

Techniques de Conteneurisation

Année Universitaire
2020-2021

Plan du module

- Chapitre 1 : Concepts et Architecture
- Chapitre 2 : Installation et Configuration
- Chapitre 3: Gestion des images et des conteneurs
- Chapitre 4 : Volumes des données
- Chapitre 5 : Administration et Sécurité

Chapitre 1 : Concepts et Architecture

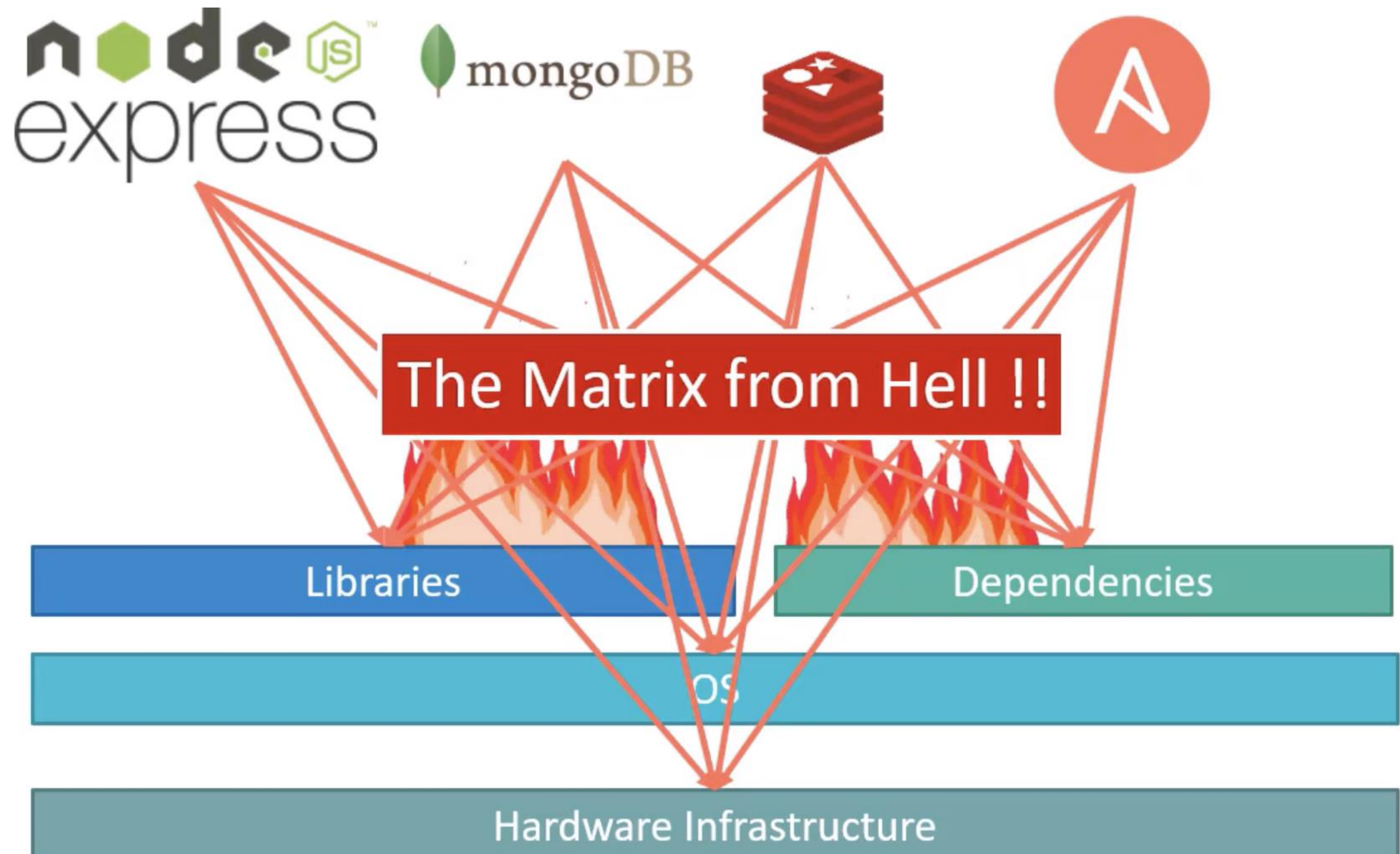
Plan

- Introduction
- Principe de conteneurisation
- Plateforme de conteneurisation : Docker

Introduction

- ❖ L'industrie du logiciel a changé.
- ❖ **Avant:**
 - ❖ Applications monolithiques
 - ❖ Cycles de développement longs
 - ❖ Mise à l'échelle lente
- ❖ **Actuellement:**
 - ❖ Services découplés
 - ❖ Améliorations rapides et itératives
 - ❖ Mise à l'échelle rapide

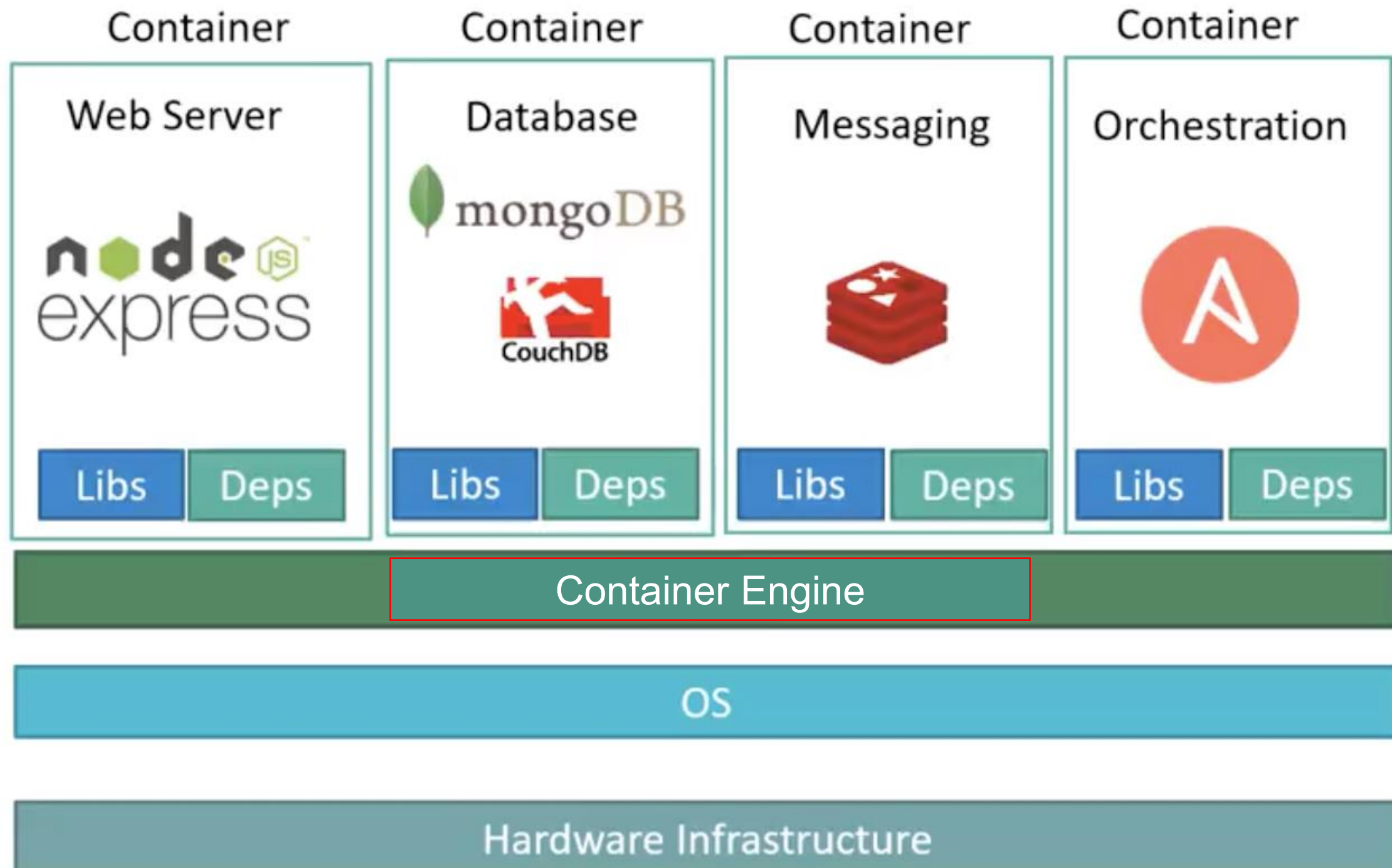
Motivation : Pourquoi des conteneurs?



Motivation

- ❖ Plusieurs défis sont envisagés tels que :
 - ❖ Compatibilité / Problèmes de dépendance.
 - ❖ Durées d'installation
 - ❖ Différence des environnements Dev / Test/ Prod.

