MODULE :
Le Cloud Computing & DevOps

Pr. F. Benabbou
Master DSBD
Faculté des Sciences Ben M'Sik Casablanca

TABLE OF CONTENTS

01

LE CLOUD COMPUTING

Virtualisation des applications

- Généralement les applications sont installées sur un poste de travail
- Mais il y a des Problèmes liés à une installation classique :
  - Non compatibilité entre différentes applications
  - Parfois impossibilité de faire cohabiter plusieurs versions de la même application
  - Problème de conflits de DLL (cas Windows : version de DLLs systèmes propre à une application)
  - Lenteur du déploiement des applications (installation/désinstallation),…
  - Impossibilité de les migrer d'un host vers un autre
  - Etc.

Virtualisation des application

- La virtualisation d'application est une solution qui consiste à séparer l'utilisation d'une application de son environnement matériels et logiciels d'exécution.
- Le but de la virtualisation d'applications est de réduire la complexité liée à la gestion et au déploiement des applications au sein d'un SI.

Avant Virtualisation

Après Virtualisation

Virtualisation des applications

- Plusieurs méthodes de virtualisation d’applications existent, on peut les grouper en 4 méthodes
  - Virtualisation de l’application en utilisant une machine virtuelle qui s ’exécute sur le hôte.
  - Virtualisation de l'application dans un espace utilisateur sur le système d'exploitation hôte
  - La virtualisation d'application basée sur l'isolation de processus ou conteneurs
  - Virtualisation d'application basée sur le cloud

Virtualisation des App. en utilisant des MVs

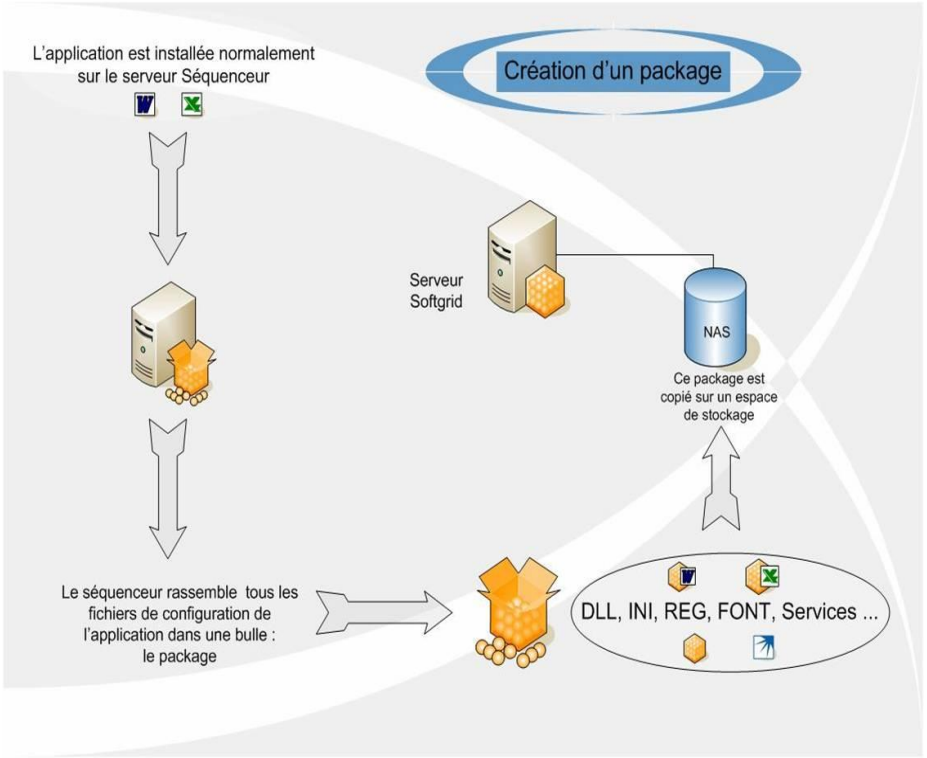
- Cette approche consiste à exécuter l'application sur une VM qui est elle-même exécutée sur le système d'exploitation hôte.
- Dans ce cas c’est la VM qui fournit un environnement d'exécution isolé pour l'application, qui est séparé du système d'exploitation hôte et des autres applications.
- L'application peut utiliser son propre système d'exploitation et ses propres dépendances, qui sont installés dans la VM.
- La VM fournit également une couche d'abstraction pour l'application, qui permet de la rendre portable entre les différents systèmes d'exploitation et les différentes plateformes.
- Exemple : Vmware Workstation, VirtualBox , QEMU, etc

Virtualisation des App. dans un espace utilisateur

- Elle permet d’exécuter l'application dans un espace utilisateur indépendant et isolé du système d'exploitation de l'hôte et des autres applications installées sur le même système.
- Cette approche utilise une couche d'abstraction pour virtualiser l'application, qui sera exécutée directement sur le système d'exploitation hôte.
- Elle encapsule les applications dans un package autonome qui contient tout ce dont l'application a besoin pour fonctionner.
- Permet l’exécution d’applications sur des systèmes où elles ne peuvent pas être installées nativement.

Fonctionnement

- L‘application virtuelle est créé sous forme d’un package par la capture d'instantanés du système physique ou elle est installée.
- On emballe l'application avec ses dépendances, éliminant les conflits entre applications.
- Les applications sont packagées sur un serveur et proposées à la demande.
- l’application est exécutée par le poste de travail, avec ses propres ressources.



App-V Application Virtualization Sequencer

Virtualisation des App. dans un espace utilisateur

- Turbo Studio, VMware ThinApp, Microsoft App-V, et cameyo permettent de créer des packages d'applications portables qui peuvent être déployés sur des systèmes Windows sans installation.
- Limites :
  - La virtualisation de l'application sur le système d'exploitation hôte est souvent limitée à un système d'exploitation spécifique
  - Certaines applications peuvent être incompatibles avec ces logiciels
  - Certaines applications ne peuvent pas être virtualisées par ces logiciels : par exemple ThinApp ne peut pas virtualiser les antivirus, pilotes d'imprimantes et scanner, applications nécessitant des pilotes de périphériques, certains VPN, etc.).
  - Il faut disposer d'une machine physique ou virtuelle propre à utiliser comme cible pour le "séquençage" (App-V) ou la "capture" (ThinApp).

Virtualisation basée sur les conteneurs

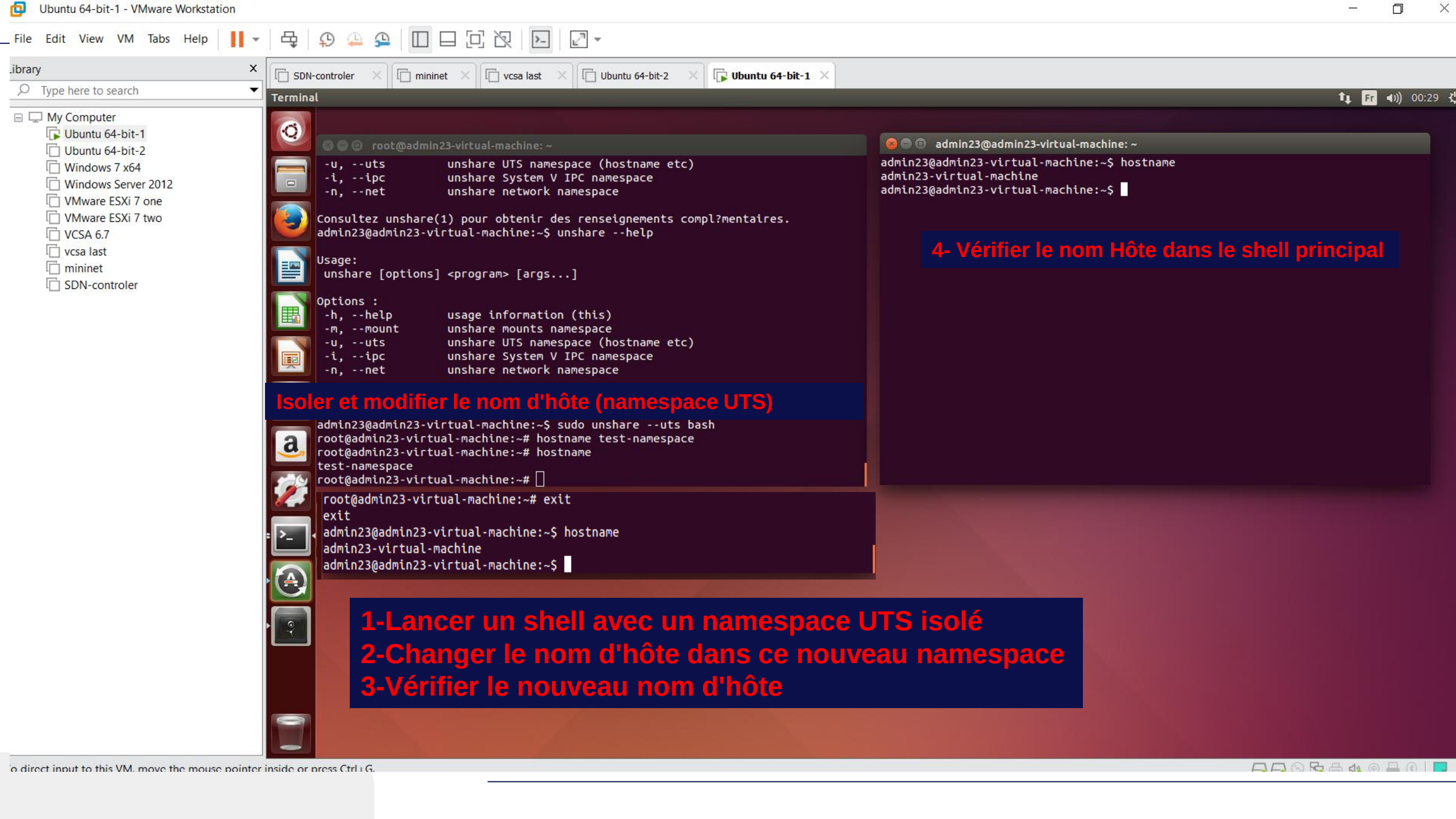
- Aussi appelé virtualisation basé sur l'isolation des processus
- Elle consiste à isoler les environnements d'exécution d'applications, le tout étant géré par un unique noyau celui du host.
- Bien que ce type de virtualisation n'est pas spécifique à Linux, elle repose sur les fonctionnalités de virtualisation du noyau Linux pour isoler les processus les uns des autres.
- L'isolation de processus est réalisée principalement grâce aux fonctionnalités du noyau Linux suivantes :
    - les espaces de noms
    - les contrôles de groupe de processus (cgroups)
    - Changer le répertoire racine : chroot.

Les espaces de noms (namespaces)

- Les **namespaces** sont un mécanisme de virtualisation du noyau Linux qui permettent d'isoler les ressources système entre différents **processus** ou **groupes** de **processus**.
- Lorsqu'un processus est dans un espace de noms, il ne peut pas voir ou accéder aux ressources des autres processus ou groupe de processus
- Le rôle des **namespaces** est **d'isoler** les ressources système en créant un **contexte système isolé** pour le processus ciblé afin qu'il ne puisse pas voir les processus des autres, leurs connections réseaux ou leurs fichiers.
- Ce mécanisme va créer pour le **processus** ciblé des **instances virtuelles** des ressources système, telles que les **UID**, **système** de fichiers, **PID**, le **réseau**, la **liste** des processus, les **utilisateurs**, les systèmes de communication interprocessus (**IPC**), etc.
- **Par exemple** : chaque groupe de processus a sa propre instance virtuelle du système de fichier permettant de monter/démonter les volumes sans incidence pour l'hôte ou les autres processus.

Les espaces de noms (namespaces)

- Dans le noyau Linux, il existe différents types d'espaces de noms avec ses propres propriétés :
  - **user namespace** namespace possède son propre ensemble d'ID d'utilisateurs et d'ID de groupes à attribuer aux processus. Cela permet à un processus d’avoir le privilège root dans son espace de noms utilisateur sans l'avoir dans d'autres espaces de noms utilisateur.
  - **Process ID (PID)** namespace qui attribue aux processus un ensemble de PID indépendant de l'ensemble des PID des autres espaces de noms.
  - **Network** namespace possède sa propre table de routage privée, son ensemble d'adresses IP, sa liste de sockets, sa table de suivi des connexions, son pare-feu et d'autres ressources liées au réseau.
  - **Mount** namespace possède une liste indépendante de points de montage. Cela signifie qu’on peut monter et démonter des systèmes de fichiers dans un espace de noms de montage sans affecter le système de fichiers hôte.
  - **Interprocess communication** (IPC) namespace possède ses propres ressources IPC, ses propres files d'attente de messages pour communiquer.
  - **UNIX Time-Sharing (UTS) namespace permet à** chaque processus d’avoir son propre nom d'hôte (hostname) et son propre nom de domaine, qui peuvent être définis indépendamment du nom d'hôte et du nom de domaine réels du système.


