Module Cloud

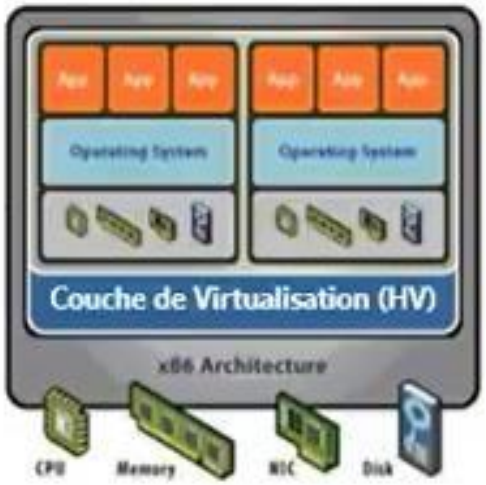
Conteneurisation - Docker

+ Plan

- Evolution de l'infrastructure
- Les conteneurs
- Conteneurs vs. Machines virtuelles
- Docker
- Pratiques Docker
- Orchestration

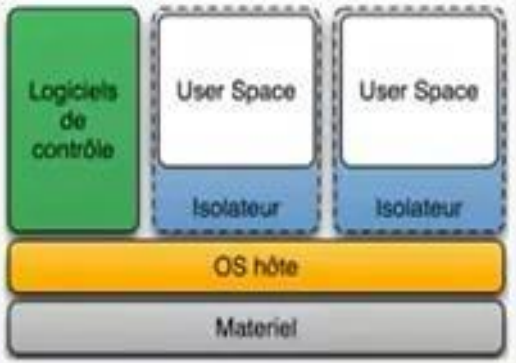
Virtualisation de l'OS

- Virtualisation de serveurs
- Virtualisation des postes de travail
  - Virtual Desk Top Infrastructure (VDI) :
    - Postes de travail hébergés dans un Data center
    - L'utilisateur accède à son poste de travail à distance en utilisant son terminal.
    - L'environnement de travail est associé au profile utilisateur
    - L'utilisateur a la même vue sur son environnement quelque soit le terminal utilisé.
  - Streaming d'OS.
    - Le système démarre à partir d'un disque sur le réseaux
    - Charge d'une manière sélectives les contenues les applications demandées par l'utilisateur à partir d'un serveur distant.
- Dans un même serveur physique on peut créer plusieurs serveurs virtuels et plusieurs postes de travail.

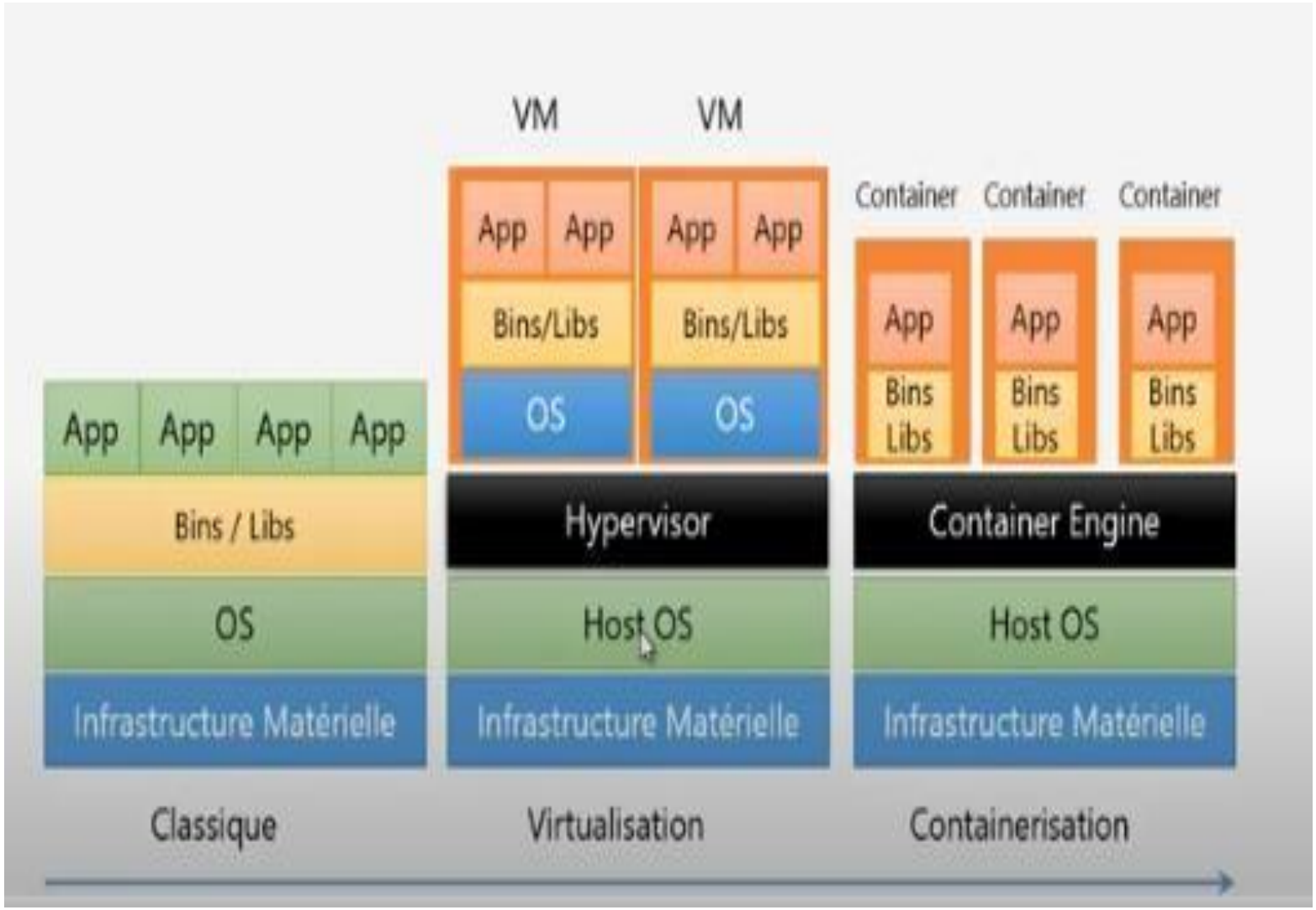


Virtualisation des applications

- **Virtualisation de sessions (Virtualisation de présentation):**
  - La virtualisation est réalisée du coté du serveur. L'ensemble des données de l'utilisateur ainsi que les applications sont hébergées dans un Data center.
  - L'utilisateur accède à ses applications par un simple navigateur web ou par un client Desttop installé sur son poste de travail en ouvrant une session sur le serveur distant.
- **Virtualisation par Isolation ou Bulles Applicatives**
  - Consiste à l'installation d'une application par streaming dans le poste de travail
  - L'administrateur déploie les applications sur un serveur distant
  - L'utilisateur se connecte au serveur et télécharge les applications autorisées sur son poste
  - Quand le client clique sur l'icone de l'application, le poste client demande au serveur de lui envoyer la bulle applicative en streaming.
  - Ensuite l'application s'exécute en local.
  - L'application ne s'installe pas sur l'OS du poste client, elle s'exécute sur une bulle, cad un environnement totalement virtualisé et cloisonné.
  - Exemples : UNIX chroot (1979-1982), BSD Jail (1998), Parallels Virtuozzo (2001), Solaris Containers (2005), Linux LXC (2008), Docker (2013) (basé sur LXC)