Solution: Conteneurs



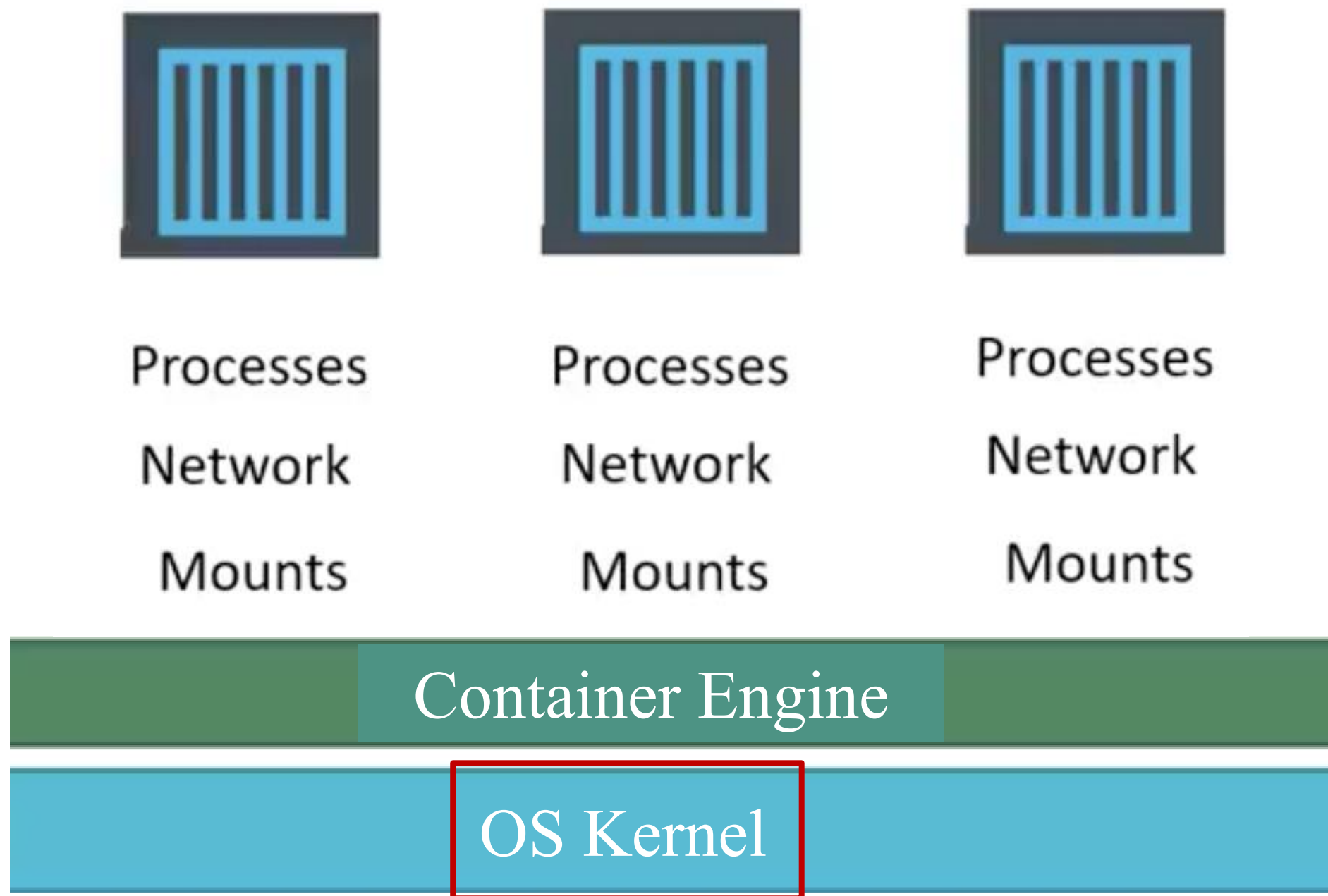
Conteneur

“A container consists of an entire runtime environment : an application, its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package.”

Source : <https://www.cio.com/>

- ❖ Chaque application est exécutée avec ses propres dépendances dans des conteneurs séparés.

Conteneurs



Systeme d'exploitation : Rappel



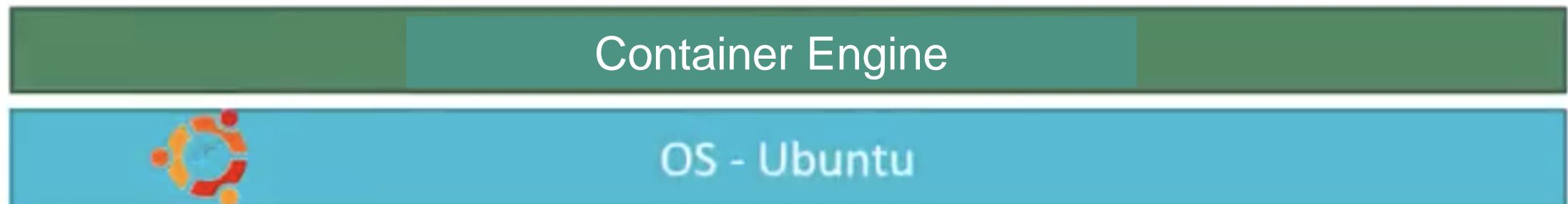
Software

Software

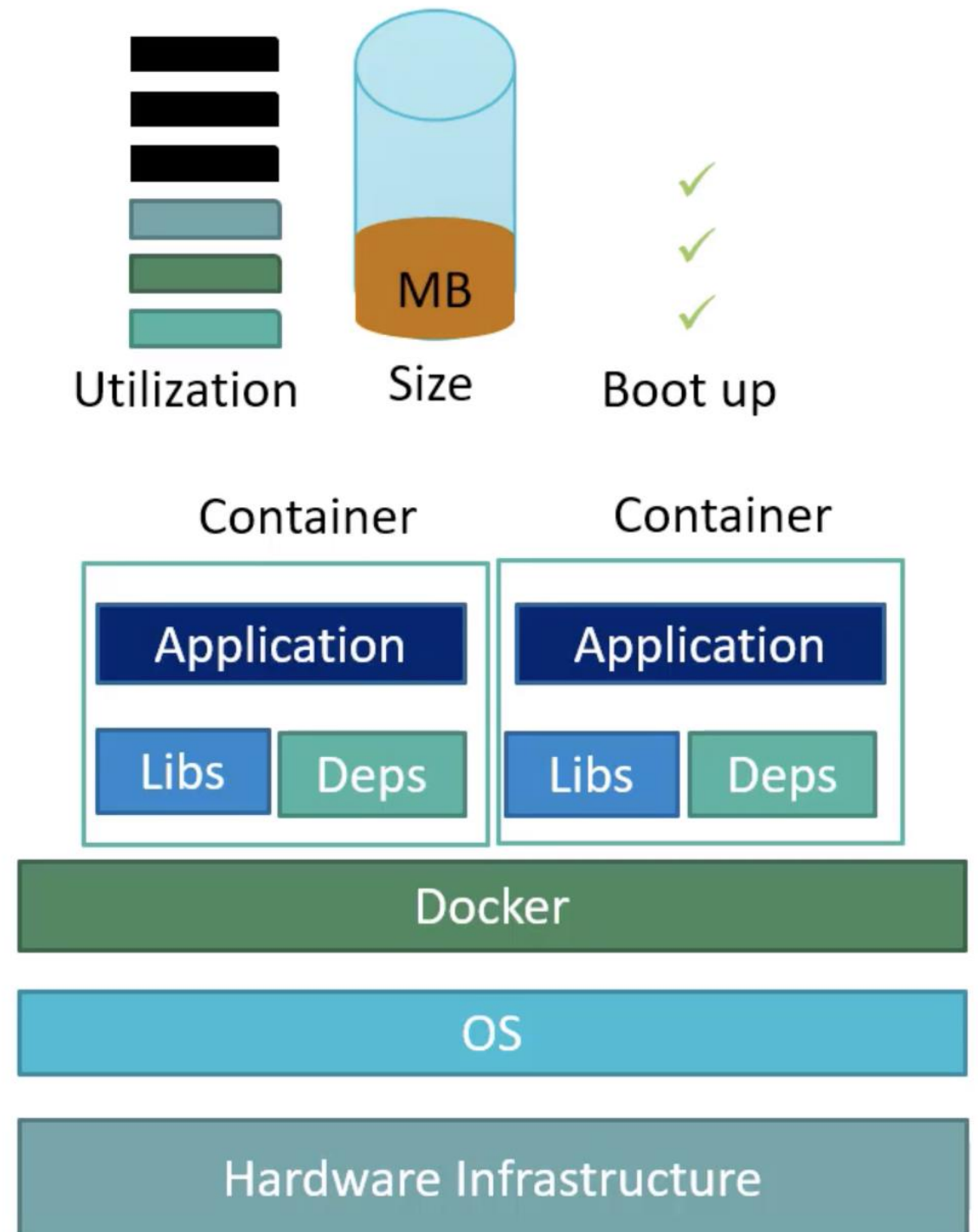
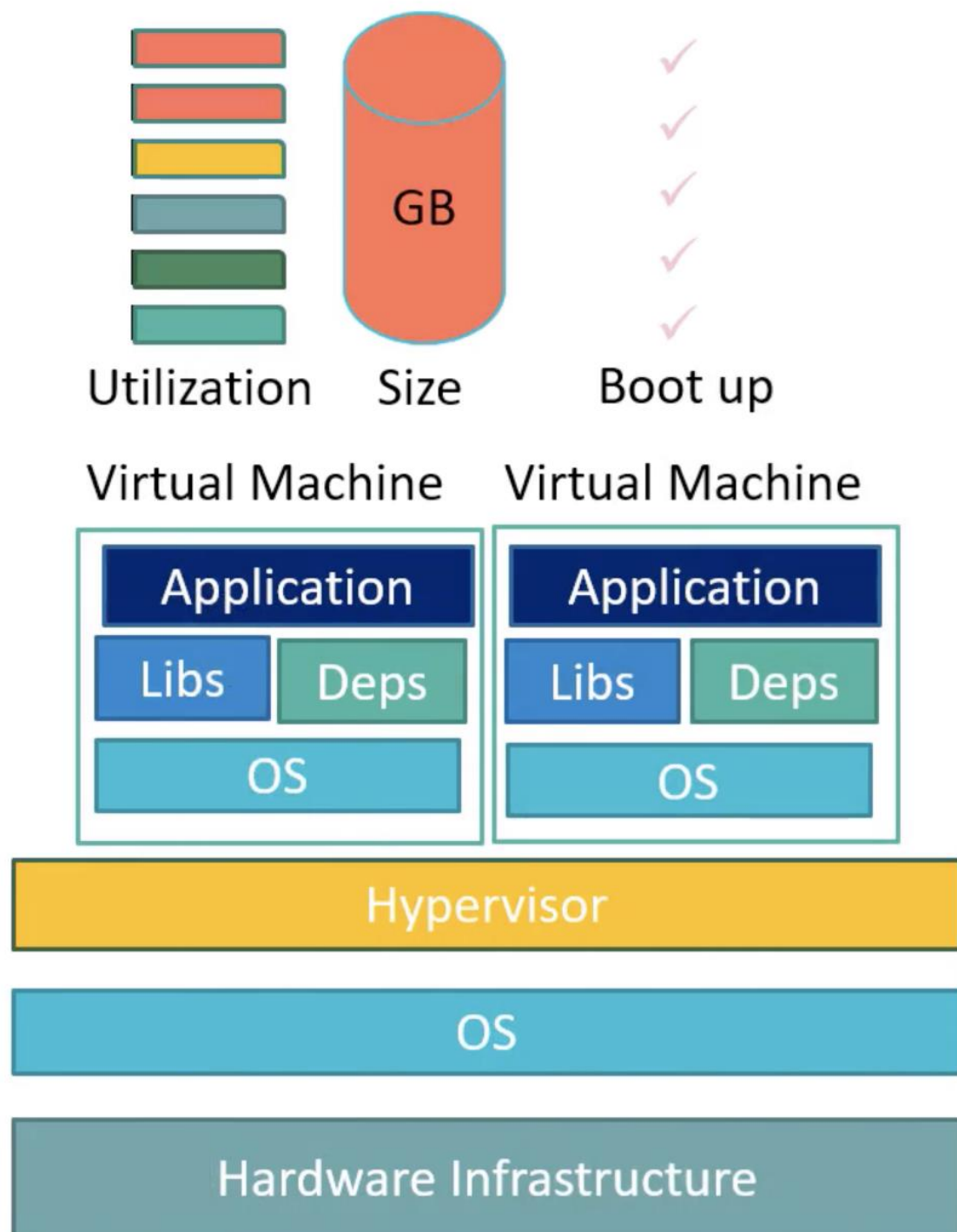
Software

