

Virtualisation & Cloud Computing

Dr. Wael Sellami
wael.sellami@gmail.com

2022-2023

Chapitre 4 : Conteneurs dans le Cloud Computing

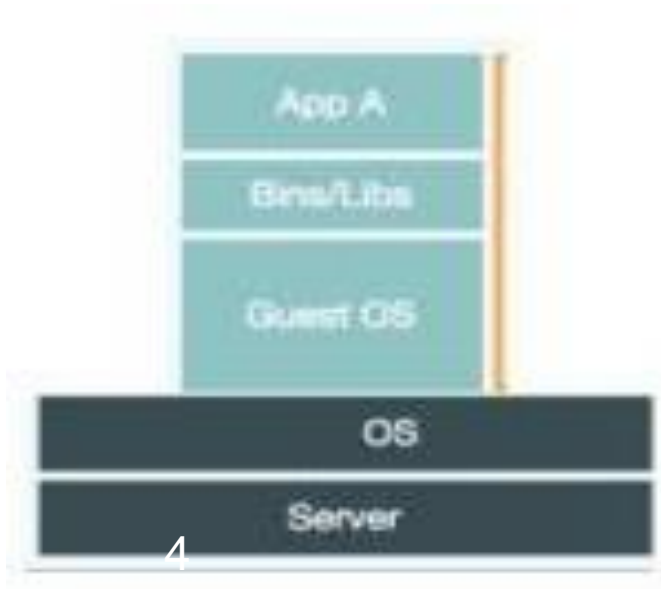
Plan du chapitre

1. Evolution des serveurs
2. Conteneur
3. Docker
4. Orchestration des conteneurs

Evolution des serveurs

Serveurs physiques

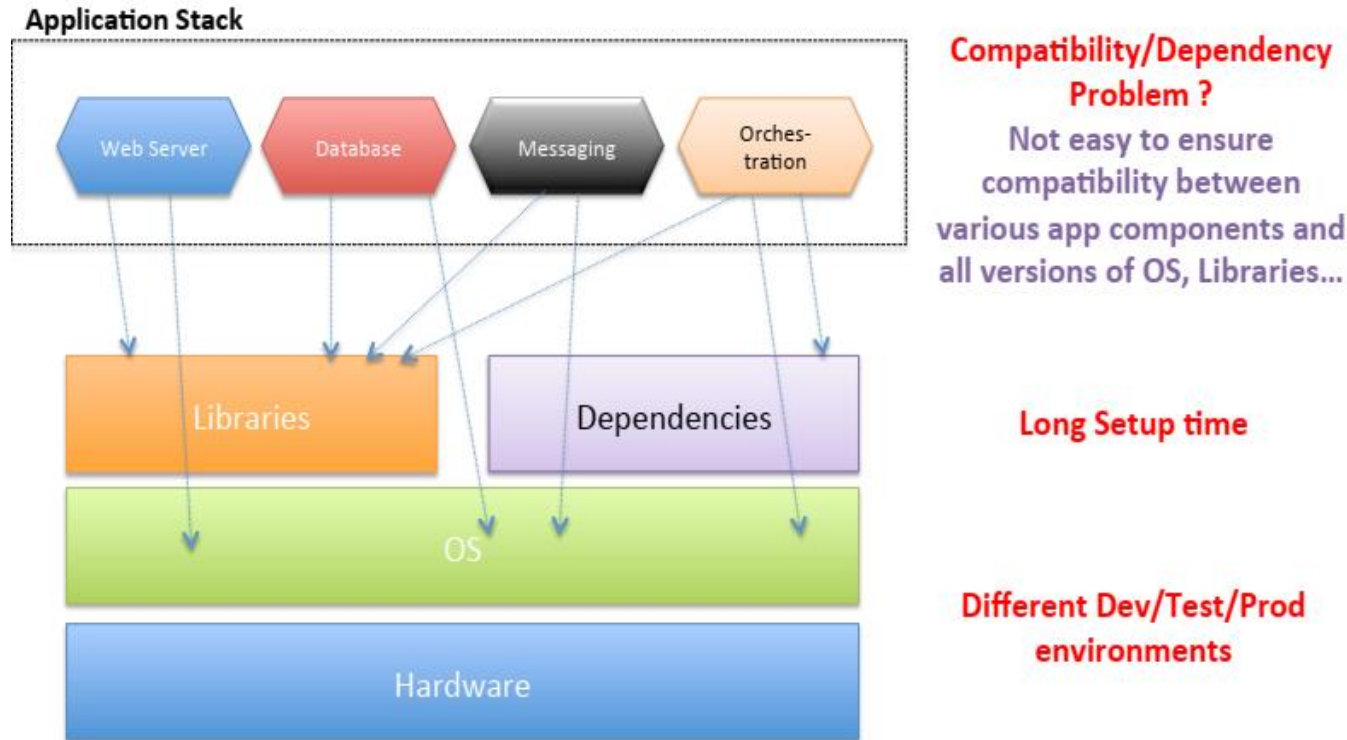
- Historiquement, quand nous avons besoin de serveurs, nous achetions des serveurs physiques avec une quantité définie de CPU, de mémoire RAM ou de stockage sur le disque.



