# Docker - Introduction à la "containerisation"

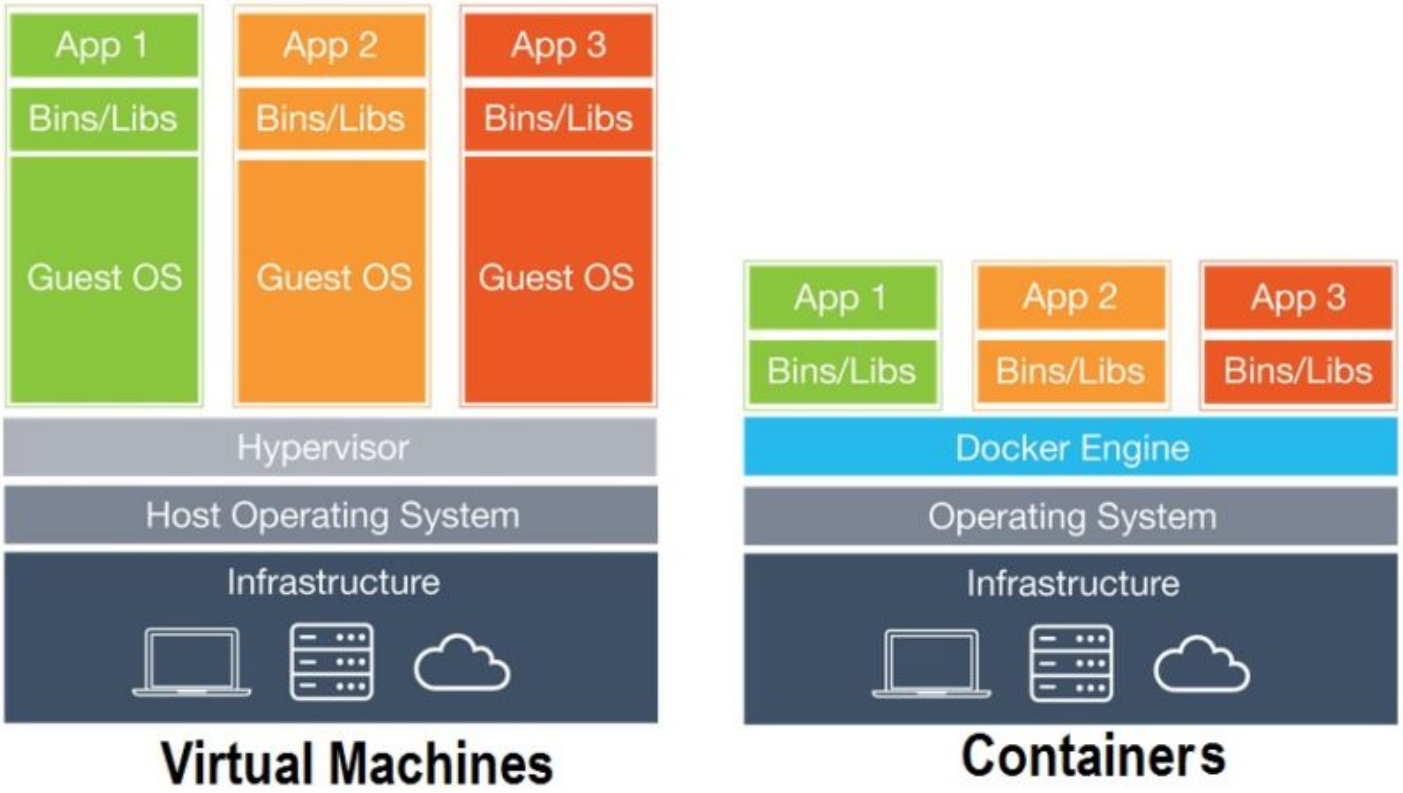
Difficulté : Intermédiaire

Notions : Containers, systèmes de containers, docker.

## I. Introduction

Tout comme la virtualisation avais permis un bond en avant en permettant de placer plusieurs machines virtuelles sur un hôte physique, la containérisation va plus loin en ne virtualisant plus un système entier mais en cloisonnant directement une couche applicative ou un service dans un container.

Basé sur un système d'image, celui-ci permet de virtualiser un morceau de système sur lequel viendront s'appuyer les containers contenant les ressources (binaires, configurations, librairies et dépendances) et auquel il sera possible de connecter des volumes persistant contenant les données.