Ubuntu 64-bit-1 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Ubuntu 64-bit-1
Ubuntu 64-bit-2
Windows 7 x64
Windows Server 2012
VMware ESXi 7 one
VMware ESXi 7 two
VCSA 6.7
vcsa last
mininet
SDN-controler
SDN-controler
mininet
vcsa last
Ubuntu 64-bit-2
Ubuntu 64-bit-1
Terminal
Fr
00:29
root@admin23-virtual-machine: ~
-u, --uts unshare UTS namespace (hostname etc)
-i, --ipc unshare System V IPC namespace
-n, --net unshare network namespace
Consultez unshare(1) pour obtenir des renseignements compl?mentaires.
admin23@admin23-virtual-machine:~$ unshare --help
Usage:
unshare [options] <program> [args...]
Options :
-h, --help usage information (this)
-m, --mount unshare mounts namespace
-u, --uts unshare UTS namespace (hostname etc)
-i, --ipc unshare System V IPC namespace
-n, --net unshare network namespace
admin23@admin23-virtual-machine: ~
admin23@admin23-virtual-machine:~$ hostname
admin23-virtual-machine
admin23@admin23-virtual-machine:~$
4- Vérifier le nom Hôte dans le shell principal
Isoler et modifier le nom d'hôte (namespace UTS)
admin23@admin23-virtual-machine:~$ sudo unshare --uts bash
root@admin23-virtual-machine:~# hostname test-namespace
root@admin23-virtual-machine:~# hostname
test-namespace
root@admin23-virtual-machine:~#
root@admin23-virtual-machine:~# exit
exit
admin23@admin23-virtual-machine:~$ hostname
admin23-virtual-machine
admin23@admin23-virtual-machine:~$
1-Lancer un shell avec un namespace UTS isolé
2-Changer le nom d'hôte dans ce nouveau namespace
3-Vérifier le nouveau nom d'hôte

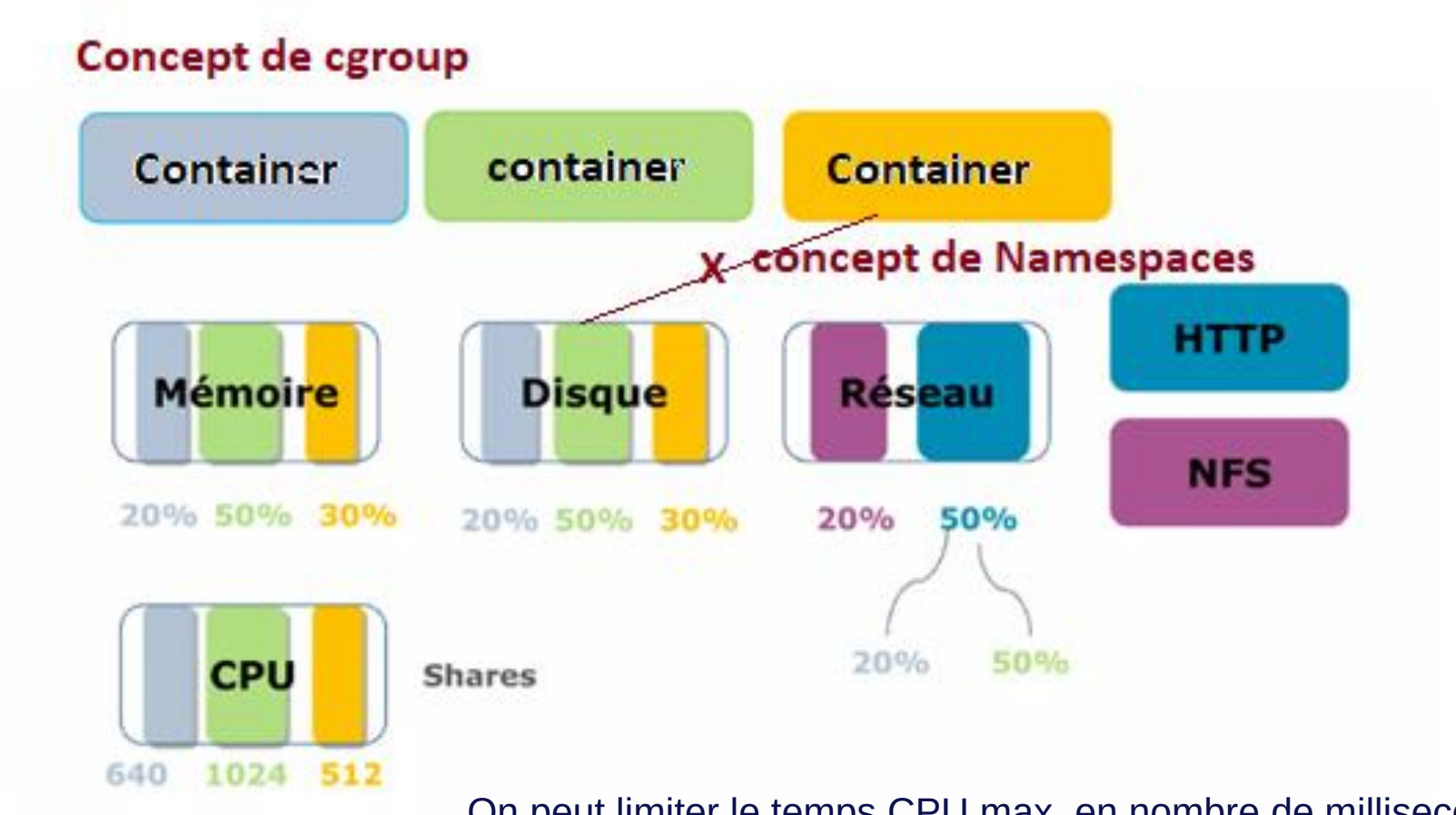
Les contrôles de groupe de processus (cgroups)

- Un groupe de contrôle est une fonctionnalité du noyau Linux qui permet de limiter, gérer et surveiller et isoler **l'utilisation des ressources** (CPU, mémoire, E/S disque, réseau, etc.) d'un ensemble de processus.
- Ce mécanisme **empêche** un groupe de processus **d'accéder aux ressources** des autres groupe ou celle du système hôte.
- Les cgroups permettent de **contrôler** la quantité d'une ressource (CPU, mémoire, réseau et E/S disque) à laquelle un processus ou un ensemble de processus peut accéder ou qu'il peut utiliser.
- Les cgroups sont un élément clé dans un environnement partagé comme celui des conteneurs car il y a souvent plusieurs processus en cours d'exécution dans un conteneur qu'il faut contrôler ensemble.

Les contrôles de groupe de processus (cgroups)

- Les Cgroups offrent les fonctionnalités suivantes :
  - **Limitation** des ressources permet de configurer un cgroup pour limiter la quantité d'une ressource particulière (mémoire ou CPU, par exemple) qu'un processus peut utiliser.
  - **Priorité** permet de contrôler la quantité d'une ressource (CPU, disque ou réseau) qu'un processus peut utiliser par rapport aux processus d'un autre cgroup lorsqu'il y a conflit de ressources.
  - **Surveillance** : Les limites de ressources sont surveillées et signalées au niveau du cgroup.
  - **Contrôle** – permet de modifier l'état (gelé, arrêté ou redémarré) de tous les processus d'un cgroup.

Cgroup et Namespaces



On peut limiter le temps CPU max en nombre de milliseconde

Changer le répertoire racine

- La commande **chroot** permet de changer le répertoire racine vers un nouvel emplacement dans le système de fichier.
- Après avoir modifié la racine, le processus ne peut plus accéder aux fichiers ou aux utilitaires qui sont **plus haut dans la hiérarchie** des fichiers.
- C'est un commande qui prend comme unique argument un répertoire et y fixe la racine du système de fichiers pour le processus qui l'invoque.
- Ainsi on peut emprisonner l'exécution des processus dans son contexte.

```
sudo chroot /home/admin23/chroot_test  /bin/ls
```

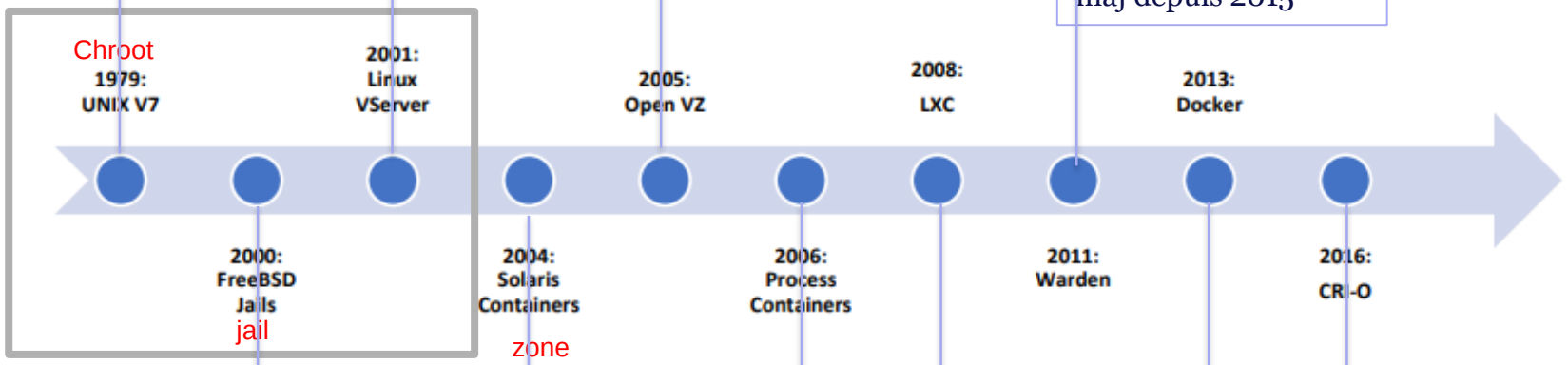
Chronologie de la Virtualisation par conteneurs

Chroot : Lors de la création d'un nouveau processus, ce dernier est isolé du système hôte

Vserver: le même concept que BSD Jail mais dans l'univers Linux, il n'est plus maintenu depuis 2006

est une technologie de virtualisation au niveau du système d'exploitation qui permet de créer plusieurs **environnements virtuels** sur un seul noyau Linux.

Warden est une technologie de conteneurisation utilisé sur Cloud Foundry, un PAAS (Platform as a Service), n'est plus maj depuis 2015



Jail :21 ans plus tard FreeBSD reprend l'idée avec les BSD Jails pour créer un environnement d'exécution isolé pour les processus (sandbox)

Zone : c'est le même concept que les BSD Jails mais pour les systèmes Solaris, Les **zones** de Solaris permettent de créer des environnements logiquement séparés, chacun ayant son propre **espace utilisateur**, **répertoire racine**, **adresses IP**, et **ressources système**.

Cgroup: Process Containers fût lancé par Google, et a été intégré au noyau Linux 2.6.24 en janvier 2008.

LXC remplace OpenVZ et inègre le cgroups et namespaces

Lancée par l'équipe de Solomon Hykes

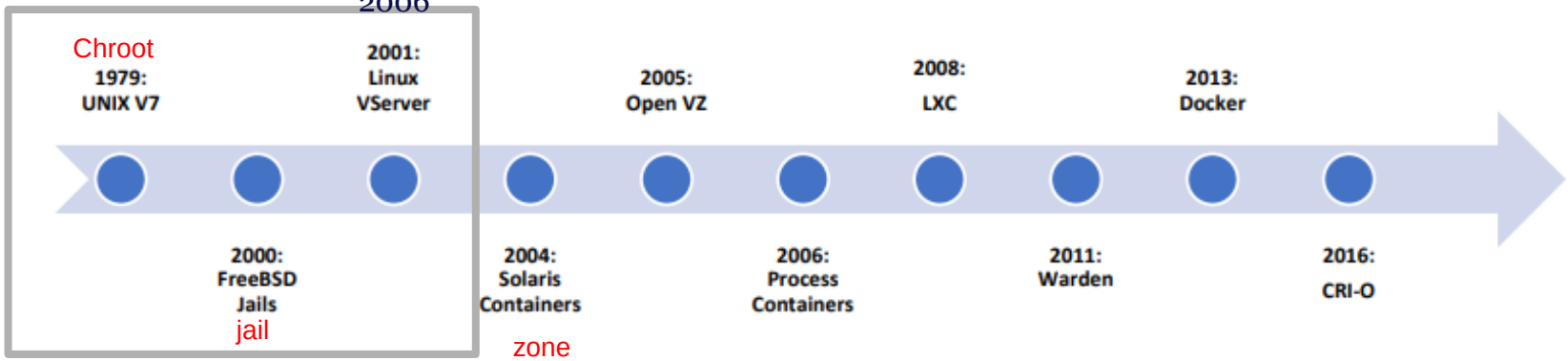
Gestion des conteneurs

Chronologie de la Virtualisation par conteneurs

- La Virtualisation basée sur les conteneurs consiste à encapsuler l'application, les librairies et ses dépendances dans un unité logiciel exécutable appelée **conteneur(container)**.
- Un conteneur peut être exécuté sur un système d'exploitation hôte sans avoir besoin d'une VM.
- Ce schéma donne l'origine des conteneurs et quelques solutions de virtualisation basées sur les conteneurs

Chroot : Lors de la création d'un nouveau processus, ce dernier est isolé du système hôte

le même concept que BSD Jail mais dans l'univers Linux, il n'est plus maintenu depuis 2006



21 ans plus tard FreeBSD reprend l'idée avec les BSD Jails pour créer un environnement d'exécution isolé pour les processus (sandbox)

Zone : c'est le même concept que les BSD Jails mais pour les systèmes Solaris, Les **zones** de Solaris permettent de créer des environnements logiquement séparés,

Deux approches de conteneurs

- Approche Système
  - LinuX Containers (LXC)
  - OpenVZ.
- Approche Processus
  - Docker.

