Partie 2 : Conteneurisation

Yaya DOUMBIA
Consultant supervising associate Cyber Risk & Security
yaya.doumbia@intervenants.efrei.net

Plan

1. Introduction à la Conteneurisation
2. Virtualisation Vs. Conteneurisation
3. Les Fondations Linux : Chroot, Cgroups, Namespace
4. Avantages et Inconvénients
5. Outils de Conteneurisation : LXC, Docker

Introduction à la Conteneurisation

Introduction à la Conteneurisation

Qu'est-ce que la conteneurisation ?

Il s'agit d'une forme de virtualisation du système d'exploitation dans laquelle vous exécutez des applications dans des espaces utilisateurs isolés appelés conteneurs qui utilisent le même système d'exploitation partagé.

- La conteneurisation permet de réduire les charges au démarrage et de supprimer la nécessité de configurer des systèmes d'exploitation invités distincts pour chaque application, car ils partagent tout un seul noyau de système d'exploitation.

- En raison de cette efficacité élevée, les développeurs de logiciels utilisent couramment la conteneurisation des applications pour regrouper plusieurs micro-services individuels constituant les applications modernes.

Introduction à la Conteneurisation

Qu'est-ce que la conteneurisation ?

Un conteneur d'applications est un environnement informatique entièrement regroupé en package et portable

- Il dispose de tout ce dont une application a besoin pour s'exécuter : fichiers binaires, bibliothèques, dépendances et fichiers de configuration, le tout encapsulé et isolé dans un conteneur.
- La conteneurisation d'une application permet d'isoler le conteneur du système d'exploitation hôte, avec un accès limité aux ressources sous-jacentes, à l'instar d'une machine virtuelle légère.
- Vous pouvez exécuter l'application conteneurisée sur différents types d'infrastructure, tels qu'un serveur bare-metal, dans le cloud ou sur des VM, sans la remanier pour chaque environnement.

Introduction à la Conteneurisation

À quoi sert la conteneurisation ?

La conteneurisation permet aux développeurs de logiciels de créer et de déployer des applications de façon plus rapide et plus sécurisée.

- Avec les méthodes traditionnelles, vous codez dans un environnement informatique spécifique, ce qui entraîne souvent des erreurs et des bogues lorsque vous transférez ce code dans un nouvel emplacement. Par exemple, lorsque vous transférez du code de votre ordinateur de bureau vers une machine virtuelle, ou d'un système d'exploitation Windows vers Linux.

- La conteneurisation élimine ce problème en vous permettant de regrouper le code de la demande avec les fichiers de configuration, les dépendances et les bibliothèques associées. Vous pouvez ensuite isoler ce package de logiciels unique (conteneur) du système d'exploitation hôte, ce qui lui permet d'être **autonome** et de devenir **portable**, c'est-à-dire de s'exécuter sur n'importe quelle plate-forme ou n'importe quel cloud sans aucun problème.

Introduction à la Conteneurisation

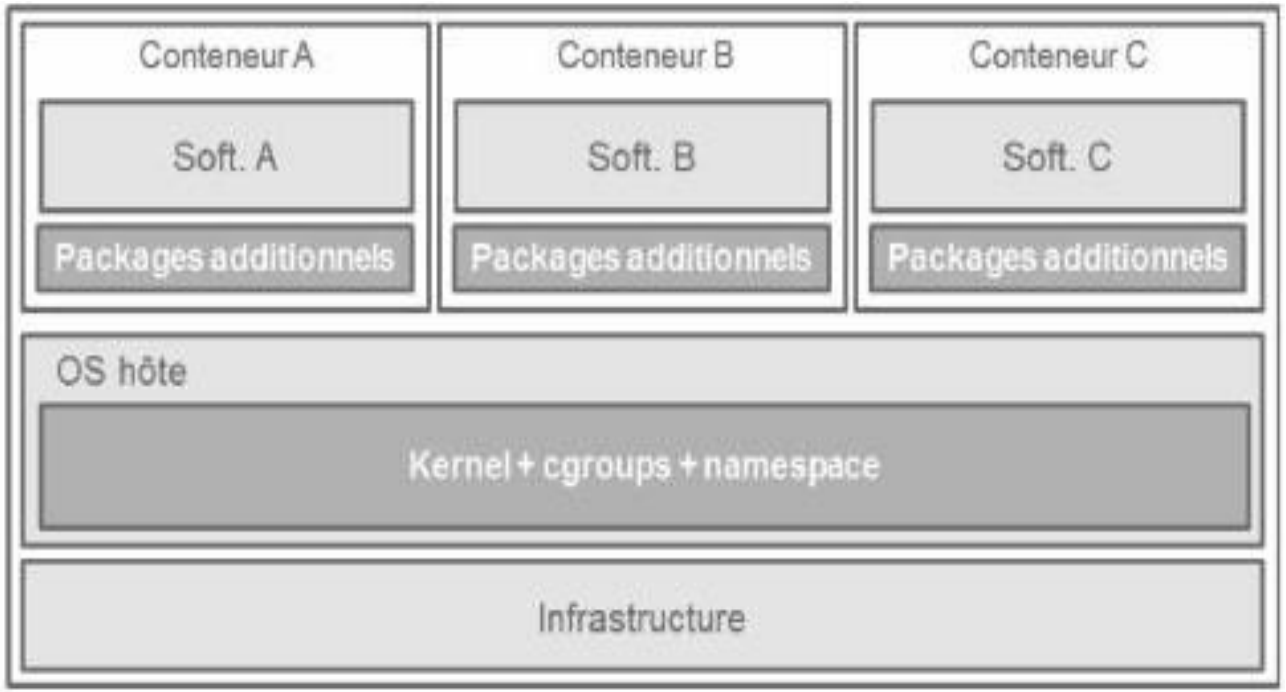
À quoi sert la conteneurisation ?

Pour faire simple, la conteneurisation permet aux développeurs d'écrire des applications une seule fois et de les exécuter partout. Ce niveau de portabilité est essentiel pour développer la compatibilité des processus et des fournisseurs. Elle présente également d'autres avantages, comme l'isolation des pannes, la sécurité et la facilité de gestion.

- L'industrie qualifie souvent les conteneurs de légers, ce qui signifie qu'ils partagent le noyau du système d'exploitation de la machine et ne nécessitent pas l'association d'un système d'exploitation à chaque application, comme c'est le cas avec la virtualisation.

- Par conséquent, les conteneurs ont une capacité intrinsèquement plus petite qu'une machine virtuelle et nécessitent moins de temps de démarrage, ce qui permet l'exécution de plus de conteneurs sur une seule capacité de calcul en tant que machine virtuelle unique. Ainsi, l'efficacité des serveurs est augmentée et les coûts des serveurs et des licences sont réduits.

Introduction à la Conteneurisation



Conteneurisation VS Virtualisation

Conteneurisation VS Virtualisation

Pourquoi la conteneurisation est-elle nécessaire maintenant ?

- La conteneurisation est devenue le dernier mot à la mode dans le domaine du cloud computing, et beaucoup pensent qu'elle peut aider à moderniser les systèmes existants en créant de nouvelles applications évolutives conçues dans le cloud.

- Pour comprendre la nécessité et la nature de la conteneurisation, commençons par la virtualisation et l'utilisation croissante des machines virtuelles (VM) dans le cloud.

  ⇒ En général, presque toutes les entreprises utilisent l'environnement cloud (public ou privé), avec des instances exécutant des VM avec des capacités d'évolutivité et d'équilibrage de charge représentant leur couche de calcul.

  ⇒ Cependant, les approches de virtualisation ont fait face à quelques défis qui ont rendu ces environnements inefficaces.

Conteneurisation VS Virtualisation

Cependant, les approches de virtualisation ont fait face à quelques défis qui ont rendu ces environnements inefficaces. En voici quelques exemples :

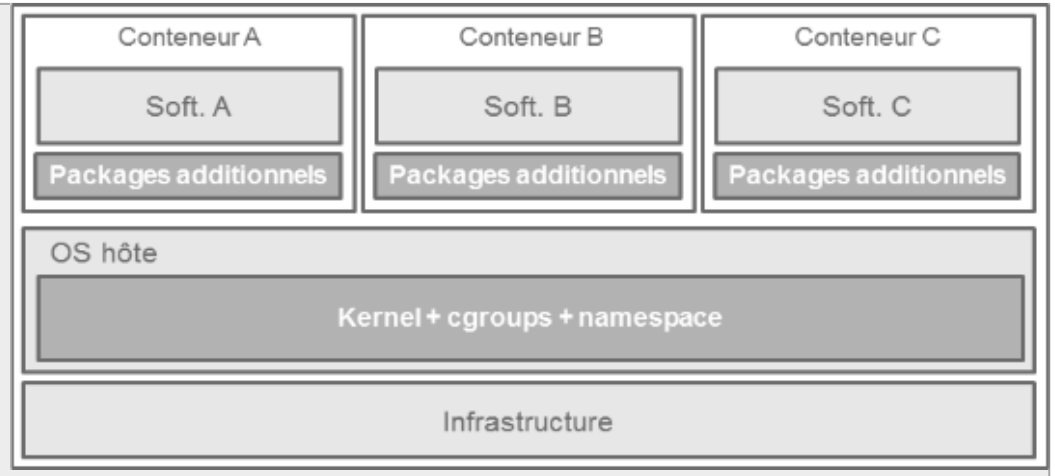
- **Manque de cohérence des environnements** : Déploiement d'applications et de packages dans des environnements virtuels;
- **Dépendance du système d'exploitation** : Les applications déployées sont exécutées uniquement sur des systèmes d'exploitation compatibles;
- **Niveau d'isolement** : Incapacité à fournir un sandbox instantané au-dessus du niveau du système d'exploitation;
- **Granularité de la consommation de calcul** : Impossibilité de déployer plusieurs applications répliquées, alors que l'équilibrage de la charge sur la couche applicative ne se produit qu'au sein d'une seule machine et non au niveau de la couche du système d'exploitation;
- **Correctifs des images dans les environnements de production** : Les déploiements canari et bleu-vert ne sont pas flexibles au niveau du cluster et sont difficiles à gérer dans plusieurs régions;

Comment pouvez-vous résoudre ces problèmes de virtualisation ? => La réponse est la conteneurisation.