Partage de l'espace logistique

Evolution des serveurs

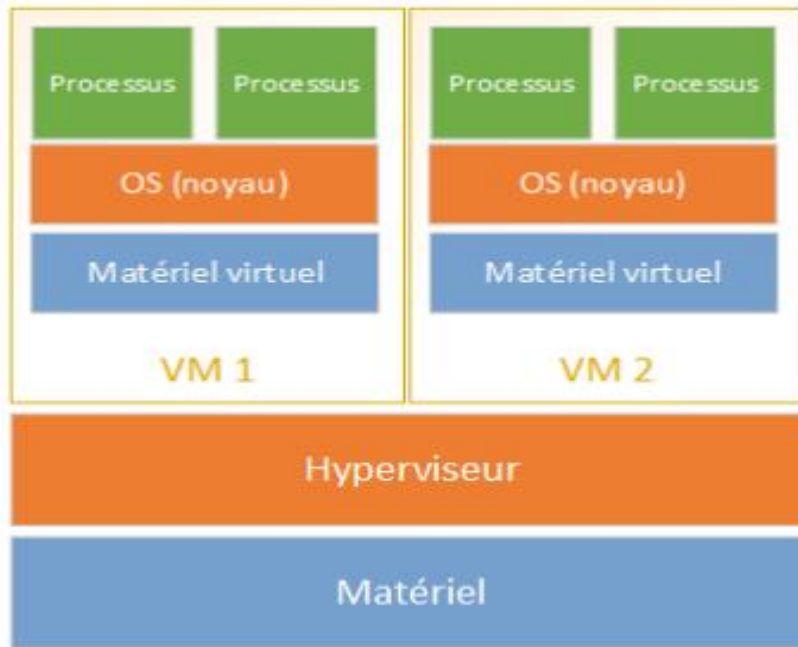
Serveurs physiques



- Besoin d'avoir de la puissance supplémentaire pour des périodes de forte charge (fête de Noël, par exemple).
- Acheter plus de serveurs pour répondre aux pics d'utilisation. Une solution a alors été créée : la machine virtuelle.

Evolution des serveurs

Virtualisation



Les machines virtuelles



Partage des ressources d'une machine

Evolution des serveurs

Virtualisation

- Cependant, avec les machines virtuelles (VM), nous faisons ce qu'on appelle de la **virtualisation lourde**.
- En effet, un système complet dans le système hôte est créé, pour qu'il ait ses propres ressources.
- L'isolation avec le système hôte est donc totale; cependant, cela apporte plusieurs contraintes :
 - ✗ une machine virtuelle prend du temps à démarrer ;
 - ✗ une machine virtuelle réserve les ressources (CPU/RAM) sur le système hôte.

Evolution des serveurs

Virtualisation

- Mais cette solution présente aussi de nombreux avantages :
 - ✓ Une machine virtuelle est totalement isolée du système hôte ;
 - ✓ Les ressources attribuées à une machine virtuelle lui sont totalement réservées ;
 - ✓ Installer différents OS (Linux, Windows, BSD, etc.).
- Mais, il arrive très souvent que l'application qu'elle fait tourner ne consomme pas l'ensemble des ressources disponibles sur la machine virtuelle. Alors est né un nouveau système de virtualisation plus léger au niveau processus : **les conteneurs**.