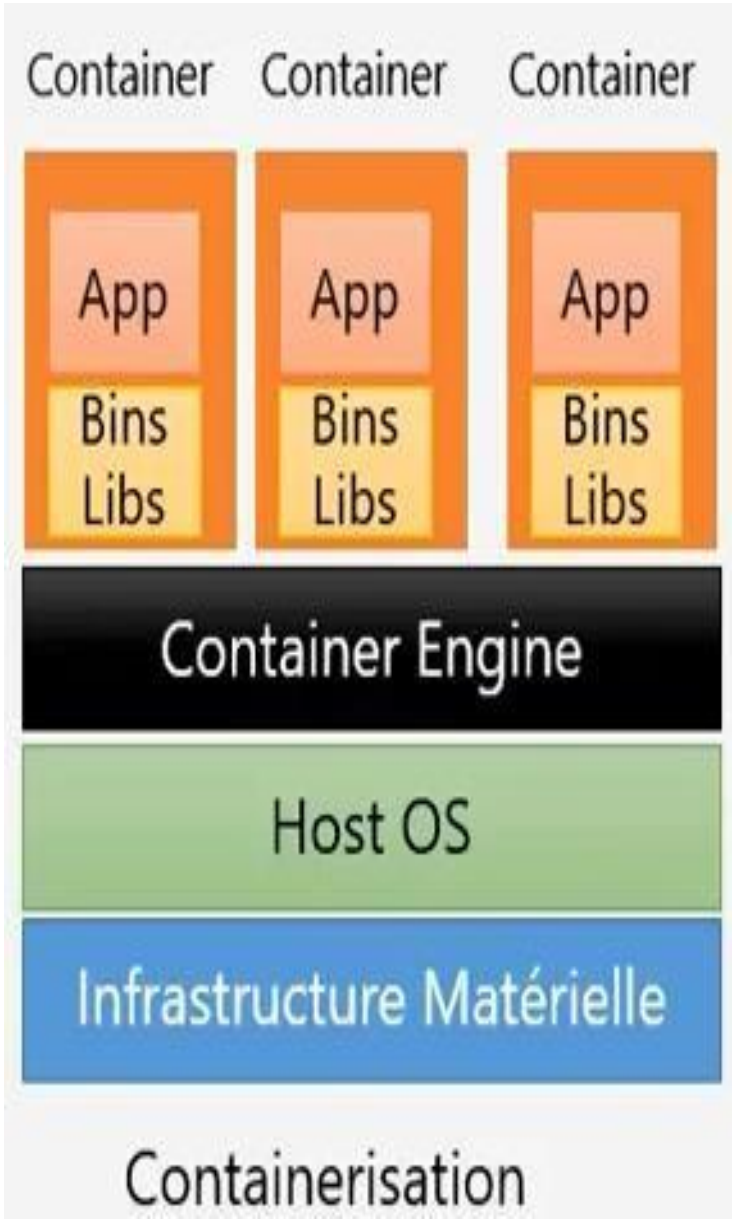
+ Evolution de l'infrastructure



+ Container

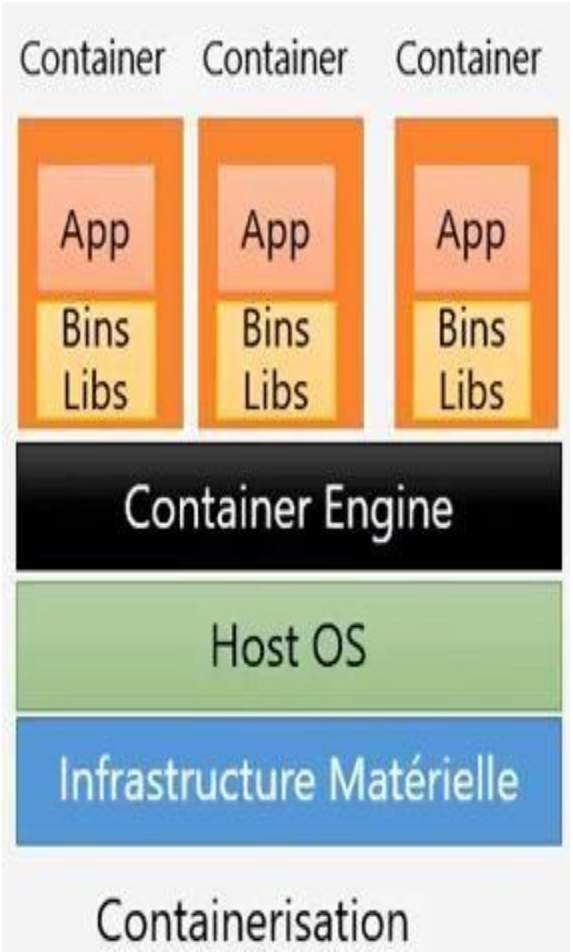
- Enveloppe permettant de packager une application avec juste ce dont elle a besoin.
- Peut être déployé tel quel dans n'importe quelle machine disposant d'un container Engine avec différents environnements (Dev, Test, Prod).
- Utilise le kernel de l'OS hote.
- A son propre espace de processus et sa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

+ Container

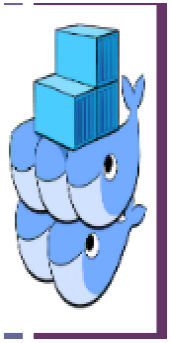

Container
Container
Container
App
App
App
Bins
Libs
Bins
Libs
Bins
Libs
Container Engine
Host OS
Infrastructure Matérielle
Containerisation

+ Pourquoi utiliser les containers

- Meilleure performance que les VMs (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 machines Virtuelles

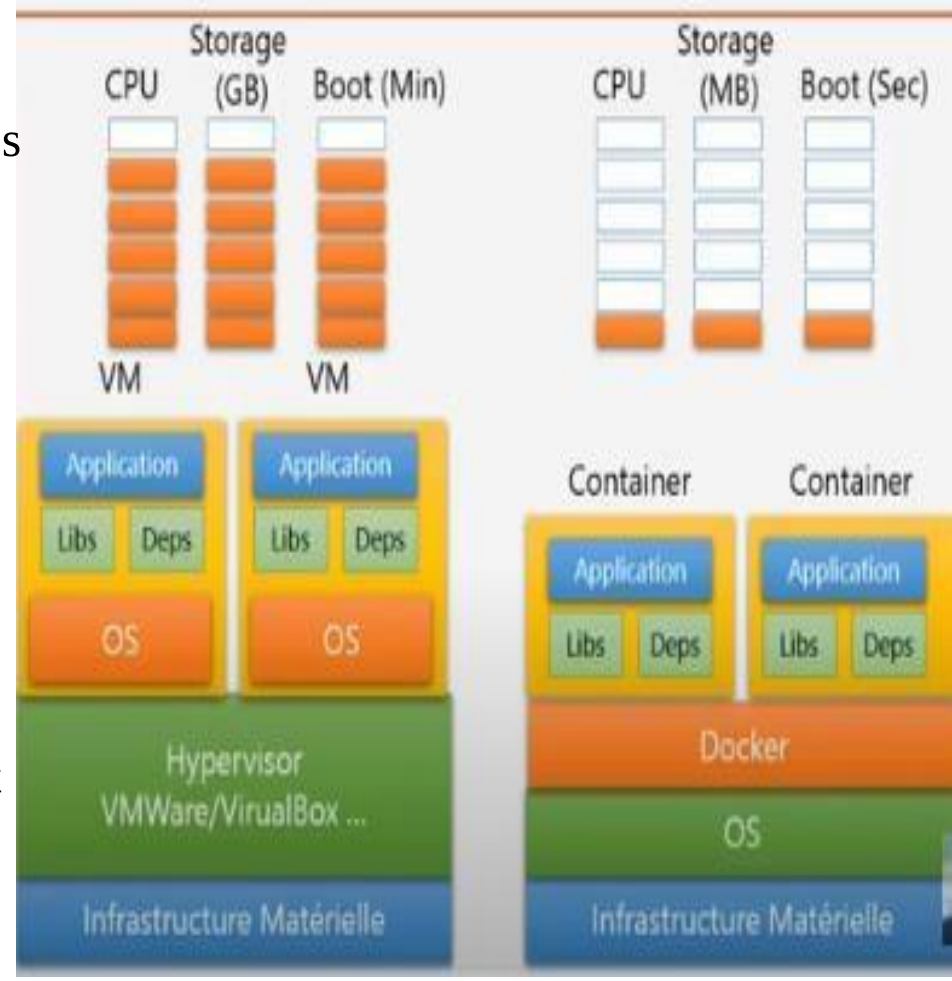


❑ **Machine Virtuelle:**

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

❑ **Conteneur**

- Permet de créer un environnement d ’exécution des applications
- Les conteneurs utilisent les mêmes OS
- Tous les conteneurs utilisent le meme kernel OS (Linux), consomme peu de ressources; boot rapide



