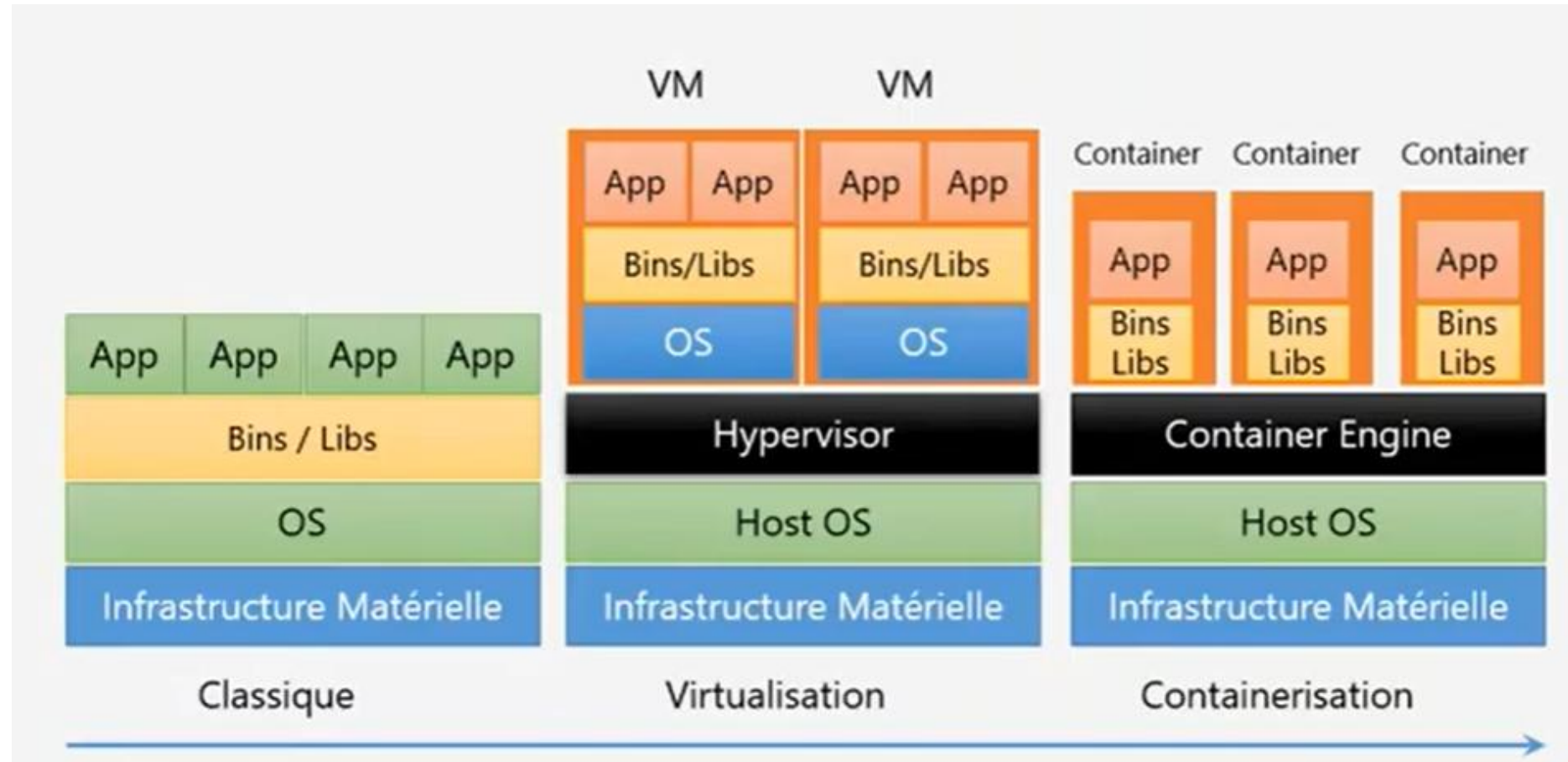


Docker

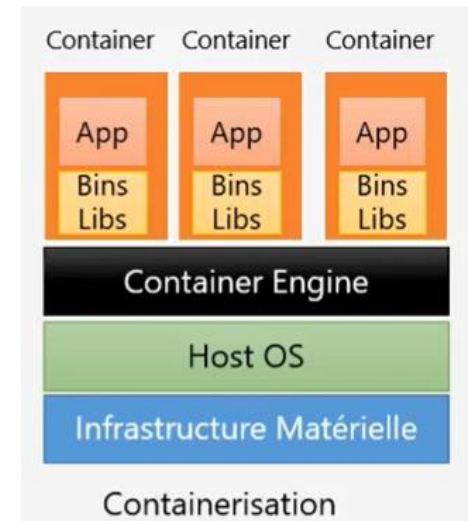
Présenté par : Bouzaien Malek

Evolution de l'infrastructure



Container

- Enveloppe permettant de packager une application avec juste ce dont elle a besoin pour fonctionner.
- Peut être déployé tel quel dans n'importe quelle machine disposant d'un Container Engine avec différents environnements (Dev, Test, Prod)
- Utilise le Kernel de l'OS Hôte.
- A son propre espace de processus et sa propre interface réseau.
- Isolé de l'hôte, mais exécutée directement dessus.