Conteneurisation VS Virtualisation

Virtualisation basée Conteneur

- ❑ Couche de virtualisation sans hyperviseur
  => Permet le fonctionnement de plusieurs applications dans un environnement isolé ou limité.
- ❑ Les applications conteneurisées communiquent directement avec le système hôte et sont capables ainsi de s'exécuter sur le système d'exploitation hôte et sur l'architecture du CPU.



Architecture à Conteneurs : 3 conteneurs sur même hôte

- ✓ Démarrage facile au bout de quelques secondes : ne nécessite pas une émulation du hardware et le démarrage d'un système d'exploitation.
- ✓ Taille des applications conteneurisées (images) très petite vis à vis celle d'une image d'une VM.
- ✓ La portabilité des conteneurs est une approche intéressante dans le développement des applications.
- ✓ Gestion facile des projets dans une machine : pas de conflit de dépendance.

Conteneurisation VS Virtualisation

Virtualisation basée Conteneur

- ❑ Une architecture qui permet au développeur d'embarquer l'ensemble des dépendances logicielles et de les isoler.
- ❑ Stockage très léger : le conteneur s'appuie sur le noyau du système hôte.
- ❑ Le nombre de conteneurs qu'un système hôte peut encapsuler est nettement très supérieur que son équivalent en VMs.
- ❑ Côté Cloud : offre des performances meilleures
- ❑ Côté entreprise : remplacer les solutions couteuses (hyperviseurs) par des conteneurs.

Conteneurisation VS Virtualisation

La virtualisation permet à plusieurs systèmes d'exploitation et applications logicielles de fonctionner simultanément tout en partageant les ressources physiques d'un ordinateur.

- Par exemple, vous pouvez exécuter les systèmes d'exploitation Linux et Windows ainsi que plusieurs applications sur le même serveur. Les développeurs regroupent chaque application et ses fichiers, dépendances et bibliothèques (y compris une copie du système d'exploitation) sous forme de machine virtuelle.

- Lorsque vous exécutez plusieurs machines virtuelles sur une seule machine physique, vous pouvez réaliser d'importantes économies sur les coûts d'investissement initial, d'exploitation et d'énergie

Conteneurisation VS Virtualisation

La conteneurisation utilise les ressources informatiques de manière efficace. Un conteneur crée un progiciel exécutable qui regroupe le code de l'application et tous les fichiers de configuration, dépendances et bibliothèques dont il a besoin pour fonctionner dans un environnement portable et défini par le logiciel.

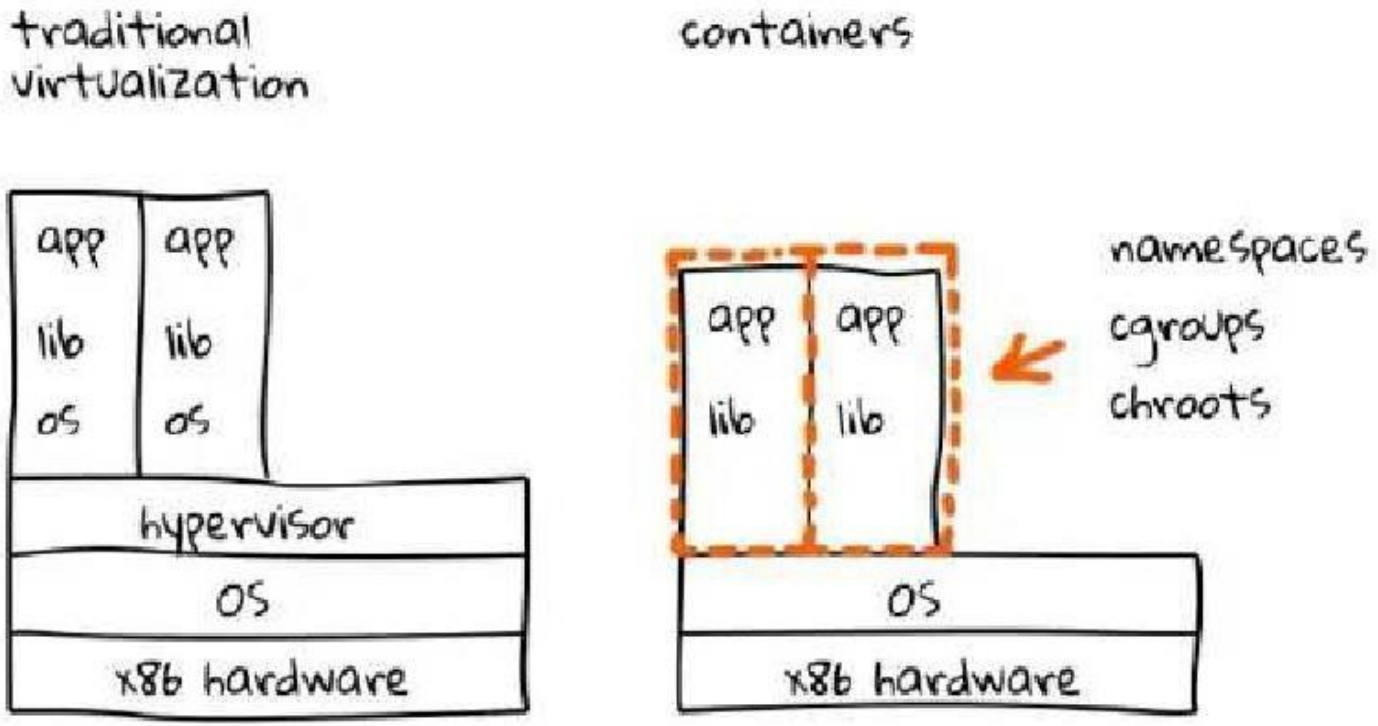
- Cependant, contrairement aux VM, les conteneurs ne sont pas regroupés dans une copie du système d'exploitation.

- Au lieu de cela, les développeurs installent leurs moteurs d'exécution sur le système d'exploitation du système hôte pour transformer celui-ci en relais et permettre à tous les conteneurs de partager un système d'exploitation similaire.

Les fondations Linux (Chroot, Cgroups, Namespace)

Fondations Linux

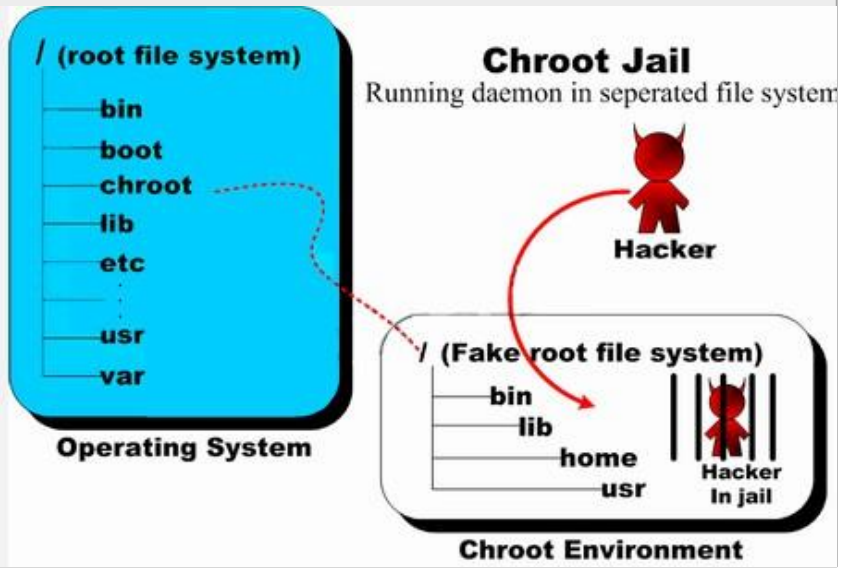
La conteneurisation utilise les propriétés existantes (fonctionnalités) du noyau du système d'exploitation pour créer un environnement isolé pour les processus.