OS Kernel

Conteneurs



Conteneur Vs Machine virtuelle

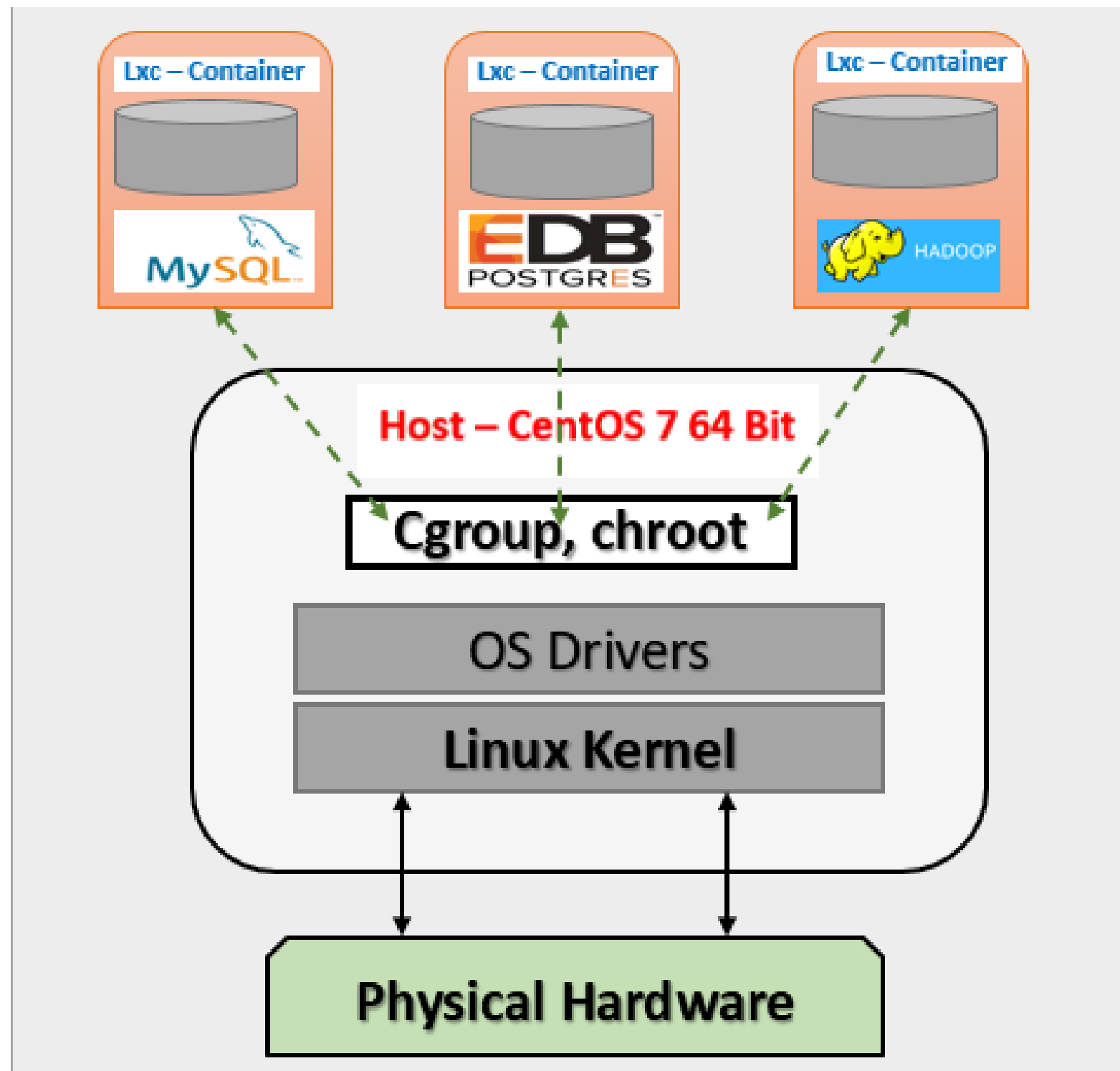


Principe de la conteneurisation

LXC « LinuX Containers »

- ❖ Le noyau de linux est doté d'un système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation pour créer des environnements virtuels appelés conteneurs.
- ❖ Les conteneurs Linux ont une isolation des systèmes de fichier, des identifiants réseau et utilisateur et également une isolation des ressources (processeur, mémoire, etc).

Principe de la conteneurisation : Architecture LXC



Principe de la conteneurisation : Namespaces

- ❖ Habituellement, tous les processus voient la même ressource et y accèdent de la même façon :
- ❖ Le hostname est le même pour tous les processus;
- ❖ Chaque processus a un PID unique ;
- ❖ La configuration réseau (IP, ports ouverts) est commune à tous les processus sur le système ;
- ❖ Tous les processus voient les mêmes montages sur les mêmes répertoires.

Principe de la conteneurisation : Namespaces

- ❖ Les **Namespaces** sont une fonctionnalité du noyau Linux.
- ❖ Les **namespaces** permettent de créer des groupes de processus isolés du reste de la machine.
- ❖ Les namespaces offrent des "*visions*" différentes des ressources (file system, network, ID, etc...) selon le processus qui en fait la demande.

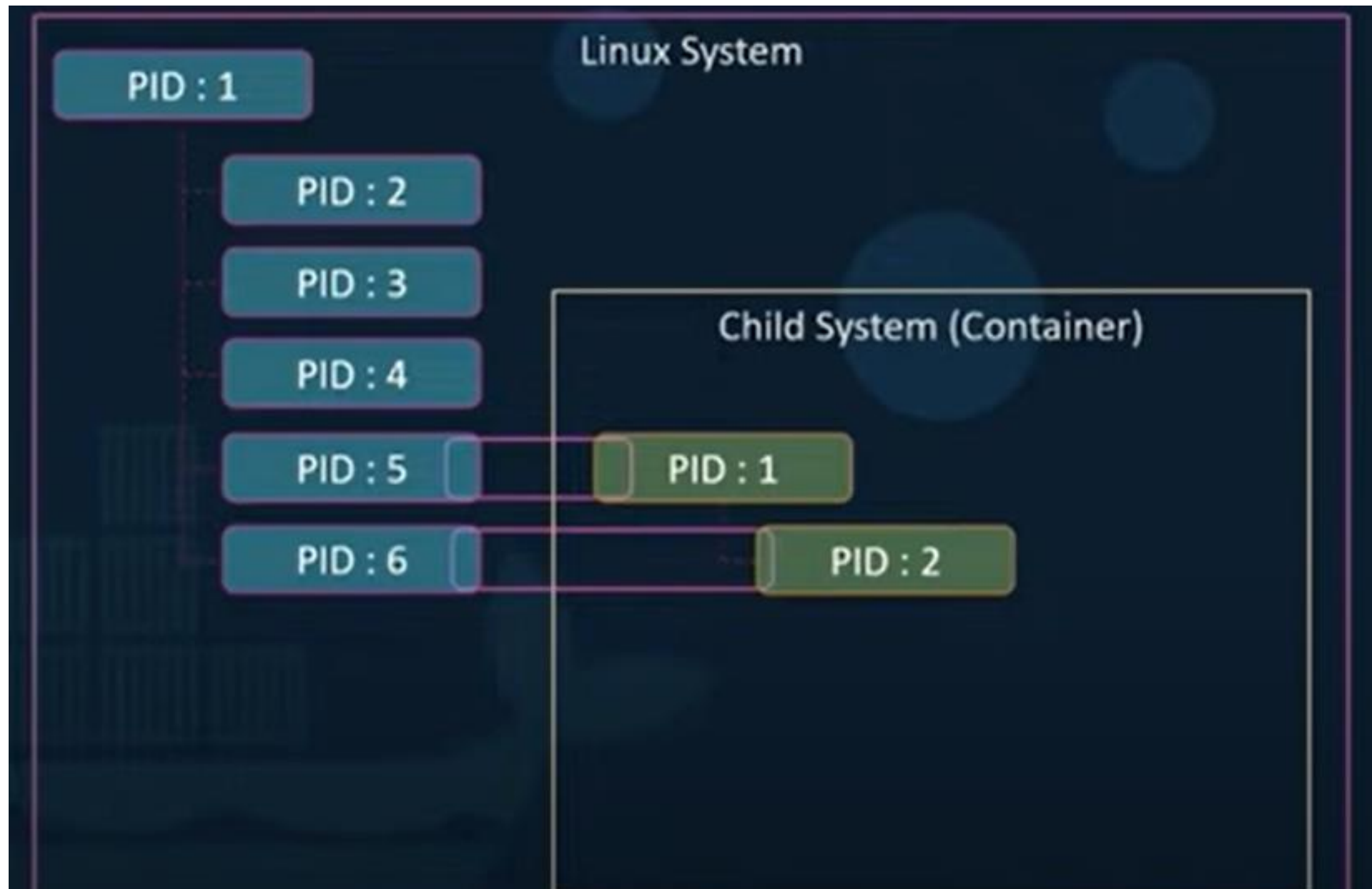
Principe de la conteneurisation : Namespaces