- Permet de décomposer l'infrastructure applicative en petits éléments légers facile à déployer et à réutiliser.
- Créer un conteneur pour chaque application.

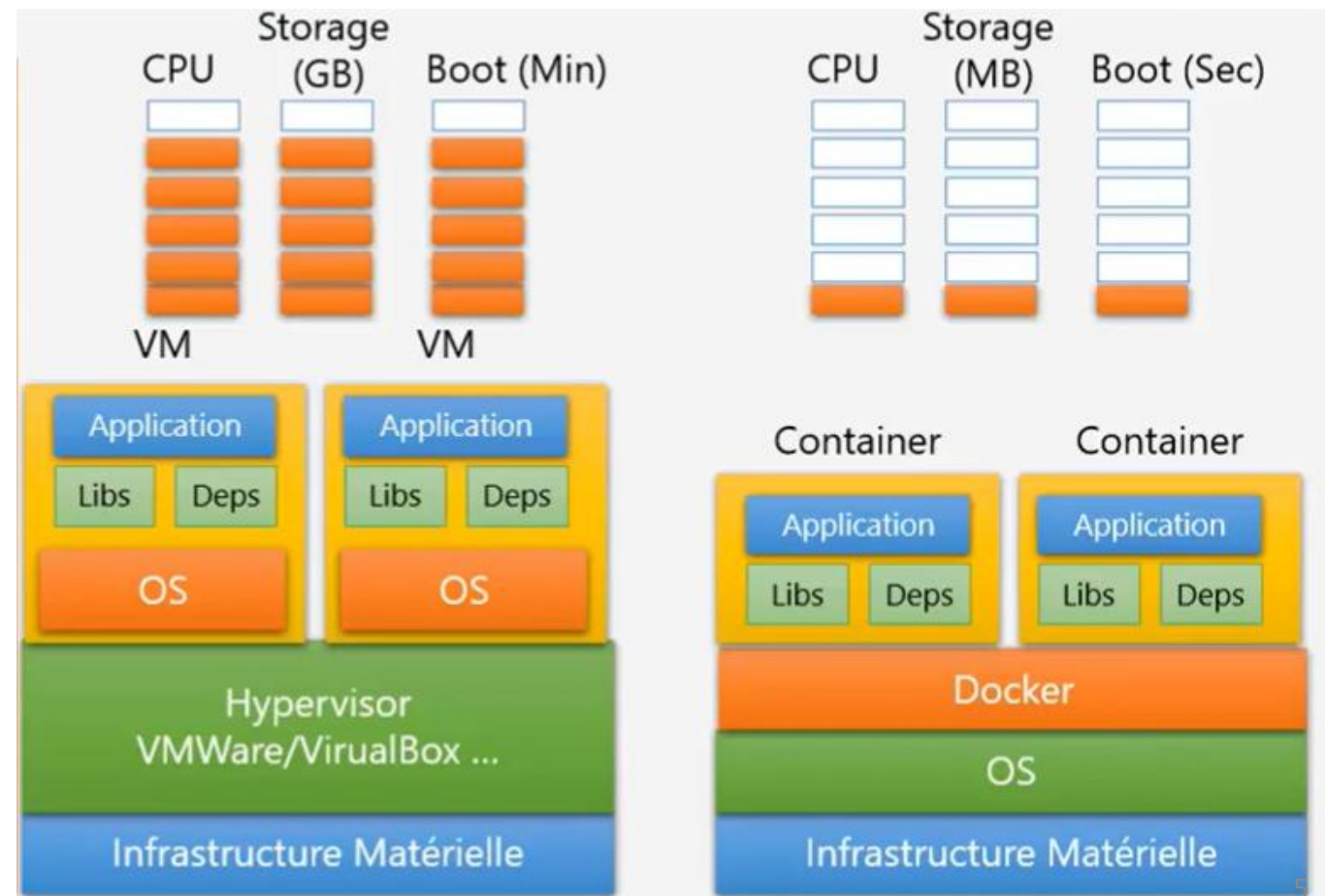


Pourquoi utiliser les Containers

- Meilleures performances que les VM (Démarrage instantané).
- Portabilité d'un environnement à l'autre (Multi cloud).
- Cohérence entre les environnements Dev, Test et Prod.
- Permet de modulariser facilement l'application.
- Gérer l'héritage technique (Ancienne application) grâce à l'isolation.