+ Utiliser des conteneurs dans des VM

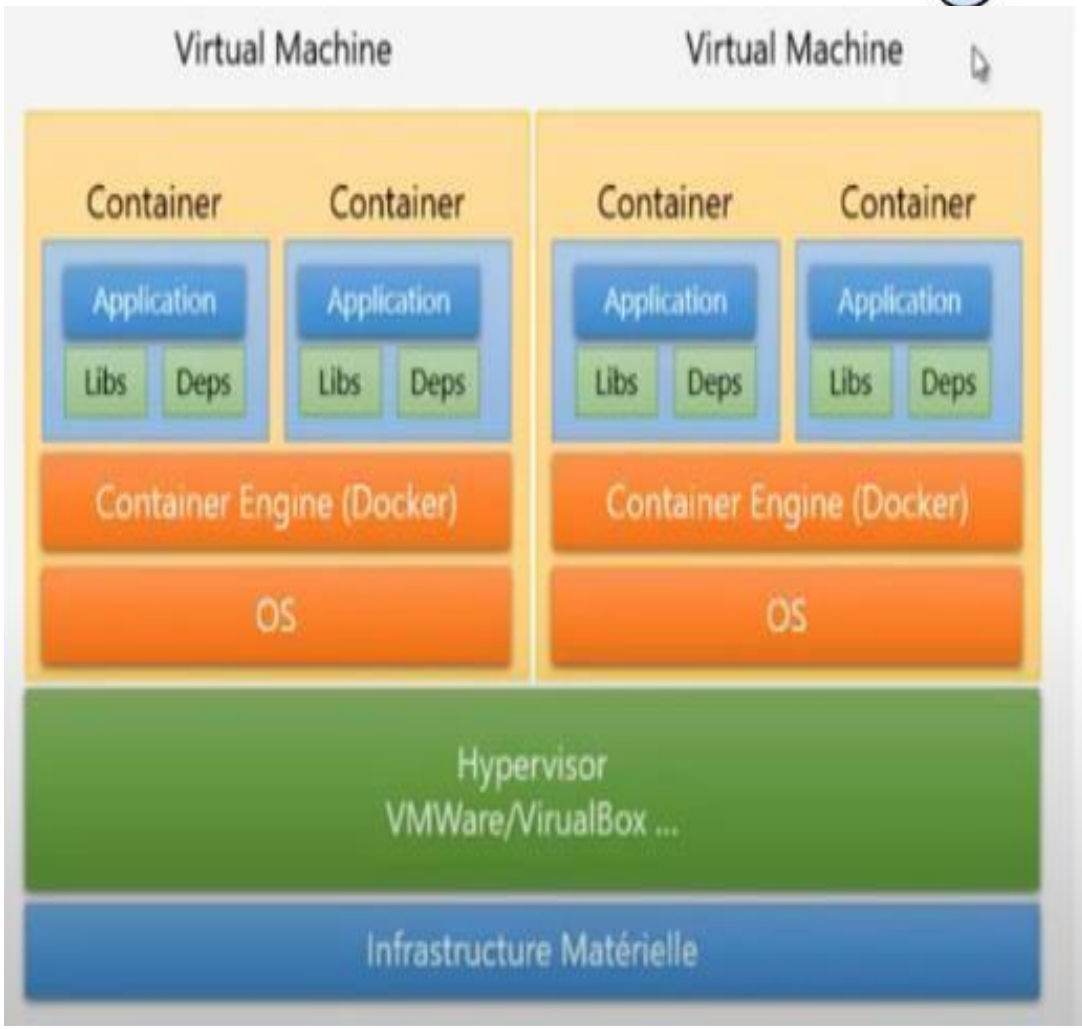
- Docker ne vient pas pour remplacer les machines virutelles
- Dans la pratique, on utilise les deux :

- Les machines virtuelles pour virtualiser les machines.

- Utiliser docker pour isoler les environnements d'exécution

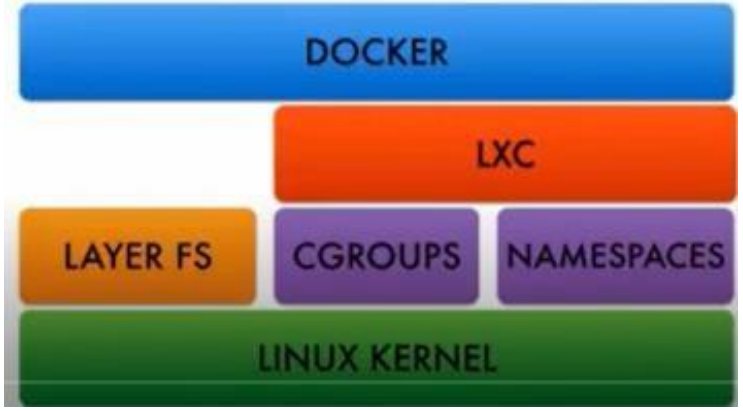
des applications dans des machines virtuelles

- Ceci pour tirer le bénéfice des technologies



Histoire des conteneurs

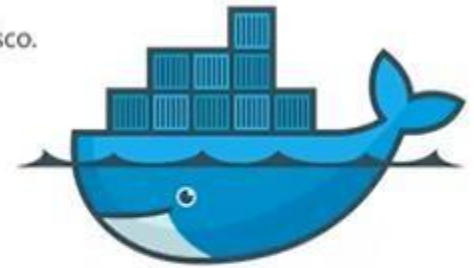
- Conteneur = unité de transport intermodal
- Virtualisation de processus :
  - UNIX chroot (1979-1982)
  - BSD Jail (1998)
  - Parallels Virtuozzo (2001)
  - Solaris Containers (2005)
  - Linux LXC (2008)
  - Docker (2013) (basé sur LXC)



+ C'est quoi Docker

- Docker permet de créer des environnements (appelés 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 qui permet le déploiement d'applications sous la forme de conteneurs logiciels.
- L'origine de docker est au départ :
  - Au départ société francaise, et maintenant basée à San Francisco
  - dotCloud : un PaaS avec un container engine écrit en Python
  - En 2012 : Ré-ecriture de langage go
  - En 2013; première version open source de docker
  - Réaction très positive de la communoté
  - DotCloud change de nom pour docker

isco.



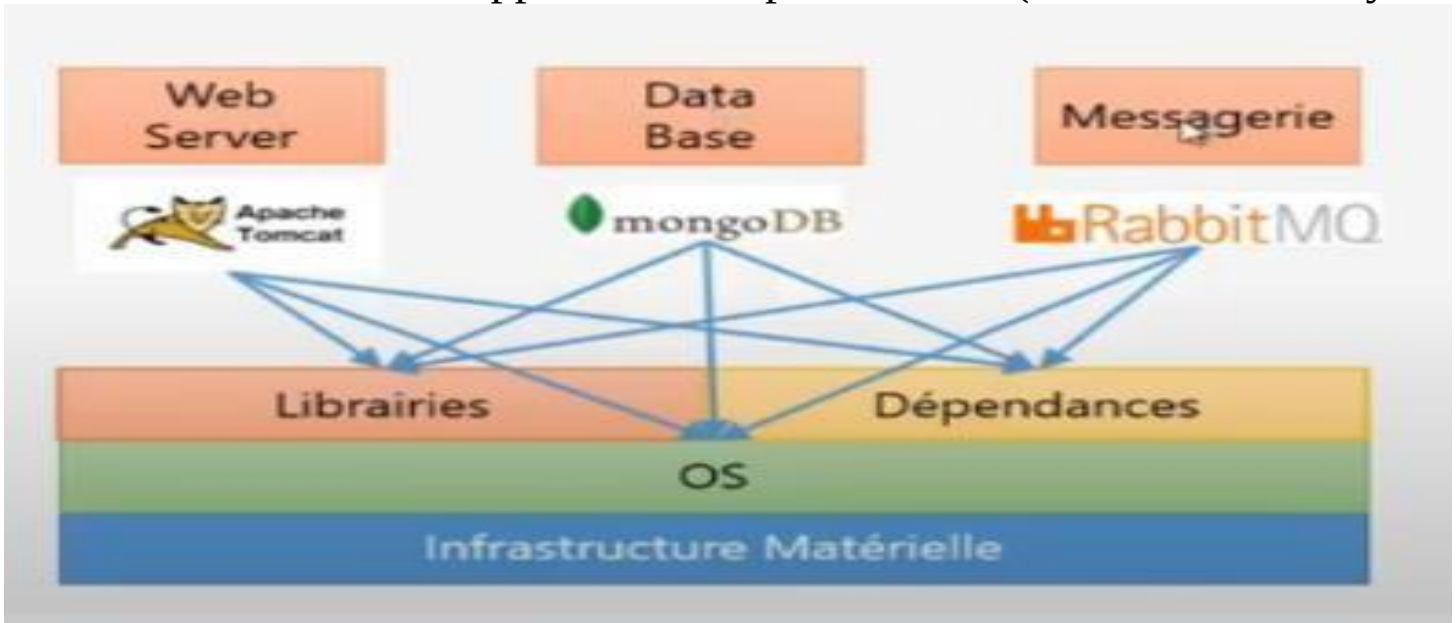
Problèmes de déploiement des applications

- Pour une application utilisant différentes technologies (Services),

  des problèmes sont posés au moment du déploiement en production :

