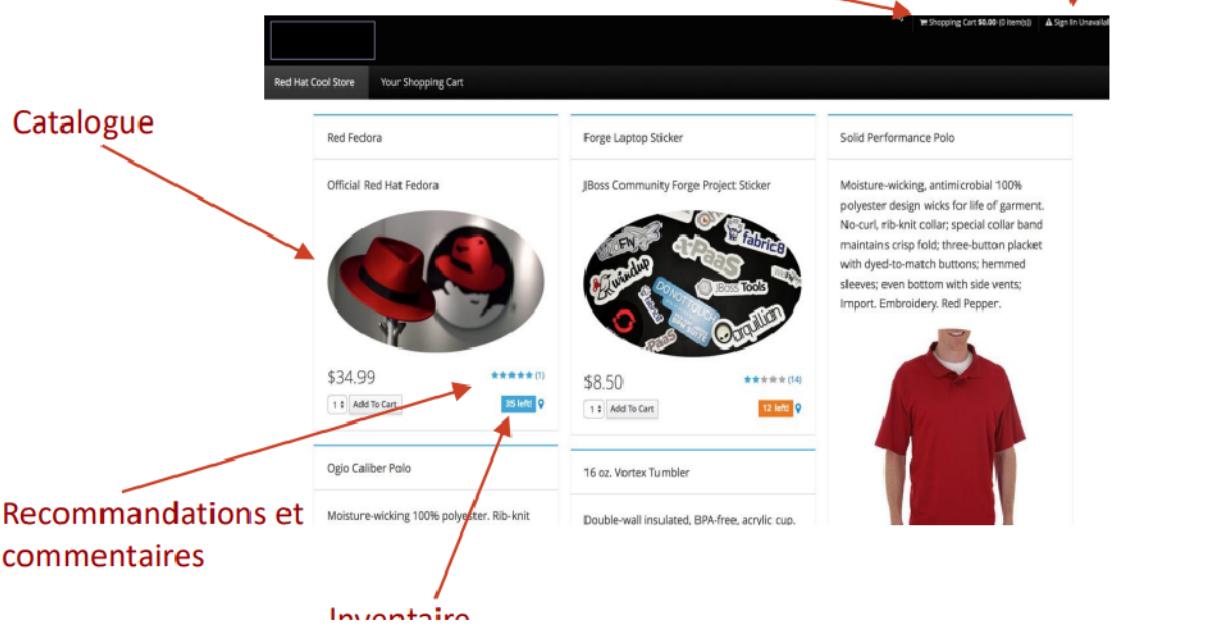
# Microservice Architecture

## Avantages :

- temps de développement plus rapides
- facile à maintenir
- De meilleurs tests : puisque les services sont plus petits et plus rapides à tester.
- De meilleurs déploiements : chaque service peut être déployé indépendamment.
- Haute évolutivité

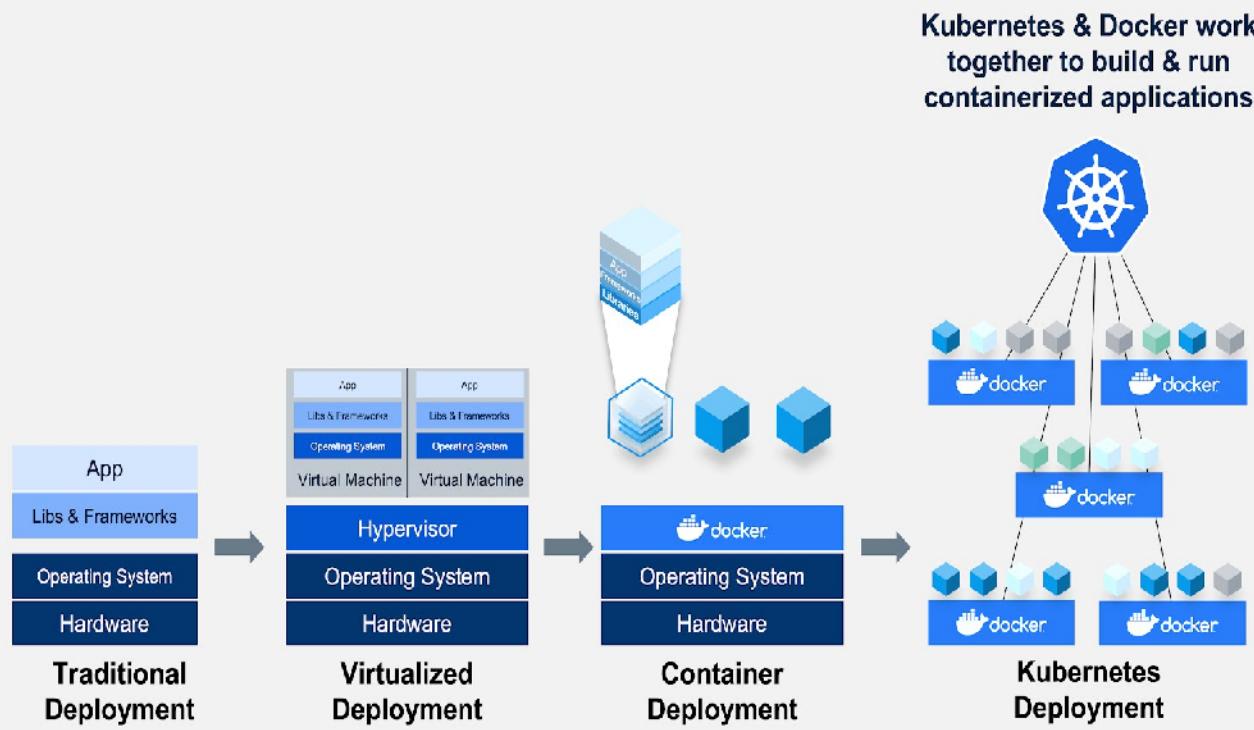
# Microservice Architecture

## Exemple :



9

## The past and the present of Apps Deployment



# Qu'est ce que Docker?

- Docker n'est pas un langage de programmation, c'est un ensemble d'outils pour construire des environnements d'exécution.
- C'est donc un ensemble d'outils pour vous aider à résoudre les problèmes d'installation, de retour-arrière, de distribution et de mise à jour de vos applications.
- Docker permet de créer des **conteneurs** à partir des images de certaines dépendances et de les démarrer instantanément.
- Au lieu d'installer une dépendance en local, on peut donc utiliser un conteneur Docker qui fonctionnera de manière identique sur tous les postes
- Il est open source, c'est à dire que tout le monde peut contribuer à son développement.