Containers vs Virtual Machines

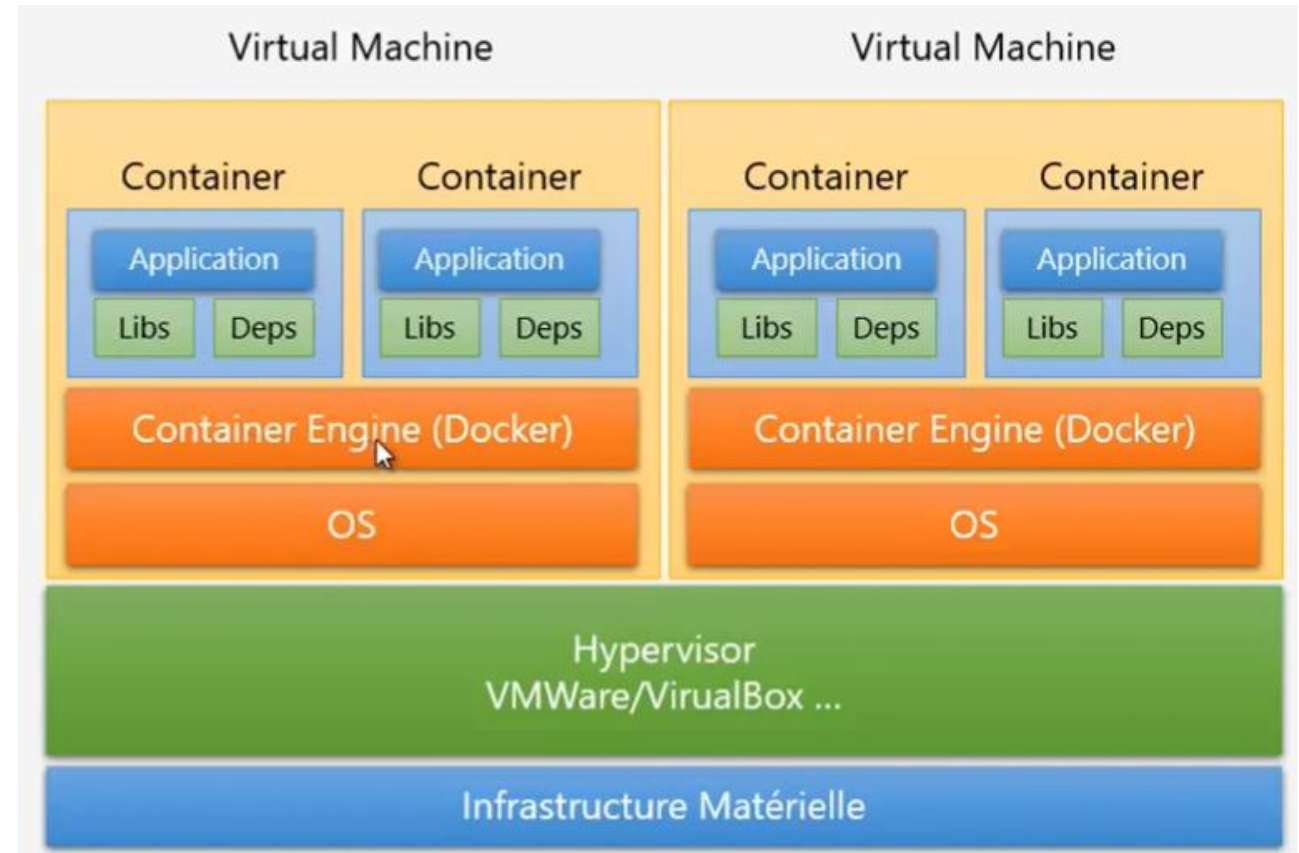
- Machine virtuelle :
 - Permet de virtualiser une machine physique.
 - Chaque VM a son propre OS.
 - Une VM consomme beaucoup de ressources (CPU, Stockage) et prend assez de temps pour booter (quelques minutes).
- Conteneur
 - Permet de créer un environnement d'exécution des applications.
 - Les conteneurs utilisent le même OS.
 - Tous les conteneurs utilisent le même Kernel (Linux), consomment peu de ressources, boot rapide (quelques secondes)



Utiliser des conteneurs dans des Machines virtuelles

POSSI

- Docker ne vient pas pour remplacer les machines virtuelles.
- Dans la pratique on utilise les deux :
 - Les machines virtuelles pour virtualiser les machines.
 - Utiliser Docker pour isoler les environnements d'exécution des applications dans des machines virtuelles.
- Ceci pour tirer le bénéfice des technologies