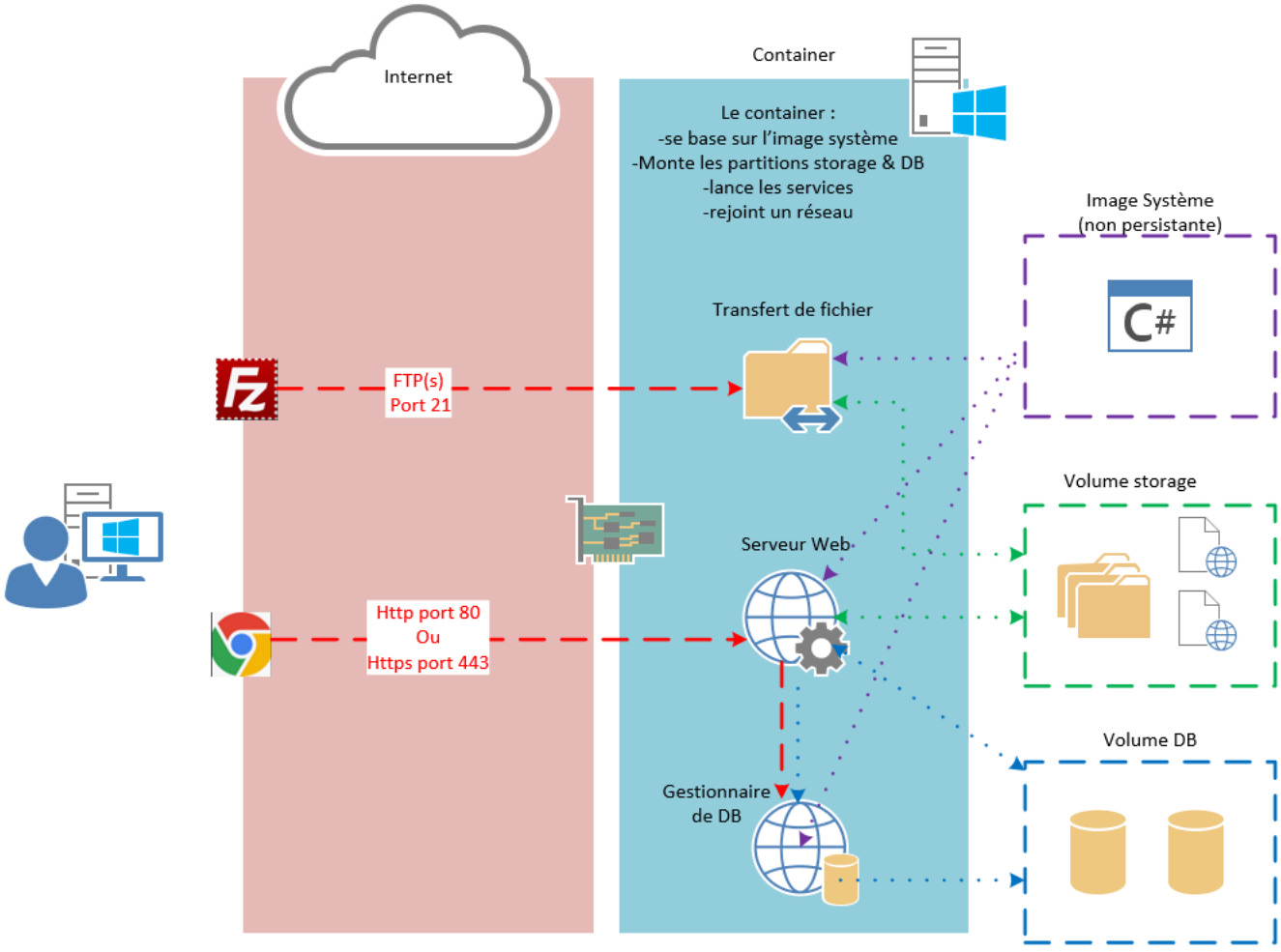

Cela est une excellente alternative en terme de performance, la mutualisation des images permettant un gain significatif du fait de la non nécessité de virtualiser une couche hardware complète dans la plupart des cas.

Mais également en terme de gestion et de déploiement. Une fois les images créées, elle sont déployables en quelques minutes et utilisables immédiatement. De plus si plusieurs containers utilisent la même image et nécessitent un mise à jour, il sera simple de mettre à jour l'image et la pousser en production. les containers pourront ensuite être liés a cette nouvelle image, réduisant le temps de mise à jour et les indisponibilités.

Enfin en terme de sécurité. Chaque application utilisant son propre environnement cloisonné, il sera très difficile d'infecter les systèmes adjacents et la compromission d'un service aura des effets uniquement sur celui-ci.

De plus, les images étant non persistantes, un rechargement de celle-ci supprimera les éventuelles traces d'attaque et permettra de revenir rapidement à une configuration sûre.

Cela permet également de personnaliser grandement l'architecture des services. Par exemple voici un container web :




Enfin Le dernier avantage et non le moindre, est la grande capacité d'automatisation de l'outil.

Les images et containers, ainsi que des architectures complètes peuvent être créées a partir de fichiers d'instructions (la plupart en YAML) qui seront utilisés comme base à travers des API.

Ce qui permet de faire de "l'infrastructure as code" et de publier en quelques étapes des services voire des infrastructures complètes architecturées en amont.