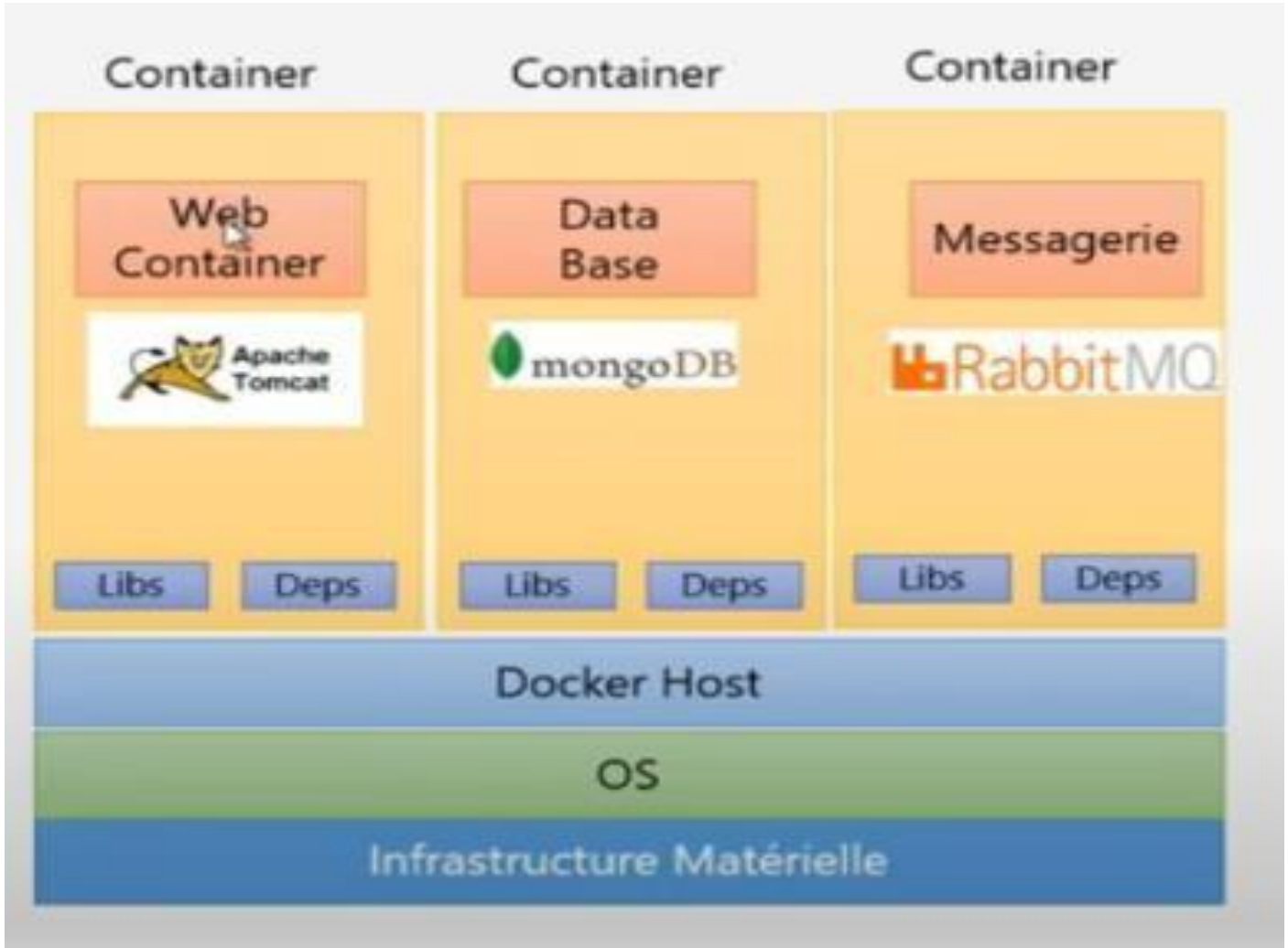
  - Compatibilité des applications avec les OS
  - Installer les dépendances et les librairies réquises avec les bonnes versions pour chaque service
  - Installer les différents environnements : Dev, Test, Prod

- Ce qui prend beaucoup de temps pour déployer des applications
- Avec des conflits entre les développeurs et les opérationnels (Administrateurs système)