Docker est une plate-forme qui permet de "construire, transporter, et exécuter toutes applications, partout"

Il est utilisé pour pallier le problème le plus coûteux en informatique : le déploiement

11

## Conteneur

- Analogie pertinente
  - Contient ce dont on a besoin
    - (Du code, de la conf, des données....)
    - cela ne concerne que le producteur/developpeur et le consommateur/end-user
  - Facile à transporter
    - Le transporteur ne se soucie pas de ce qu'il contient
  - Facile à stocker/héberger



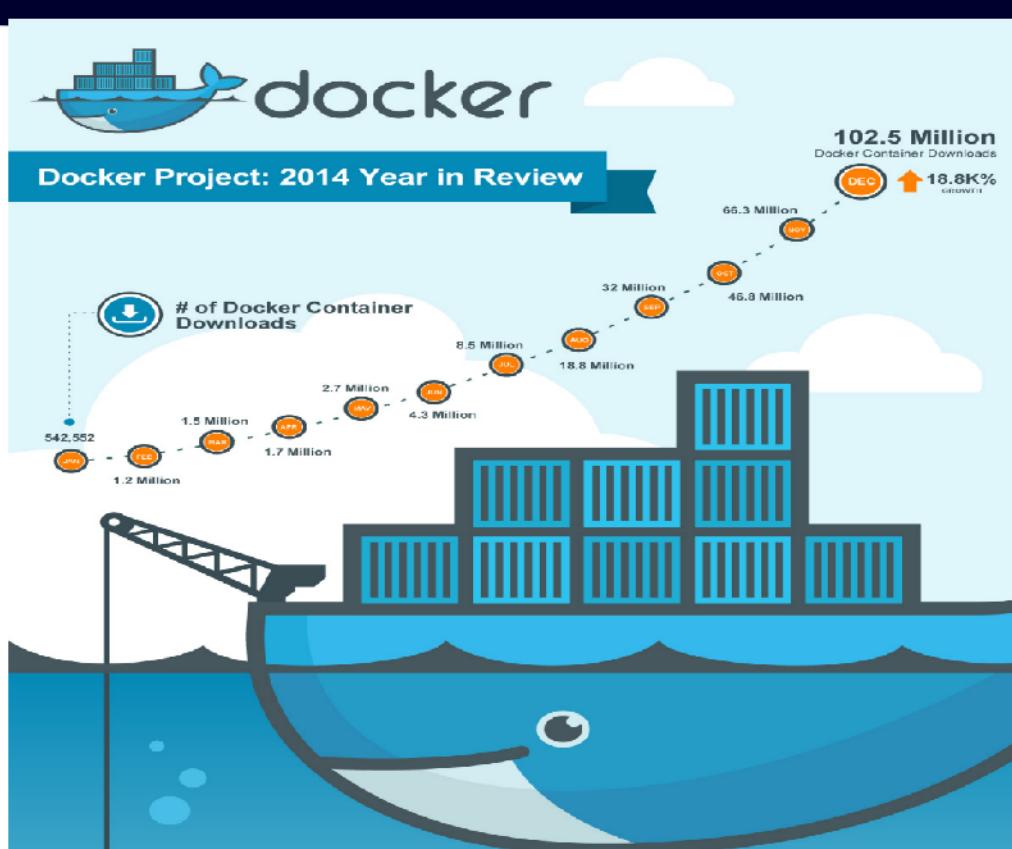
12

# Historique de Docker

- Docker a été développé au début des années 2009 par Solomon Hykes et 2 autres personnes passionnées par Linux.
- Première release en mars 2013.
- Initialement créé avec une base historique de librairies LXC.
- Docker est aujourd'hui développé en langage Go (Goland) de Google.

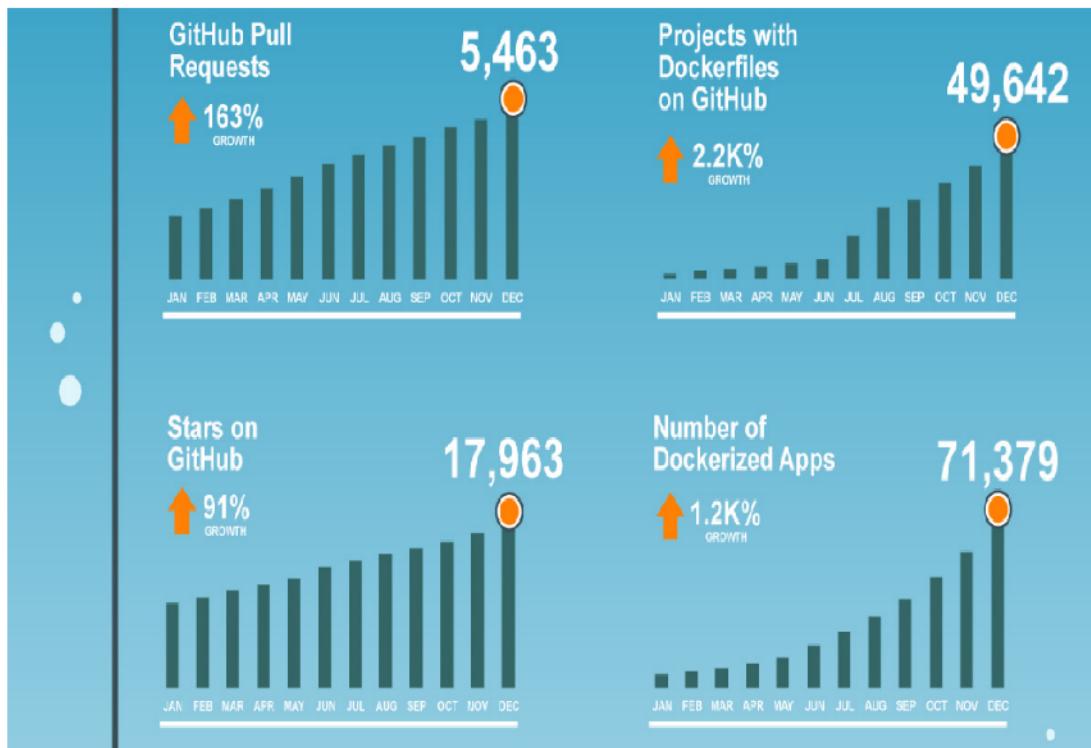
13

## Docker : Le Buzz



14

# Docker : Le Buzz



15

# Docker : Google

- Chez Google, tout fonctionne dans des conteneurs. Selon The Register, **deux milliards de conteneurs sont lancés chaque semaine**. Google utilise des conteneurs depuis une dizaine d'années, quand les technologies de conteneurisation n'étaient pas encore démocratisées;
- C'est l'un des secrets de la performance et la régularité des opérations du moteur de recherche Google, GMail et autres services.