- ❖ Dans le noyau Linux, il existe différents types de namespaces.
- ❖ **User Namespace** : chaque container comprendra ses propres utilisateurs, ainsi qu'un utilisateur root.
- ❖ **PID Namespace**, chaque conteneur a ses propres id de processus, le PID namespace attribue un ensemble de PID aux processus qui sont indépendants de l'ensemble de PID dans d'autres namespaces.
- ❖ **Network Namespace**, chaque conteneur peut avoir sa propre interface réseau, son ip, ses règles de filtrage.

Principe de la conteneurisation : Namespaces

- ❖ **Mount Namespace** : chaque conteneur comprendra son propre système de fichier et ainsi sa propre arborescence de fichier non visible par les autres containers
- ❖ **UNIX Time-Sharing (UTS) Namespace** : permet à un seul système d'apparaître comme ayant différents host names pour différents processus (containers) .
- ❖ **Interprocess communication (IPC) Namespace** : permet d'isoler les communications inter-processus.

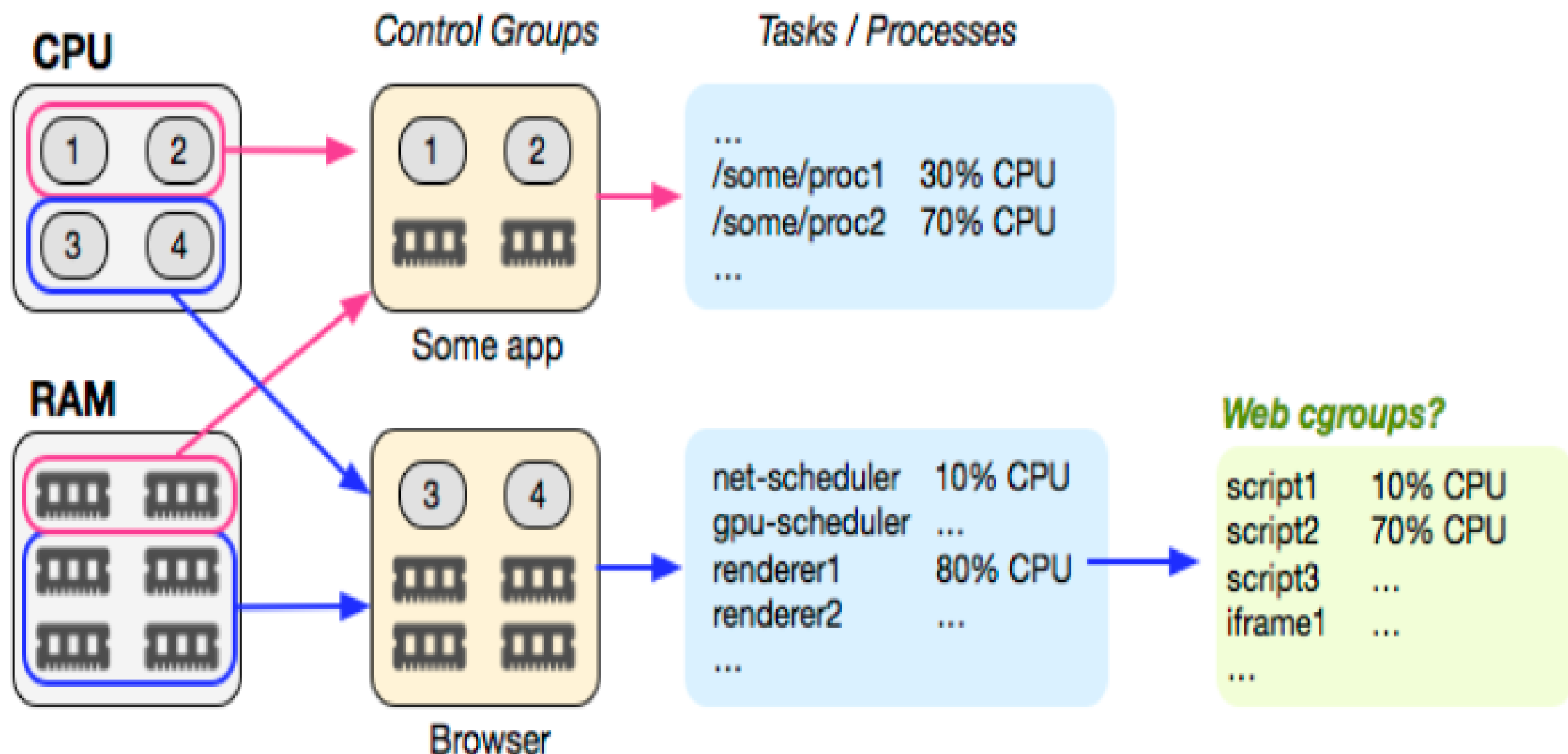
Principe de la conteneurisation : PID Namespaces



Principe de la conteneurisation : Cgroups

- ❖ Les **Cgroups** ou Control Groups, est une fonctionnalité du noyau Linux pour limiter, compter et isoler l'utilisation des ressources.
- ❖ Cgroups fournit :
 - ❖ **Limitation des ressources** : des groupes peuvent être mis en place afin de ne pas dépasser une limite de mémoire.
 - ❖ **Priorisation** : certains groupes peuvent obtenir une plus grande part de ressources processeur ou de bande passante d'entrée-sortie.
 - ❖ **Comptabilité** : permet de mesurer la quantité de ressources consommées par certains systèmes en vue de leur facturation par exemple.
 - ❖ **Isolation** : séparation par espace de nommage pour les groupes, afin qu'ils ne puissent pas voir les processus des autres, leurs connexions réseaux ou leurs fichiers.
 - ❖ **Contrôle** : figer les groupes ou créer un point de sauvegarde et redémarrer.

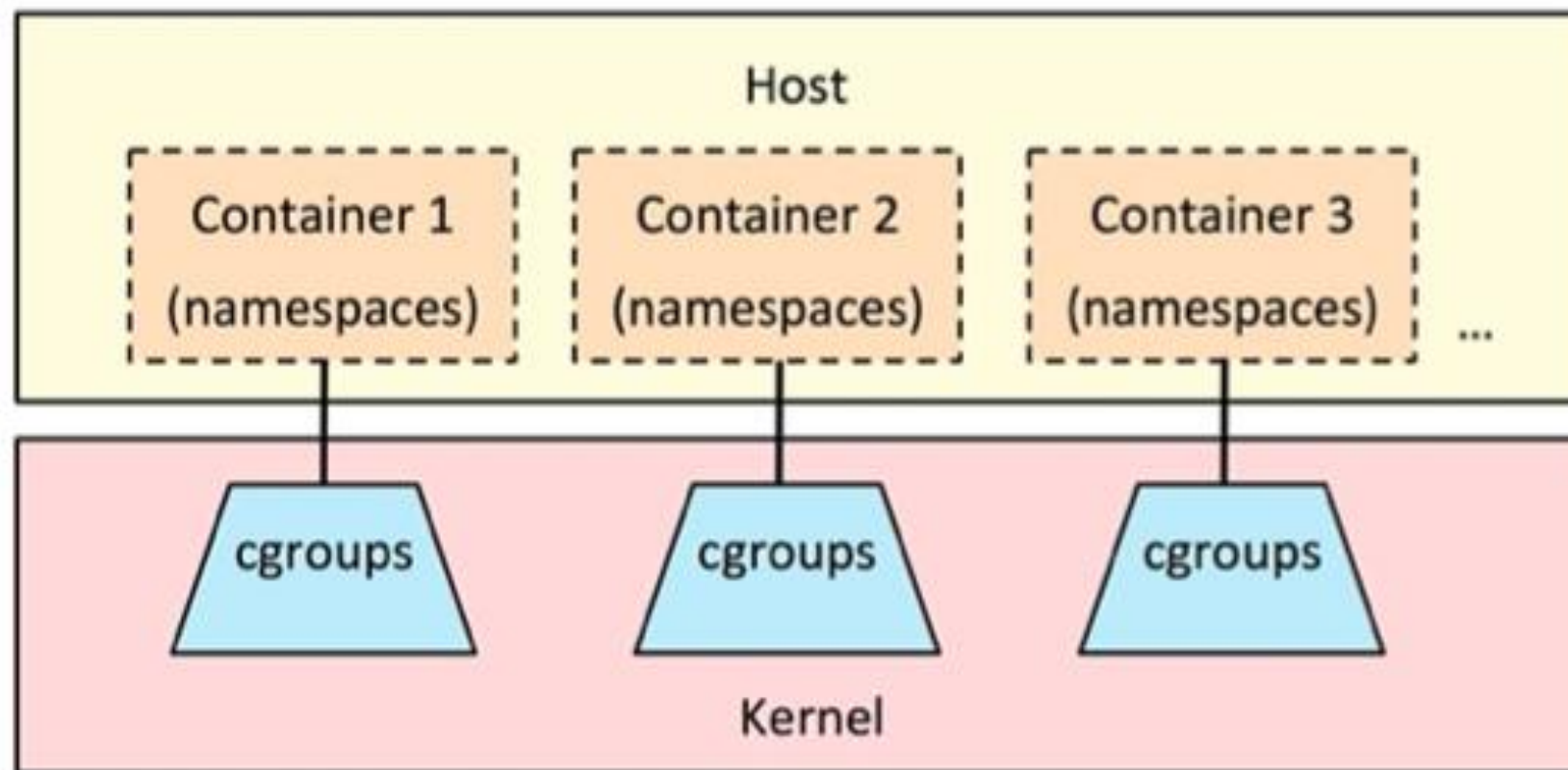
Principe de la conteneurisation : Cgroups