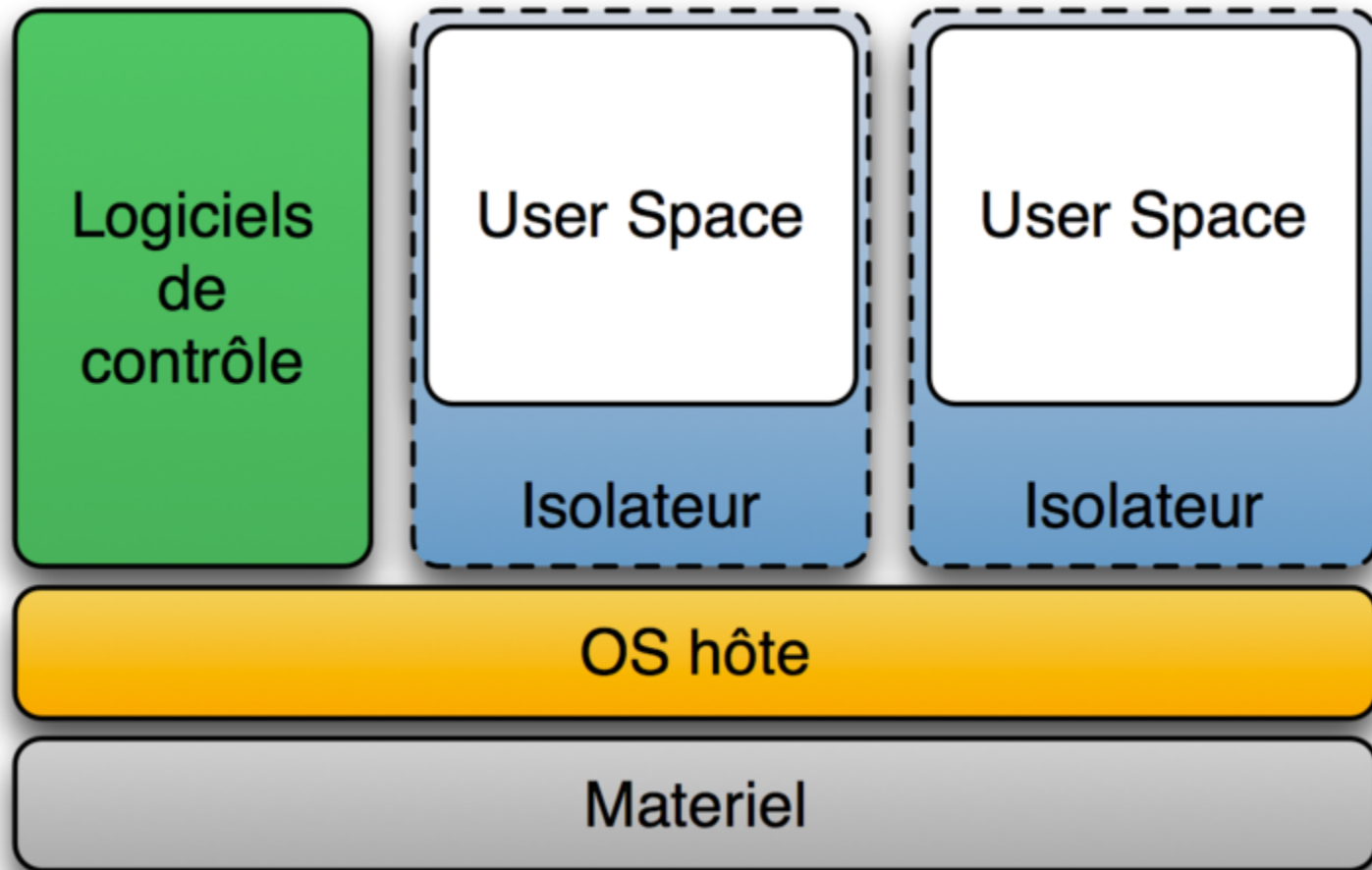
Virtualisation des processus

Présentation de l'isolation

- **L'isolation** (aussi appelé cloisonnement) est une technique qui intervient au sein d'un même système d'exploitation.
- Elle permet de séparer un système en plusieurs contextes ou environnements. Chacun d'entre eux est régi par l'OS hôte, mais les programmes de chaque contexte ne peuvent communiquer qu'avec les processus et les ressources associées à leur propre contexte.
- Il est ainsi possible de partitionner un serveur en plusieurs dizaines de contextes, presque sans ralentissement.
- Il est possible également de lancer des programmes dans une autre distribution que celle du système principal.