LXC (LinuX Containers)

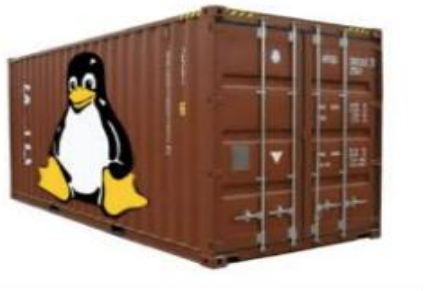
LXC

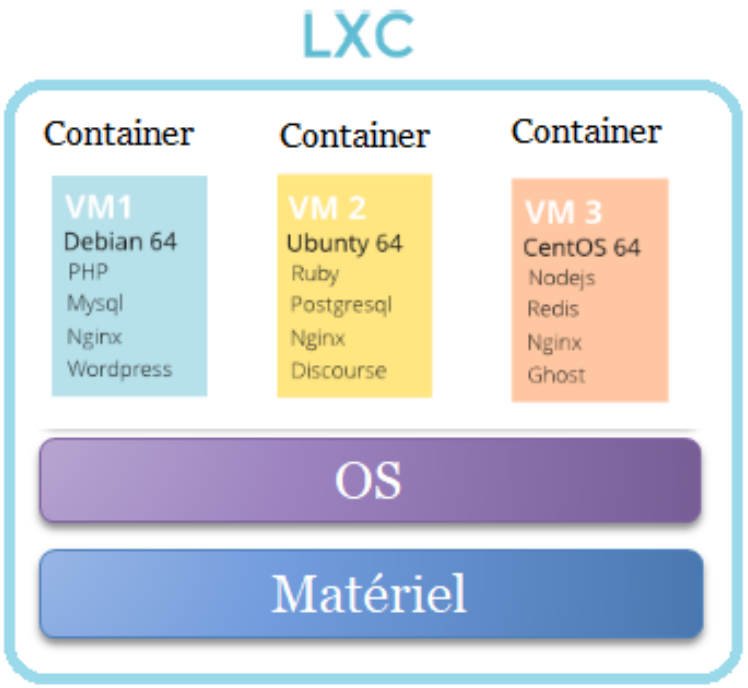
- C’est une technologie open source de virtualisation au niveau du système d'exploitation qui permet la création et l'exécution de plusieurs environnements virtuels Linux isolés (VE) appelés conteneurs sur un seul hôte de contrôle.
- C’est une solution de virtualisation de niveau système d'exploitation basée sur le noyau Linux
- Les conteneurs LXC partagent les ressources du noyau de la machine physique de l'hôte, ce qui les rend plus légers et plus rapides que les machines virtuelles.
- LXC se base sur les **cgroups** et des **namespaces** dans le noyau Linux afin de prendre en charge des environnements OS virtualisés légers (conteneurs).
- Les conteneurs peuvent être utilisés soit pour cloisonner des **applications** spécifiques, soit pour émuler un tout nouvel **hôte**.

Virtualisation LinuX Containers (LXC)

LXC

- Il est possible de se connecter à un conteneur LXC, de le traiter comme un système d'exploitation, d'y installer une ou plusieurs applications et des services.





Le système d'exploitation invité s'exécute sur le **même noyau que l'hôte**.

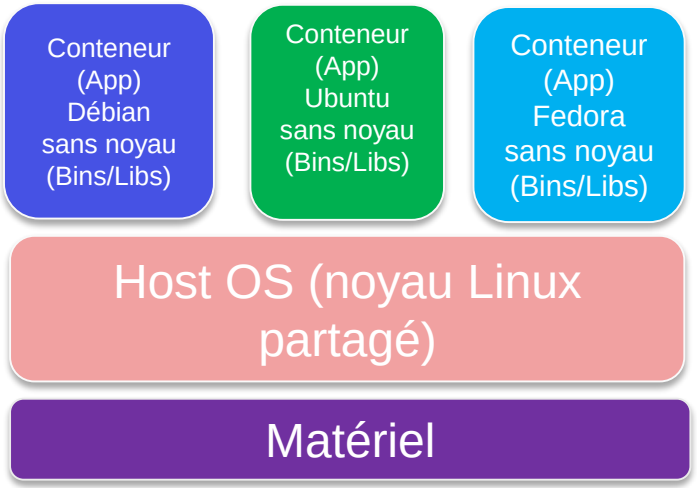
Virtualisation LinuX Containers (LXC)

LXC

- Les conteneurs LXC ne sont pas des machines virtuelles dans le sens traditionnel du terme, il n' y a pas d'hyperviseur et le matériel n'est pas virtualisé.
- Un conteneur apporte une virtualisation de l'environnement d'exécution (processeur, mémoire vive, réseau, système de fichier…) et non pas de la machine physique
- LXC fournit des modèles de conteneurs pour les systèmes d'exploitation de base et un ensemble complet d'outils pour la gestion du cycle de vie des conteneurs.
- LXC prend en charge d'autres formats de conteneurs tels que les conteneurs **Docker** et les conteneurs **OCI** (Open Container Initiative).

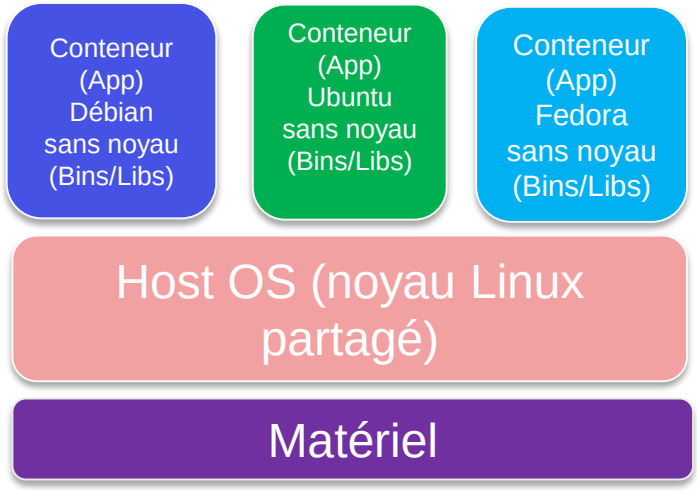
Conteneur OpenVZ

- OpenVZ est une solution de virtualisation Open Source développée par Parallels Virtuozzo.
- C'est une solution de virtualisation de niveau système d'exploitation basée sur le noyau Linux
- OpenVZ a **intégré** les **espaces** de **noms**, les **cgroup** et les chroots, pour isoler les conteneurs les uns des autres et du système hôte.
- OpenVZ est principalement utilisé pour la virtualisation de systèmes complets qu'on appelle Virtual Private Servers (VPS).
- Ces VPS sont des conteneurs qui n'ont pas de noyau, ils partagent le noyau de l'hôte à partir duquel ils sont déployés.
- Il n'y a aucune charge supplémentaire due a une émulation



Conteneur OpenVZ

- Chaque conteneur dispose de son propre système de fichiers, de ses propres utilisateurs et groupes, de ses processus, d'interfaces réseau etc…
- Pour chaque conteneur il est possible de mettre des quotas sur la quantité mémoire ou le temps processeur qui peuvent être consommés.
- Tous les conteneurs ou VPS doivent avoir une distribution linux compatibles avec la version du noyau et l'architecture du système hôte physique,
- Un VPS ressemble à un vrai serveur physique

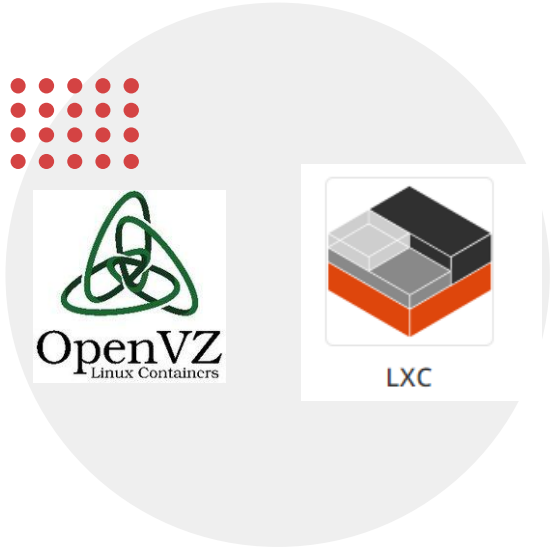


Conteneur OpenVZ

- Les VPS supportent la migration et leur cycle de vie est géré par un outil propriétaire « vzctl »
- OpenVZ propose une large gamme de templates VPS préconfigurés pour les distributions Linux populaires, telles que Debian, Ubuntu, CentOS, Fedora ...
- Différentes distributions peuvent fonctionner en parallèle sur un même nœud hôte
- Il permet d'exécuter des centaines de conteneurs sur une petite machine
- les environnements hôte et invité doivent tous deux avoir **le même noyau linux**
- Impossible d'installer un OS comme Windows server, Bsd..
- Les conteneurs OpenVZ ne sont pas compatibles avec les autres plateformes

OpenVZ vs LXC

- LXC et OpenVZ sont des technologies de virtualisation légère
- Ils sont tout deux principalement utilisé pour la virtualisation de systèmes complets contrairement à Docker.
- Les VPS de OpenVZ et les conteneurs de LXC peuvent être configurés pour exécuter différents distributions Linux et offre des performances de système d'exploitation similaires à un système d'exploitation installé sur un serveur physique.
- Tous deux prennent en charge la migration en direct d'un conteneur d'un host vers un autre host du même type



OpenVZ vs LXC

OpenVZ

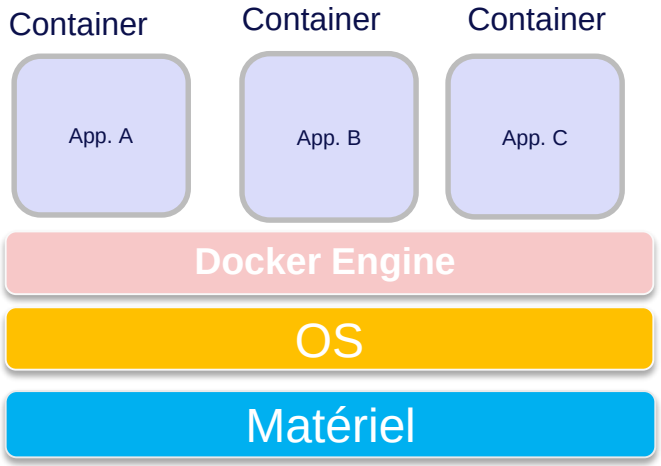
- **OpenVZ**, nécessite un noyau modifié
- OpenVZ est conçu pour fonctionner avec des systèmes invités qui utilisent la même famille de noyau que l'hôte.
- OpenVZ n'est pas compatible avec les technologies de conteneurs tels que Docker ou LXC.
- Cela peut limiter la portabilité des applications entre différents environnements de conteneurs.

LXC

- **LXC** fonctionne avec le noyau Linux standard, il peut être utilisé sur toute distribution Linux sans nécessiter de modifications au noyau.
- LXC prend en charge d'autres formats de conteneurs tels que les conteneurs Docker et les conteneurs OCI.

Docker

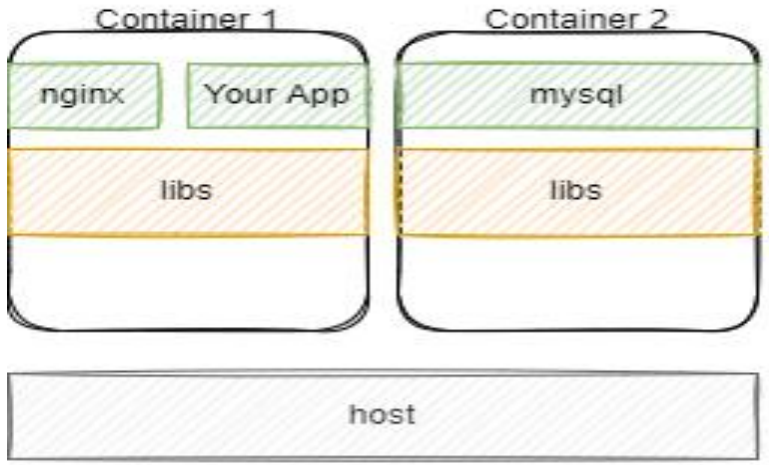
- Docker est un logiciel libre de virtualisation des applications basé sur l'isolation de processus.
- C’est une plateforme open source qui automatise le déploiement, la gestion et l'exécution d'applications à l'intérieur de conteneurs logiciels.
- Les conteneurs Docker encapsulent une application avec son environnement d'exécution et toutes ses dépendances, ce qui permet de les exécuter de manière cohérente et portable sur n'importe quel environnement compatible Docker,
- En théorie, les conteneurs Docker sont conçus pour contenir une seule application ou un service spécifique.