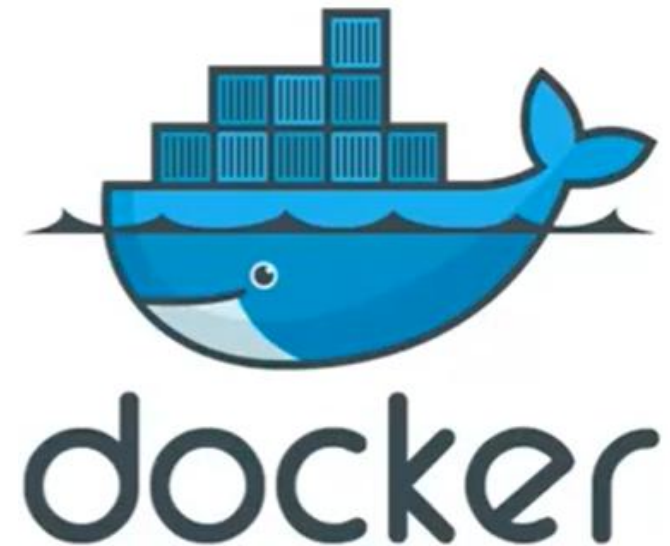


Histoire des conteneurs

- Conteneur = unité de transport intermodal.
- Virtualisation de processus :
 - UNIX chroot (1979-1982)
 - BSD Jail (1998)
 - Parallels Virtuozzo (2001)
 - Solaris Containers (2005)
 - Linux LXC (2008)
 - Docker (2013) (basé sur LXC)

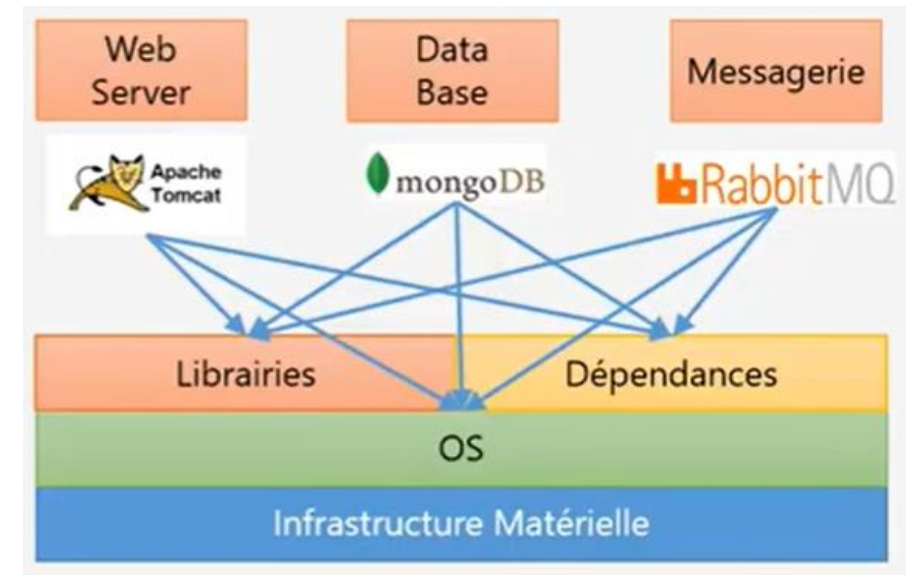
C'est quoi Docker

- Docker permet de créer des environnements (appels containers) de manière à isoler des applications.
- Il permet d'empaqueter une application ainsi que les dépendances nécessaires dans un conteneur virtuel isolé qui pourra être exécuté sur n'importe quelle machine supportant docker.
- Docker est un logiciel libre qui permet le déploiement d'applications sous la forme de conteneurs logiciels.
- L'origine de docker est :
 - Au départ, société française et maintenant basée à San Francisco.
 - dotCloud : un PaaS avec un container engine écrit en python.
 - En 2012 : Réécriture le langage GO.
 - Réaction très positive de la communauté.
 - dotCloud change de nom pour Docker.
 - En 2014, Levée de fonds : 40 millions \$.
 - En 2015 , Levée de fonds : 95 millions \$.