La commande “jail” :: chroot

Définition : chroot (change root) (1979) — Commande qui permet le changement du répertoire racine d’un processus et de ses processus fils (sous-processus)

- chroot fournit un espace disque isolé pour chaque processus;
- chroot peut prévenir les logiciels malveillants de l’accès à des fichiers à l’extérieur de l’espace créé (faux répertoire /)
- chroot n’offre aucun autre mécanisme d’isolation processus à part le changement du répertoire racine (/) du processus.



Fondations Linux

Contrôle de ressources : cgroups (Control Groups)

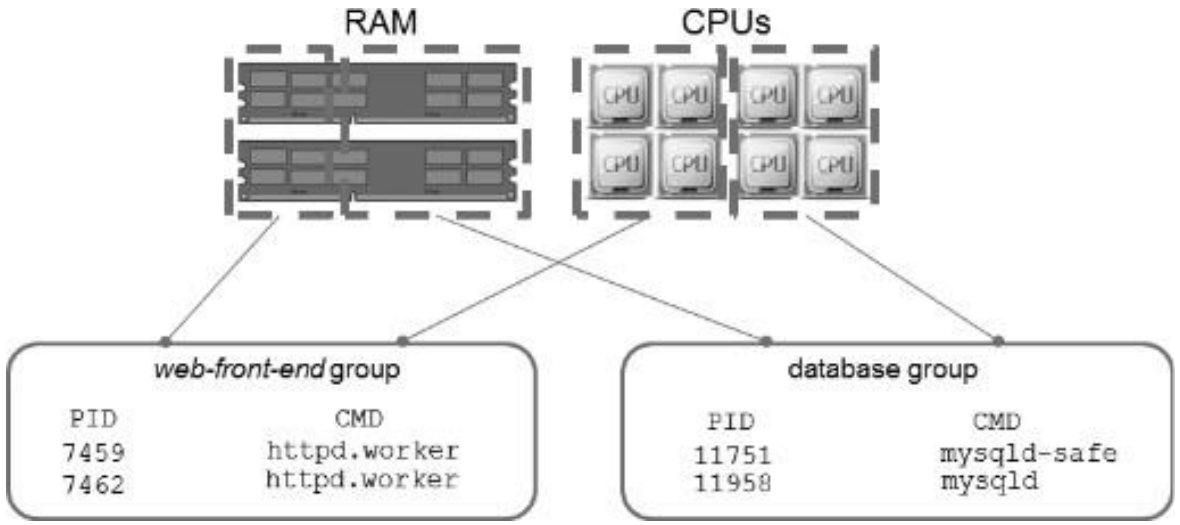
Définition : une fonctionnalité du noyau Linux pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.

- cgroups (2006) fournit un mécanisme pour rassembler des processus dans un groupe et limiter leur accès aux ressources système.
- permet de partitionner les ressources d'un hôte.
- L'objectif est de contrôler la consommation des ressources par processus.
- Les ressources sont : Temps CPU, Mémoire système Disque dur (stockage), Bande passante (réseau).

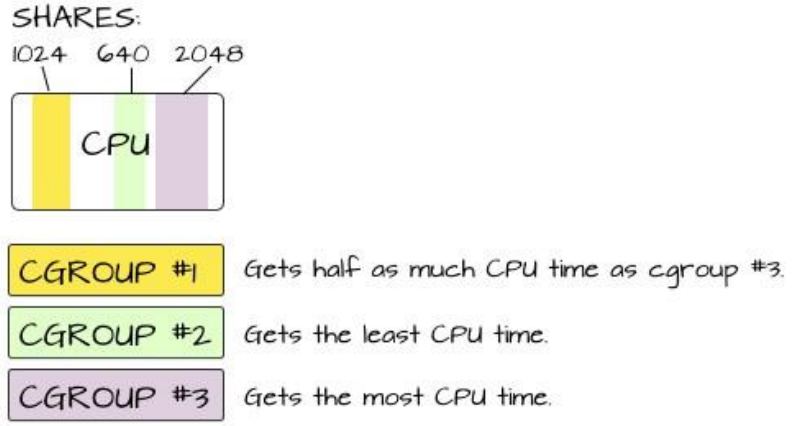
Contrôle de ressources : cgroups (Control Groups)

Prenons par exemple une machine sur laquelle serait installée une application web avec un front-end PHP et une base de données MySQL.

=> CGroups permettrait de répartir la puissance de calcul (CPU) et la mémoire disponible entre les différents processus, afin de garantir une bonne répartition de la charge (et donc probablement des temps de réponse)



Répartition des Ressources avec Cgroups



Les espaces de nom: namespaces

Définition : Les namespaces (ou "espaces de noms") isolent les ressources partagées. Ils donnent à chaque processus sa propre vue unique du système, limitant ainsi leur accès aux ressources système sans que le processus en cours ne soit au courant des limitations.

- Permettent de disposer d'un environnement distinct pour : PID, réseau, points de montage, utilisateurs et groupes, etc.
  - Par défaut, chaque système Linux possède un unique espace de nom (Namespace);
  - Toutes les ressources système (système de fichiers, processus PID, IDs des utilisateurs, interfaces réseau, etc) appartiennent à cet espace de nom.
  - On peut créer des espaces de nom additionnels pour organiser les ressources à l'intérieur.

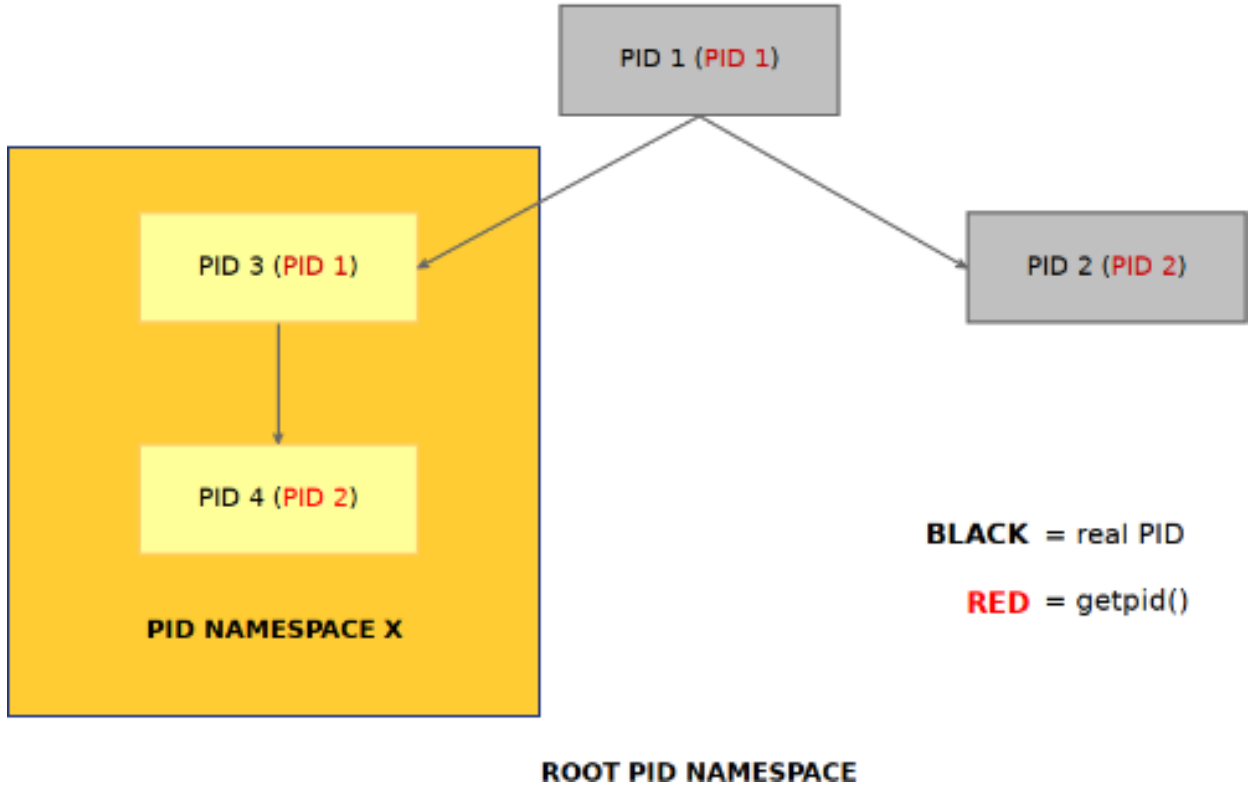
Une utilisation importante des espaces de nom, "namespaces", est l'implémentation d'une virtualisation légère (conteneurisation)

Fondations Linux

Les espaces de nom: namespaces

Tableau 1.1 — Namespaces Linux

Namespace	Isole
IPC	Communication interprocessus
Network	Terminaux réseau, ports, etc.
Mount	Point de montage (système de fichiers)
PID	Identifiant de processus
User	Utilisateurs et groupes
UTS	Nom de domaines



Fondations Linux

Qu’est ce qu’un container au final ?