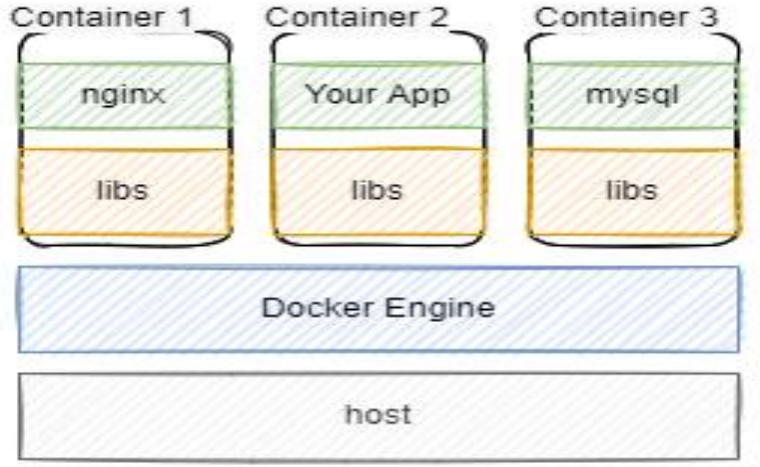
Le rôle de l’isolateur est assuré par Docker engine

LXC vs Docker

- Docker a été initialement développé en utilisant les conteneurs LXC, mais il a évolué en adoptant une autre technologie de virtualisation appelée **libcontainer**.
- Cependant, Docker continue d'utiliser LXC pour certaines fonctionnalités telles que l'isolation des ressources et des processus.
- Le Docker Engine permet de créer, exécuter et gérer des conteneurs Docker
- Docker utilise une architecture client-serveur qui se base sur l'API REST.

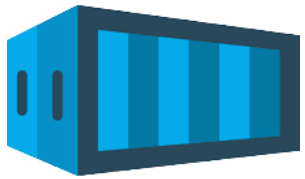


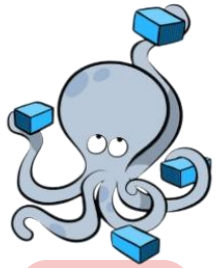
LXC

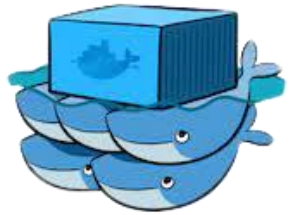


Docker

Concepts Clés



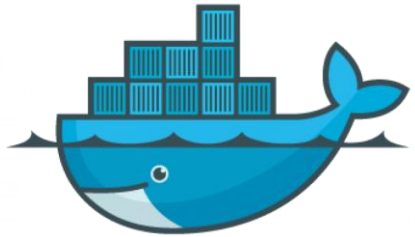


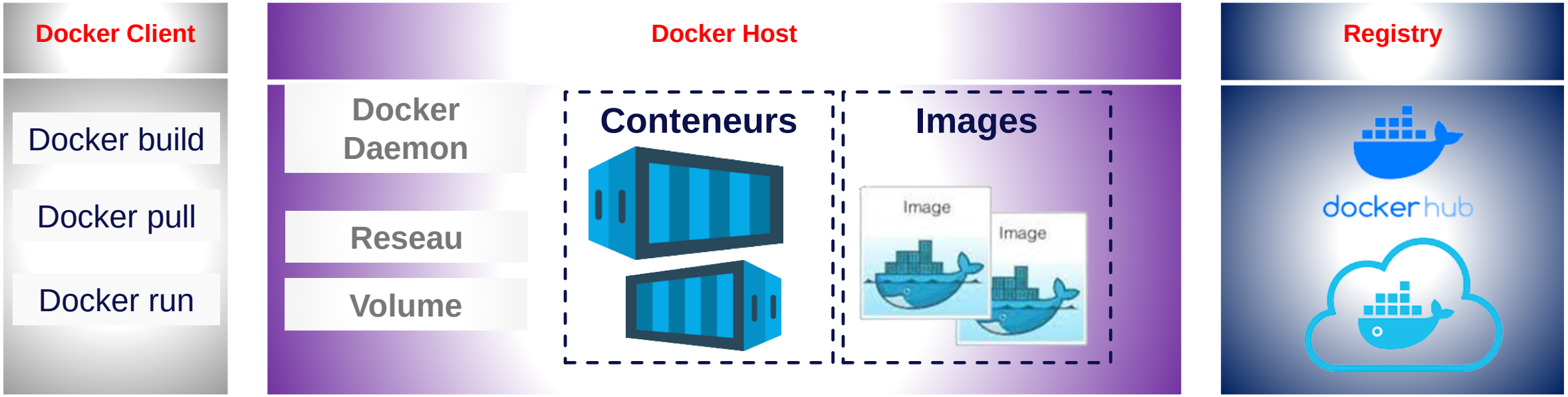


1 2 3 4 5 6 7

Conteneur | Image | Dockerfile | Registre Docker | Docker compose | Docker Swarm | Docker Engine

Architecture Docker





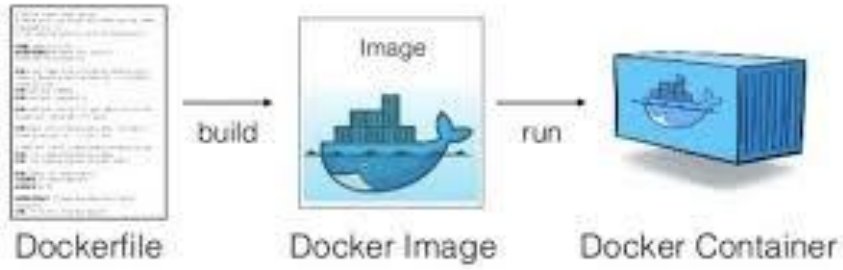
Docker client

- Un **conteneur** est une **instance** d'une image Docker qui peut être exécutée de manière isolée sur un système hôte. Chaque conteneur contient une **application** et toutes les dépendances nécessaires pour son exécution
- Une **image** Docker est un package exécutable (sorte de Template prête à l'emploi) qui contient toutes les dépendances et les fichiers nécessaires pour exécuter une application.
- Les images Docker sont créées à partir d'un fichier de configuration appelé **Dockerfile**.

Docker build

Docker pull

Docker run



Dockerfile

- Un **Dockerfile** est un fichier de configuration qui contient les instructions nécessaires pour créer une image Docker.
- Il spécifie les dépendances, les fichiers et les configurations nécessaires pour exécuter une application.
- Ces instructions décrivent les actions que l'image doit exécuter une fois qu'elle sera créée.
- Elles seront lues une à une, puis exécutées indépendamment les unes des autres pour obtenir l'image souhaitée.
- Un Dockerfile précise le système d'exploitation sur lequel sera basé le conteneur, les langages, variables environnementales, emplacements de fichiers, ports réseaux et autres composants requis.

Dockerfile

- Voici un exemple de Dockfile, avec explications des instructions :
  - **From**: permet de définir l'image source, ici une image de base Debian 9
  - **RUN** : pour exécuter une commande dans votre conteneur **ADD** afin de copier ou de télécharger des fichiers dans l'image
  - **WORKDIR** qui permet de modifier le répertoire courant. La commande est équivalente à une commande cd en ligne de commande. L'ensemble des commandes qui suivront seront toutes exécutées depuis le répertoire défini.
  - **RUN** npm install : permet d'installer le package du projet Node.js.
  - **EXPOSE** permet d'indiquer le port sur lequel l'application est à l'écoute.
  - **VOLUME** permet d'indiquer quel répertoire sera partagé avec le hos et servir à stocker les logs persistants.
  - **CMD** qui permet de définir la commande par défaut lors de l'exécution de des conteneurs Docker..

```
# Utiliser l'image officielle Node.js avec la version
FROM node:18-bullseye
# Mettre à jour les paquets et nettoyer après
l'installation
RUN apt-get update -yq && apt-get install -yq curl &&
apt-get clean
# Copier tous les fichiers dans le répertoire /app
ADD . /app/
WORKDIR /app
# Copier uniquement package.json et installer les
dépendances via npm
COPY package.json .
RUN npm install
# Exposer le port de l'application
EXPOSE 2368
# Déclarer un volume pour les logs
VOLUME /app/logs
# Lancer l'application
CMD ["npm", "run", "start"]
```

Dockerfile

- Docker utilise un fichier package.json
  - Installer les dépendances : Lorsqu'on construit une image Docker, le package.json permet d'installer automatiquement toutes les dépendances.
  - Définition des scripts de démarrage: Le package.json définit les scripts qui contrôlent comment l'application est lancée, il peut utiliser ces scripts pour démarrer l'application.
  - Déclaration des métadonnées du projet: ce fichier contient des informations importantes sur l'application comme: nom, version, auteurs, etc., nécessaire pour la documentation et l'organisation du projet.
  - Automatiser les tâches liées à Docker avec des scripts personnalisés comme la création et exécution.

```
{
 "name": "my-app",
 "version": "1.0.0",
 "description": "Une application Node.js
conteneurisée",
"author": "DSBD",
 "license": "FSBM",
 "scripts": {
  "start": "node app.js"
 },

  "dependencies": {}
}
```

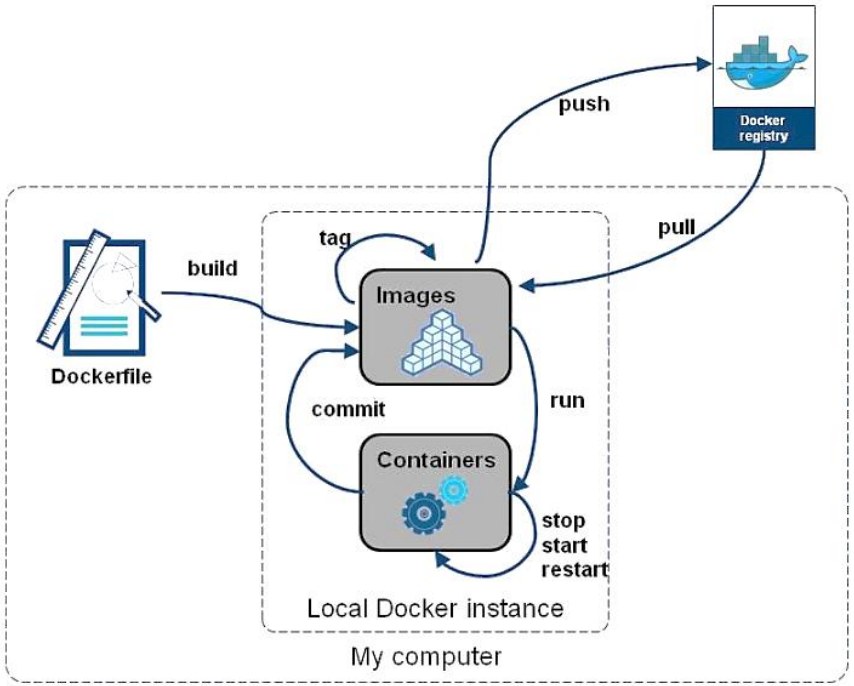
La ligne "dependencies": {} signifie simplement qu'il n'y a aucune dépendance nécessaire pour exécuter l'application à ce moment-là elle sera remplie dès qu'une dépendance sera ajoutée via **npm install**.

Docker client

- **build** : permet de construire une image Docker à partir d'un Dockerfile.
- Lorsqu'on exécute la commande **build**, Docker va créer un conteneur pour chaque instruction, et le résultat sera sauvegardé dans une layer.