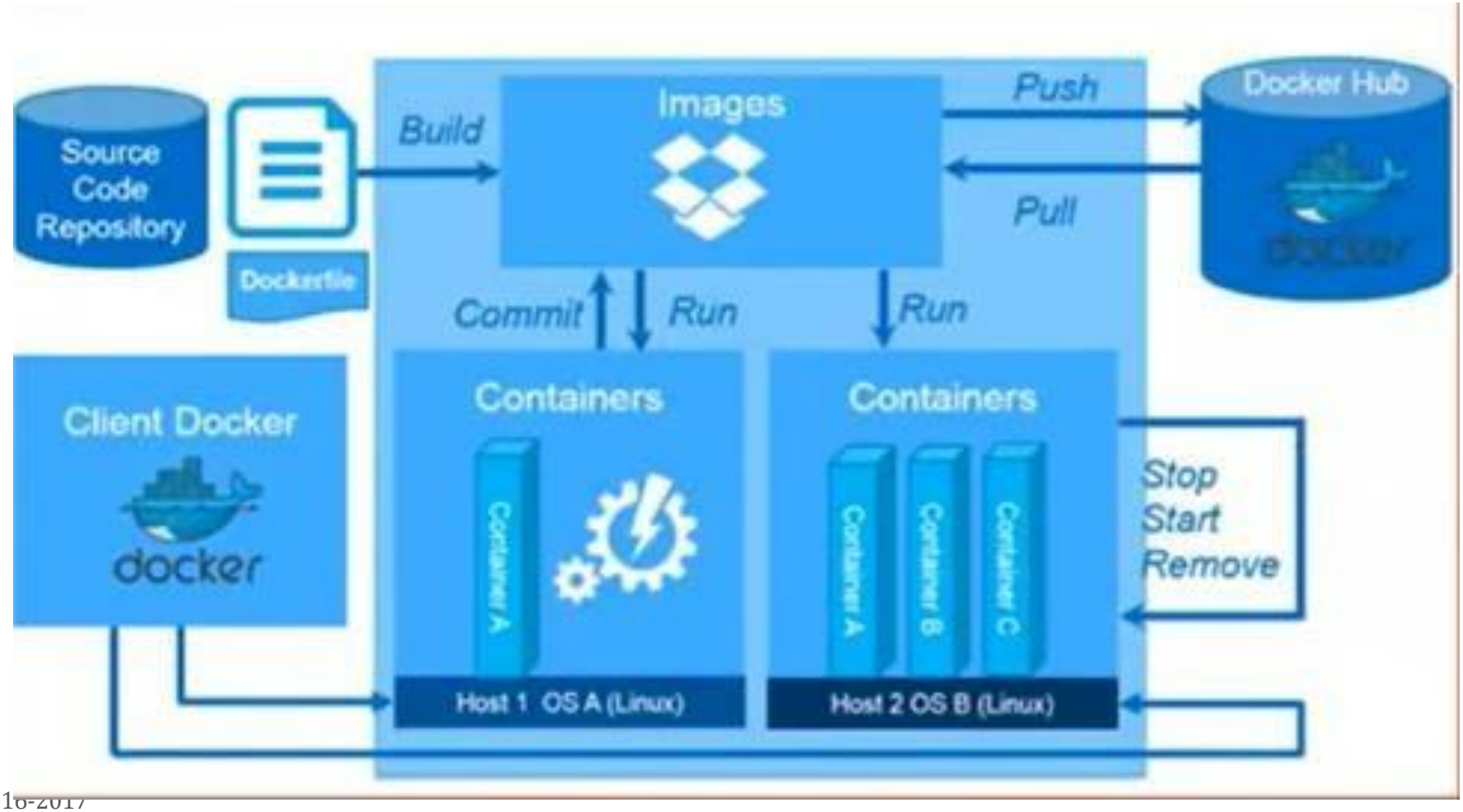


Solution : Conteneurs des applications

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



Architecture globale de Docker

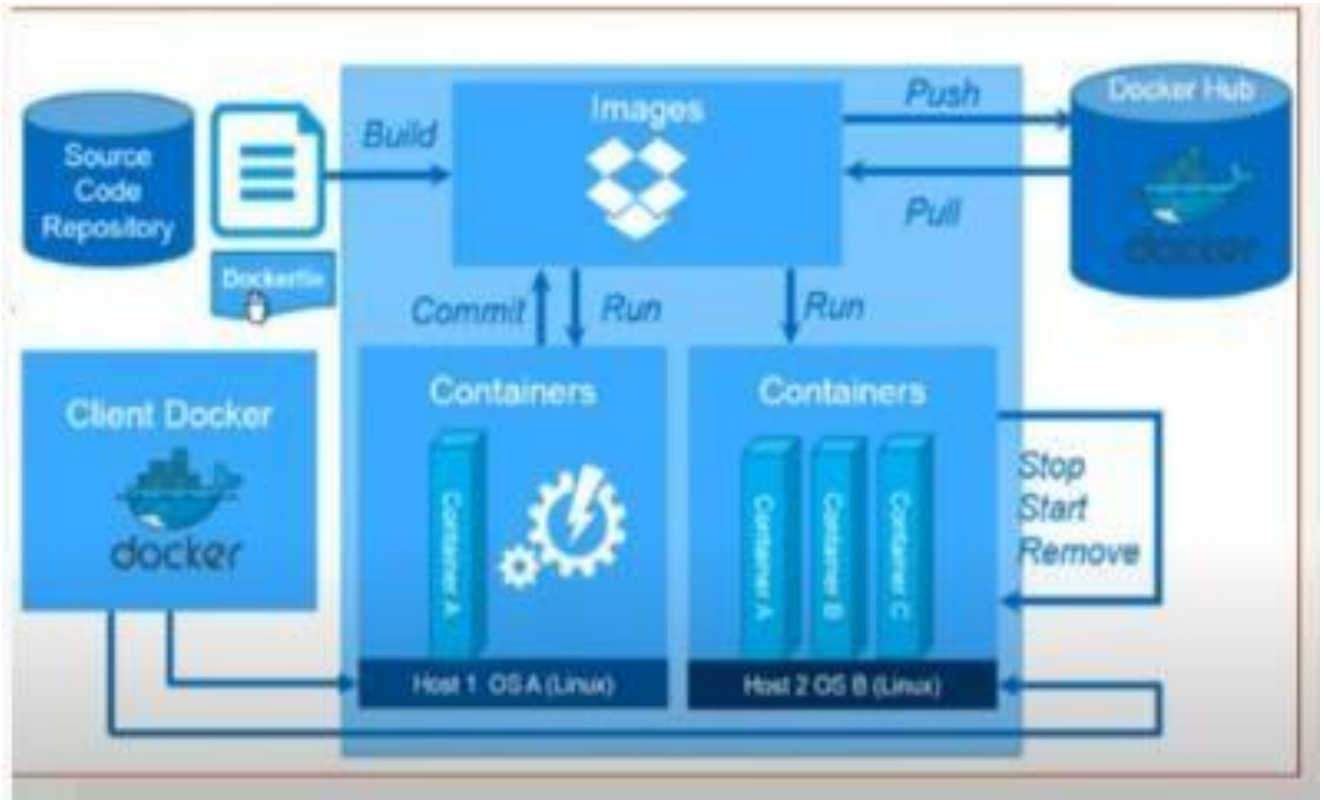


Architecture globale de docker

- Docker se compose de :

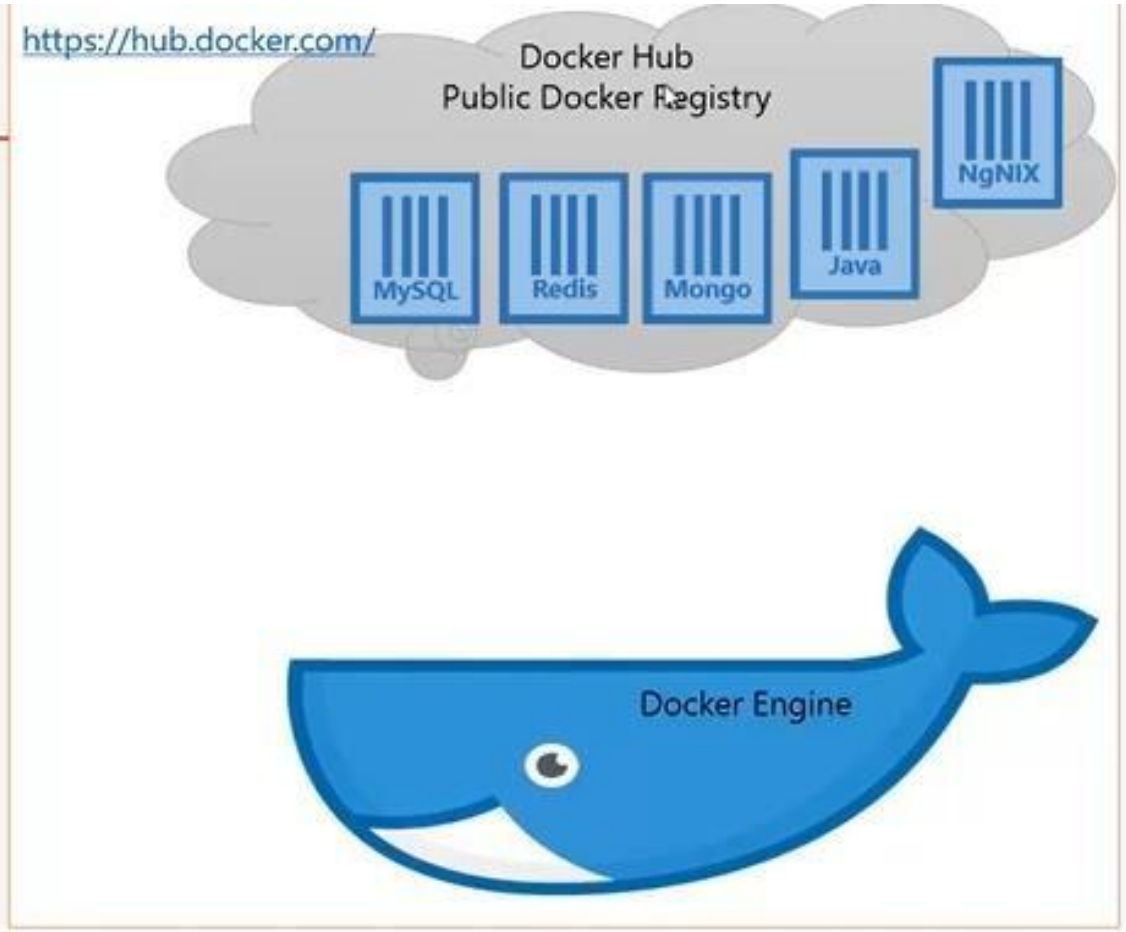
  - **Docker engine :** qui permet de créer le host docker sur une machine Linux (Docker Daemon)

  - **Un Client Docker :** qui peut se trouver dans n'importe quelle autre machine et qui est connecté à Docker Engine via différents connecteurs exposés par (Socket, REST API, etc).