Un conteneur est un système de fichiers (RootFS) sur le quel s’exécutent des processus en respectant les règles suivantes

- ❑ L’isolation : les conteneurs ne se voient pas les uns des autres;
- ❑ La limite en terme de ressources grâce à Cgroups.

Container Runtime : environnement d’exécution des conteneurs.

Avantage et Inconvénients

Avantages et Inconvénients

Avantages

1) **Portabilité**

- ❑ Un conteneur d'application crée un progiciel exécutable qui est isolé par rapport au système d'exploitation hôte. Ainsi, il ne dépend pas du système d'exploitation hôte et n'est pas lié à celui-ci, ce qui le rend portable et lui permet de s'exécuter de manière cohérente et uniforme sur n'importe quelle plate-forme ou cloud.

- ❑ Les méthodes de consolidation du système d'exploitation utilisées par les développeurs permettent également d'éviter les incohérences telles que l'intégration qui cherche à entraver la fonctionnalité de l'application.

Avantages et Inconvénients

Avantages

2) Vitesse

- La légèreté des conteneurs permet d'améliorer l'efficacité des serveurs et de réduire les coûts liés aux serveurs et aux licences.
- Elle réduit également le temps de lancement, car il n'y a pas de système d'exploitation à démarrer.
- L'utilisation d'un conteneur Docker vous permet de créer une version principale d'une application (image) et de la déployer rapidement sur demande.

Avantages et Inconvénients

Avantages

3) Evolutivité

- La conteneurisation offre une grande évolutivité. Un conteneur d'application peut gérer des charges de travail croissantes en reconfigurant l'architecture existante afin d'activer les ressources à l'aide d'une conception d'application orientée vers les services. Par ailleurs, un développeur peut ajouter d'autres conteneurs dans un cluster d'ordinateurs distribués.

- Un environnement conteneur permet l'ajout de nouvelles fonctions, mises à jour et caractéristiques instantanément sans que cela affecte les applications d'origine. Par conséquent, les conteneurs permettent l'évolutivité des applications avec une utilisation des ressources minimale.

Avantages et Inconvénients

Avantages

4) Agilité

- Le Docker Engine, qui permet l'exécution des conteneurs, est devenu la norme de l'industrie en matière de conteneurs d'applications grâce à des outils de développement simples et à une approche universelle fonctionnant avec les systèmes d'exploitation Windows et Linux.

- Les développeurs peuvent donc continuer à utiliser les outils et les processus DevOps pour accélérer le développement et l'amélioration des applications.

Avantages et Inconvénients

Avantages

5) Efficacité

- Étant donné que les logiciels exécutés dans les environnements conteneurisés partagent le noyau du système d'exploitation de la machine hôte, les développeurs peuvent partager les couches d'application entre les conteneurs.

- De plus, les conteneurs ont une capacité intrinsèquement plus petite que les machines virtuelles.

- Leur temps de démarrage est réduit, ce qui permet aux développeurs d'exécuter plus de conteneurs sur la même capacité de calcul qu'une machine virtuelle unique. Ainsi, l'efficacité des serveurs est augmentée et les coûts des serveurs et des licences sont réduits.

Avantages et Inconvénients

Avantages

6) Sécurité

- L'isolation des applications via les conteneurs empêchent le code malveillant d'affecter d'autres applications conteneurisées ou le système hôte. Vous pouvez également définir des autorisations de sécurité pour bloquer automatiquement l'accès aux composants indésirables qui cherchent à s'introduire dans d'autres conteneurs ou à limiter les communications.

- L'isolation des applications aide les développeurs à partager des fonctionnalités supplémentaires sans facteur de risque. Par exemple, si vous travaillez avec une équipe de développement à l'extérieur de votre réseau, vous pouvez partager les ressources nécessaires sans que les informations critiques ne se trouvent dans votre réseau.

Avantages et Inconvénients

Avantages

7) Facilité de Gestion

- En utilisant une plate-forme d'orchestration des conteneurs, vous pouvez automatiser l'installation, la gestion et l'évolution des charges de travail et des services conteneurisés.

- L'orchestration des conteneurs permet de faciliter les tâches de gestion, comme le déploiement de nouvelles versions d'applications, l'évolution d'applications conteneurisées ou la mise à disposition de fonctions de surveillance, de consignation et de débogage.

Avantages et Inconvénients

Avantages

8) Continuité des Opérations

- ❑ Différents conteneurs s'exécutent indépendamment, de sorte que la défaillance de l'un n'aura pas d'impact sur la continuité des autres.

- ❑ Les équipes de développement bénéficient de la flexibilité nécessaire à la correction des erreurs dans un conteneur sans provoquer l'arrêt des autres. Par conséquent, la conteneurisation assure la continuité des opérations.

Avantages et Inconvénients

Avantages

9) Facilité d'Utilisation

- ❑ Les conteneurs sont conviviaux pour les développeurs, car il est possible d'utiliser un seul environnement pour le développement et la production (un obstacle courant dans le développement d'applications web => écrire une app sur Windows, mais ne s'exécute pas sur Mac).

- ❑ Avec la conteneurisation, l'image que votre équipe construit localement est la même que celle exécutée en production. Lorsqu'elles sont combinées à un workflow approprié, les applications conteneurisées peuvent aider à minimiser les bogues liés au changement d'emplacement.

Avantages et Inconvénients

Inconvénients

Les conteneurs ne sont pas parfaits et ont leurs inconvénients et leurs limites.

- Tout d'abord, une quantité étonnamment élevée de travail de mise en place est nécessaire pour développer et lancer une stratégie de conteneurisation et la gérer efficacement.
- Il n'y a pas assez de prise en charge et de dépendance pour les applications, et malgré les technologies émergentes dans le domaine, il n'existe toujours aucune solution complète pour le moment.
- En outre, il n'y a pas assez d'experts qualifiés, compétents et expérimentés dans ce domaine.

Avantages et Inconvénients

Inconvénients

En augmentant la flexibilité des applications, la conteneurisation apporte de la complexité de différentes manières. Cette complexité peut être liée à la sécurité, à l'orchestration, à la surveillance et au stockage des données

- **Sécurité** : par rapport aux VM traditionnelles, les conteneurs présentent un risque de sécurité potentiellement plus important. Ils ont besoin d'une sécurité multi niveau, car ils ont plusieurs couches. Par conséquent, vous devez sécuriser l'application conteneurisée ainsi que le registre, le daemon Docker et le système d'exploitation hôte.

- **Orchestration** : vous pouvez utiliser un seul outil d'orchestration pour les machines virtuelles, fourni avec une solution virtualisée (par exemple, un outil d'orchestration VMware pour VMware). Toutefois, lorsqu'il s'agit de conteneurs, vous devez choisir parmi différents outils d'orchestration comme Kubernetes, Mesos ou Swarm.

Avantages et Inconvénients

Inconvénients

- **Stockage** : le stockage des données sur les VM est simple, mais se complexifie lorsqu'il s'agit de conteneurs. Pour les données persistantes du conteneur, vous devez les déplacer hors du conteneur d'application vers le système hôte ou vers un endroit disposant d'un système de fichiers persistant. La conception des conteneurs est à l'origine de la perte de données. En effet, les données du conteneur peuvent disparaître à jamais une fois celui-ci supprimé, à moins d'en faire une sauvegarde ailleurs.

- **Surveillance** : il est également essentiel de surveiller les conteneurs pour détecter les problèmes de performance et de sécurité. Vous pouvez utiliser différents outils de surveillance, services de surveillance externes et analyses pour faire face à ce problème. L'environnement cloud étant complexe, vous devez surveiller de près les problèmes de sécurité.

Outils de Conteneurisation

Outils de Conteneurisation

==> Historique

- 1979 : Unix V7 chroot
- 2000 : FreeBSD Jails
- 2001 : LinuxVServer
- 2004 : Solaris Containers
- 2005 : OpenVZ
- 2006 : cgroups
- 2008 : namespaces
- 2008 : **LXC**
- 2011 : Warden (Cloudfoundry)
- 2013 : Lmctfy (google)
- 2013 : **Docker**

Outils de Conteneurisation

==> LXC (Linux Container)