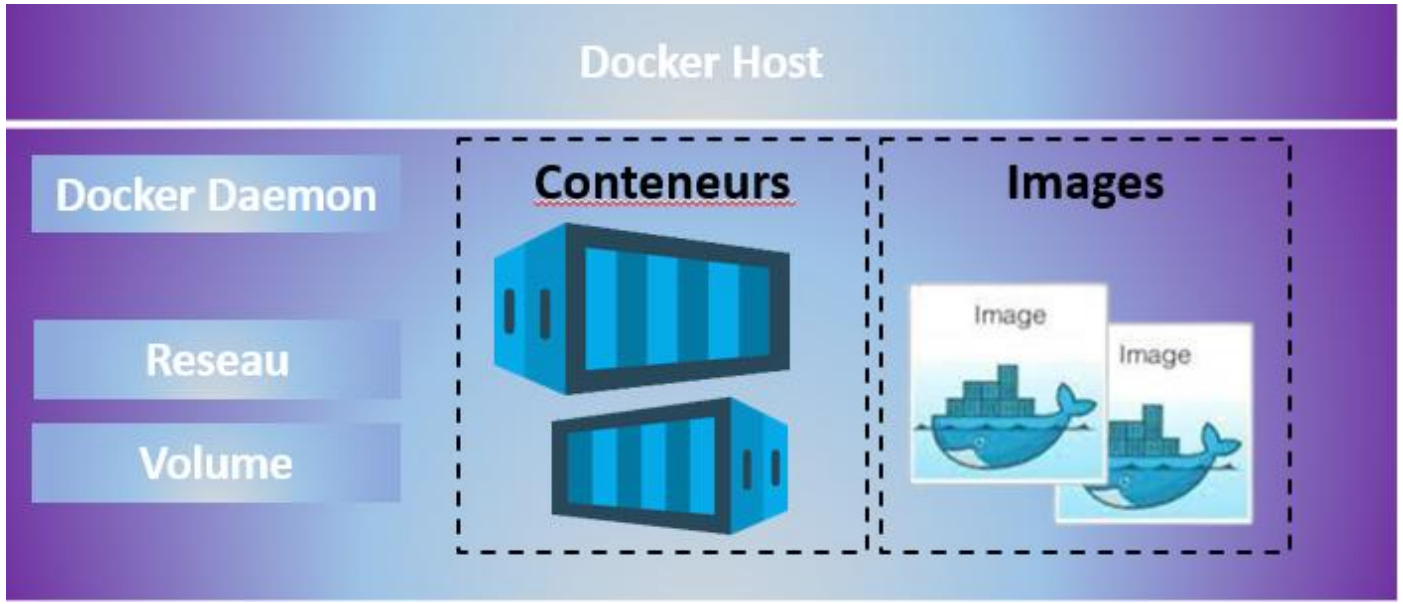
```
devdocker build -t Myapp-dev
```

- Le résultat final étant un ensemble de layers qui construisent une image Docker complète.
- Si une layer ne bouge pas entre deux builds, Docker ne la reconstruira pas. Seules les layers situées après une layer qui se reconstruit seront elles aussi reconstruites.
- **Run**: permet de lancer un conteneur Docker à partir d'une image
- **Pull**: permet de télécharger une image Docker à partir d'un registre Docker public ou privé
- **Commit** : représente une "capture" des modifications dans un projet



Architecture Docker : Docker Host

- Le Docker Host représente la machine physique ou virtuelle qui exécute le moteur Docker.
- Il fournit l'environnement et les ressources nécessaires pour créer, exécuter et gérer des conteneurs
  - Il assure aussi des fonctionnalités comme l'installation, la suppression, la mise à jour et la surveillance des conteneurs.
  - Plusieurs conteneurs Docker peuvent être exécutés simultanément sur un seul Docker host
- **Démon Docker** : processus d'arrière-plan chargé de recevoir les commandes et de les transmettre aux conteneurs.



Architecture Docker : Docker Host

- **Réseaux** : Les réseaux Docker sont des réseaux virtuels qui assure la communication entre les conteneurs isolés.
- Il existe principalement cinq pilotes réseau dans le docker:

•bridge: Les conteneurs utilisant ce pilote ne peuvent communiquer qu'entre eux, donc ils ne sont pas accessibles de l'extérieur.

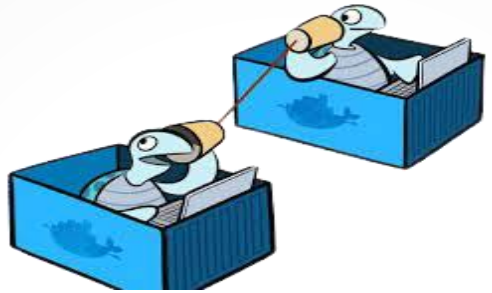
•none: interdit toute communication interne ou externe avec le conteneur car ce dernier n'aura pas d'interface réseau.

•host: permet aux conteneurs d'utiliser la même interface réseau que l'hôte. Dans ce cas, les conteneurs sont accessibles de l'extérieur et ne seront pas isolés les uns des autres dans le réseau.

•overlay: ce mode est utilisé dans le cas de mise en réseau multi-hôte, dans ce cas les conteneurs s'exécutent sur différents hôtes Docker

•macvlan: idéal lors de l'utilisation des applications nécessitant une connexion directe au réseau physique. Le macvlan attribue une adresse mac a un conteneur pour que celui-ci apparaisse comme un périphérique physique sur le réseau.

Réseau



Le système réseau de Docker utilise des drivers (pilotes). Plusieurs drivers existent et fournissent des fonctionnalités différentes.

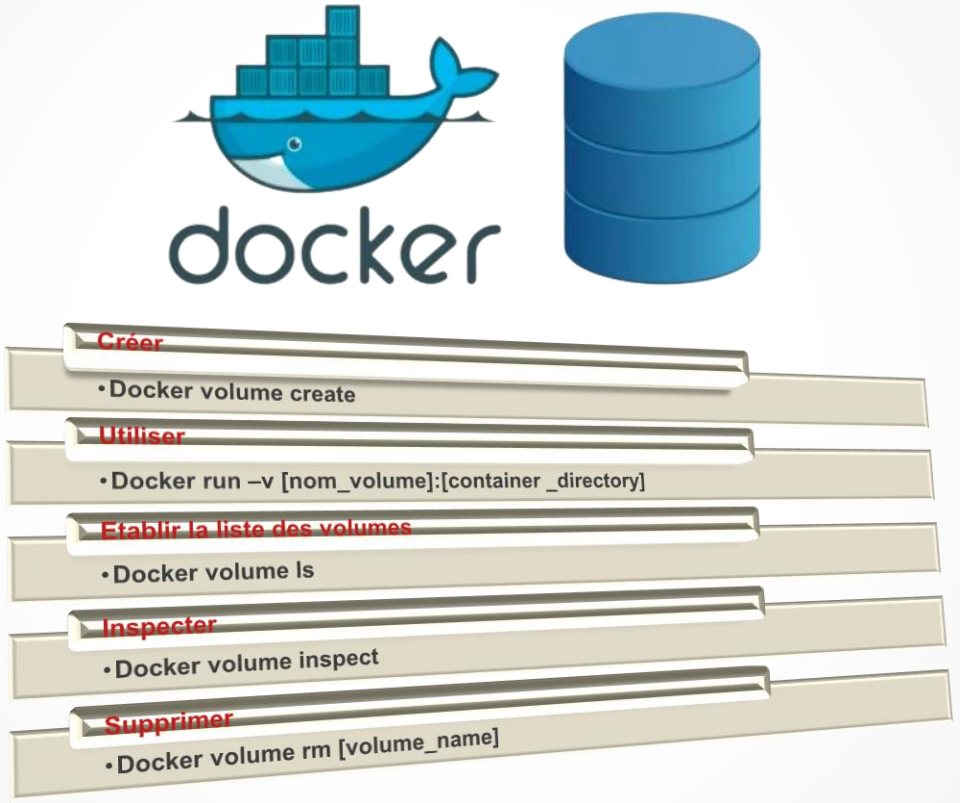
Bridge | None | Host | Overlay | Macv

Dans Docker, chaque conteneur possède sa propre interf réseau virtuelle qui est connectée à un réseau spécifiqu

Architecture Docker : Docker Host

- **Volumes** : Les volumes Docker permettent de stocker des données persistantes pour les conteneurs Docker.
- Les volumes Docker peuvent être montés dans des conteneurs pour stocker des données telles que des fichiers de configuration, des bases de données, des fichiers de log, etc.
- Les volumes sont indépendants des conteneurs, ils peuvent être utilisés pour stocker des données même si le conteneur est supprimé
- Cette approche permet d'éviter de sauvegarder les données sur un conteneur.

Volume

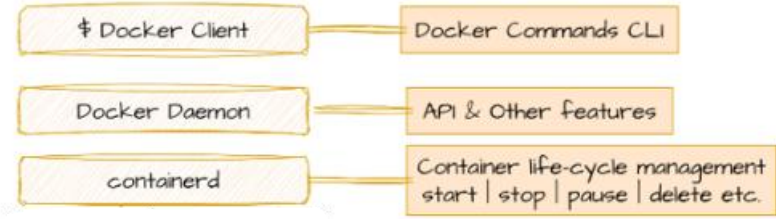


Architecture Docker: Registry

- Registry, appelé aussi Docker Hub est un service qui permet stocker, gérer et distribuer des images de conteneurs (dépôt d'images).
- Il Fournit un emplacement centralisé pour que les développeurs publient les images de conteneurs qui seront accessible pour les utilisateurs à travers un API.
- Le registre public de Docker est appelé Docker Hub, mais il existe également des registres privés pour les entreprises.
- Docker Hub offre plusieurs services :
    - L'hébergement d'images Docker
    - Authentification des utilisateurs
    - Intégration avec GitHub

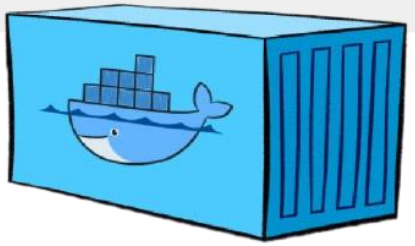
Architecture Docker: Docker Engine

- **Docker Engine** est le composant principal de Docker qui permet de créer, de gérer et d'exécuter des conteneurs Docker sur un système hôte.
- Cela inclut plusieurs composants :
  - Le Docker Daemon (ou dockerd), qui est le processus principal du Docker Engine.
  - L'API Docker, qui permet aux applications d'interagir avec Docker.
  - L'interface en ligne de commande Docker (CLI), qui permet aux utilisateurs d'interagir avec le Daemon via des commandes dans le terminal.



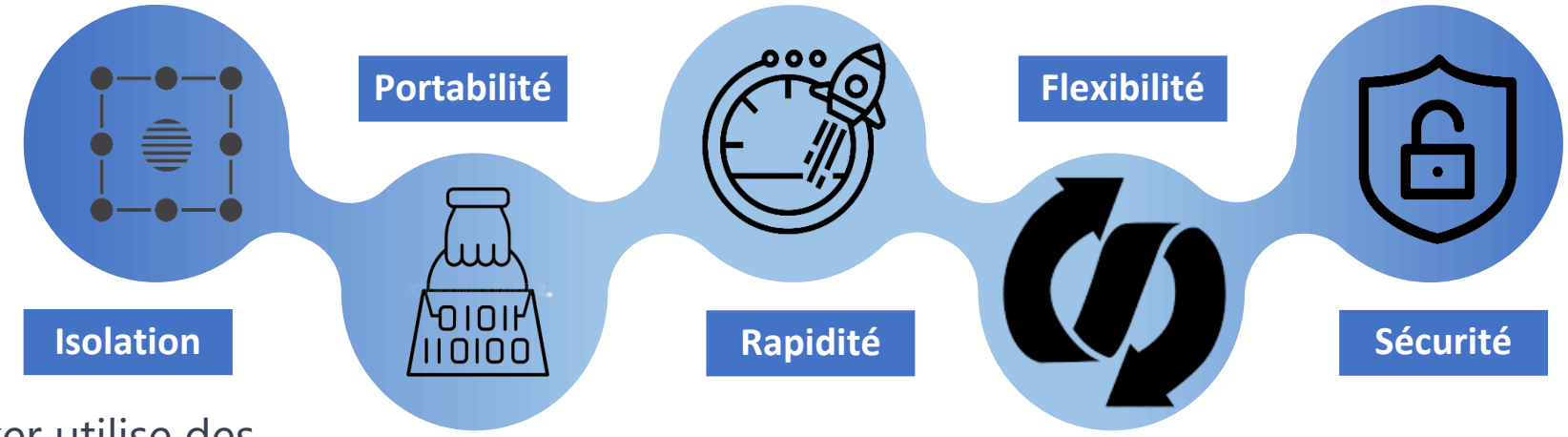
containerd est un **runtime** de conteneur, c'est un composant de bas niveau responsable de l'exécution des conteneurs.