16

# Alternatives à Docker

- LXC  
<https://linuxcontainers.org/fr/>

- OpenVZ  
<https://openvz.org/>

- Les Zones de solaris

<https://docs.oracle.com/cd/E19253-01/820-2318/zones.intro-1/index.html>

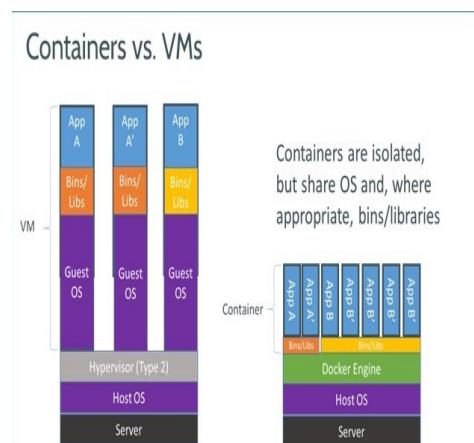
- rkt (rocket) CoreOS  
<https://coreos.com/rkt/>

17

## Conteneurs vs Machines Virtuelles

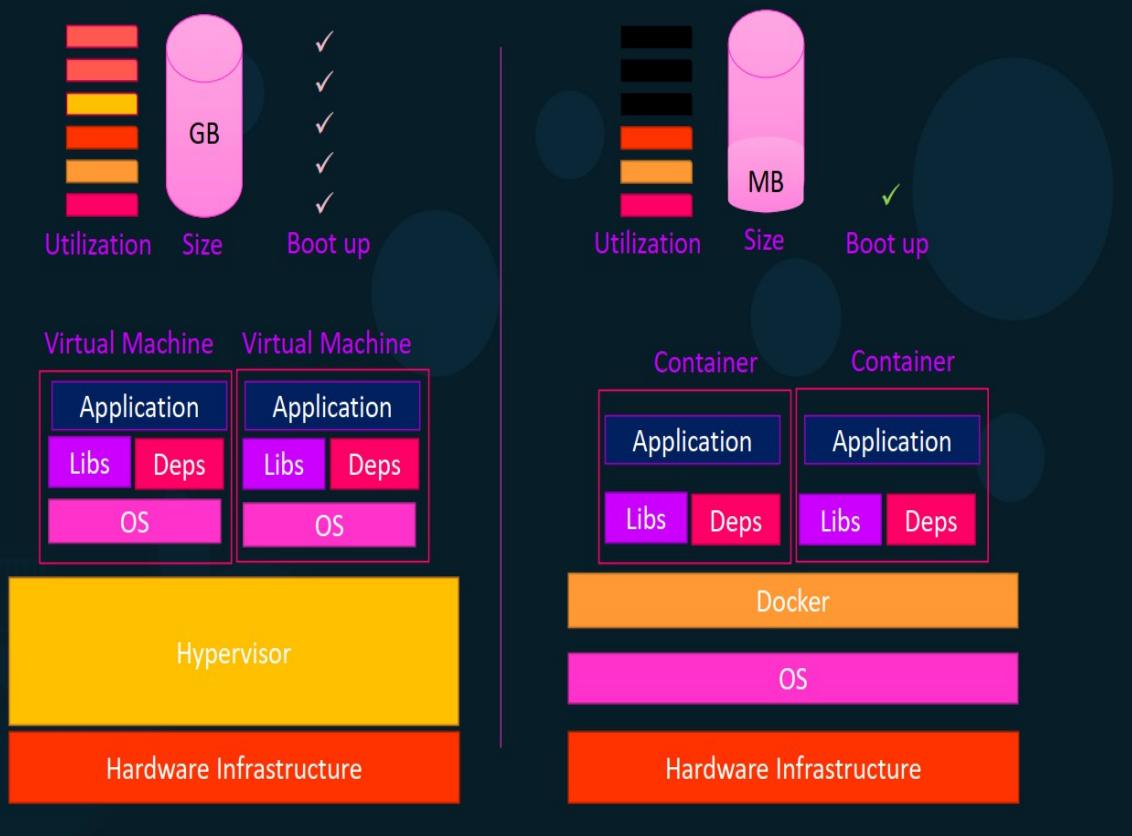
### ○ Un Conteneur

- Ca ressemble à une VM
  - Connexion via shell (ssh ou autre)
  - Ses propres process
  - Sa propre interface réseau
  - Son propre root
  - On peut installer des packages (apt-get install ...)
  - On peut démarrer des services
  - On peut faire du routing réseau (iptables)



18

# Conteneurs vs Machines Virtuelles



## Sans ou Avec Docker

**Dev:** il fonctionne bien dans mon système !

**Testeur:** il ne fonctionne pas dans mon système !

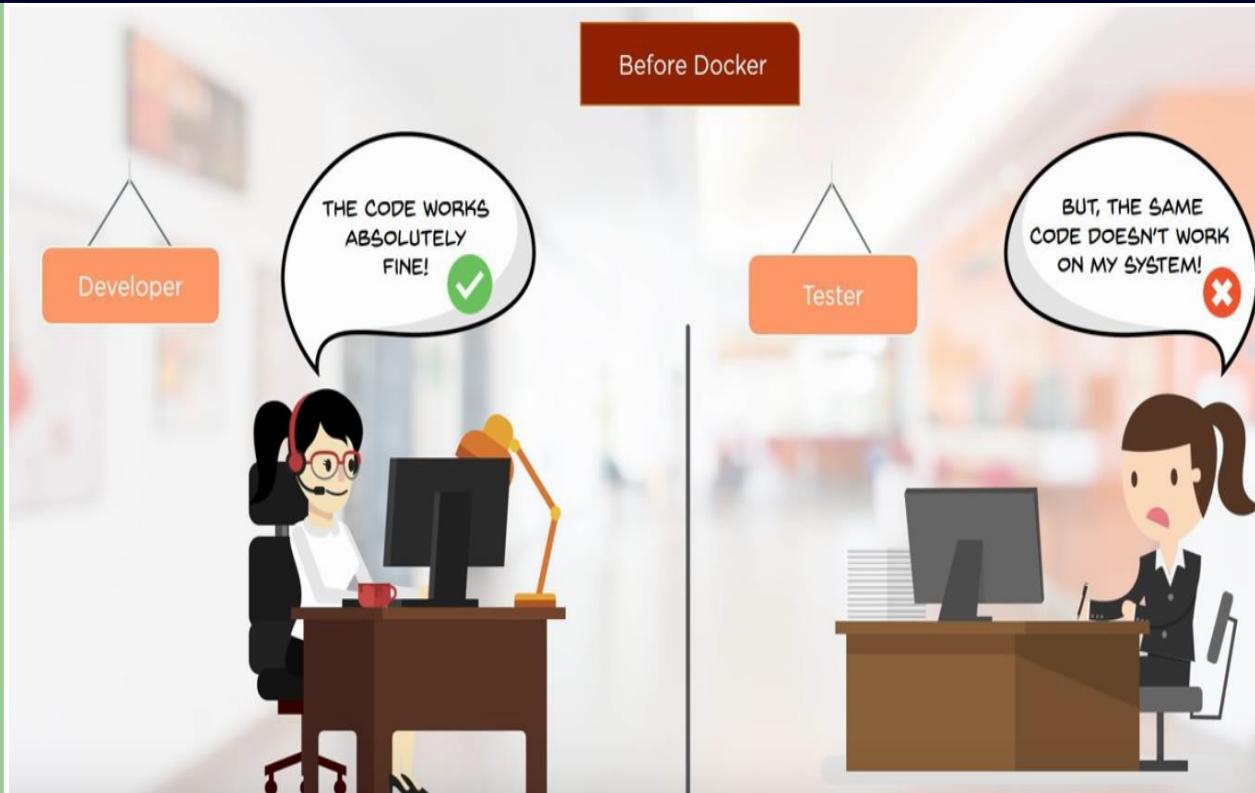
### Sans Docker

Un développeur envoie du code à un testeur mais il ne s'exécute pas sur le système du testeur en raison de divers problèmes de dépendance, mais cela fonctionne bien du côté du développeur.