Virtualisation des processus

Présentation de l'isolation



Virtualisation des processus

Unix chroot / BSD Jail

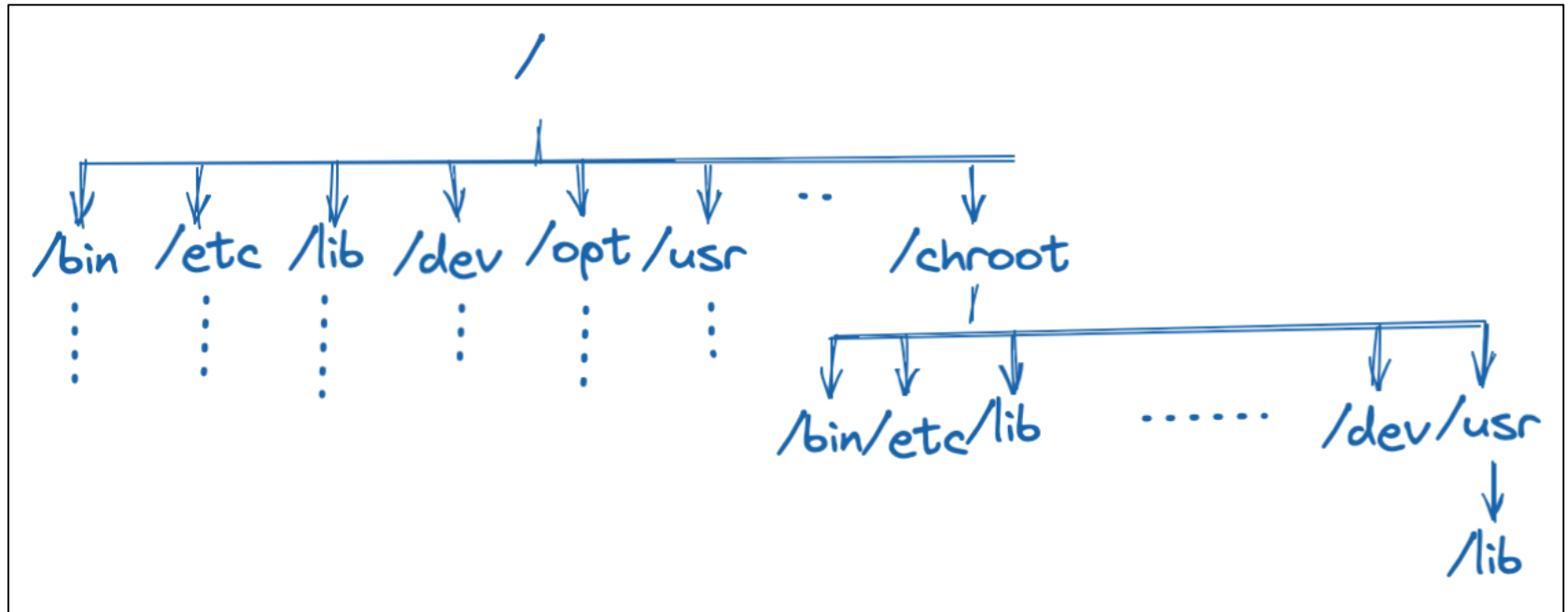
- Via des mécanismes comme Unix chroot (1982) ou BSD jail (1998), il est possible d'exécuter des applications dans un environnement qui n'est pas celui du système hôte, tout en étant légers. Il s'agit d'un « mini système » ne contenant que ce dont l'application a besoin, et n'ayant que des accès limités aux ressources.

chroot

- chroot signifie « **change root** », traduisez changement de racine
- Elle permet d'isoler la racine du système de fichier (le / de l'arborescence) pour une commande spécifique. La racine du système de fichier visible par la commande "chrootée" est une sous-arborescence du système de fichier complet. Ceci permet, par exemple, de sécuriser un serveur en lui donnant accès à un système de fichier restreint.