Autres Objets Docker



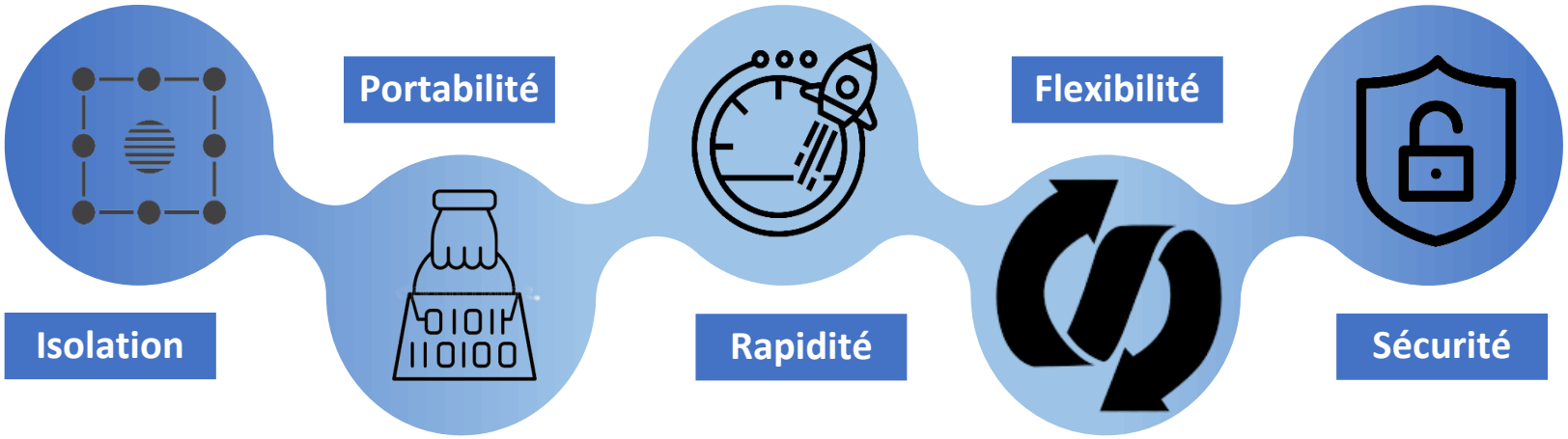
- **Docker Compose** : Docker Compose est un outil qui permet la gestion de multi-conteneurs sur **une seule machine**.
  - Il permet de définir un environnement de développement multi-conteneurs dans un fichier YAML (docker-compose.yml) et de déployer ces conteneurs avec une seule commande
- **Docker Swarm** : Docker Swarm est un outil d'orchestration de conteneurs.
  - Il permet de gérer des clusters de machines Docker et de déployer des applications sur **plusieurs hôtes Docker** de manière distribuée et hautement disponible

Avantages



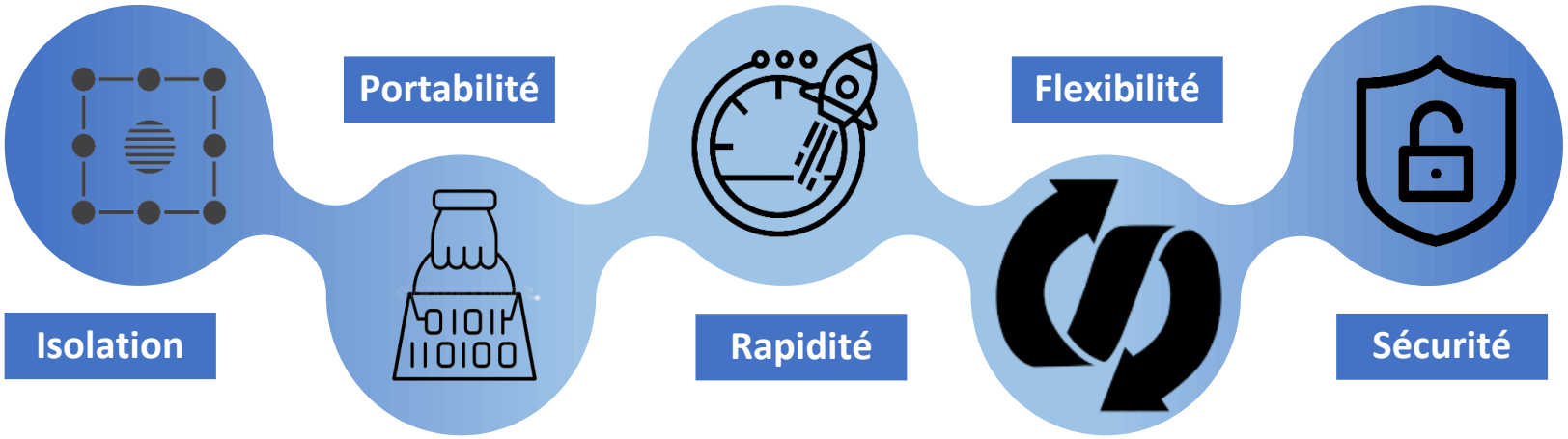
Isolation : Docker utilise des conteneurs pour isoler les applications et leurs dépendances, ce qui permet d'éviter les conflits entre les différentes applications et de garantir que chaque application fonctionne de manière prévisible.

Avantages



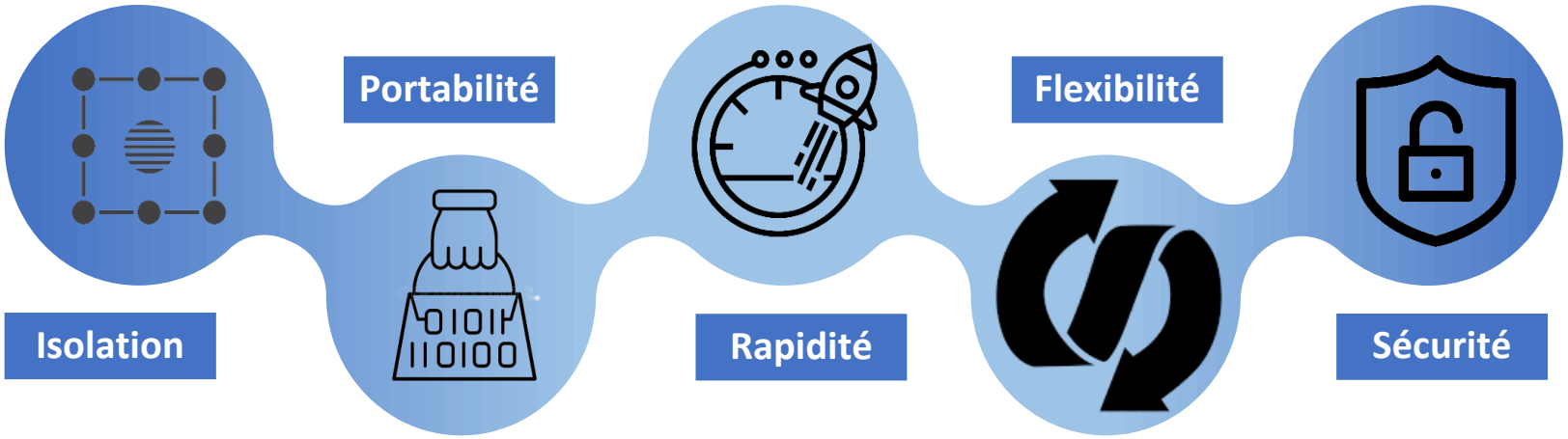
Portabilité : Les conteneurs Docker sont conçus pour fonctionner sur n'importe quel système d'exploitation, ce qui permet aux développeurs de créer une application une fois, puis de la déployer facilement sur différents environnements.

Avantages



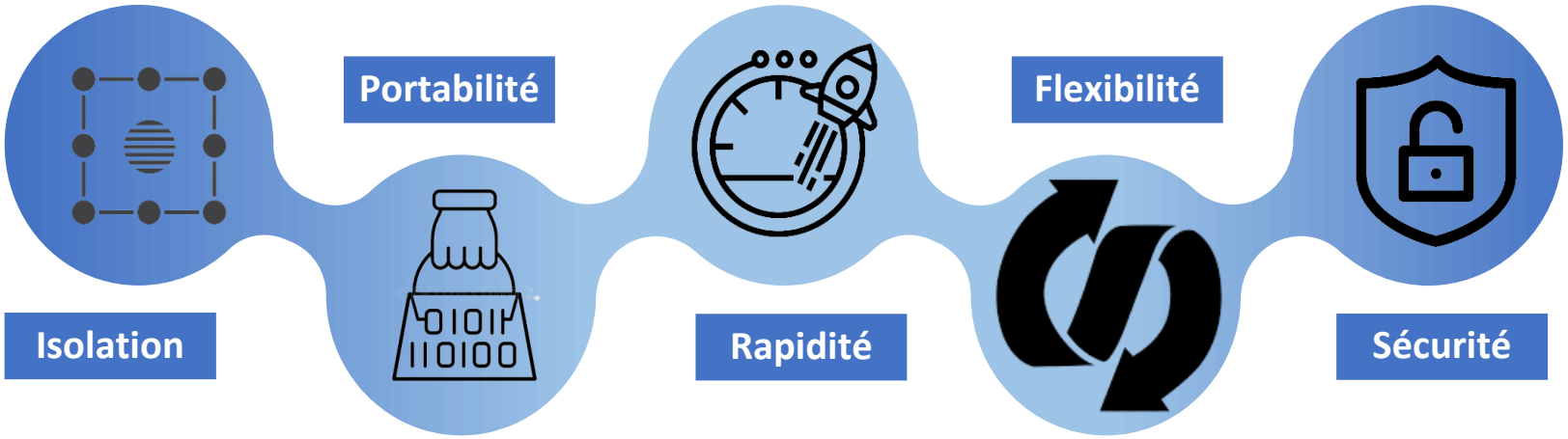
Rapidité : Les conteneurs Docker sont extrêmement légers et peuvent être démarrés en quelques secondes, ce qui permet aux développeurs de tester et de déployer rapidement des applications.

Avantages



Flexibilité : Docker permet aux développeurs de créer des microservices, qui sont de petites applications indépendantes qui peuvent être combinées pour créer des applications plus complexes.

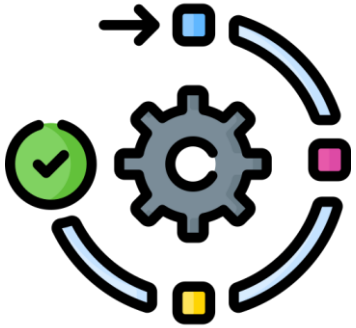
Avantages



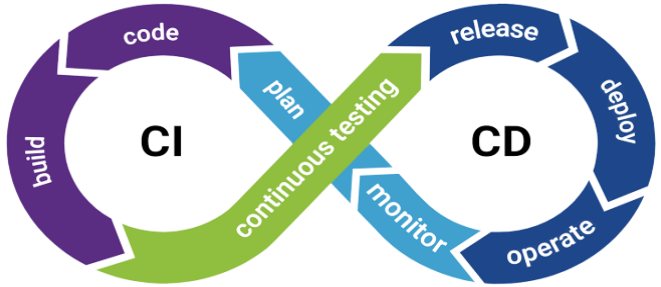
Sécurité : Docker utilise des mécanismes de sécurité intégrés pour garantir que les conteneurs sont isolés les uns des autres et que les applications sont protégées contre les menaces externes.

Docker et CLoud

Le Cloud Computing et Docker sont deux technologies qui peuvent être utilisées conjointement pour fournir des environnements de développement et de déploiement flexibles et évolutifs.



Orchestration des conteneurs

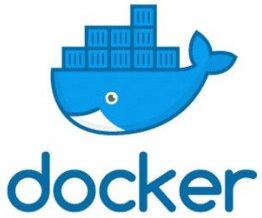


Intégration continue CI et Déploiement continu CD

Inconvénient

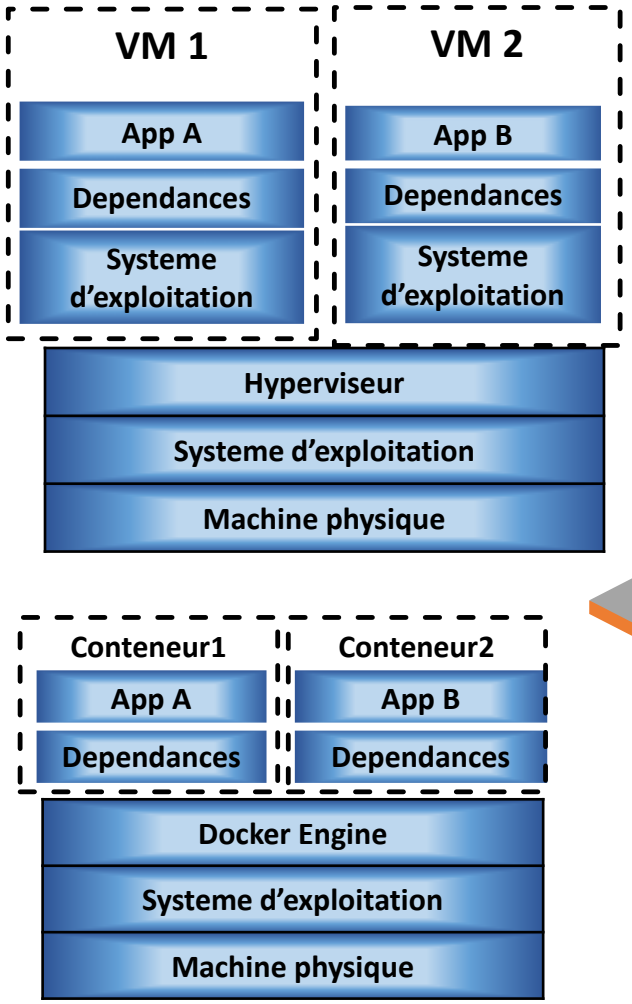
- La plateforme Docker présente quelques inconvénients.
  - Performance: Il peut être difficile de gérer de façon efficace un grand nombre de conteneurs simultanément, et les conteneurs Docker peuvent présenter une surcharge de performance vu qu'ils s'exécutent sur le même Host.
  - Sécurité : Les containers sont isolés, mais partagent le même système d'exploitation.
    - ✓ Une attaque ou une faille de sécurité sur l'OS peut compromettre tous les conteneurs.
    - ✓ Les images de conteneurs peuvent contenir des vulnérabilités si elles ne sont pas gérées correctement, qui peuvent être exploitées pour accéder au host ou aux autres conteneurs.
  - Taille des images : Les images de conteneur Docker peuvent être de taille très importante et occuper une taille disque très grande.
  - Dépendances : Les conteneurs Docker peuvent avoir des dépendances qui doivent être mises à jour régulièrement.