### Avec Docker

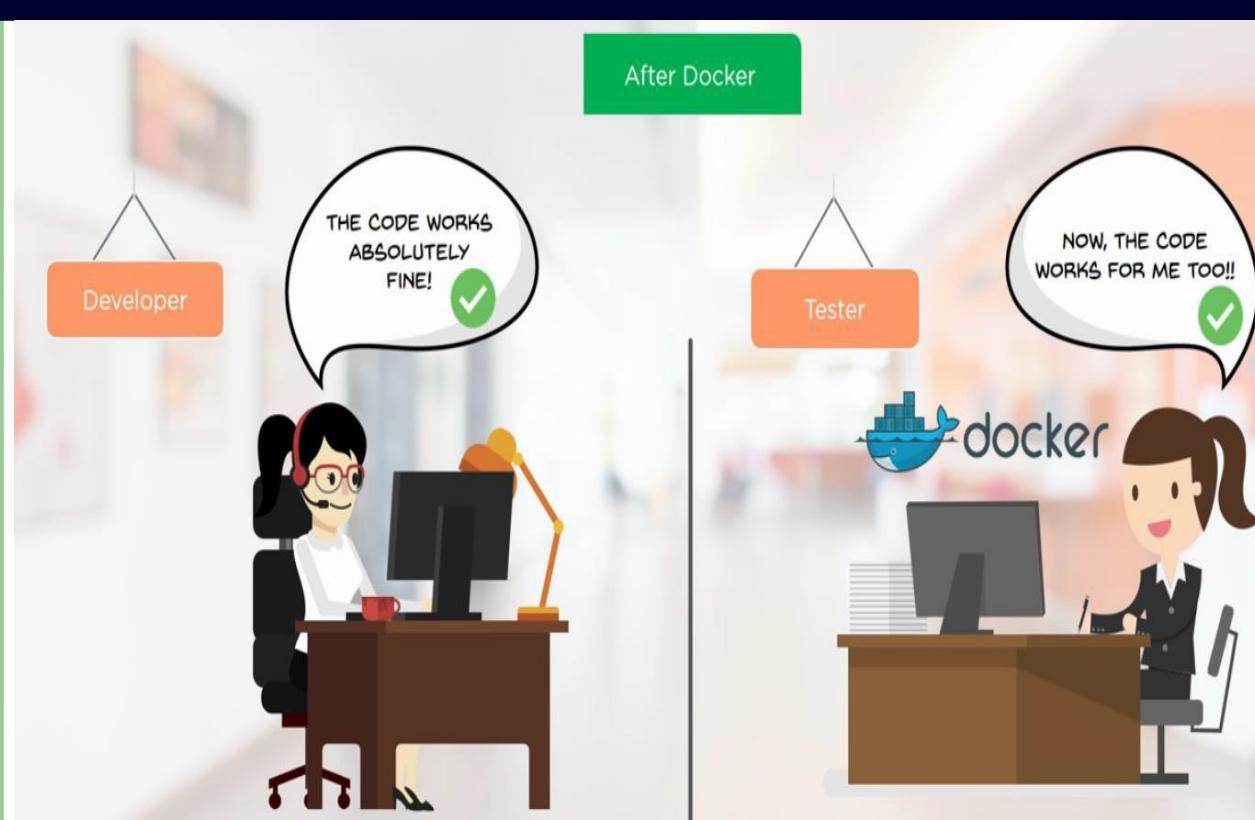
Comme le testeur et le développeur disposent désormais du même système exécuté sur le conteneur Docker, ils sont tous deux capables d'exécuter l'application dans l'environnement Docker sans avoir à faire face à des problèmes de différences de dépendances comme auparavant.