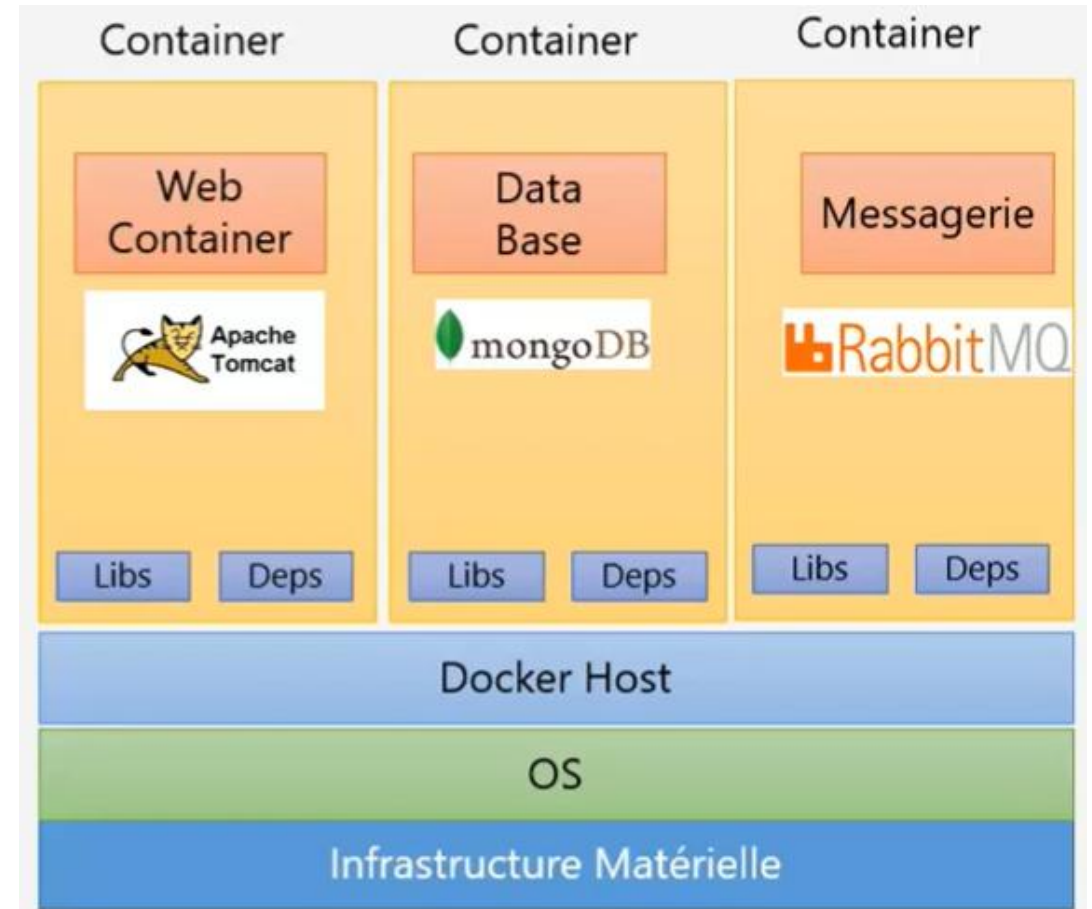
Problèmes de déploiement des applications

- Pour une application utilisant différentes technologies (services), des problèmes sont posés au moment du déploiement en production.
 - Compatibilité des applications avec les OS
 - Installer les dépendances et les librairies requises avec les bonnes versions pour chaque service.
 - Installer les différents environnements :
 - Dev
 - Test
 - Prod
- Ce qui prend beaucoup de temps pour déployer les applications.
- Avec des conflits entre les développeurs et les opérationnels (Administrateurs Systèmes)

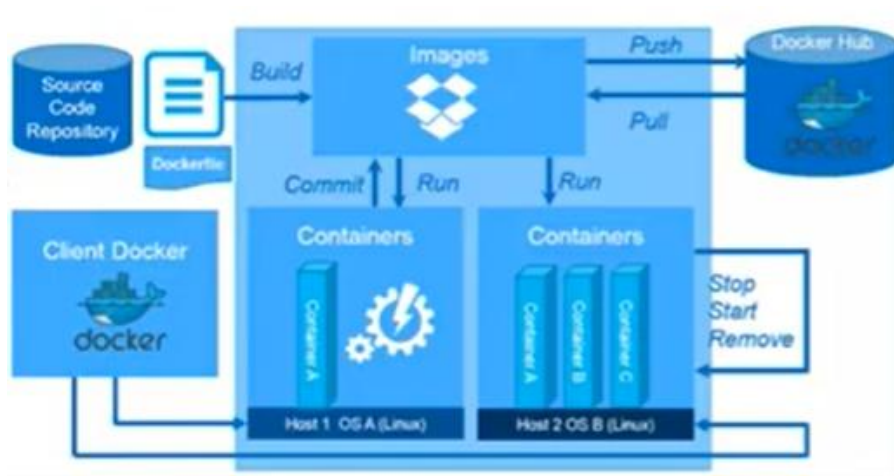


Solution : conteneurs d'applications

- Embarquer les applications dans des conteneurs.
- Exécuter chaque service avec ses propres dépendances dans des conteneurs séparés.