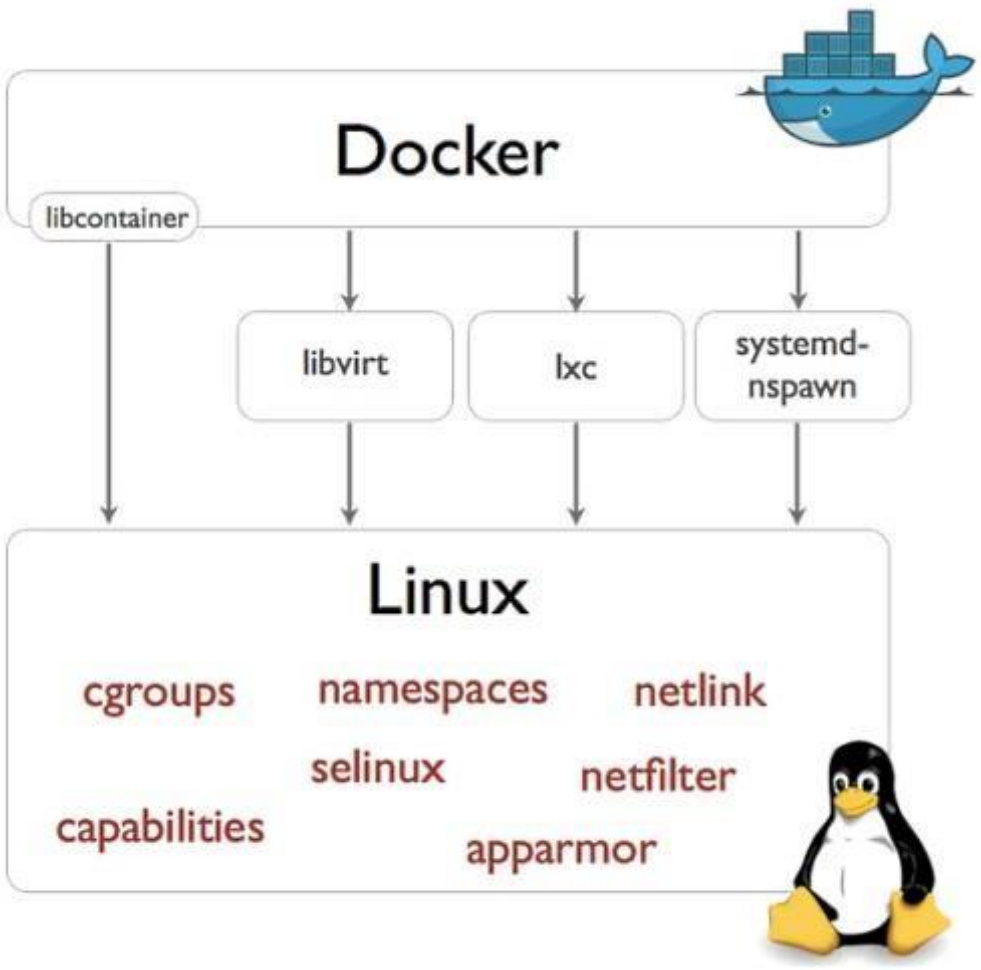
LXC est un projet en open source soutenu financièrement par Canonical.

- Site web : http://linuxcontainers.org/
- Plus de 60 contributeurs
- Mainteneurs : Serge Hallyn(Ubuntu), Stéphane Graber(Ubuntu)
- Intégré au noyau 2.6.24 : création des instances isolées baptisées “Conteneur Linux” ou LXC (LinuX Containers)
- LXC utilise la notion de contrôle de ressource “cgroups” et propose également une isolation au niveau des espaces de nom (Namespaces) et les modules ‘Application Armor’ (AppArmor) et Security-Enhanced Linux (SELinux).
- LXC est une technologie de conteneurisation qui offre des conteneurs Linux légers

Outils de Conteneurisation

==> Docker

- **Docker** est un projet open source écrit en GO et hébergé sur GitHub (https://github.com/docker).
- Le projet a été initialement développé par la startup DotCloud en 2013.
- Docker a utilisé LXC à ses débuts et qui a été remplacé par la suite par sa propre bibliothèque, libcontainer.



Outils de Conteneurisation

Docker Vs. LXC

- **Docker** restreint l'utilisation du conteneur à une seul service ("one service per container" philosophy) : il faut construire X conteneurs pour une application qui consiste de X services.
  ⇒ On peut utiliser un seul conteneur Docker pour lancer plusieurs services via un script mais il est généralement recommandé d'utiliser un service par conteneur.
- **LXC** peut contenir plusieurs processus à l'intérieur d'un conteneur;
- **Docker** construit sur LXC pour y ajouter des services de gestion et de déploiement d'images.
  - **Portabilité** des conteneurs entre deux machines;
  - **Automatisation** : création des conteneurs à partir des fichiers Dockerfile;
  - Docker est un outil de création et distribution d'application portable via le **concept d'image**;
- l'objectif principal de **Docker** est de faire le paquetage, la conteneurisation, la distribution et l'exécution des applications autant de fois, n'importe où et à tout moment.

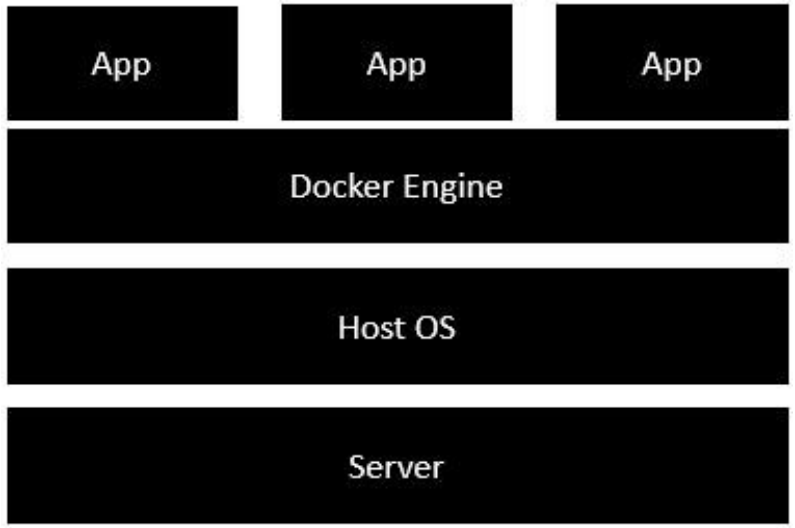
Outils de Conteneurisation

==> Docker

L'image suivante montre la nouvelle génération de virtualisation activée via **Dockers**.

- **Server** : est le serveur physique utilisé pour héberger plusieurs machines virtuelles.
- **Host OS** : est une machine de base telle que Linux ou Windows
- Au-dessus du système d'exploitation, on déploie le moteur Docker ou **Docker Engine** .
- Puis on crée un conteneur où on exécute les **applications**.

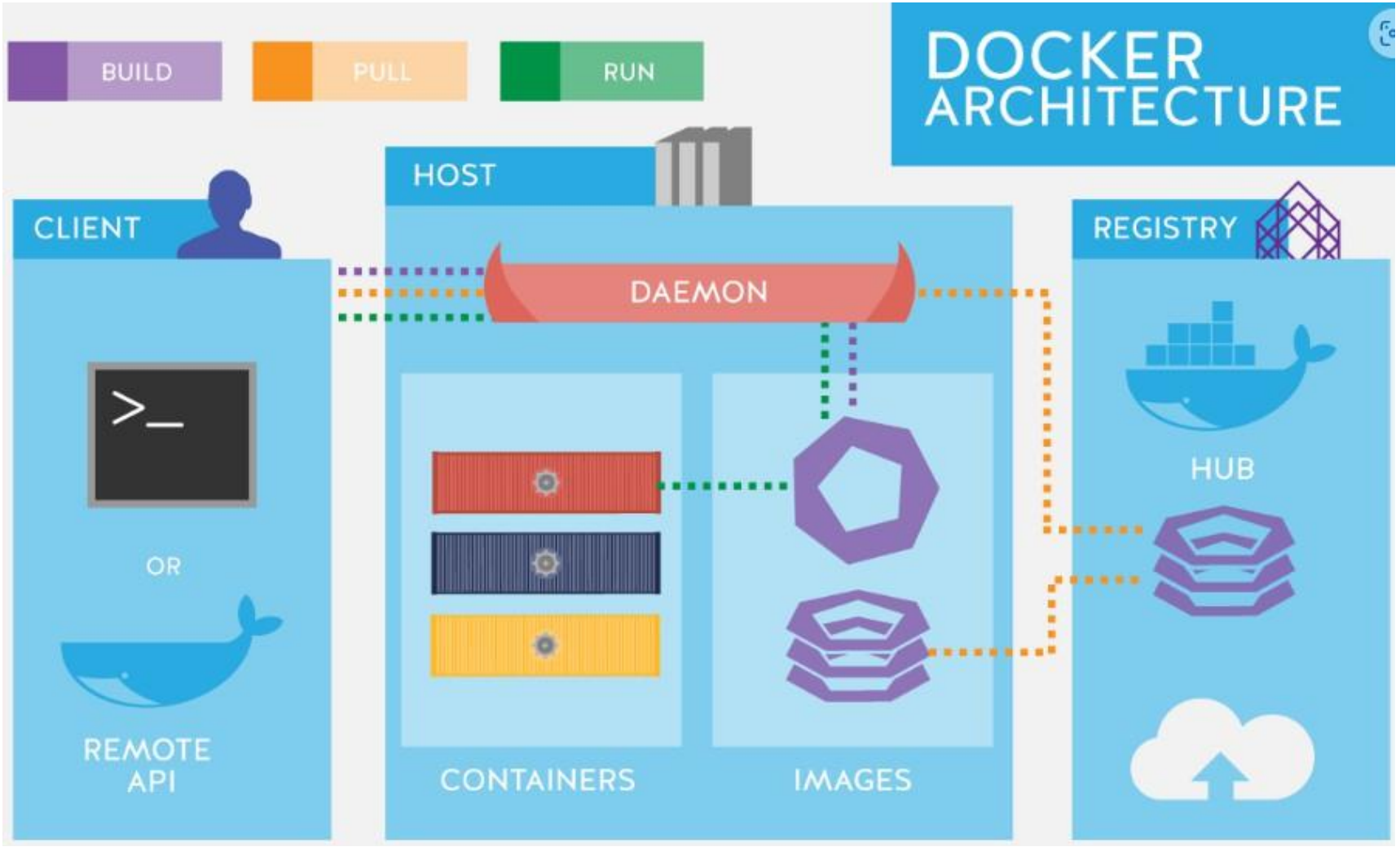
Le principal avantage de cette architecture est que vous n'avez pas besoin de matériel supplémentaire pour le système d'exploitation invité.