Architecture globale de Docker

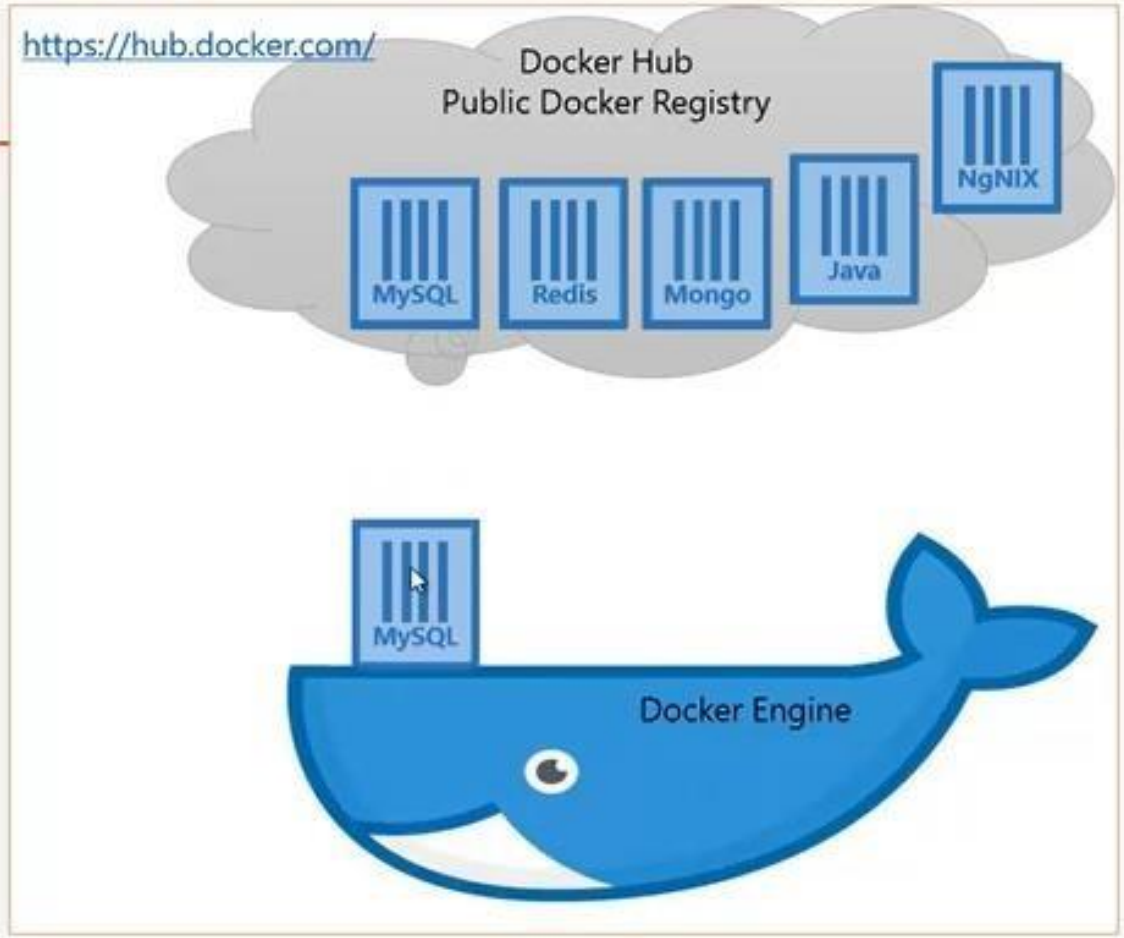
Docker => Docker Hub



Architecture globale de Docker

Docker => Docker Hub

```
$ docker run mysql
```



Architecture globale de Docker

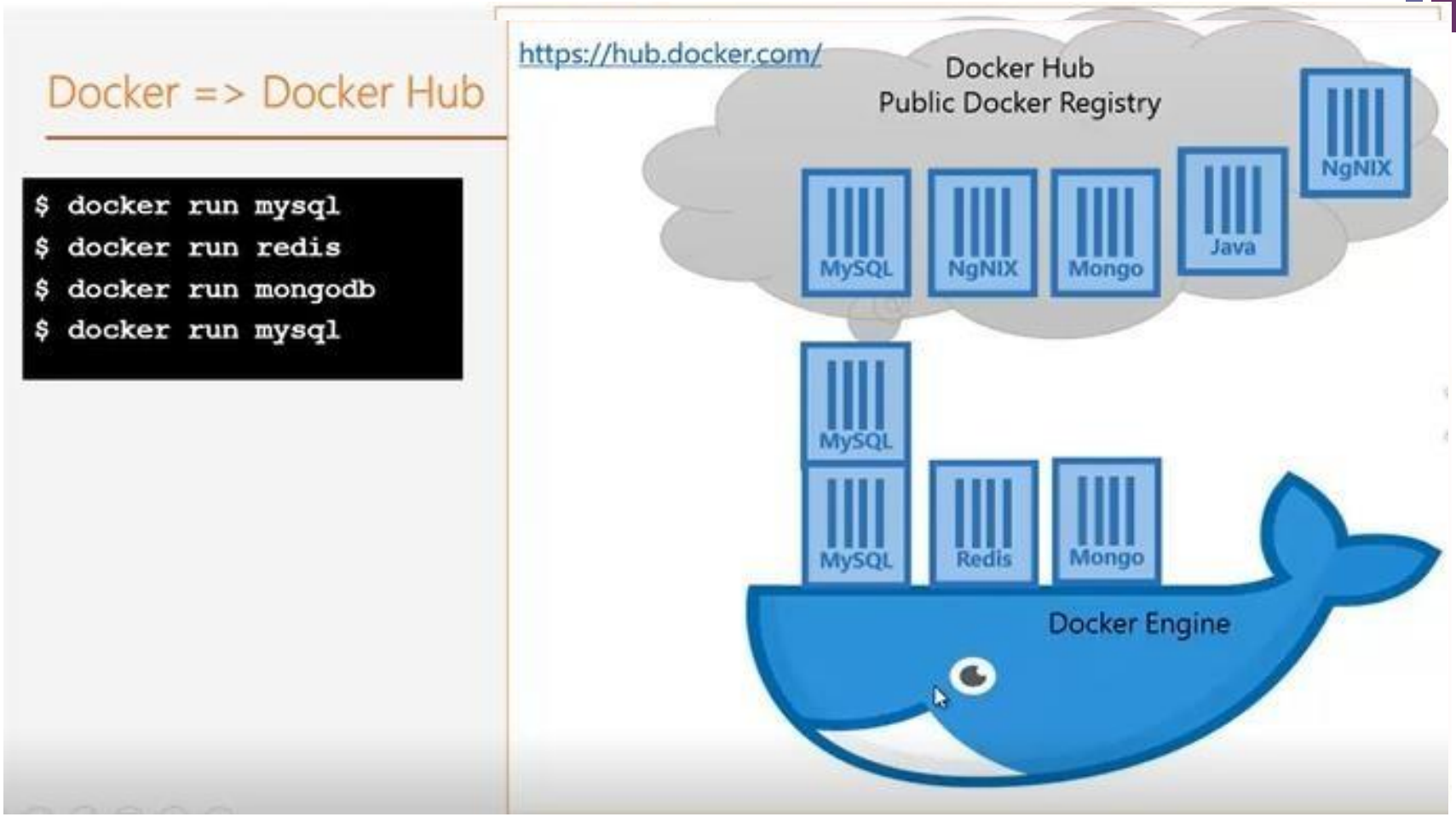


Image vs. Container

- Une image Docker est juste un fichier package représentant la template des Conteneurs. Elle définit la structure du conteneur. Elle englobe l'application containérisée et l'ensemble 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.

Etapes

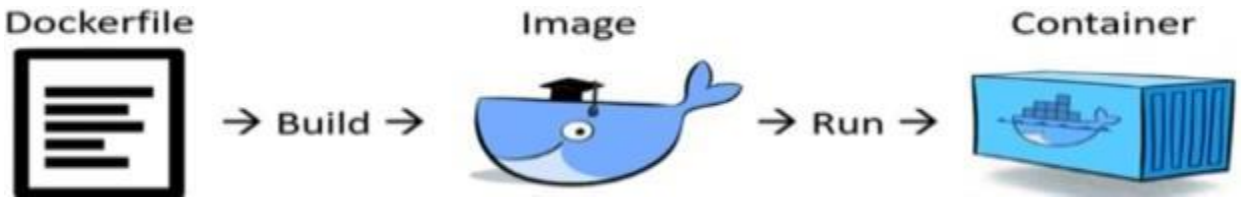
- Le développeur crée un fichier **Dockerfile** contenant les commandes que docker va exécuter pour construire une image docker de cette application.
  - La création de l'image à partir de dockerfile se fait à l'aide de $ Docker build

=> L'image docker contient tout ce dont l'application a besoin pour s'exécuter correctement.

  - Les images Docker peuvent être publiées dans un registre publique (Docker Hub) ou privé.

    $ Docker push image-name
  - Pour télécharger une image docker d'une application dans un host docker, il suffit d'utiliser:

    $ Docker pull image-name
  - La création et l'exécution d'un conteneur se fait par l'instanciation 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 de celle-ci avant d'en créer et exécuter un conteneur docker.