---

# II. Principes de bases

**Les images :** Les images sont des versions packagées de services. Elles sont construite à l'aide de fichiers que l'on appelle 'dockerfiles'. Le plus souvent, elle partent d'une couche OS sur laquelle le(s) service(s) et le contenu seront préconfigurés et packagés puis fournis sous forme d'image pour un déploiement rapide et une grande souplesse dans la gestion des containers utilisant ces

images. Lors de l'utilisation par un container, le contenu de l'image est dit 'non persistant'. En effet, si le container est arrété ou recontruit, l'image redémarre sur son état initial.

**Les volumes et points de montage :** Du fait de la non persistance des images, il n'est pas possible d'y stocker des données dites 'vivantes' dans le container. Pour cela, il faudra lui attacher des volumes ou des fichiers. Il s'agit des ressources qui seront partagées entre l'hôte et le container. Cela peut être des fichiers de configurations par exemple qui seront montés dans le container sur un chemin précis ou dans le cas des volumes. Lors de la création de ceux-ci, un espace dédié est créé sur l'hôte et sera utilisé ensuite dans le container sur un point de montage. L'on y placera toutes les données qui feront vivre le service (site web, base de donnée, fichiers du service, logs...)

**Les containers :** Le container est donc ce qui résulte de l'utilisation des éléments ci-dessus. Il va utiliser une image pour fournir un service et les volumes pour stocker la donnée.

**Les services :** Sur un cluster (swarm), les containeurs ne sont plus addressés directement en tant que tel mais font partie d'un service. Celui-ci permet de disposer de propriétés de déploiement comme de la haute dispo, des contraintes de placement sur des noeuds du cluster, le nombre de copies de containers, etc...

**Les stacks :** Une architecture rendant un service peut être constituées de plusieurs containers ou services. par exemple dans le cadre d'un hébergement web, il peut y avoir un container wordpress, puis un container DB et un container SSH, tous constitutifs du 'stack' hebergement.

**Les réseaux :** Les réseaux permettent à différents services de comuniquer entre eux. Par exemple permettre à un container web de se connecter à un container SQL, ou encore à un reverse proxy d'adresser un serveur web.

**L'exposition des ports :** Par défaut les containers sont dans des réseaux isolés ou bridgés non accessible de l'extérieur. Il faut pour cela natter des ports. Par exemple un port 8082 en TCP sur le port 80 d'un container. dans ce cas, c'est l'hôte qui devra être addressé avec le port natté (ici: 8082). On parkle alors d'une "exposition de port".

**Les secrets :** Certaines informations sont par nature sensibles. Les comptes ou mots de passes, des clés, des tokens, etc... Pour se passer ses informations entre les noeuds ou les monter dans un ou plusieurs container sans exposer ces informations il est possibles de créer des secrets. Ce sont des fichiers chiffrés qui sont directement généré à partir d'une saisie d'information protégées et ne seront ensuite utilisable qu'entre les membres du cluster et aux containers. Cela permet d'utiliser ces informations en toute sécurité.

**Les dockerfiles :** Ce sont des fichiers contenant une séquence d'instructions qui vont être utilisés pour générer une image. Cela sert dans le cadre de l'automatisation. Exemple :

```
FROM image_name #partir de l'image désignée (ou FROM scratch) pour recompiler une image en partant de zéro.

LABEL label1="valeur1" label2="valeur2" label3="valeur3" #définit les tags de l'image

VOLUME ['/chemin/vers/dossier'] # si pas de volume associé, par défaut monte un volume à cet endroit

ARG ARG1="value" #variable 1
ARG ARG2="value" #variable 2

WORKDIR /chemin/racine #chemin sur lequel on se place pour exécuter les commandes

COPY /chemin/vers/fichier /chemin/dans/container #copie un fichier présent sur la machine dans le container

RUN commande1 # lance une commande dans le container
RUN commande2
RUN commande3

EXPOSE 80/tcp #expose le port voulu dans le protocole voulu

ENTRYPOINT ["command", "arg1", "arg2"] #donne le processur ou service à observer pour le démarrage du container
```

> Pour plus d'informations : Dockerfile reference

**Les fichiers compose :** Les fichiers compose sont des fichier descriptif de l'architecture d'un service ou d'un stack. Ils permettent de générer les services/stacks à partir de ce fichier de manière automatisée. Cela rentre dans le cadre de "l'infrastructure as code".

Exemple de fichier compose pour un stack wordpress :

```
services:
  db:
    # We use a mariadb image which supports both amd64 & arm64 architecture
    image: mariadb:10.6.4-focal
    # If you really want to use MySQL, uncomment the following line
    #image: mysql:8.0.27
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
  wp_data:
```

**Le swarm :** le swarm est un cluster composant un ensemble de noeuds sous docker. Un swarm est initialisé par un noeud 'manager' et comportera ensuite un ensemble de 'worker'. Cela permettra de répartir des charges de travail et/ou de fiabiliser certains services en les répliquants sur plusieurs noeuds.

---

Revision #3
Created 2026-07-17 20:45:24 UTC by Olivier
Updated 2026-07-17 20:46:49 UTC by Olivier