Outils de Conteneurisation

==> Docker

Architecture Docker



Outils de Conteneurisation

==> Docker

Architecture Docker

Docker Deamon (dockerd) écoute les requêtes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux et les volumes. Un démon peut également communiquer avec d'autres démons pour gérer les services Docker.

Docker Client (docker) est le principal moyen utilisé par de nombreux utilisateurs Docker pour interagir avec Docker. C'est une application cliente sous forme d'interface de ligne de commande (CLI). Lorsque vous utilisez des commandes telles que docker run, le client envoie ces commandes à dockerd, qui les exécute. Docker Client peut communiquer avec plusieurs démons.

Container est une instance exécutable d'une image. Vous pouvez créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de l'API ou de la CLI Docker. Vous pouvez connecter un conteneur à un ou plusieurs réseaux, y attacher un stockage ou même créer une nouvelle image en fonction de son état actuel.

Docker Engine est une application client-serveur avec ces composants majeurs : un serveur (dockerd), une API REST et une interface cliente de ligne de commande (CLI : Client Line Interface).

Outils de Conteneurisation

==> Docker

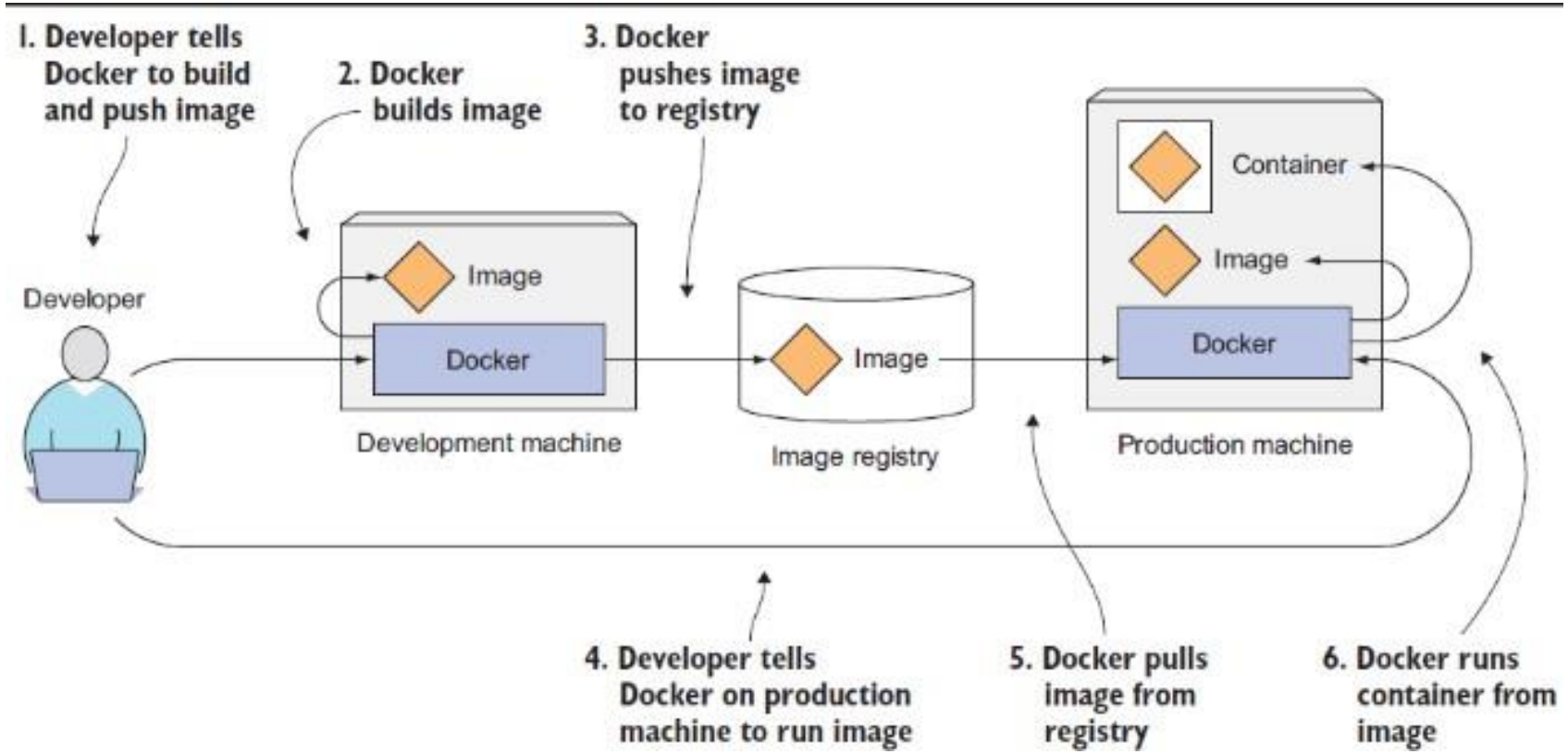
Workflow Docker

1. **Docker Client** demande à **Docker daemon** de créer un conteneur en utilisant une image spécifique;
2. **Docker daemon** télécharge l'image depuis le dépôt **Registry**;
3. **Docker daemon** conteneur communique avec le noyau Linux via **libcontainer** pour créer le conteneur:
    - Libcontainer permet l'accés aux APIs du noyau (cgroups, namespaves, etc) sans dépendance externe.
    - Pour construire un espace isolé pour l'application, le conteneur est crée avec les fonctionnalités.

Outils de Conteneurisation

==> Docker

Workflow Docker



Outils de Conteneurisation

==> Docker

Notion d'image

- **Image** : c'est le template / modèle, les données de l'application prêtes à l'emploi:
  => Archive qui peut être échangée entre plusieurs hôtes et réutilisée tant de fois.
- **Conteneur** : instance qui fonctionne / peut être démarrée

Outils de Conteneurisation

==> Docker

Notion d'image

- Les images sont disponibles dans des dépôts : https://hub.docker.com ou https://store.docker.com
- Un dépôt contient les images déposées par les autres;
- Choix multiple : possibilité de télécharger des versions;
- Le dépôt git d'une image contient les informations sur les options possibles et les explications;
- **Mais** !! Contraintes de sécurité liées à l'utilisation des images publiques.

Une **image** Docker peut être :

- Soit **téléchargée** depuis un dépôt;
- Soit **construite** à partir d'un DockerFile, puis envoyée sur un registry ou serveur

Outils de Conteneurisation

==> Docker

Gestion d'image : Docker Pull // Docker Push

1. docker pull : télécharger une image

```
docker pull [OPTIONS] ImageName:TAG
```

- Télécharger / extraire une image depuis le dépôt (registry) Docker Hub
- TAG est la version (par defaut latest)
- Exemple : docker pull debian:8.5

2. docker push : envoyer (uploader) l'image sur un serveur

```
docker push ImageName:TAG
```

Outils de Conteneurisation

==> Docker

Gestion d'image : Construction d'une image (1)

- Construction d'une image à partir d'un conteneur : `docker commit`

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

```
$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED
c3f279d17e0a        ubuntu:22.04        /bin/bash           7 days ago
197387f1b436        ubuntu:22.04        /bin/bash           7 days ago

$ docker commit c3f279d17e0a  svendowideit/testimage:version3

f5283438590d

$ docker images

REPOSITORY                      TAG                 ID
svendowideit/testimage          version3            f5283438590d
```

Outils de Conteneurisation

==> Docker

Gestion d'image : Construction d'une image (2)

- Construction d'une image à partir du fichier Dockerfile
  - Une image est construite à partir d'un fichier de description appelé Dockerfile.
  - Le fichier Dockerfile contient les commandes nécessaires pour :
    - télécharger une image de base
    - appliquer une série de commandes spécifiques
    - décrire l'environnement d'exécution