Virtualisation des processus

Unix chroot / BSD Jail



Virtualisation des processus

Conteneur Linux LXC

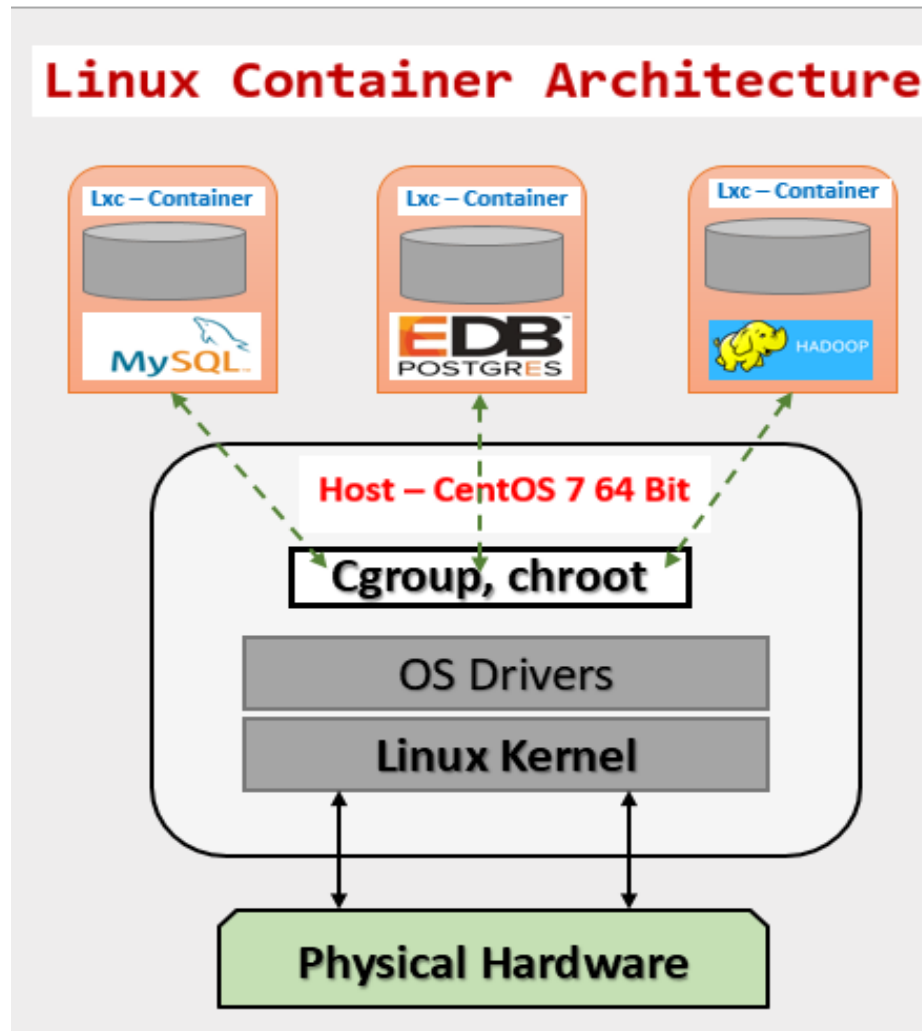


LXC : C'est quoi?

- La virtualisation par conteneurs se base sur la virtualisation Linux LXC, pour Linux Containers (2008).
- Un conteneur Linux est une enveloppe virtuelle qui permet de packager une application avec tous les éléments dont elle a besoin pour fonctionner (fichiers source, run-time, librairies et dépendances).
- Le conteneur permet de faire de la **virtualisation légère**, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une isolation des processus. Le conteneur **partage** donc **les ressources** avec le **système hôte**.

Virtualisation des processus

Conteneur Linux LXC



Virtualisation des processus

Conteneur Linux LXC



LXC : C'est quoi?

- Il s'agit d'une méthode de cloisonnement au niveau du système d'exploitation.
- Les conteneurs sont isolés du reste du système. Ils sont packagés en un ensemble cohérent et prêt à être déployé sur un serveur et son OS.
 - ✓ portables et fonctionnent de la même manière dans les environnements de développement, de test et de production.
 - ✓ déplacer l'application jusqu'en production sans aucun effet secondaire.
- Les conteneurs partagent entre eux le **kernel Linux** ; ainsi, il n'est pas possible de faire fonctionner un système Windows ou BSD dans celui-ci.