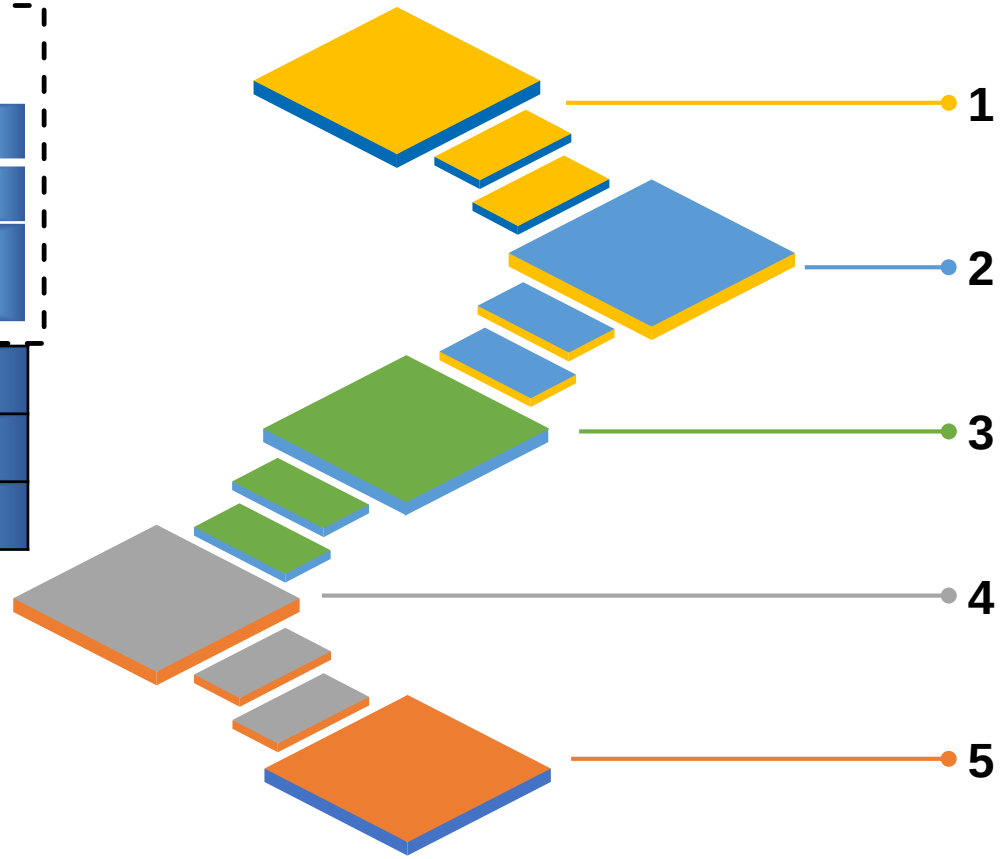
OpenVZ vs Docker



- OpenVZ voit un conteneur comme un vrai serveur virtuel VPS et Docker voit un conteneur comme une application ou un service.
- Les conteneurs Docker ne sont pas strictement liés à la même version du noyau que le système hôte
- Les VPS d'OpenVZ et les conteneurs LXC doivent avoir la même version du noyau que celle du système hôte.
- Les conteneur Docker fonctionnent sur n'importe quel hôte Docker indépendamment du système d'exploitation et de la configuration matérielle.
- Le moteur Docker ne prend en charge que son propre format de conteneur.
- les conteneurs openVZ peuvent contenir n'importe quel nombre de processus
- Les conteneurs Docker contiennent généralement une application avec ses dépendances.

Machines virtuelles Vs. Conteneurs





1 Architecture

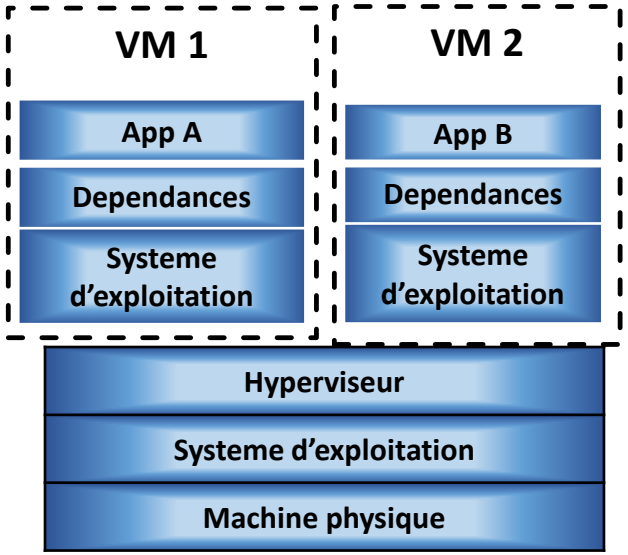
2 Consommation des ressources

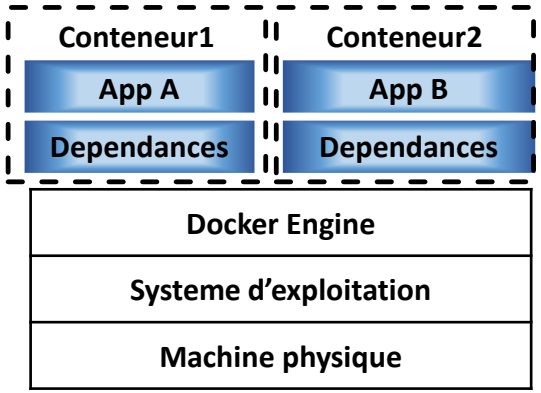
3 Taille

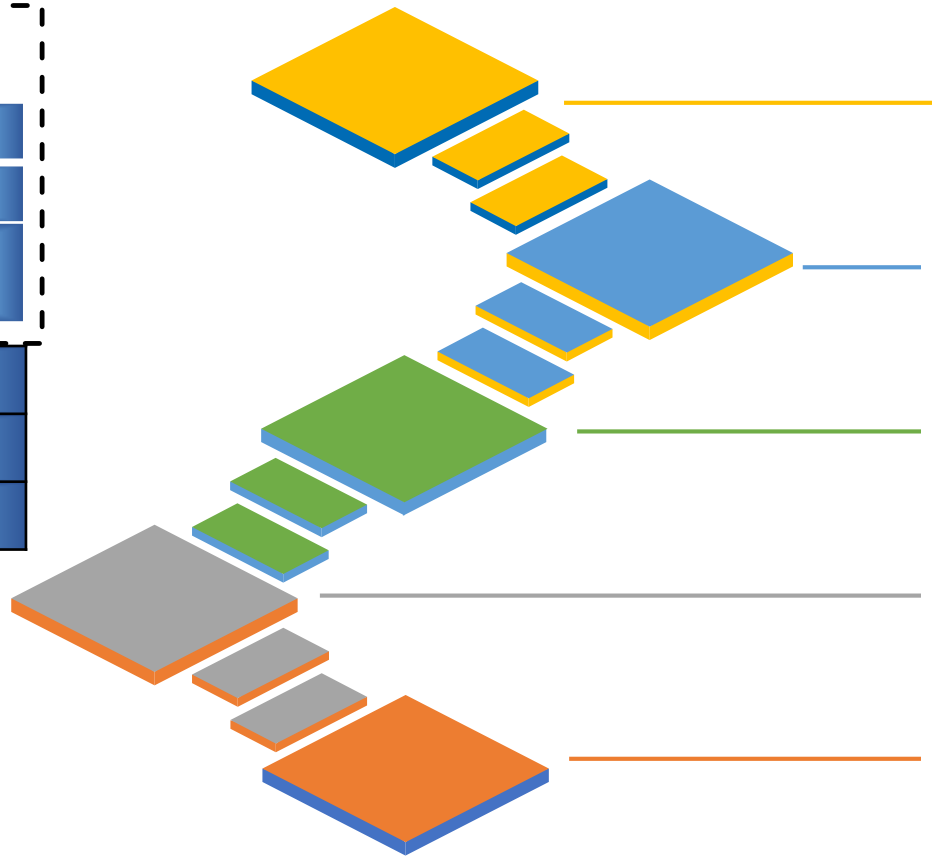
4 Déploiement

5 Isolation

Machines virtuelles Vs. Conteneurs



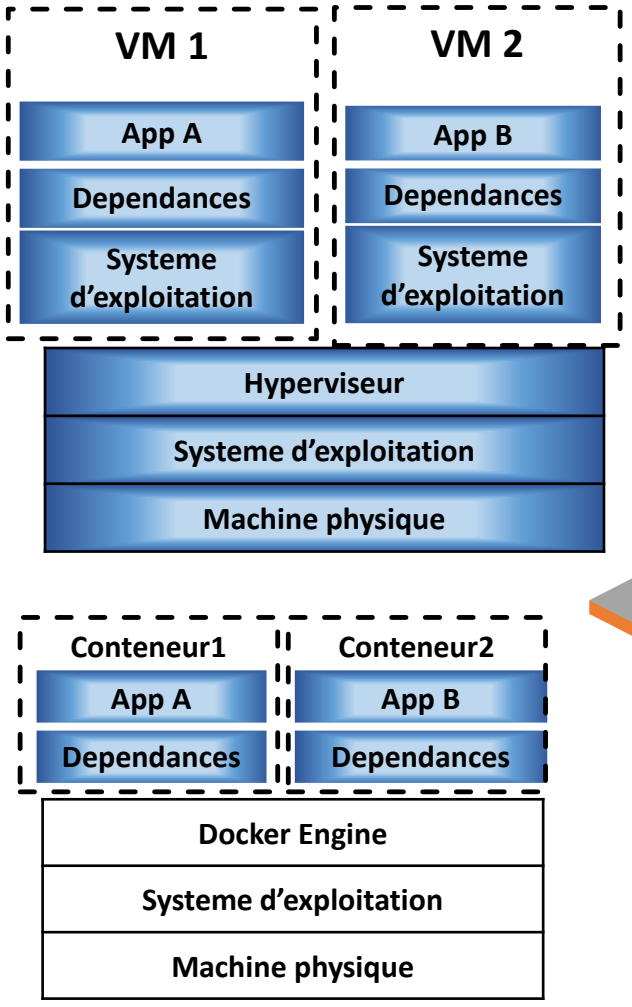


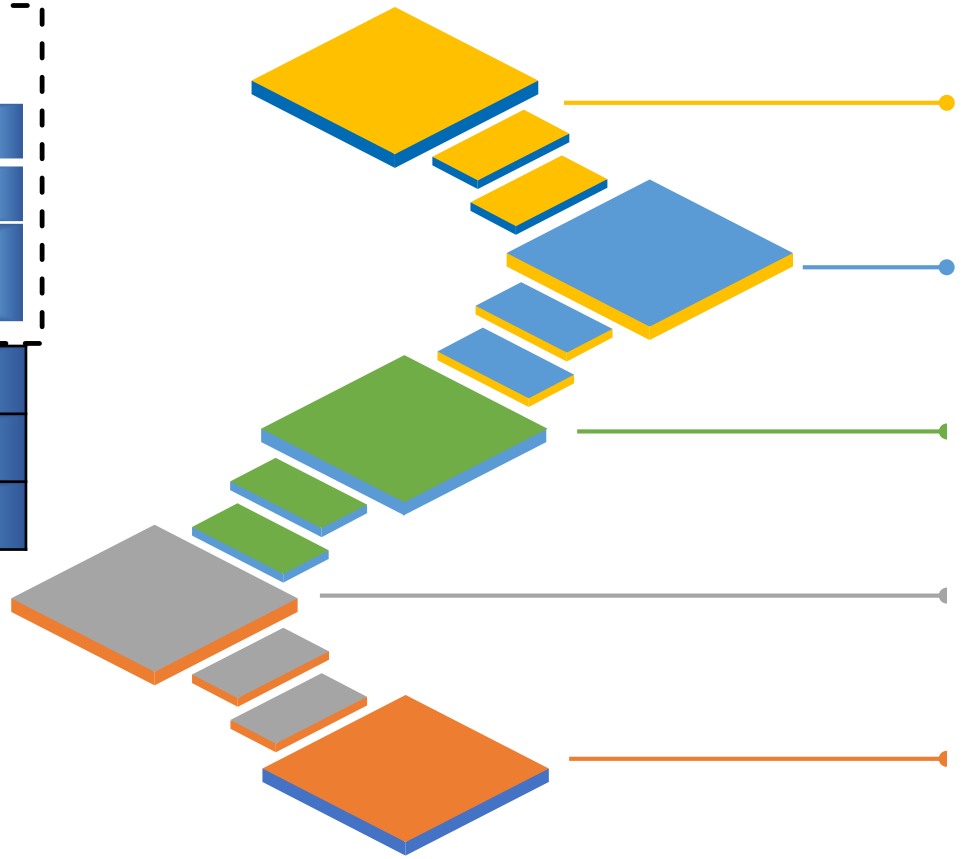


1 Architecture

Les machines virtuelles simulent une machine physique complète, avec son propre système d'exploitation, alors qu'un conteneur partage l'OS du hôte