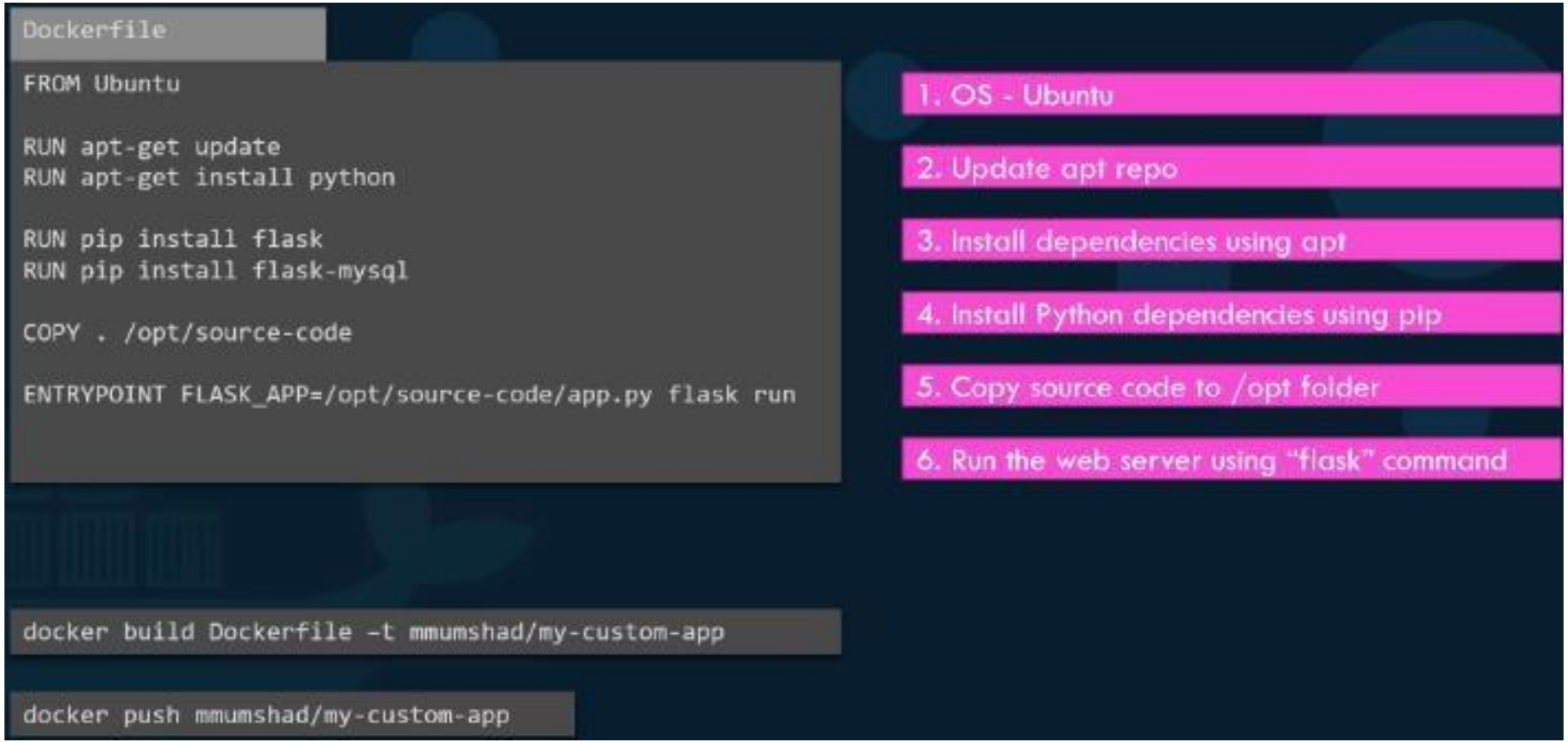
```
docker build [OPTIONS] PATH
```

Outils de Conteneurisation

==> Docker

Gestion d'image : **Construction d'une image**

- Construction d'une image à partir du fichier Dockerfile



FLASK : framework open-source de développement web en Python.

Outils de Conteneurisation

==> Docker

Gestion de conteneur : Création de conteneur

- Construction d'un conteneur à partir d'une image :

```
docker create --name ContainerName [OPTIONS] ImageName:TAG [COMMAND]
```

- ContainerName — Nom du conteneur à créer
- ImageName:TAG — Nom de l'image à exécuter et version
- COMMAND — Commande à exécuter dans le docker

- Exemples d'options :

-v HostVolume : ContainerVolume — Crée un volume partagé entre l'hôte et le conteneur
- - network — Choisir le réseau du Docker
p HostPort : DockerPort — mapping de port entre l'hôte et le conteneur

Outils de Conteneurisation

==> Docker

Gestion de conteneur : **Commandes (1/2)**

- Lancer un conteneur à partir d'une image :

```
docker run --name ContainerName ImageName:Tag command
```

- Lancer un conteneur existant :

```
docker start ContainerName
```

(docker run = docker create + docker start)

- Arrêter un conteneur :

```
docker stop ContainerName
```

Outils de Conteneurisation

==> Docker

Gestion de conteneur : **Commandes (2/2)**

- Lister les conteneurs exécutés :

```
docker ps
```

- Lister les conteneurs existants :

```
docker ps -a
```

- Exécuter une commande dans un docker existant :

```
docker exec -it ContainerName Command
```

Outils de Conteneurisation

==> Docker

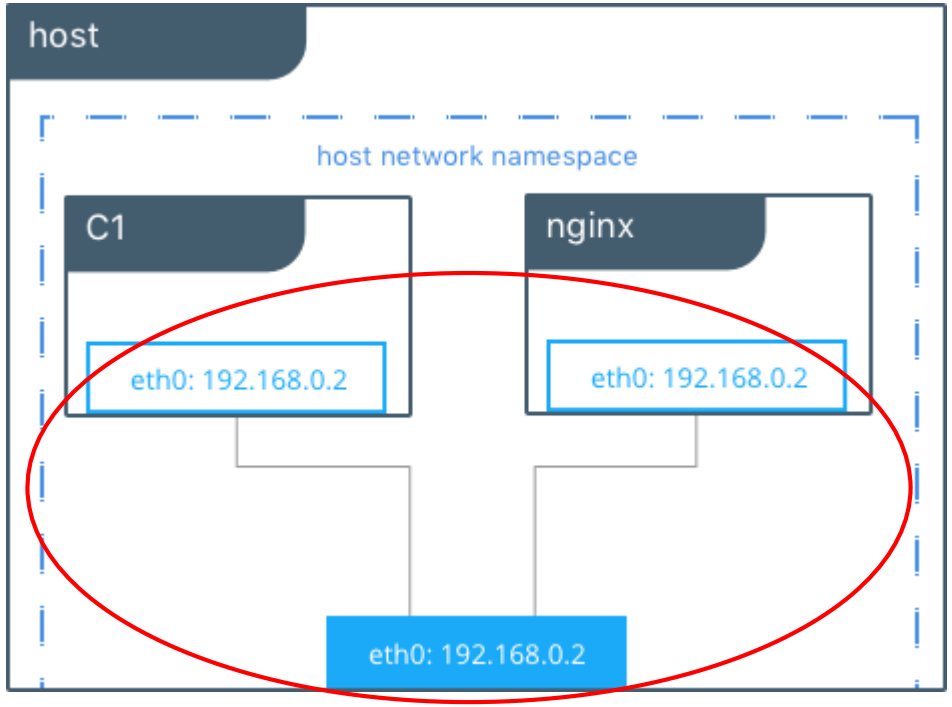
Gestion de conteneur : Réseau Docker

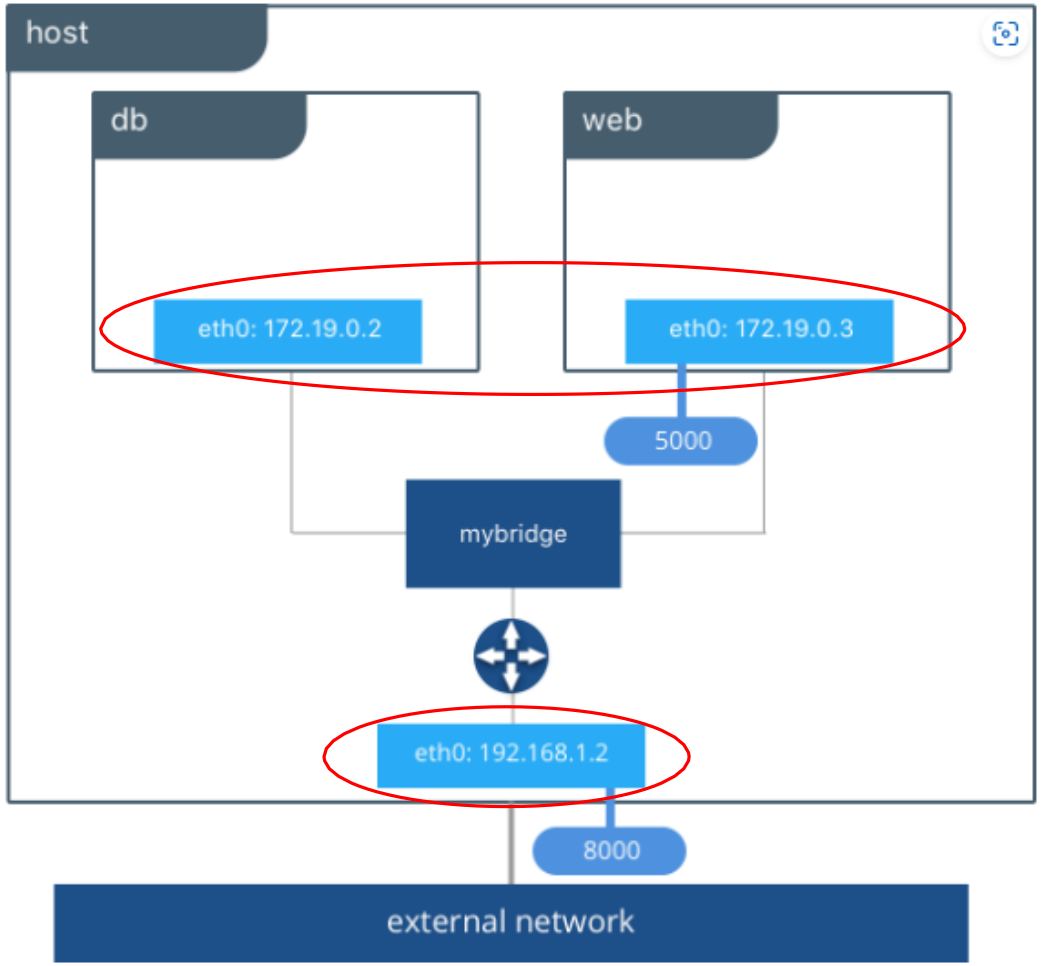
- Lors de la création d'un conteneur, trois réseaux sont automatiquement définis :
  - **Bridge** : le réseau interne par défaut attaché au conteneur
  - **Host :** réseau de la machine hôte
  - **None** : réseau isolé, le conteneurs est complètement et n'a pas d'accès à aucun réseau et n'a pas d'accès au monde externe
- Tous les conteneurs sont associés par défaut au réseau Bridge et ont des adresses à l'intérieur
- Pour permettre l'accès aux conteneurs depuis l'extérieur, il faut faire le mapping de port entre l'hôte et le conteneur ou associer le conteneur au réseau "host"
- Deux conteneurs qui sont crées sur le même réseau peuvent communiquer directement sans aucune configuration soit via leurs adresses IP ou leurs hostnames