Architecture globale Docker



- Le développeur crée un fichier Dockerfile contenant les commandes que docker va exécuter pour construire une image docker de cette application.
 - `$ docker build`
- L'image docker contient tout ce dont l'application a besoin pour s'exécuter correctement.
 - `$ docker push image_name`
- Pour télécharger une image docker d'une application dans Host Docker, il suffit d'utiliser.
 - `$ docker pull image_name`
- La création et l'exécution d'un conteneur d'une application se fait par instantiation et exécution de l'image en utilisant :
 - `$ docker run image_name`
- Avec docker run, si l'image n'existe pas dans le host, elle va procéder au téléchargement celle-ci d'Internet et créer et exécuter un conteneur docker.
- Docker se compose de :
 - Docker Engine qui permet de créer le Host Docker sur une machine Linux (Docker daemon).
 - Un client Docker qui peut se trouver dans n'importe quelle autre machine et qui est connecté à Docker Engine via différents connecteurs par dockerd (socket, REST API, etc.)

Docker => Docker Hub

<https://hub.docker.com/>

```
$ docker run mysql  
$ docker run redis  
$ docker run mongod
```

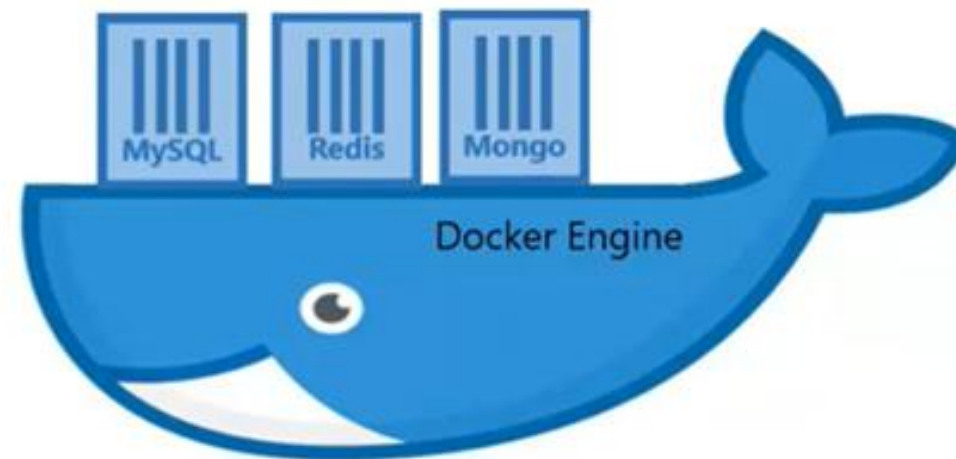
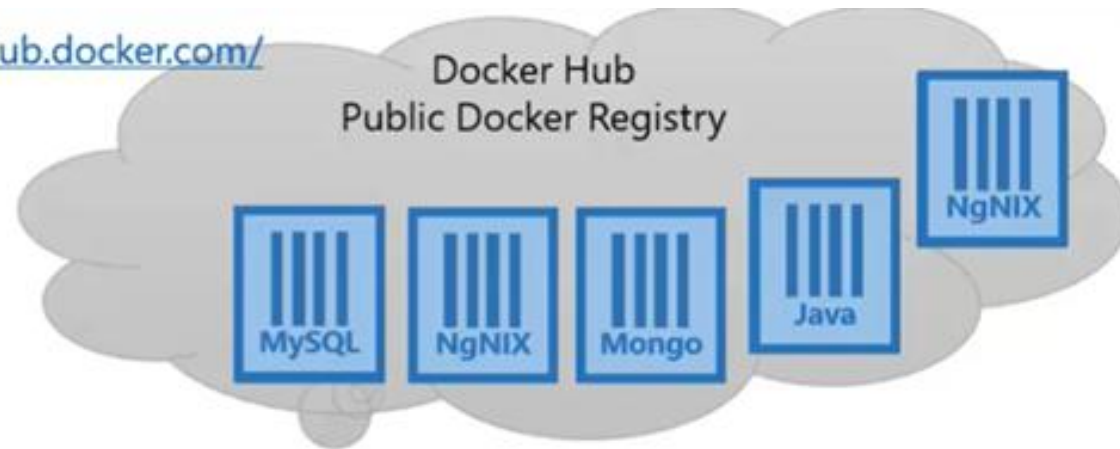


Image vs Container

- Une image docker est juste un fichier package représentant la Template des conteneurs. Elle définit la structure du conteneur en englobe l'application containerisée de ses dépendances.
- Un Conteneur représente une instance d'une image. Un conteneur est exécuté par le Docker Host. Ce qui implique l'exécution de l'application qu'il transporte dans un environnement isolé fourni par le conteneur.