Virtualisation des processus

Conteneur Linux LXC



LXC : Technologies de base

- LXC repose sur la notion de groupes de contrôle Linux (**cgroups**) :
 - permet de limiter et d'isoler l'utilisation des ressources qu'un processus peut utiliser (processeur, mémoire, réseau, système de fichier, etc), et ce sans recourir à des machines virtuelles à part entière.
- LXC repose aussi sur une isolation des espaces de nommage du noyau (**namespace**) :
 - Permet d'empêcher qu'un groupe puisse « voir » les ressources des autres groupes (systèmes de fichiers, les ID réseau et les ID utilisateur)
- LXC repose sur les **bibliothèques Profils Apparmor** (Application Armor) et **SELinux** (Security-Enhanced Linux) pour la sécurité en termes des restrictions, permissions et droits utilisateur

Virtualisation des processus

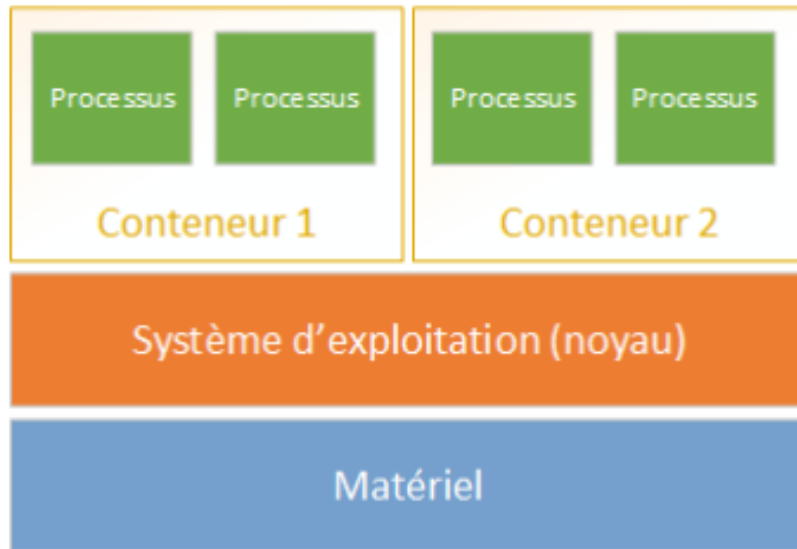
Conteneur Linux LXC



- Intégré officiellement au noyau Linux :
 - ✓ Mount namespace (Linux 2.4.19)
 - ✓ PID namespace (Linux 2.6.24) - x PID/process
 - ✓ Net namespace (Linux 2.6.19-2.6.24)
 - ✓ User namespace (Linux 2.6.23-3.8)
 - ✓ cgroups (Linux 2.6.24) - gérer la limitation de ressource