Outils de Conteneurisation

==> Docker

Gestion de conteneur : Réseau Docker





Outils de Conteneurisation

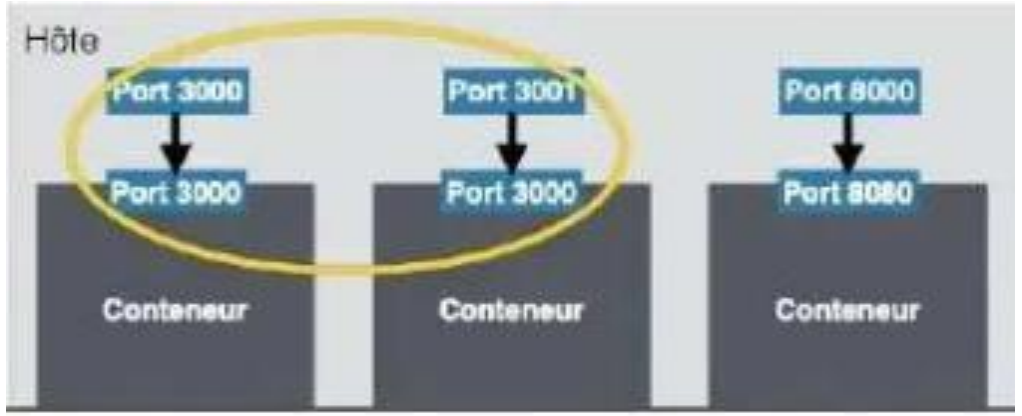
==> Docker

Gestion de conteneur : Port Publishing par défaut et lors de la création, un conteneur ne partage aucun port avec l'extérieur.

⇒ La publication de ports réseau est un moyen de permettre au conteneur de communiquer avec d'autres conteneurs et le monde externe.

- Publier un port (- - publish / -p)
  - Publie un port conteneur à un port host (mapping)
  - Exemple d'utilisation : Transfert de tout le trafic de données entrant sur le port hôte 8000 au port conteneur 8080

```
docker run --publish 8000:8080 image
```



✓ 8000 : port usuel de l'application conteneurisée
✓ 8080 : port hôte (accès conteneur)

Outils de Conteneurisation

==> Docker

Gestion de conteneur : Port EXPOSE

- Exposer un port (EXPOSE, utilisé dans le DOCKERFILE)
  - Indique le port sur le quel le conteneur écoute / accepte tous le trafic entrant.
  - Correspond généralement au port usuel de l'application
  - Exemple d'utilisation : **EXPOSE 80** (Apache web server) / **EXPOSE 3306** (MySQL server)
- **Rôle de EXPOSE**
  - **(1) Fournir de la documentation**
    - Indique le port sur le quel le conteneur écoute / accepte tous le trafic entrant.

      " The EXPOSE instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published."— docs.docker.com
  - **(2) Informe Docker Deamon sur le port de l'application si le flag '-P' est utilisé**
    - Le flag '-P' (—publish-all) indique à Docker de publier tous les ports de l'application : Docker a besoin de connaitre ces ports (see exposed ports)

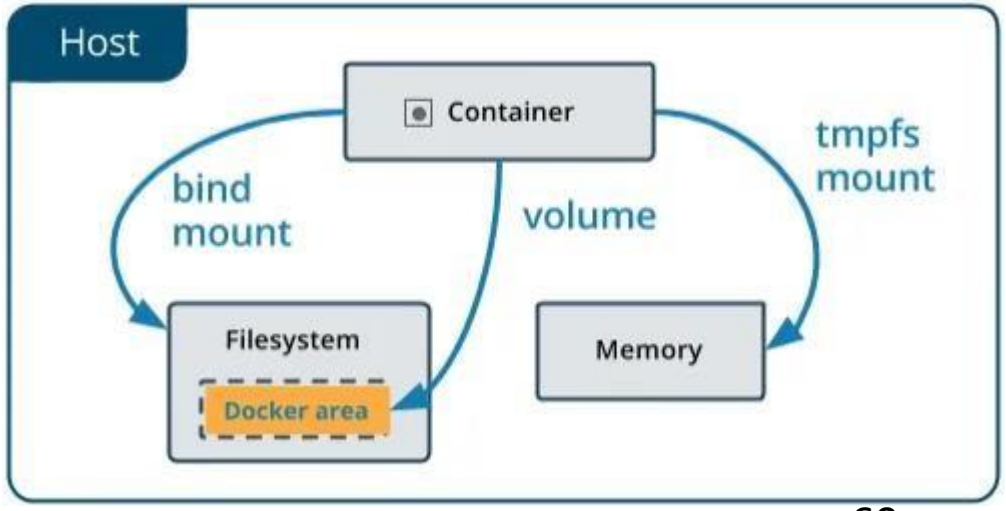
Outils de Conteneurisation

==> Docker

Gestion de conteneur : Gestion des Données

- Les données dans un conteneur sont éphémères : les fichiers créés à l'intérieur d'un conteneur sont stockés sur un espace de travail : couche de conteneur inscriptible (writable).e).
  - Cela signifie que : Les données ne sont pas conservées lorsque ce conteneur est supprimé

- Trois méthodes de stockage :
  - **tmpfs mount** : (éphémère) permet de stocker les données dans la mémoire du système hôte en plus de la couche inscriptible.
  - **volume** : configurer un volume Docker commun entre hôte et conteneur au moment du lancement.
  - **bind mount** : monter un répertoire du système de fichier hôte au conteneur lors de son lancement.



Outils de Conteneurisation

==> Docker

Ecosystème Docker

L'écosystème Docker regroupe tous les utilitaires utilisés avec Docker :

- **Docker Engine** : c'est l'application client-serveur open source dont les composants centraux sont : démon et client Docker.
- **Docker machine** : outil de gestion et provisioning des machines virtuelles exécutant Docker.
  C'est un script utilisé pour la création des machines virtuelles utilisant docker.
- **Docker Swarm** : un swarm est un groupe de machines exécutant Docker et appartenant à un Cluster.
- **Docker Compose** : outil permettant de définir le comportement de plusieurs conteneurs qui dépendent les uns des autres et qui permet leur lancement.

Outils de Conteneurisation

==> Docker

Docker Compose

```
version: '3.7'
services:
  db:
    image: mysql:5.7
    container_name: mysql_c
    restart: always
    volumes:
      - db-volume:/var/lib/mysql
      - ./articles.sql:/docker-entrypoint-initdb.d/articles.sql
    environment:
      MYSQL_ROOT_PASSWORD: test
      MYSQL_DATABASE: test
      MYSQL_USER: test
      MYSQL_PASSWORD: test
  app:
    image: myapp
    container_name: myapp_c
    restart: always
    volumes:
      - ./app:/var/www/html
    ports:
      - 8080:80
    depends_on:
      - db
volumes:
  db-volume:
```

Services : conteneurs

Conteneur1 :
Nom:db
image: mysql:5.7

Conteneur2 :
Nom:app
image: myapp

Commande	Description
docker-compose -f docker-compose.yml up	Lancement des conteneurs
docker-compose -f docker-compose.yml down	Destruction des conteneurs