# Sans ou Avec Docker



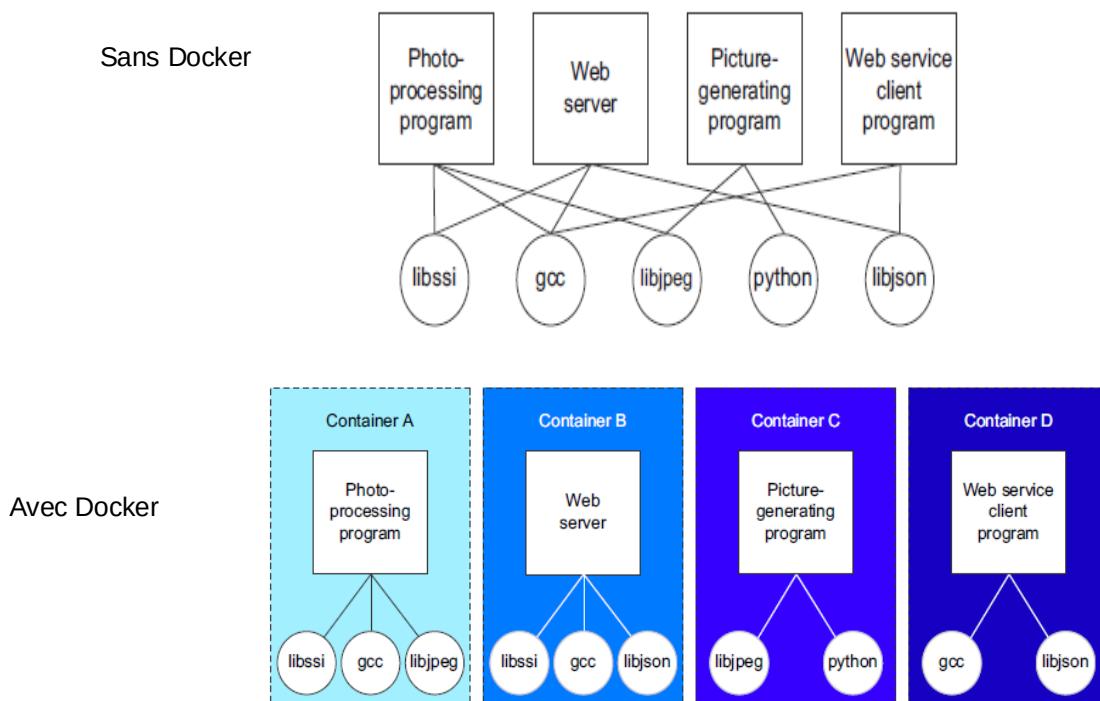
21

# Sans ou Avec Docker



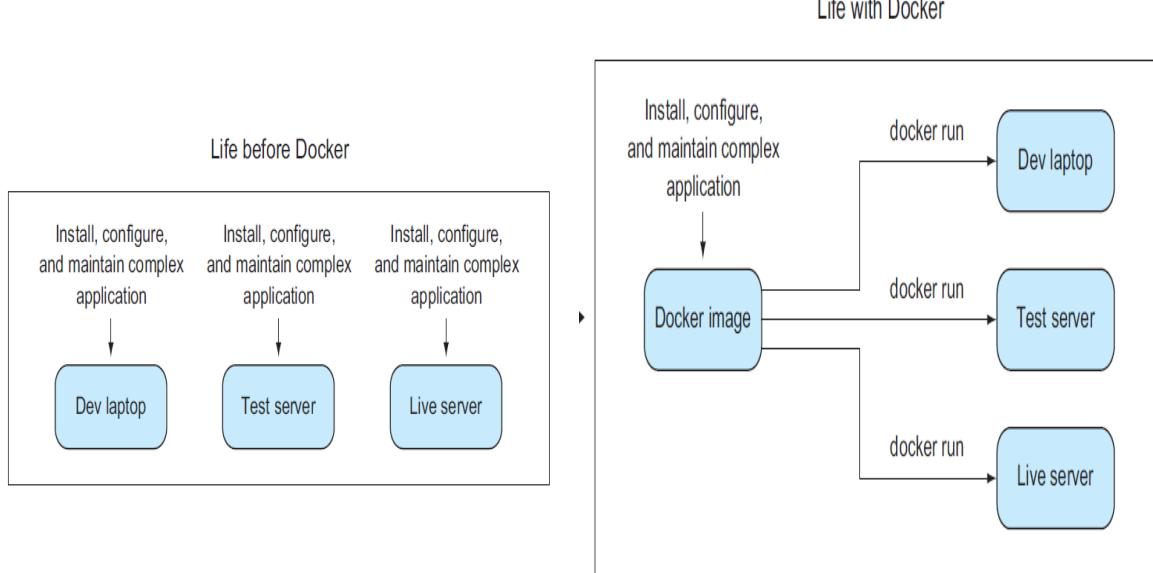
22

# Sans ou Avec Docker



23

# Sans ou Avec Docker



24

# Avantages des conteneurs

- Remplace les machines virtuels (VM).
- Permet le packaging d'applications.
- Ouverture vers les microservices.
- Meilleures performances
  - Accès direct au matériel
- Démarrage beaucoup plus rapide
  - Pas besoin de démarrer un système complet
- Permet la mise en place du Continuous Delivery (CD).

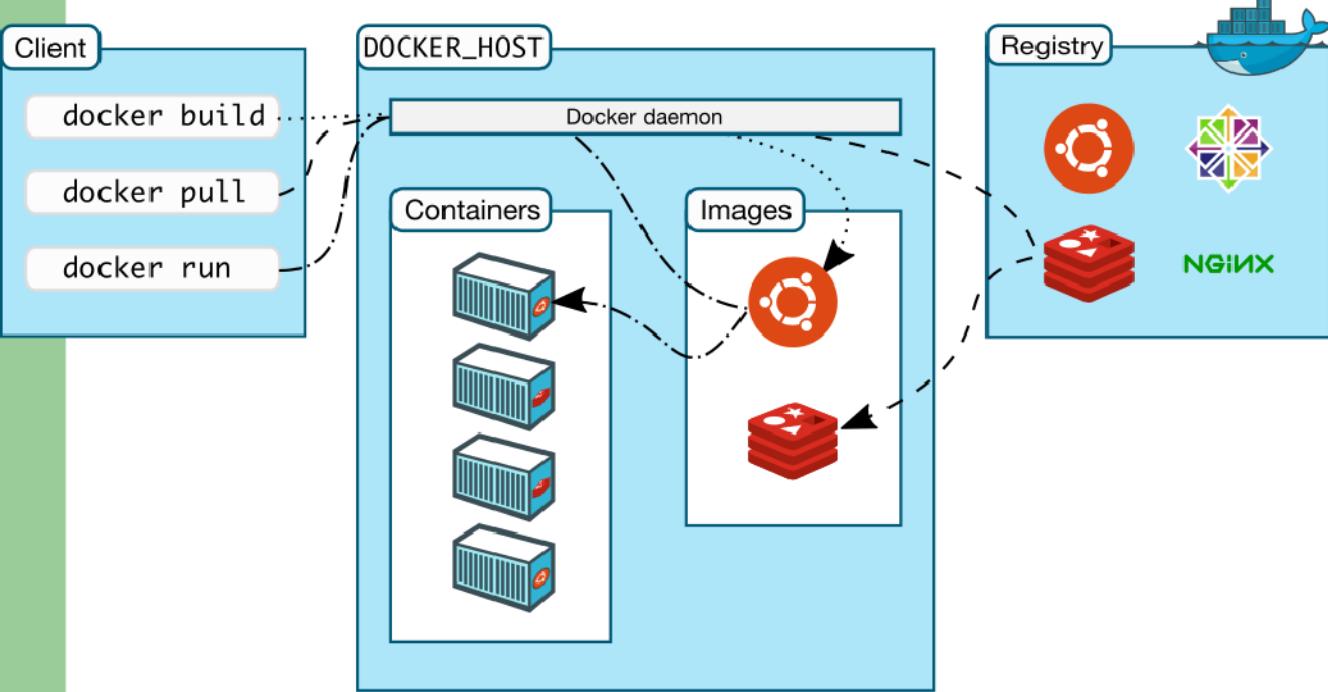
25

# Avantages des conteneurs

- 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
- Passage du dev à la production (DevOps)
  - Créer une image de conteneur
  - L'image peut être directement déployé en production
    - Même environnement pour le dev, le test, et la production
    - Déploiement simplifié
    - Les devs peuvent se charger du déploiement
  - Le passage en production est fluidifié