Machines virtuelles Vs. Conteneurs



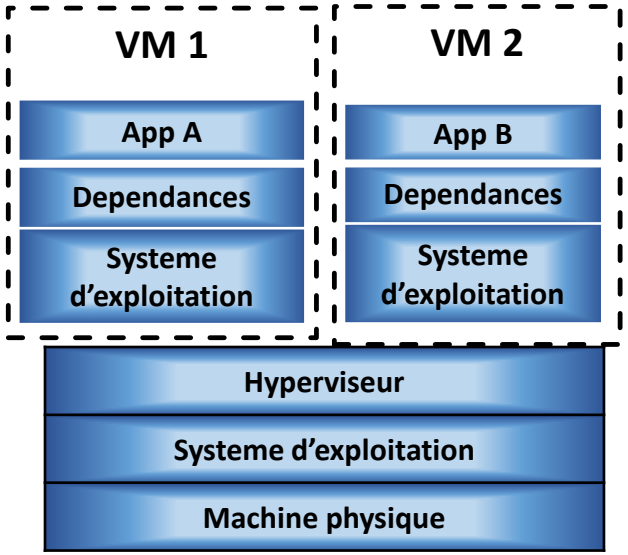


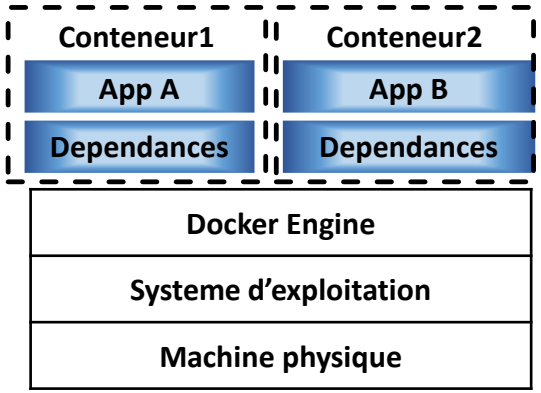
1 **Architecture**

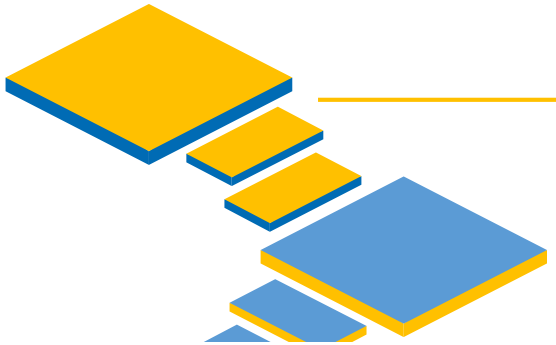
2 **Consommation des ressources**

Les machines virtuelles ont besoin d'un hyperviseur pour simuler les ressources matérielles nécessaires.

Machines virtuelles Vs. Conteneurs







Les machines virtuelles sont plus volumineuses et prennent plus de temps à déployer car elles nécessitent un système d'exploitation complet alors que les conteneurs Docker sont plus petits et plus rapides à déployer.

1 Architecture

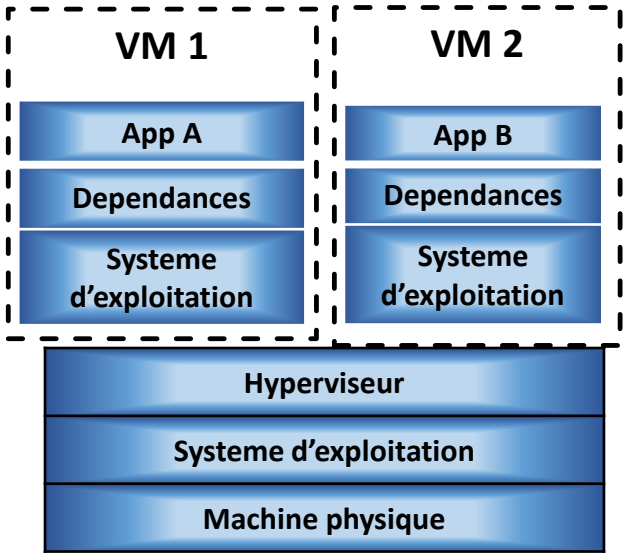
2 Consommation des ressources

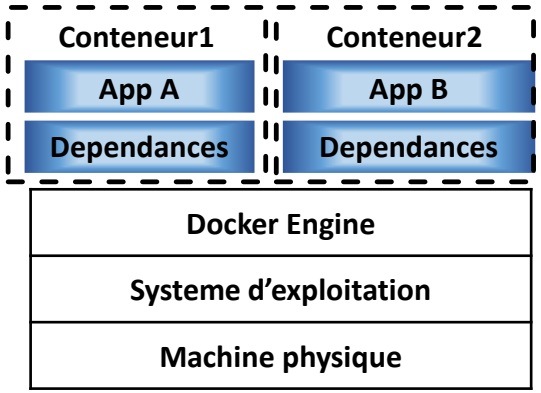
3 Taille

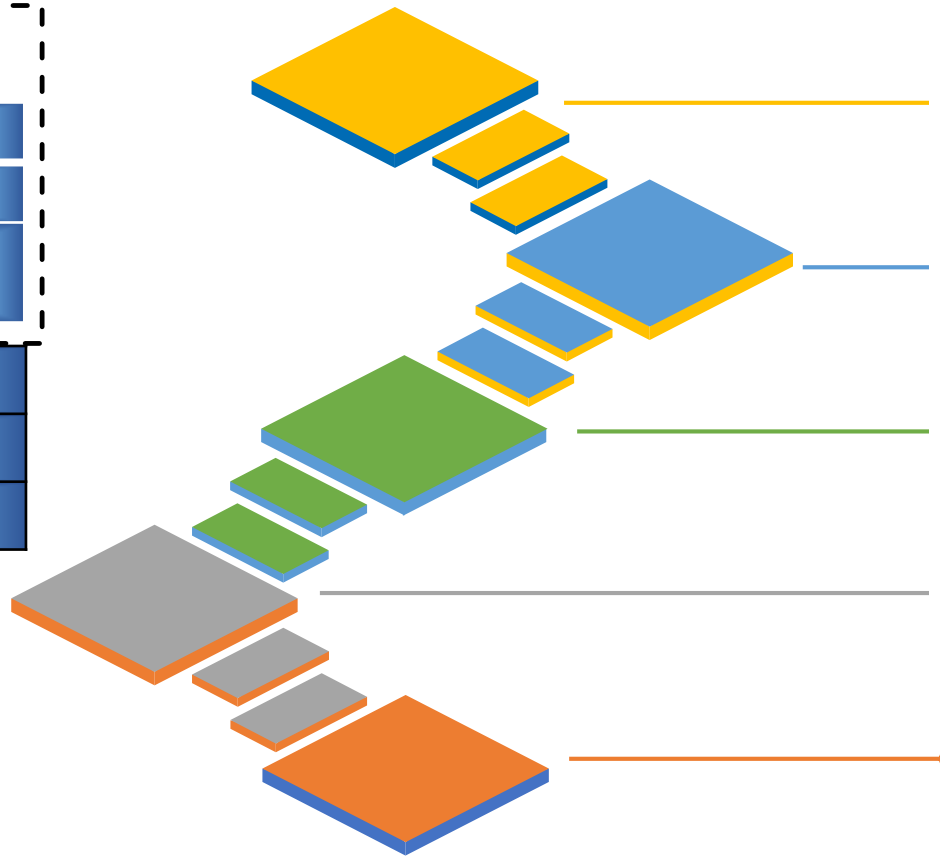
4 Déploiement

5 Isolation

Machines virtuelles Vs. Conteneurs



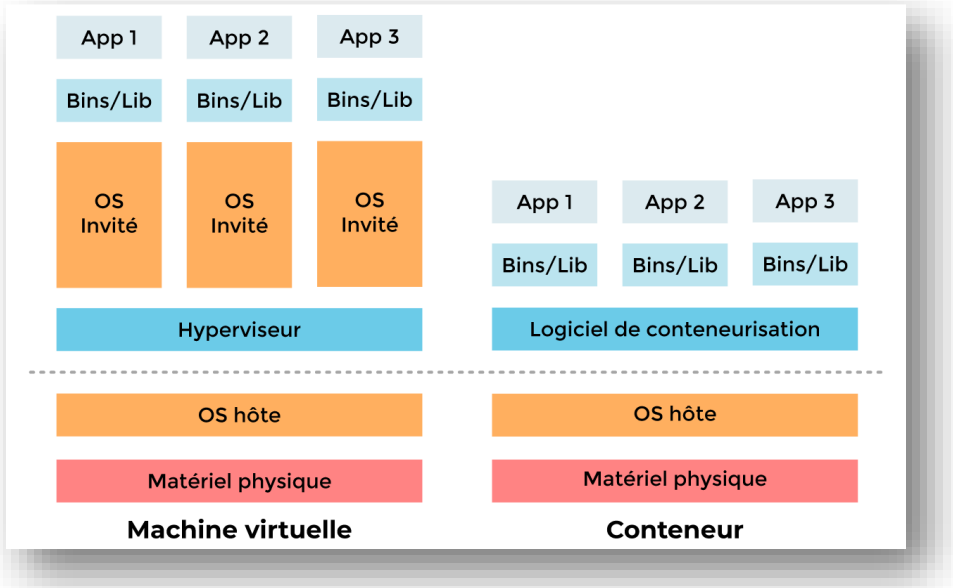




VM est complètement séparée de l'hôte et des autres machines virtuelles alors que
les conteneurs Docker offrent une isolation moins stricte, car ils partagent le même système d'exploitation que l'hôte Docker.

5 **Isolation**

VM vs Conteneur



Machine Virtuelle

- ✓ On virtualise la **machine**
- ✓ **Isolation totale** avec l'os hôte et les autres OS.
- ✓ Exécute un OS complet (Kernel inclus) donc on utilise plus de ressources système (CPU, mémoire…).
- ✓ OS hôte et OS invité indépendants.

Conteneurs

- ✓ On virtualise **l'environnement d'exécution des App** (proc, mémoire vive, sys de fichier).
- ✓ **L'isolation au niveau processus**
- ✓ Exécute uniquement les services nécessaires pour l'application qu'il contient.
- ✓ Les conteneurs démarrent plus rapidement qu'une VM

Virtualisation d'application basée sur le cloud

- SaaS: La virtualisation des applications basée sur le cloud est une approche de virtualisation dans laquelle les applications sont encapsulées dans des conteneurs et exécutées sur une infrastructure cloud.
- Les applications sont installées à distance sur un serveur virtuel ou physique au sein du datacenter
- Elles sont fournit comme service à la demande
- Des exemples de plateformes de ce genre il y a Amazon Web Services, Microsoft Azure et Google Cloud Platform .

Avantage de la virtualisation

- La consolidation des ressources
- Meilleure Flexibilité
- Portabilité
- Facilité des Migration des systèmes et applications
- Optimisation de la consommation d'énergie
- Monitoring très simple
- Licence avantageuse
- Reprise d'activité rapide suite à un désastre : Suite à un désastre, toutes les MVs peuvent être redéployées et la reprise d'activité peut être instantanée sur un autre DataCenter.
- Plus de sécurité et meilleure fiabilité : Il n'y a pas de risque de déstabiliser la machine physique

Risques liés à la virtualisation

- Les risques « classiques » d'un SI
  - les risques liés aux vulnérabilités des systèmes d'exploitation
  - les risques d'attaques basées sur le matériel
  - les risques liés à une administration à distance
- Les risques liés à un système virtualisé
  - les risques d'attaques d'un système virtuel
  - les risques liés aux vulnérabilités de la couche d'abstraction
  - les risques induits par la combinaison des deux

Exemple : manque de cloisonnement

- Les flux de données de chaque machine virtuelle sont traités par cette unique carte réseau.
- il n'est pas possible de garantir un cloisonnement des flux au niveau de la ressource partagée.
- En cas d'erreur ou de compromission de la carte réseau, un accès aux données des différents flux d'information est possible.