Principe de la conteneurisation

Linux Containers

Container = combination of namespaces & cgroups

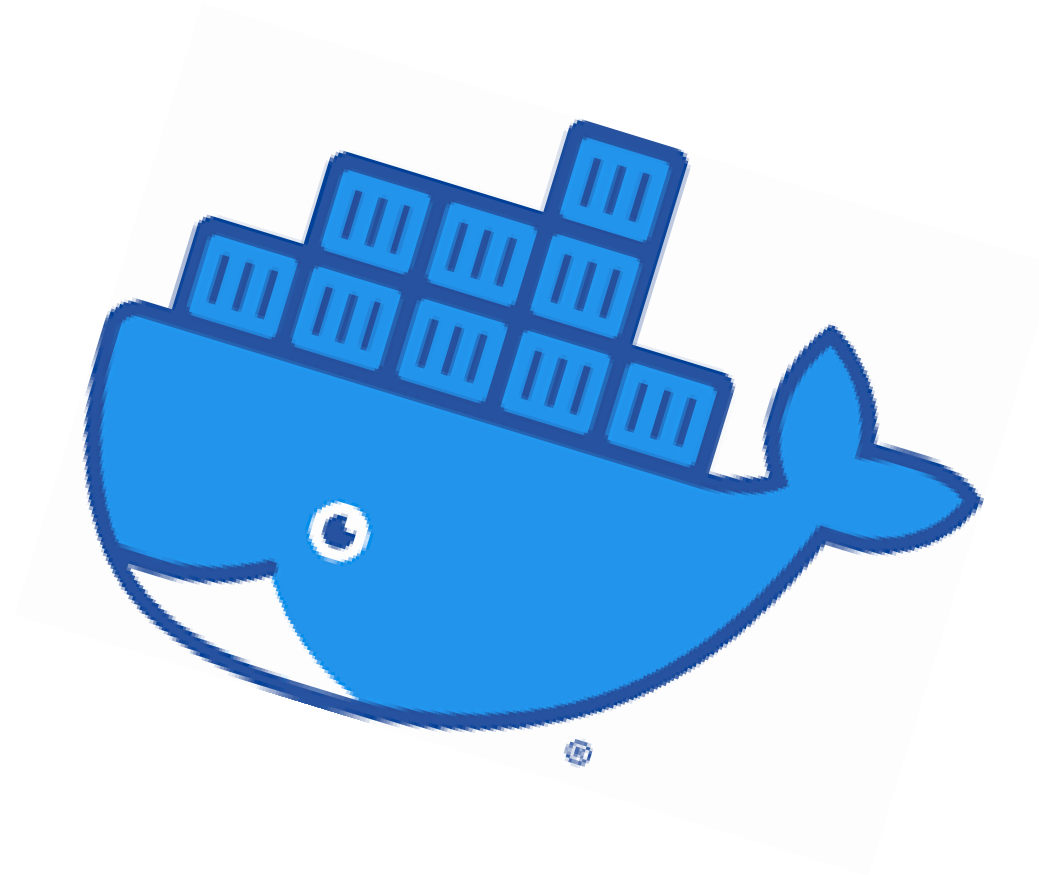


Avantages de la conteneurisation

- ❖ La conteneurisation est de plus en plus populaire car les conteneurs sont:
- ❖ **Flexible:** même les applications les plus complexes peuvent être conteneurisées.
- ❖ **Léger:** les conteneurs exploitent et partagent le noyau hôte.
- ❖ **Interchangeable:** vous pouvez déployer des mises à jour à la volée
- ❖ **Portable:** vous pouvez créer localement, déployer sur le cloud et exécuter n'importe où votre application.
- ❖ **Évolutif:** vous pouvez augmenter et distribuer automatiquement les répliques (les clones) de conteneur.

Plateforme de conteneurisation : Docker

- ❖ Docker est une plate-forme ouverte permettant d'empaqueter une application et ses dépendances dans un conteneur isolé qui pourra être exécuté sur n'importe quel serveur.



Plateforme de conteneurisation : Docker

- ❖ Un Gestionnaire de Conteneur :
 - ❖ Virtualisation légère : les systèmes hôte et invité partagent le même noyau.
 - ❖ Basé sur les **Namespaces** Linux et les **Cgroups**.
 - ❖ Massivement Copy-on-Write (COW) :
 - ❖ Déploiement instantané
 - ❖ Adapté aux micro-services

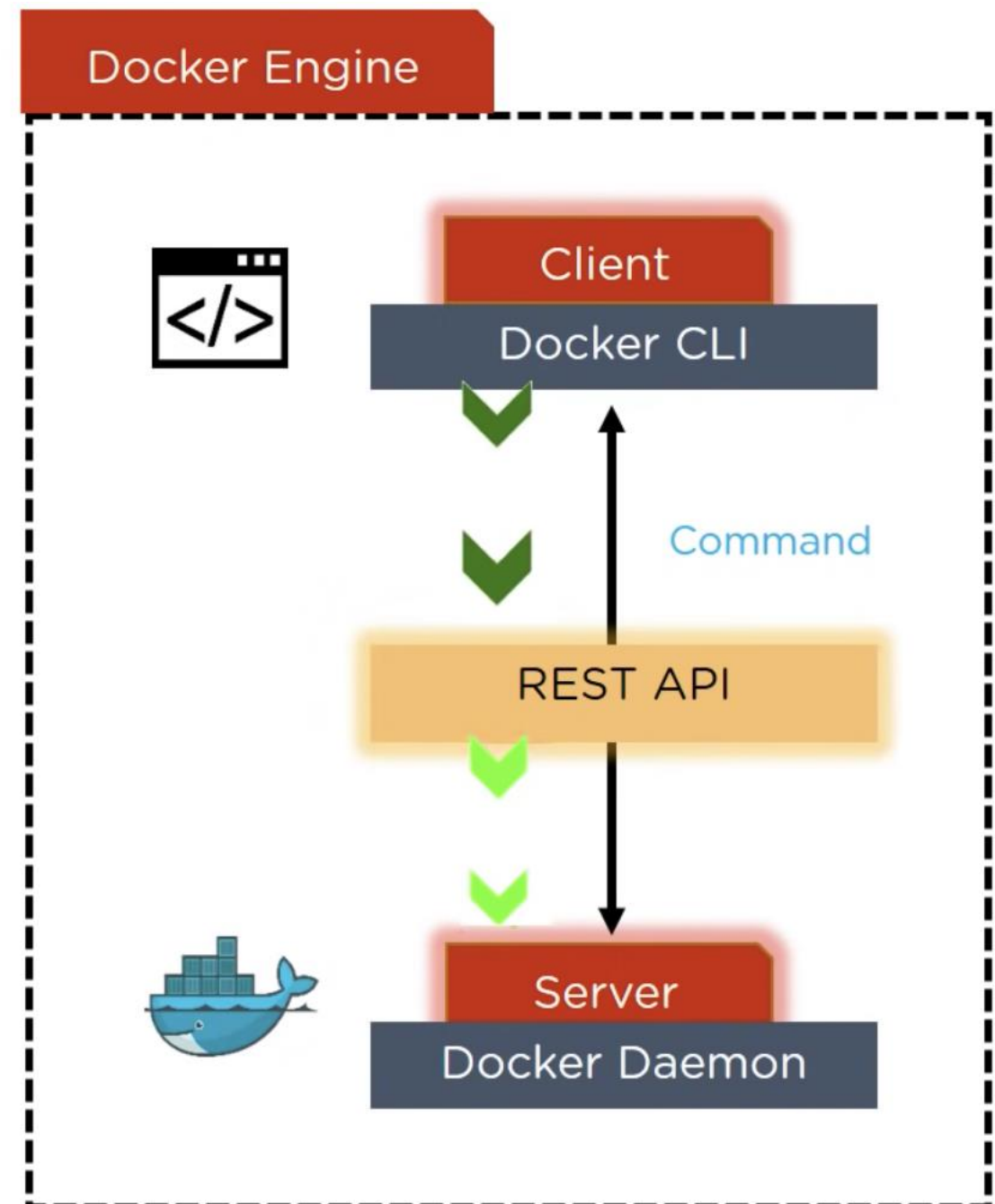
Plateforme de conteneurisation : Docker

Les origines du projet Docker

- ❖ **dotCloud** utilisait un **PaaS** à l'aide d'un moteur de conteneur personnalisé.
- ❖ Ce moteur était basé sur OpenVZ (et plus tard, LXC) et AUFS.
- ❖ Cela a commencé (vers 2008) en tant que script Python unique.
- ❖ En 2012, le moteur avait plusieurs composants (~ 10) Python. (et ~ 100 autres micro-services!)
 - ❖ Fin 2012, dotCloud lance moteur de conteneur.
 - ❖ Le nom de code pour ce projet est "**Docker**"

Architecture Docker

- ❖ L'architecture de Docker est basée sur une architecture de type *client-serveur*.
- ❖ **Client** : Docker CLI (Linux), Remote API (Mac, Windows)
- ❖ **Server** : Docker Daemon.
- ❖ **Docker Engine** = Client (CLI) + Rest API + Docker server.