26

# Docker Architecture



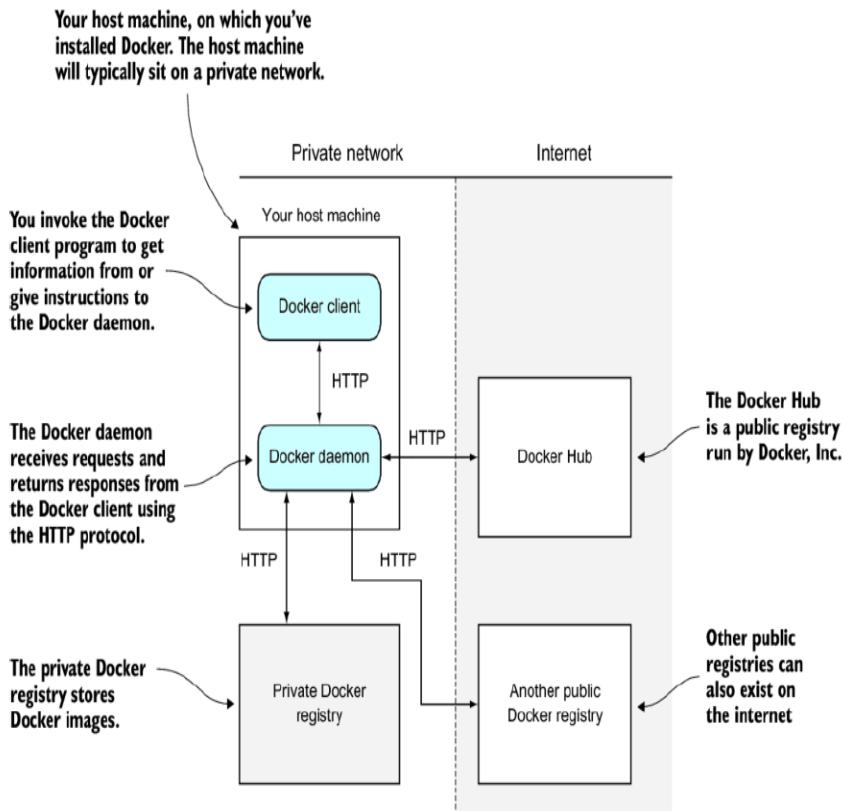
27

# Docker Architecture

- **Docker Engine** : c'est une application qui suit architecture **client-serveur**. Il est installé sur la machine hôte. Il existe trois composants dans le moteur Docker:
  - Server: C'est le démon docker appelé docker. Cela peut créer et gérer les images Docker. Conteneurs, réseaux, etc.
  - API REST: Il est utilisé pour indiquer au démon docker ce qu'il doit faire.
  - CLI: C'est un client qui sert à entrer commandes docker.
- **Client Docker** : Les utilisateurs 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.
- **Registres Docker** : C'est l'emplacement où les images Docker sont stockées. Il peut s'agir d'un registre public ou privé. Docker Hub est l'emplacement par défaut des images Docker

28

# Docker Architecture



30

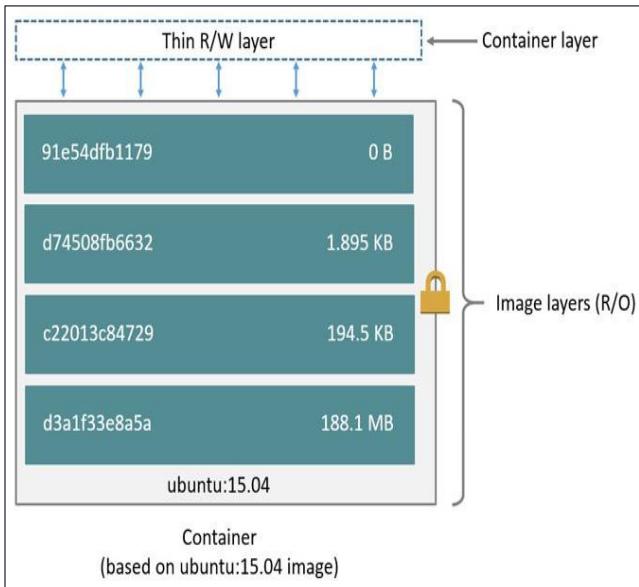
## Images & Containers

- Une **image** est une collection ordonnée de modifications de layer Linux avec les paramètres d'exécution correspondants qui seront utilisés pour lancer un container. Les images sont toujours Read-Only.
- Un **container** est une instantiation dynamique d'une image.
- Un Conteneur
  - Une instance d'une image de conteneur
  - S'exécute dans un environnement isolé
- Analogie POO
  - Une image = une classe
  - Un conteneur = une instance

31

# Images & Containers

## Docker Image:



### Image:

Une image est constituée d'une série de couches et chaque couche représente une instruction dans l'image.

### Container layer:

Lorsqu'un conteneur est créé à partir d'une image, il ajoute une nouvelle couche accessible en écriture au-dessus des couches d'image. Cette couche est appelée « couche conteneur ».

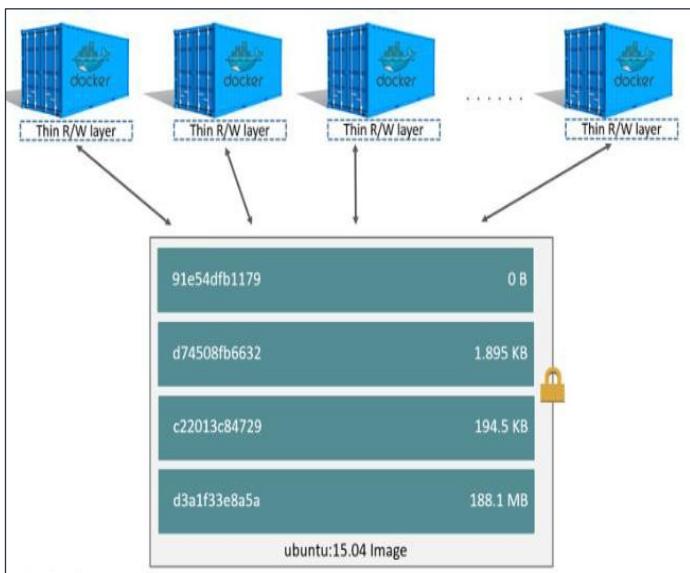
La différence majeure entre un conteneur et une image est la couche supérieure du conteneur.

Image Source: <https://docs.docker.com/storage/storagedriver/#images-and-layers>

32

# Images & Containers

## Plusieurs conteneurs partageant la même image :



Le diagramme montre plusieurs conteneurs partageant la même image Ubuntu.

Lorsque vous créez des conteneurs à partir d'une image, le conteneur et l'image deviennent dépendants l'un de l'autre et vous ne pouvez pas supprimer l'image tant que tous les conteneurs attachés à cette image n'ont pas été supprimés.

Lorsque le conteneur est supprimé, la couche conteneur est également supprimée. Cependant, l'image sous-jacente reste inchangée.

Image Source: <https://docs.docker.com/storage/storagedriver/#images-and-layers>

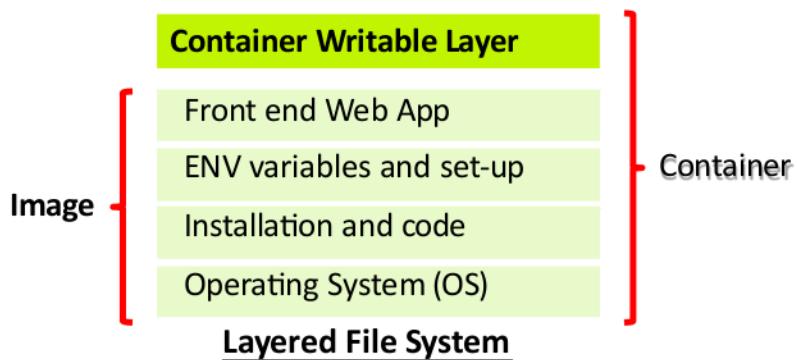
33

# Images & Containers

## IMAGES:

L'image Docker est un fichier qui contient les dépendances, les binaires et les configurations requises pour exécuter un logiciel dans un conteneur.