+ Docker File

```
FROM debian:9

RUN apt-get update -yq \
    && apt-get install curl gnupg -yq \
    && curl -sL https://deb.nodesource.com/setup_10.x | bash \
    && apt-get install nodejs -yq \
    && apt-get clean -y

ADD . /app/
WORKDIR /app
RUN npm install

EXPOSE 2368
VOLUME /app/logs

CMD npm run start
```

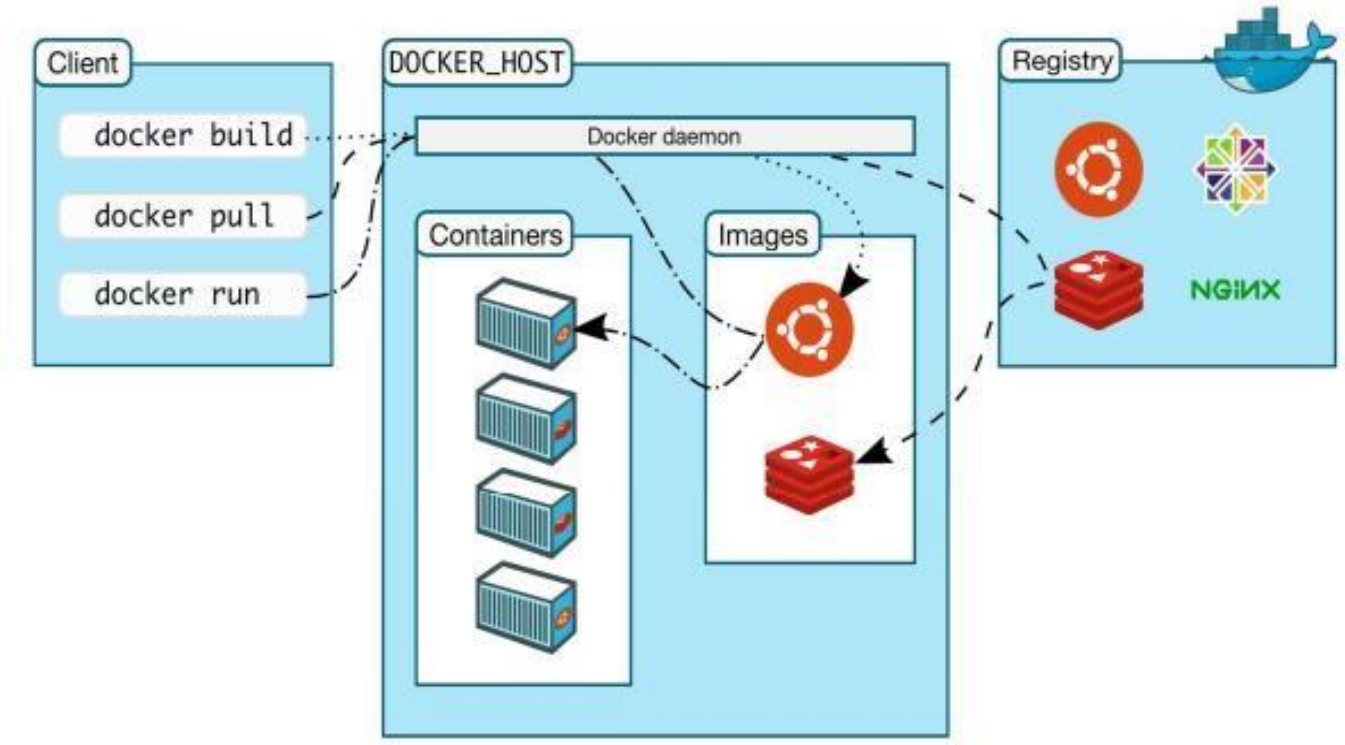
+ Docker File

- **FROM** : permet de définir l'image source ;
- **RUN** : permet d'exécuter des commandes dans votre conteneur;
- **ADD** : permet d'ajouter des fichiers dans votre conteneur ;
- **WORKDIR** : permet de définir votre répertoire de travail ;
- **EXPOSE** : permet de définir les ports d'écoute par défaut ;
- **VOLUME :** qui permet de définir les volumes utilisables ;
- **CMD** : permet de définir la commande par défaut lors de l'exécution de vos conteneurs Docker.

+ Registres Docker

❑Un registre de docker est une bibliothèque d'images. Il peut être public ou privé, et peut être sur le même serveur que le Docker Daemon ou Docker Client, ou sur un serveur totalement distinct.

❑Un registre est le composant responsable de la distribution d'un docker.



+ Docker contribue à instaurer la culture deveops

❑ 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+ descriptif des dépendances qu'il faut installer et configurer)
- **L'opérationnel :**
  - doit se débrouiller pour satisfaire les exigences de l'application,
  - pour chaque mise à jour, les mêmes histoires se répètent

=> Résultats : beaucoup de conflits entre les développeurs qui tentent d'améliorer l'application et les administrateurs qui doivent redéployer les mises à jour.

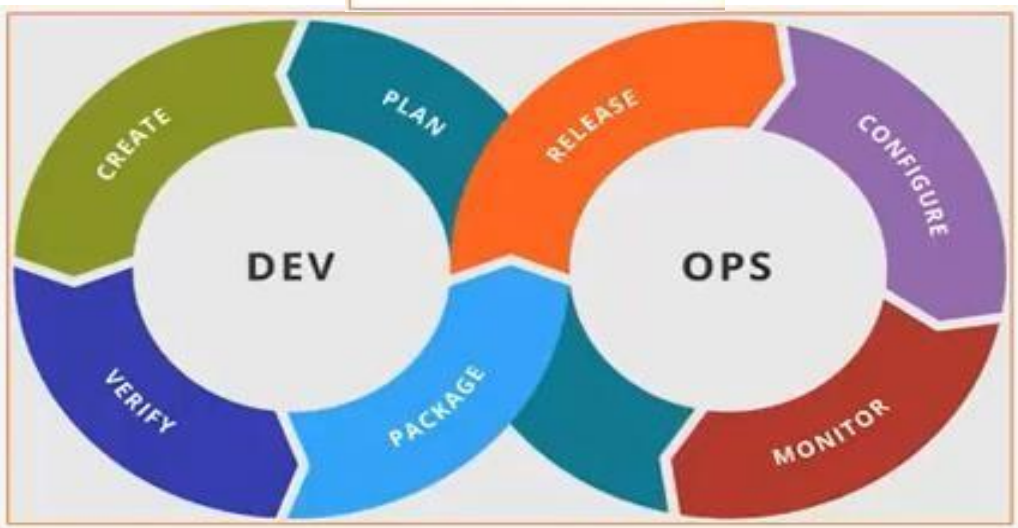
+ Docker contribue à instaurer la culture deveops

❑ 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
- **L'opérationnel :**

-déploie l'application en instanciant des conteneurs à partir de l'image docker récupérée à partir de repository docker

+ Docker contribue à instaurer la culture deveops



$ docker build $ docker run

Choisissez votre version de Docker

- Docker Inc distribue 3 versions de Docker différentes :
  - Docker Community Edition (Linux seulement) ;
  - Docker Desktop (Mac ou Windows) ;
  - Docker Enterprise (Linux seulement).
- **Docker Desktop** et **Docker Community Edition** (CE) sont deux versions de Docker **gratuites**. Avec les deux solutions, vous aurez un Docker **fonctionnel** sur votre ordinateur.
- Si vous êtes sous Windows ou macOS, utilisez Docker Desktop qui va créer pour vous l'ensemble des services nécessaires au bon fonctionnement de Docker.
- Si vous êtes sous Linux, prenez la version Community Edition (CE) ; vous utiliserez aussi cette version pour vos serveurs.

Comment fonctionne la technologie Docker ?

- La technologie Docker utilise le noyau Linux et des fonctions de ce noyau, telles que les groupes de contrôle cgroups et les espaces de noms, pour séparer les processus afin qu'ils puissent s'exécuter de façon indépendante.
- Cette indépendance reflète l'objectif des conteneurs : exécuter plusieurs processus et applications séparément les uns des autres afin d'optimiser l'utilisation de votre infrastructure tout en bénéficiant du même niveau de sécurité que celui des systèmes distincts.

Comment fonctionne la technologie Docker ?

- Les outils de conteneurs, y compris Docker, sont associés à un modèle de déploiement basé sur une image. Il est ainsi plus simple de partager une application ou un ensemble de services, avec toutes leurs dépendances, entre plusieurs environnements. Docker permet aussi d'automatiser le déploiement des applications (ou d'ensembles de processus combinés qui forment une application) au sein d'un environnement de conteneurs.

- Ces outils conçus sur des conteneurs Linux (d'où leur convivialité et leur singularité) offrent aux utilisateurs un accès sans précédent aux applications, la capacité d'accélérer le déploiement, ainsi qu'un contrôle des versions et de l'attribution des versions.