Architecture Docker

Les composants du Docker

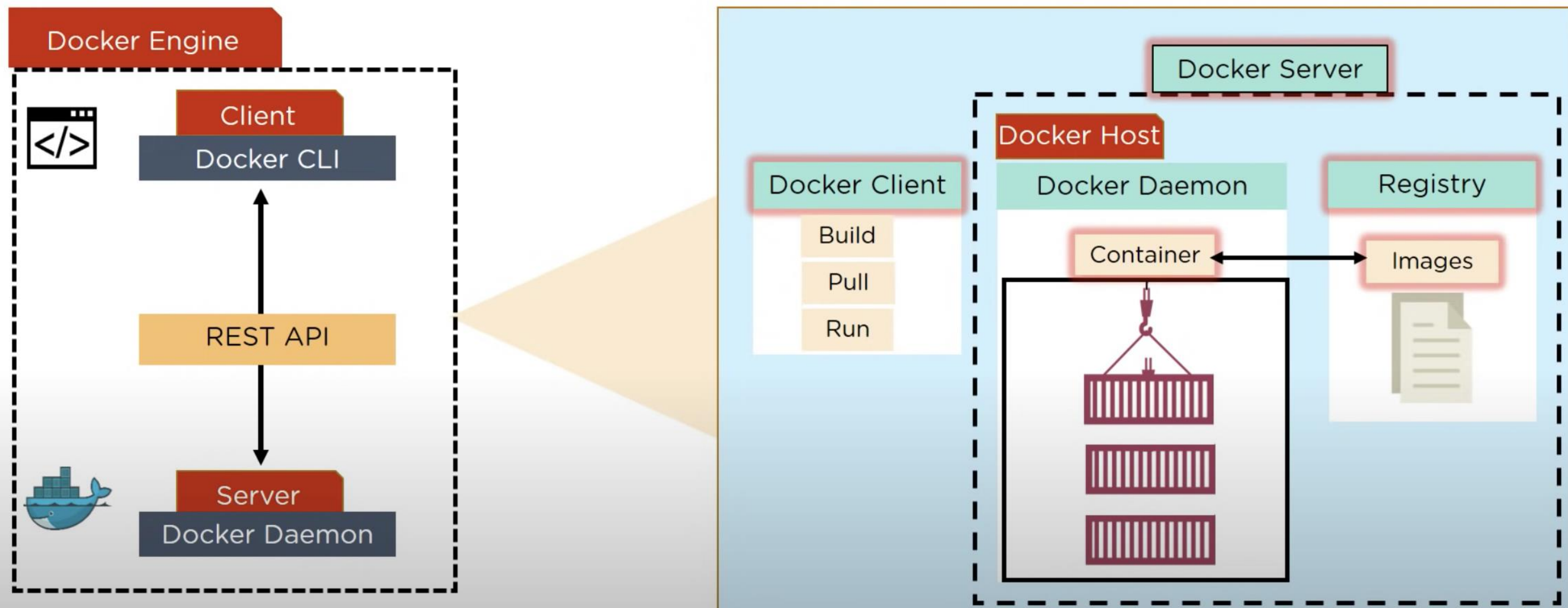
- ❖ Le **Docker Engine** est une partie fondamentale de la plateforme Docker. C'est lui qui permet notamment d'exécuter un conteneur, de construire une Docker image, ou encore de partager ses Docker images avec d'autres via Docker Hub.
- ❖ Un **outil de ligne de commande** qui a le rôle du client
- ❖ Une **API REST** qui gère les interfaces que les programmes utilisent pour communiquer avec le Docker daemon
- ❖ Un serveur appelé **Docker daemon**

Architecture Docker

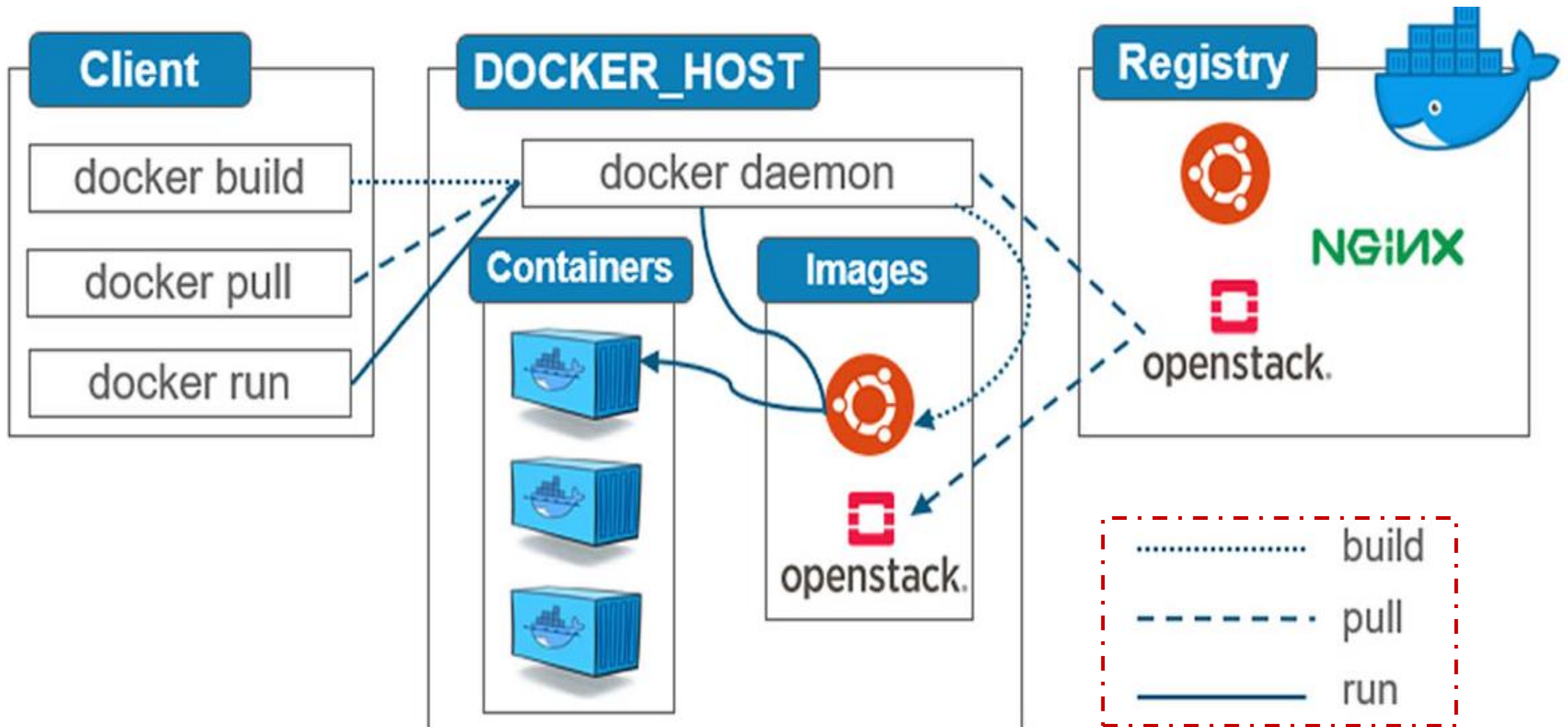
Les composants du Docker

- ❖ **Client/server** : outil utilisant l'API du serveur/Daemon
- ❖ **Index** : répertoire public (<https://index.docker.io/>)
- ❖ **Image** : conteneur en lecture seule (Template, Snapshot)
- ❖ **Conteneur** : élément manipulable (instance)