Docker contribue à instaurer la culture DevOps

- Sans Docker :
 - Le développeur
 - Développe l'application.
 - Génère le package de l'application à déployer (App war)
 - Envoie à l'opérationnel (Administrateur système)
 - App war
 - Un descriptif des dépendances qu'il faut installer et configurer pour que l'application s'exécute normalement.
 - L'opérationnel
 - Doit se débrouiller pour satisfaire les exigences de l'application.
 - Pour chaque mise à jour, c'est toujours les mêmes histoires qui se répètent.
 - Ce qui rend la vie dur au administrateur systèmes (Opérationnels).
 - Ce qui crée beaucoup de conflits entre les développeurs qui tentent d'améliorer constamment les applications et les opérationnels qui doivent redéployer les mises à jour.
- Avec Docker :
 - Le développeur
 - Développe l'application
 - Construit une image docker de son application contenant toutes les dépendances dont l'application a besoin.
 - Publie l'image docker dans le registre docker.
 - L'opérationnel déploie l'application en instanciant à partir de l'image docker récupérée à partir du repository docker.



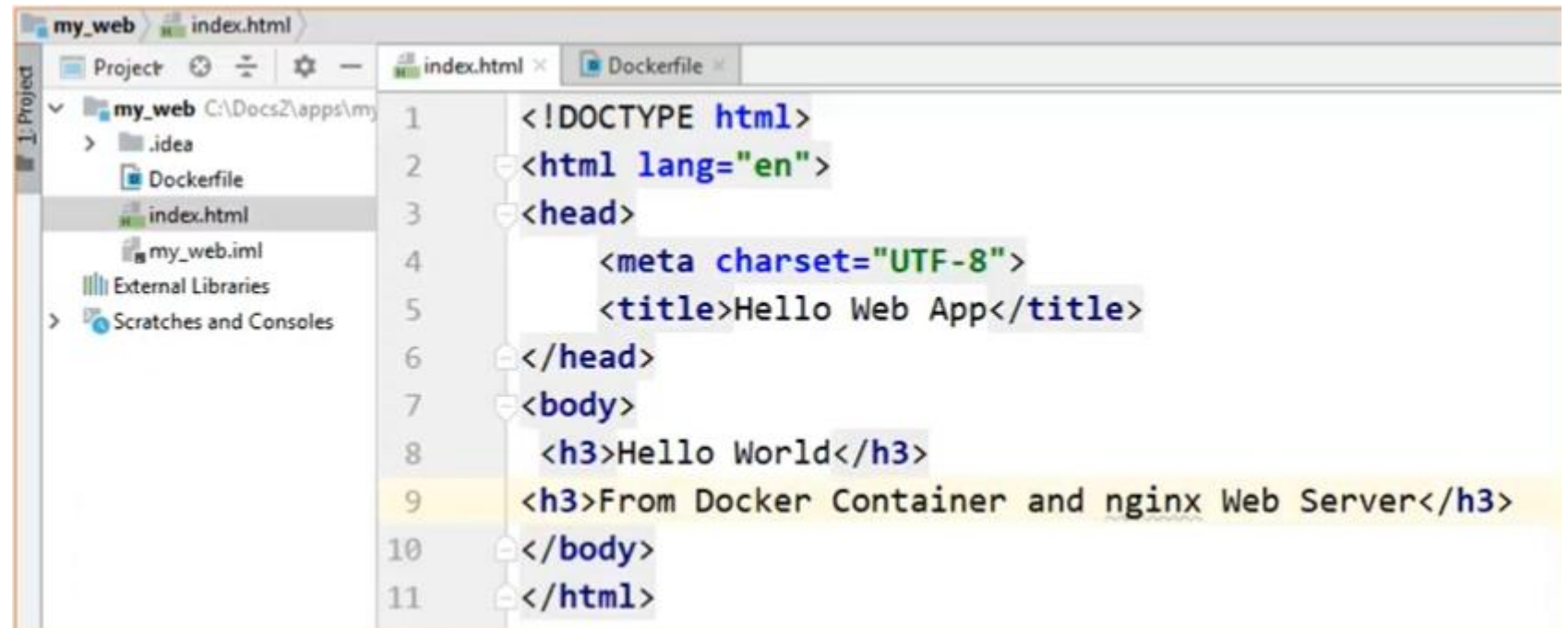
Commandes de base docker : run et ps

- `$ docker run image_name`
 - Permet de créer et démarre un conteneur nouvelle instance de l'image.
 - Si l'image n'existe pas, elle sera d'abord téléchargée à partir du docker hub
- `$ docker run -d image_name`
 - Permet de démarrer le conteneur comme service de fond.
- La valeur par défaut de la version de l'image est **latest**
- Pour spécifier la version (tag) :
 - `$ docker run image_name:tag`
 - Exemple : `$ docker run redis:4.0`
- `$ 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.
- `$ docker ps -a`
 - Permet de lister tous les conteneur avec leurs status (Up, Exited, Created)