- docker pull [**Image Name**]:[tag]
- docker image pull [**Image Name**]:[tag] (**Recommended way**)



Reference : <https://docs.docker.com/storage/storagedriver/>

34

## Docker Installation

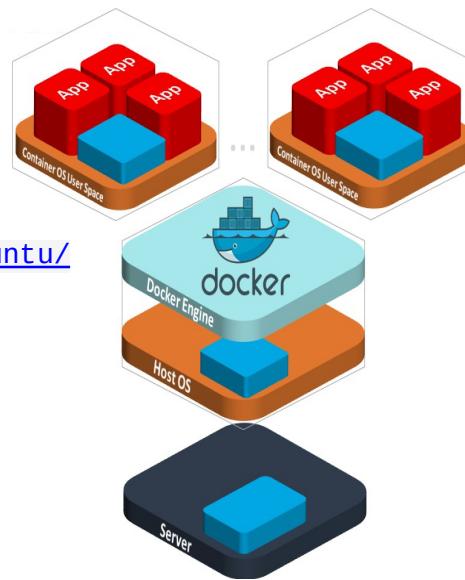
36

# Docker Installation

- Docker for Windows
- Docker for MAC
- Docker for Linux
- Docker for Ubuntu :
  - <https://docs.docker.com/engine/install/ubuntu/>

Online Emulator:

<https://labs.play-with-docker.com/>



<https://docs.docker.com/get-docker/>

37

## Installation de docker sous Ubuntu

### 1. Add Docker's official GPG key:

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

### 2. Add the repository to Apt sources:

```
echo \
  "deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

### 2. install the latest version, run:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

38

# Vérification de l'installation

```
khalid@PC:~$ docker --version  
Docker version 20.10.23, build 7155243
```

Pour vérifier l'installation tapez:  
khalid@PC:~\$ docker run hello-world

```
Hello from Docker!  
This message shows that your installation appears to be working  
correctly.
```

39

## Docker : Commandes de Base

40

# Commandes de Base

## Docker run:

Instanciez un conteneur à l'aide de la commande

- **docker container run [OPTION1 OPTION2 ... OPTIONn] [Image]:[TAG] [COMMAND] [ARGUMENT]**

- **IMAGE**: Docker Image.
- **TAG**: Run specific version of an image.
- **COMMAND**: Command to run inside the container.
- **ARGUMENT**: Arguments for the COMMAND.

## Run Container:

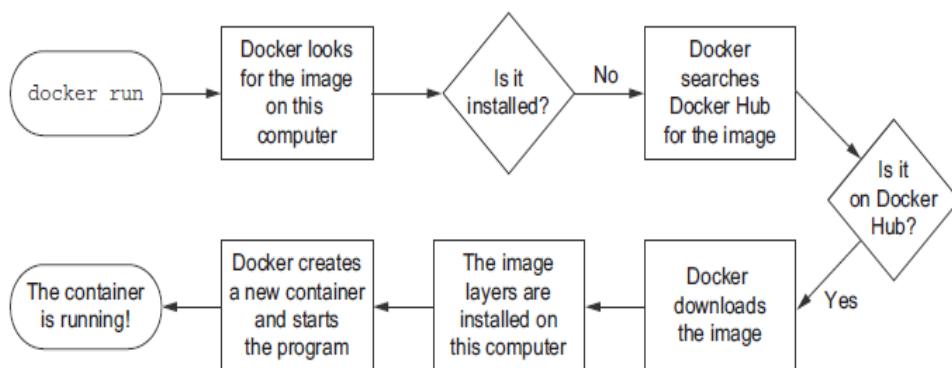
- docker run hello-world
- docker **container** run hello-world (**Recommended way**)
- docker run nginx
- docker **container** run nginx (**Recommended way**)

Reference : <https://docs.docker.com/engine/reference/run/>

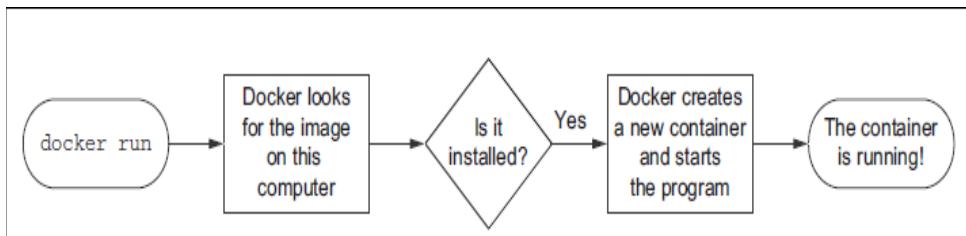
41

# Docker run

La commande: **docker run hello-world**



Lors d'un second Run



42

# Commandes de Base

## Run a container with **COMMAND** and **ARGUMENT**:

- docker run ubuntu **echo** Hello Students!
  - echo: Command run inside the ubuntu container.
  - Hello Students!: Argument for the Command.

## List all containers (**Running and stopped**):

- docker ps -a
  - -a: All

43

## Commande docker ps

La commande **docker ps** et **docker ps -a** listera les détails suivants:

- ID CONTAINER: Indique l'ID du conteneur associé au conteneur. L'ID du conteneur est un nombre aléatoire à 64 chiffres hexadécimaux. Par défaut, la commande docker ps affichera seulement 12 chiffres hexadécimaux. Vous pouvez afficher tous les 64 chiffres en utilisant le drapeau **--no-trunc** (par exemple: sudo docker ps --no-trunc).
- IMAGE: affiche l'image à partir de laquelle le conteneur Docker a été fabriqué
- COMMAND: Ceci vous montre la commande exécutée pendant le lancement du container
- CREATED: cela vous indique quand le conteneur a été créé.
- STATUS: Ceci vous indique l'état actuel du conteneur.
- PORTS: cela vous indique si un port a été affecté au conteneur.
- NAMES: le moteur Docker génère automatiquement un nom de conteneur aléatoire. Le nom du conteneur peut être configuré manuellement en utilisant l'option **--name** dans l'exécution du docker sous-commande

44

# Commandes de Base

## Donner un nom au conteneur:

- docker run --name webserver nginx
- docker container run --name mynginx
  - --name : fournissez le nom souhaité.

## Lister des conteneurs en cours d'exécution :

- docker container ls

## Lister tous les conteneurs (en cours d'exécution et arrêtés) :

- docker container ls -a

## Supprimer un conteneur :

- docker container rm [ID du conteneur]

46

# Commandes de Base

pour arrêter tous les containers qui tournent sur votre machine

`docker stop $(docker ps -aq)`

pour détruire tous vos containers

`docker rm $(docker ps -aq)`

pour détruire toutes les images

`docker rmi $(docker images -q)`

47

# Commandes de Base

**Publier des ports:** Il en existe 2 types :

- **--publish (or) -p:**

Publiez le(s) port(s) d'un conteneur sur l'hôte.

- docker container run -d --name [container name] **-p** [Host port] [Container port] [Image]

- Exemple:

- ✓ docker container run -d --name mynginx -p 8080:80 nginx

- **--publish-all (or) -P:**

Publiez tous les ports exposés sur des ports aléatoires.

- docker container run -d --name [container name] **-P** [Image]

- Exemple:

- ✓ docker container run -d --name mynginx2 -P nginx

48

# Commandes de Base

**Afficher les informations détaillées d'un conteneur :**

- docker container **inspect** [Container ID/Container name]
  - Exemple:
    - ✓ docker container inspect mynginx

**Lister le mappage des ports :**

- docker container **port** [Container ID/Container name]
  - Exemple
    - ✓ Docker container port mynginx

Reference : <https://docs.docker.com/engine/reference/run/>

<https://docs.docker.com/engine/reference/commandline/container/>

49

# Commandes de Base

## --interactive (or) -i and --tty (or) -t:

Lorsque vous vous détachez du conteneur, le conteneur **s'arrête**.

- --interactive (ou) -i : garder STDIN ouvert même s'il n'est pas attaché

- --tty (ou) -t : allouer un pseudo-TTY