Architecture Docker

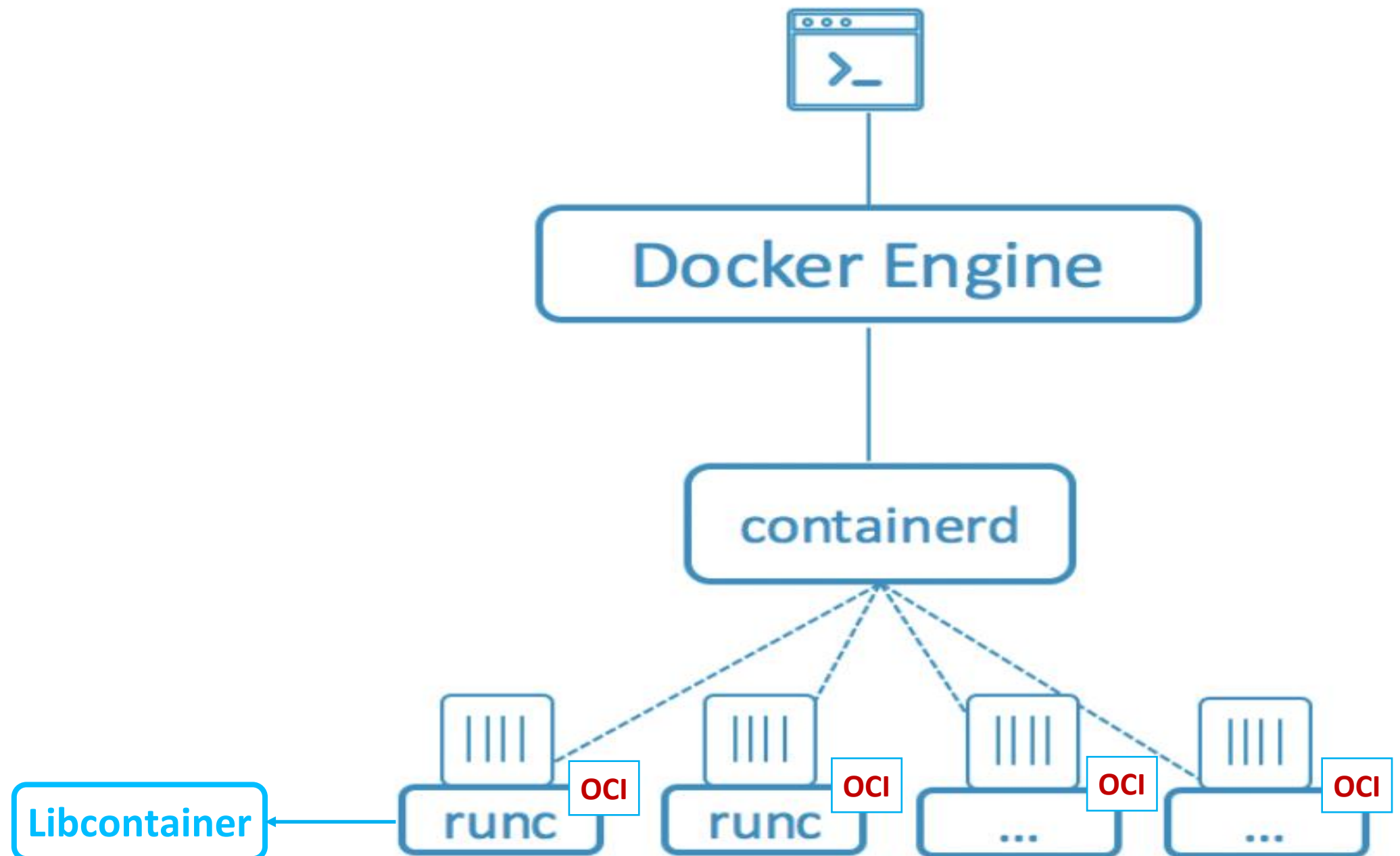


Architecture Docker



Container Runtime

La runtime est responsable de créer et de faire fonctionner le conteneur



Container Runtime: containerd

❖ Containerd

- ❖ Un moteur d'exécution de conteneur (container runtime) de haut niveau
- ❖ Un daemon lancé par docker daemon
- ❖ Une abstraction des fonctionnalités du noyau qui fournit une interface de conteneur. Il permet de gérer les conteneurs (Push, Pull...).

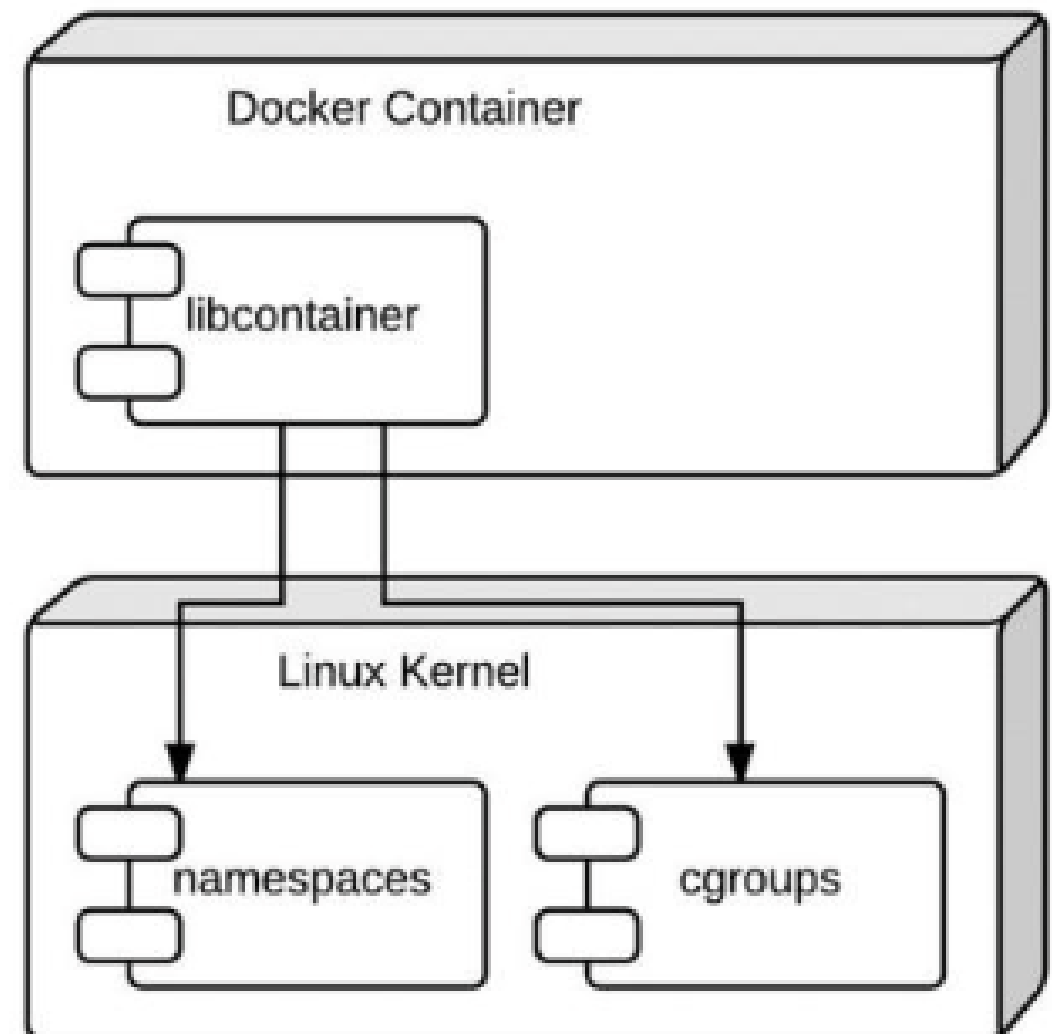
Container Runtime:runc

❖ Runc:

- ❖ Un moteur d'exécution de conteneur (container runtime) universel et léger de bas niveau
- ❖ Un outil en ligne de commande permettant de créer et d'exécuter des conteneurs en utilisant libcontainer
- ❖ L'implémentation de la specification OCI
- ❖ OCI (**Open Container Initiative**): est l'organe chargé de définir les normes relatives aux conteneurs (à quoi doit ressembler un conteneur...)

Libcontainer

- ❖ **Libcontainer** est une bibliothèque de pilotes permettant de lier directement, sans autre dépendance, un conteneur Docker avec le noyau Linux et ses composants.



Libcontainer

- ❖ Libcontainer est une abstraction qui prend en charge une gamme plus large de technologies d'isolation.
- ❖ Avec la bibliothèque libcontainer, runc peut manipuler les Namespace , les cgroups, les interfaces réseau et les règles de pare-feu sans s'appuyer sur LXC et d'autres packages externes.

Docker Container

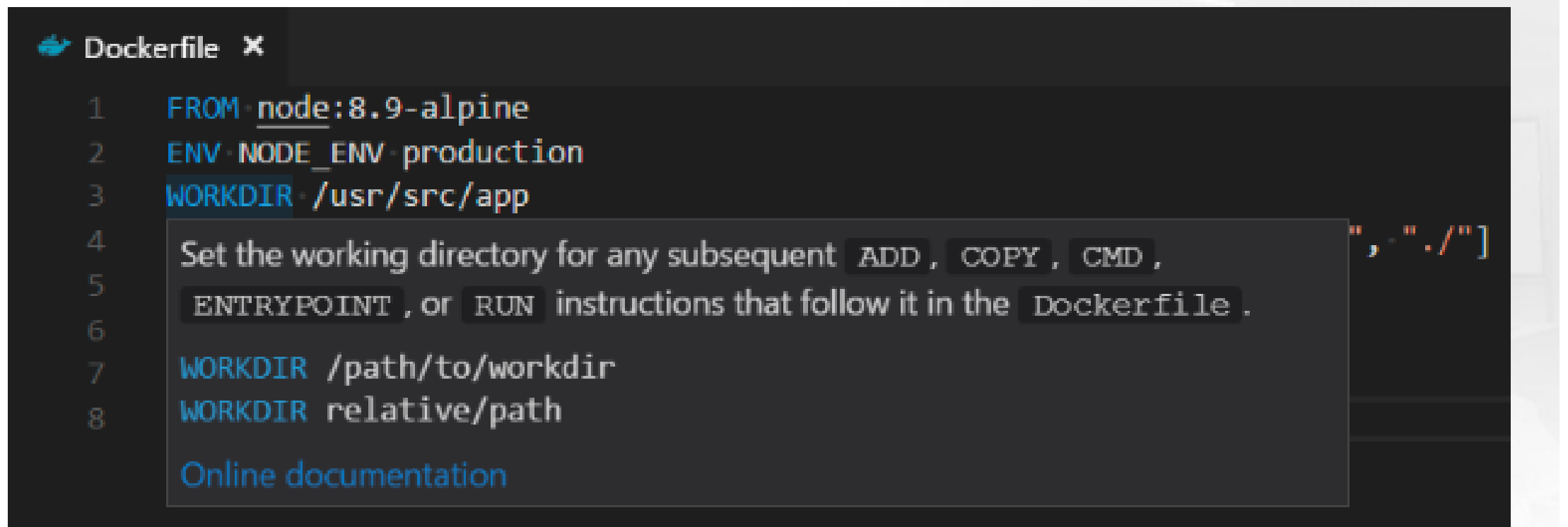
- ❖ **Un conteneur Docker** est une instance exécutable d'une **Docker image**.
- ❖ Différentes actions peuvent être effectuées sur un conteneur à partir de l'outil de ligne de commande de Docker ou d'une API :
 - ❖ Exécuter un conteneur
 - ❖ Arrêter un conteneur
 - ❖ Déplacer un conteneur
 - ❖ Supprimer un conteneur
- ❖ Chaque conteneur est une plateforme d'applications indépendante et sécurisée. à partir de laquelle on ne peut pas accéder à des ressources présentes dans un autre conteneur..