Instruction de Dockerfile

- A Dockerfile uses the following commands for building the images :
 - **ADD** : Copy files from a source on the host to the containers own file system at the set destination
 - **CMD** : Exécute a specific command within the container.
 - **ENTRYPOINT** : Set a default application to be used every time a container is created with the image.
 - **ENV** : Set environment variables.
 - **EXPOSE** : Expose a specific port to enable networking between the container and the outside world.
 - **FROM** : Define the base image used to start the build process.
 - **MAINTAINER** : Define the full name and email address of the image creator.
 - **RUN** : Central executing directive for Dockerfiles.
 - **USER** : Set the UID (the username) that will run the container.
 - **VOLUME** : Enable access from the container to a directory on the host machine.
 - **WORKDIR** : Set the path where the command defined with CMD is to be executed.

Créer une image Docker



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Hello Web App</title>
6 </head>
7 <body>
8     <h3>Hello World</h3>
9     <h3>From Docker Container and nginx Web Server</h3>
10 </body>
11 </html>
```

Dockerfile

```
FROM nginx
```

```
COPY index.html /usr/share/nginx/html
```

```
EXPOSE 80
```

Créer une image Docker

- FROM nginx : mon image va se baser sur nginx
- COPY index.html /usr/share/nginx/html : pour déployer mon application dans nginx il faut copier le fichier index.html dans le dossier utilisé par défaut par nginx pour déployer les applications.
- EXPOSE 80 : exposer l'application sur le port 80

```
$ sudo docker build -t myApp .
```

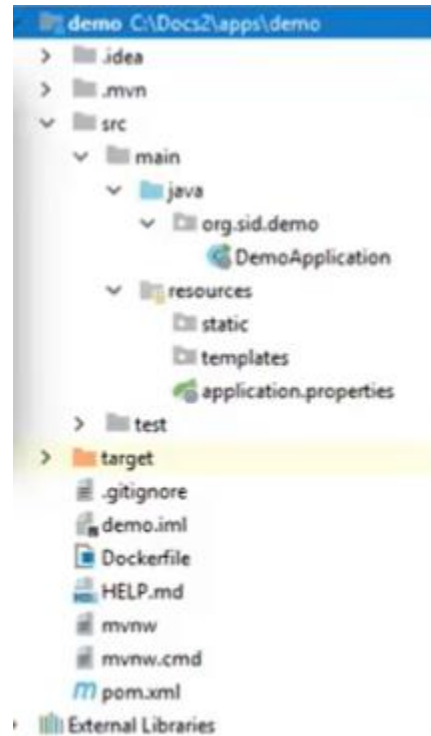
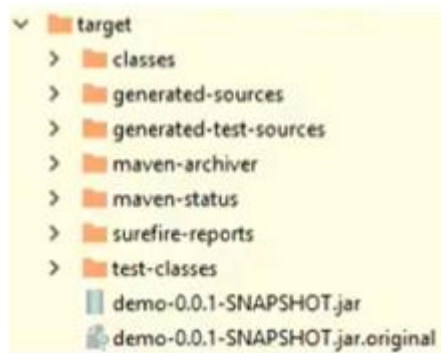
```
$ sudo docker run -d -p 8089:80 myApp
```

← → ↻ ⓘ Non sécurisé | 192.168.57.3:8089

Hello World

From Docker Container and nginx Web Server

Créer une image Docker



```
1 package org.sid.demo;
2 import org.springframework.boot.SpringApplication;
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6 @SpringBootApplication
7 @RestController
8 public class DemoApplication {
9     @GetMapping("/")
10    public String sayHello(){
11        return "Hello From Spring Boot in Docker...";
12    }
13    public static void main(String[] args) {
14        SpringApplication.run(DemoApplication.class, args);
15    }
16 }
```

Dockerfile

```
FROM openjdk:8-jdk-alpine
```

```
VOLUME /tmp
```

```
ADD target/demo*.jar /app.jar
```

```
CMD ["java", "-jar", "/app.jar", "--spring.profiles.active=prod"]
```

```
EXPOSE 8080
```

```
C:\Docs2\apps\demo> mvn package
```