```
docker container run --name [container name] -it [Image]
```

- Exemple

- ✓ docker container run --name myubuntu -it ubuntu

## attach:

Attachez les flux d'entrée, de sortie et d'erreur standard locaux à un conteneur en **cours d'exécution**. Lorsque vous vous détachez du conteneur, cela va **arrêter** le conteneur.

- docker container **attach** [Container name/Container ID]

- Exemple

- ✓ Docker container attach myubuntu

50

# Commandes de Base

Pour lancer un container en mode interactif :

```
docker run -it --name mycontainer centos /bin/bash
```

```
[root@386ec090afdc /]# hostname
```

```
386ec090afdc
```

```
[root@386ec090afdc /]# exit
```

# stop le container

```
docker start mycontainer
```

```
docker attach mycontainer
```

```
[root@386ec090afdc /]#
```

ensuite pour le mettre en background

```
faire Ctrl-p Ctrl-q
```

```
et
```

```
docker ps
```

Pour lancer un container en tâche de fond en ligne de commande faire

```
docker run -d --name mycontainer centos /bin/bash
```

51

# Commandes de Base

## exec:

- Exécutez une commande dans un conteneur en **cours d'exécution**.
  - exec n'arrêtera pas le conteneur lorsque vous vous détacherez du conteneur en cours d'exécution.
- docker container **exec** [Options] [Container ID/Container name] [Command] [Arguments]
    - Exemple:
      - ✓ docker container exec -it myubuntu /bin/bash

52

# Commandes de Base

## ● Lister les conteneurs en cours d'exécution :

- docker container ls
- docker ps

## ● Lister tous les conteneurs (en cours d'exécution et arrêtés) :

- docker container ls -a
- docker ps -a

## ● Arrêter un conteneur :

- docker container **stop** [container ID/Container name]

## ● Demarrer un conteneur :

- docker container **start** [container ID/Container name]

## ● Suspendre un conteneur :

- docker container **pause** [container ID/Container name]

## ● Relancer un conteneur ::

- docker container **unpause** [container ID/Container name]

53

# Commandes de Base

- **Récupérer les logs d'un conteneur :**
  - docker container **logs** [Container name/Container ID]
- **Pour afficher les statistiques d'utilisation des ressources du conteneur**
  - docker container **stats** [Container name/Container ID]
- **Pour voir les processus en cours d'exécution d'un conteneur :**
  - docker container **top** [container ID/Container name]

## Commandes de base des images :

- **Importer une image:**
  - docker image **pull** [Image]
- **Lister les images:**
  - docker image **ls**
- **Pour voir les informations détaillées d'une image :**
  - docker image **inspect** [Image]

54

# Commandes de Base

## Nettoyer : supprimez les images et les conteneurs.

- **Supprimer un conteneur arrêté :**
  - docker container **rm** [Container Name/Container ID]
- **Supprimez tous les conteneurs arrêtés :**
  - docker container **prune**
- **Supprimer un conteneur en cours d'exécution :**
  - docker container **rm -f** [Container Name/Container ID]
- **Supprimez tous les conteneurs arrêtés et en cours d'exécution :**
  - docker container rm -f `**docker ps -a -q**`
  - docker container rm -f `**docker container ls -a -q**`
- **Supprimer une image :**
  - docker image **rm** [Image]
- **Supprimer automatiquement un conteneur lorsqu'il s'arrete:**
  - docker container run **--rm** [Image]

55

# Docker images : Commandes

- Importer une image:

- docker image pull nginx
- docker image ls

- Exporter une image:

- docker image pull -a nom\_image

- Chercher une Image:

- docker search nginx

- Limiter le nombre de résultat :

- docker search --limit 10 nginx

- Filtrer le résultat de la recherche :

- docker search --filter stars=200 nginx
- docker search -f stars=100 -f is-official=true nginx

Reference Doc: <https://docs.docker.com/engine/reference/commandline/image/>

56

# Docker images : Commandes

- Lister les images:

- docker images
- Docker image ls
- Docker image ls -a

- Taguer une image:

- docker image tag [Source Image]:[tag] [Reference to source image]:[tag]
- ✓ **docker tag ubuntu myubuntu:v1**

- Supprimer une image:

- docker image rm nginx
- docker rmi nginx

Reference Doc: <https://docs.docker.com/engine/reference/commandline/image/>

57

# Docker images : Commandes

- **Supprimer l'image suspendue :**

- docker image prune

- **Supprimez toutes les images inutilisées et suspendues:**

- docker image prune -a

- **Inspecter une image :**

- docker image inspect nginx

- docker image inspect nginx --format"{{.ContainerConfig.Hostname}}"

Reference Doc: <https://docs.docker.com/engine/reference/commandline/image/>

58

## Commandes de base : Recap

docker ps	liste les containers actifs
docker ps -a	liste tous les containers qui sont created, restarting, running, removing, paused, exited ou dead
docker ps -aq	liste tous les UUID des containers
docker stop <nom ou ID>	arrête un container
docker rm <nom ou ID>	détruit un container
docker pause <nom ou ID>	suspend un container
docker start <nom ou ID>	démarrer un container
docker images	liste toutes les images présentes sur le disque local
docker rmi <nom ou ID>	détruit une image

59

# Commandes de base : Recap

docker create	créer un container
docker build	créer une image à partir d'un fichier Dockerfile
docker import	importer un fichier tar dans un image
docker detach	detach le process du container
docker exec	Fait exécuter une commande par un container
docker save	Crée une image à partir d'un container
docker export	Crée un fichier tar d'une image
docker commit	Crée une image de l'état courant d'un container

60

# Commandes de base : Recap

docker inspect	Retourne des information de bas niveau sur les objets docker; images, containers, network
docker load	charge dans une image un fichier tar préalable exporté avec docker export.
docker kill	kill le container
docker info	affiche l'état de Docker
docker search <nom>	rechercher dans le localhost et docker hub les images disponibles
docker logs <nom du container>	Affiche message log du container
docker pull <image_name>	importe une image d'un repository local ou de docker hub
docker pull -a <image_name>	exporte une image vers un repository

61