Docker Image

- ❖ Une **image Docker** est un modèle permettant de créer un conteneur. Par exemple, une image peut contenir un système d'exploitation Ubuntu avec un serveur web et une application web.
- ❖ Il est possible de créer des images ou de télécharger et d'utiliser des images créées et mises à disposition par d'autres personnes dans Docker Registry.

Docker Image : Dockerfile

- ❖ Un **Dockerfile** est un fichier de configuration contenant plusieurs instructions permettant de construire une Docker image. Les instructions concernent plusieurs tâches comme la création d'un répertoire, la copie de fichiers, etc.



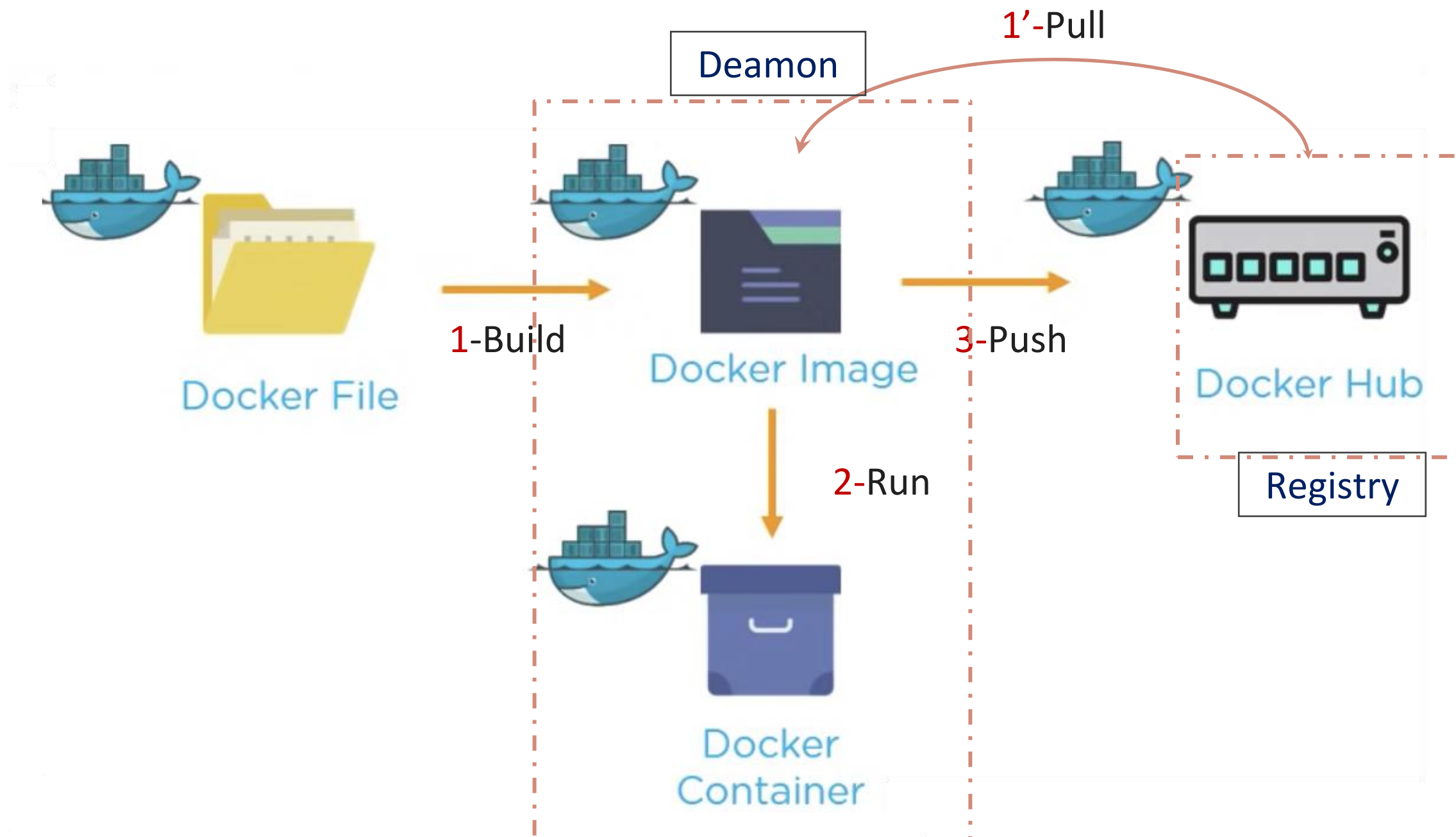
```
Dockerfile X
1 FROM node:8.9-alpine
2 ENV NODE_ENV=production
3 WORKDIR /usr/src/app
4
5 Set the working directory for any subsequent ADD, COPY, CMD,
6 ENTRYPOINT, or RUN instructions that follow it in the Dockerfile.
7 WORKDIR /path/to/workdir
8 WORKDIR relative/path
Online documentation
```


Docker Registry

- ❖ Un **Docker Registry** est une bibliothèque de Docker images. Il peut être public ou privé
- ❖ Le Docker Registry public géré par Docker s'appelle **Docker Hub** ; il contient les images officielles et permet aux développeurs de stocker et de partager des images qu'ils ont créé

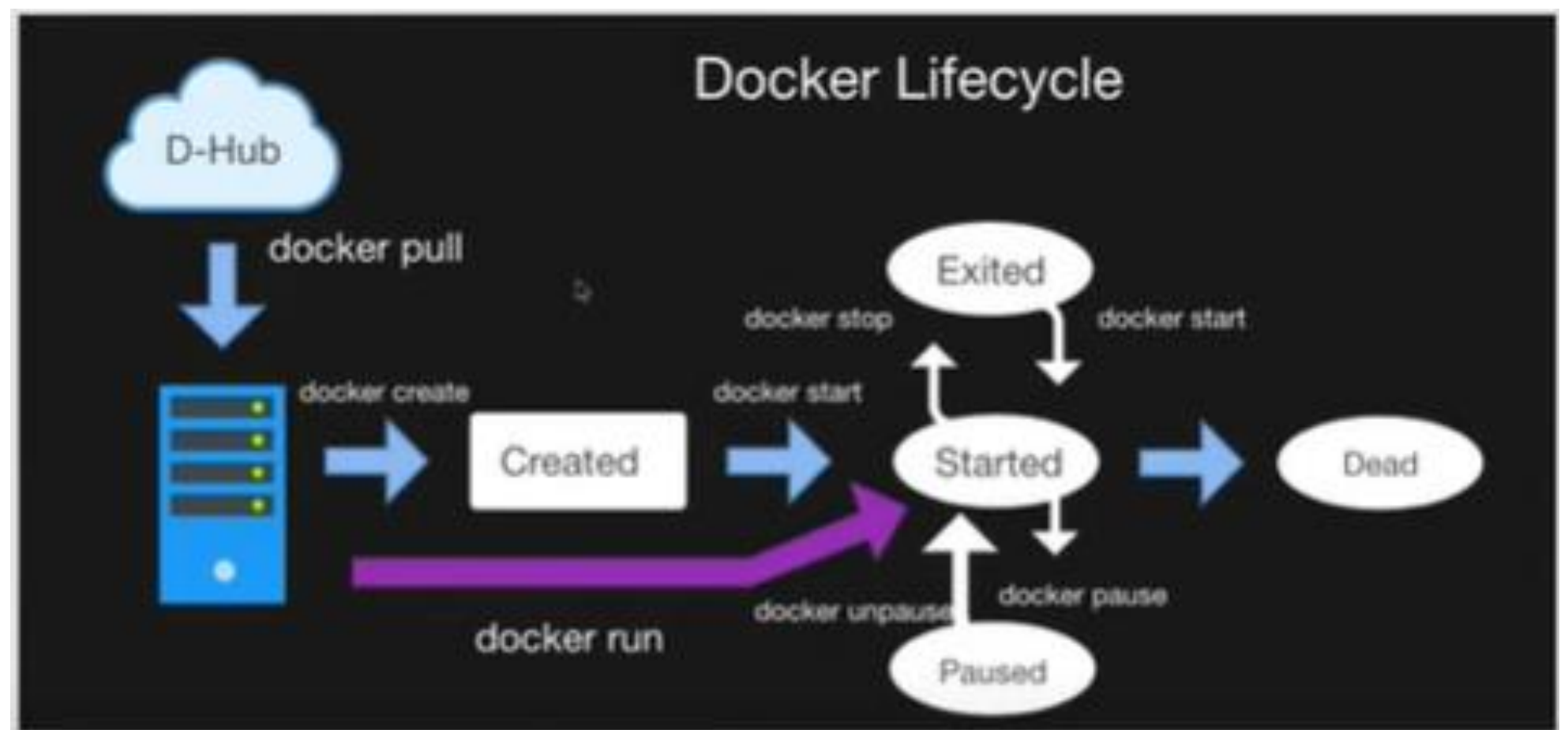


Interaction du docker avec Docker Hub



Cycle de vie d'un conteneur

- ❖ Un **conteneur** utilise le noyau Linux de la machine hôte et doit être composé de tous les fichiers liés à la création d'une image, ainsi que des métadonnées concernant le conteneur lorsque celui-ci est créé ou exécuté



Docker : Conteneur vs Image