Les espaces de noms ou namespace

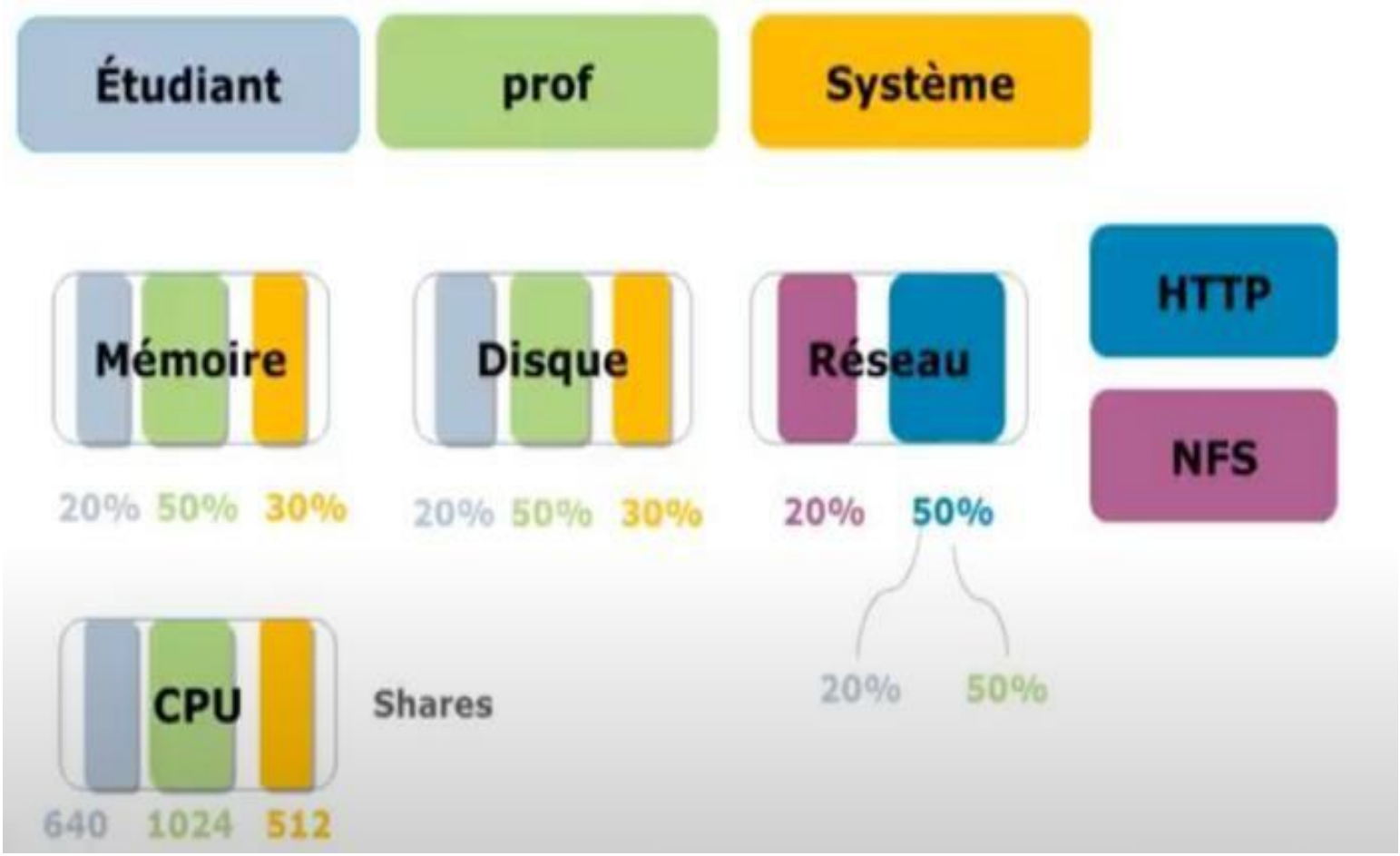
- Les namespaces sont une fonctionnalité du noyau Linux.
  - **Process namespace :** isole les processus, le conteneur dispose de sa liste des processus.
  - **Network namespace :** isole les interfaces réseau, le conteneur dispose de ses propres interfaces.
  - **Mount namespace :** isole les systèmes de fichiers.
  - **UTS namespace** : permet au conteneur de disposer de ses noms d'hotes et de domaines.
  - **IPC namespace** : ses propres processus inter-communications
  - **User namespace** : permet au conteneurs de disposer de ses utilisateurs, root avec pid et gid=0
- Commande : lsns ; Exemple : lsns - - type mnt

+ Introduction aux groupes de contrôle

- Les groupes de contrôle cgroups est une fonctionnalité du noyau linux qui a pour but de contrôler :

  - **Les ressources :** Allocation, interdiction, prioritisation
  - Surveiller et mesurer la quantité de ressources consommées.

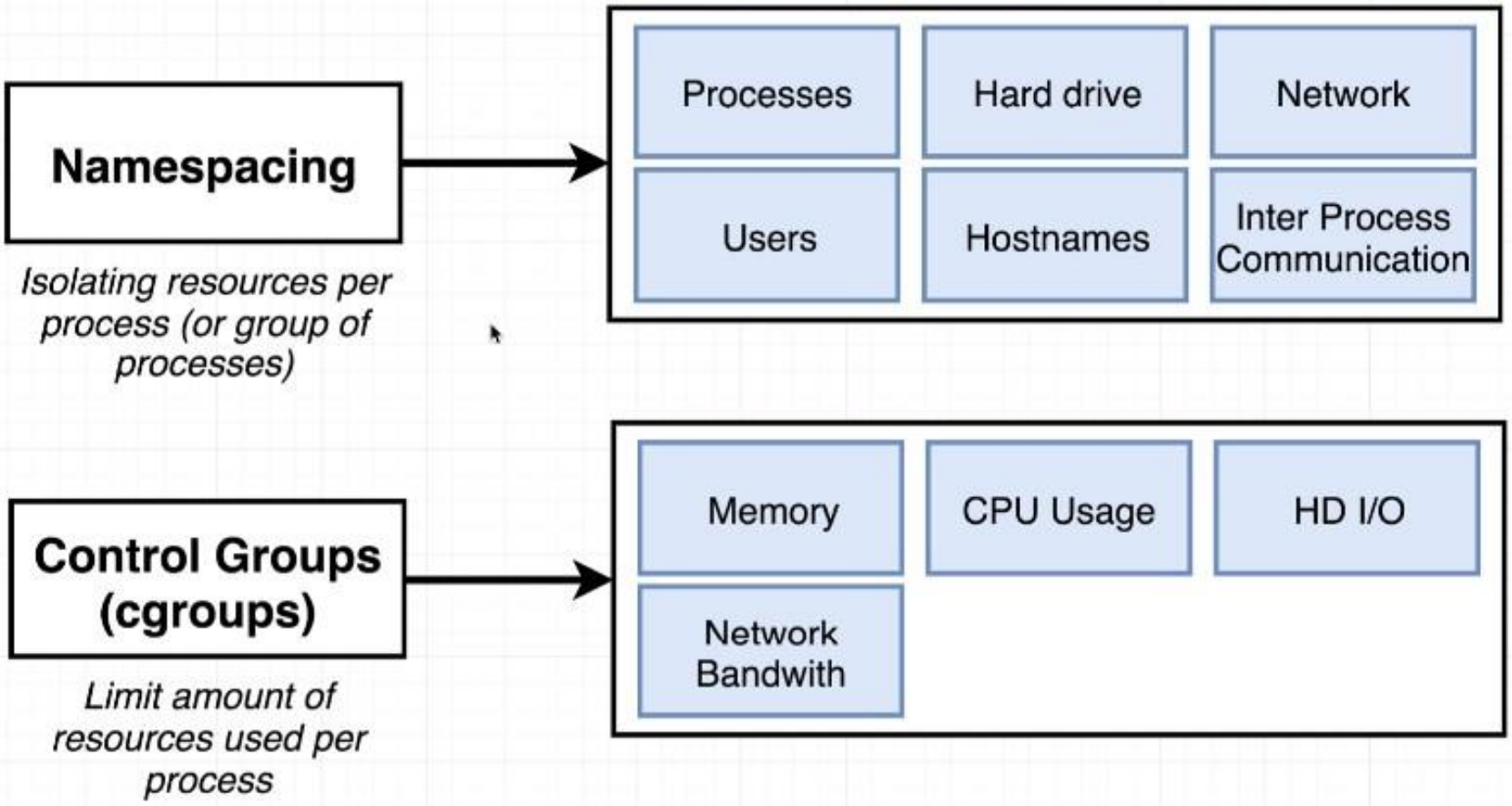
+ Introduction aux groupes de contrôle



Introduction aux groupes de contrôle

- Les cgroups sont organisés en sous-systèmes ou modules,
- Un sous système est un contrôleur de ressources.
  - **Blkio:** surveille et contrôle l'accès des taches aux entrées/sorties sur des périphériques block
  - **CPU**: planifie l'accès de la CPU
  - **Cpuset** : assigne des CPU à des taches.
  - **Devices** : autorise ou refuse l'accès aux périphériques
  - **Memory** : utilisation mémoire.

+ Résumé: cgroupes & namespaces



Merci