Virtualisation des processus

Conteneur Linux LXC

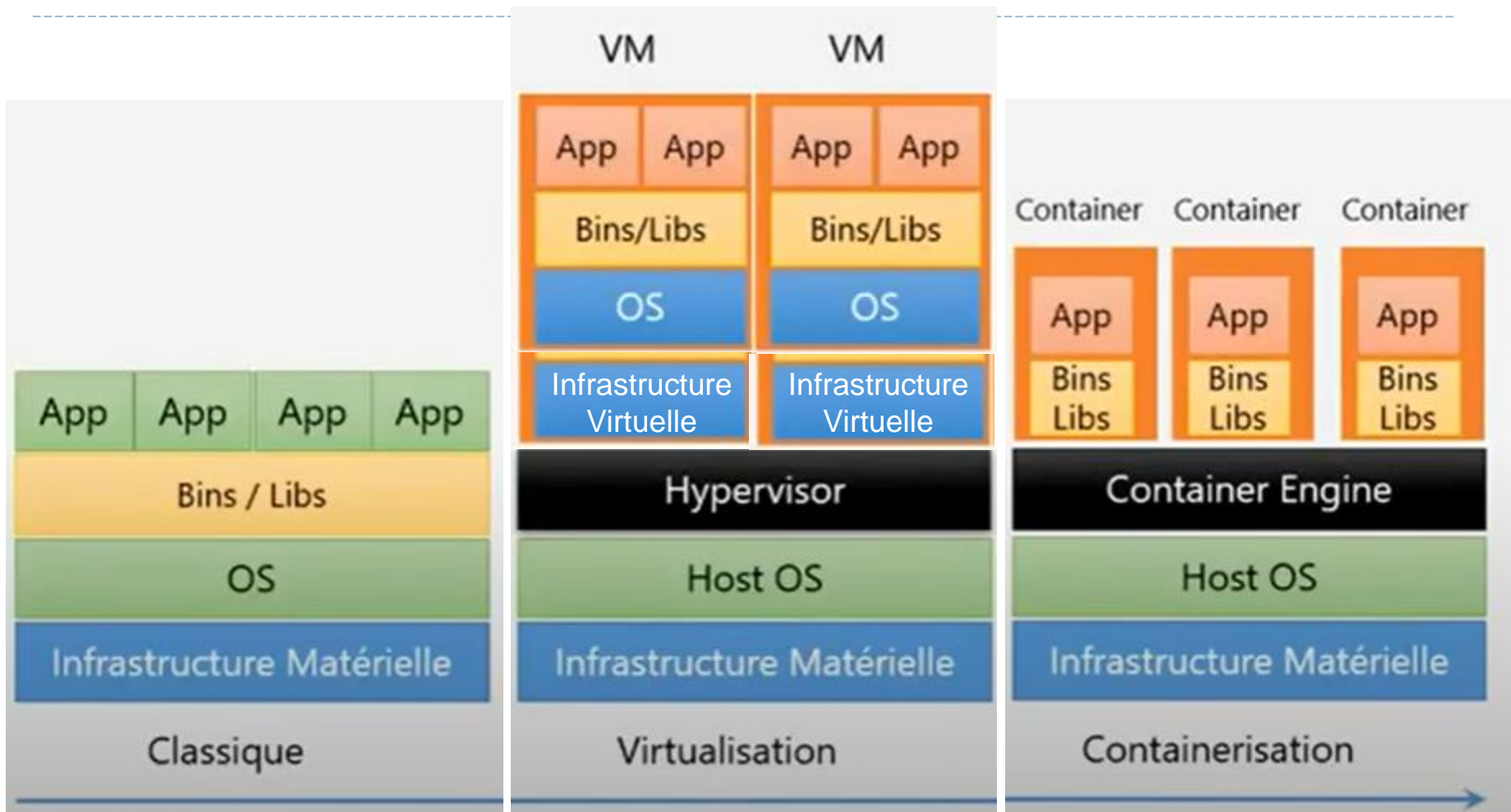


Les conteneurs



Optimisation des ressources sans perte du cloisonnement

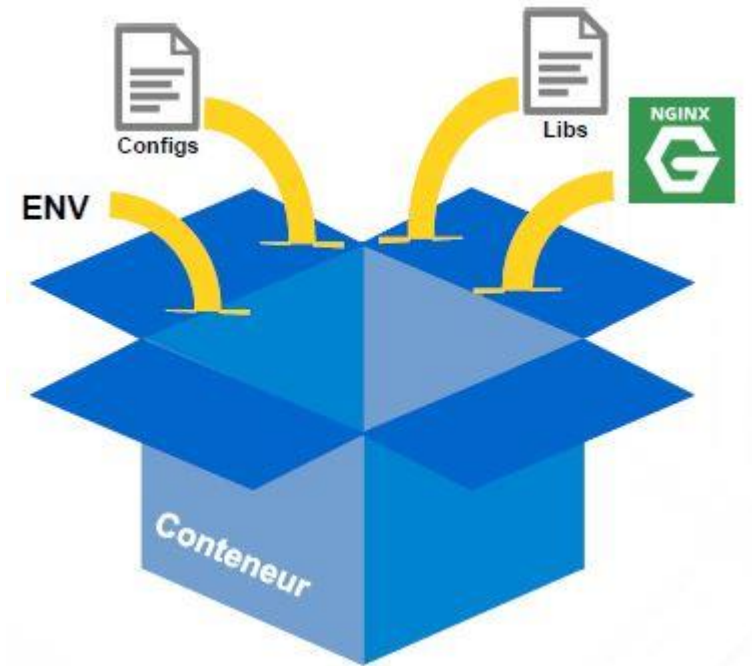
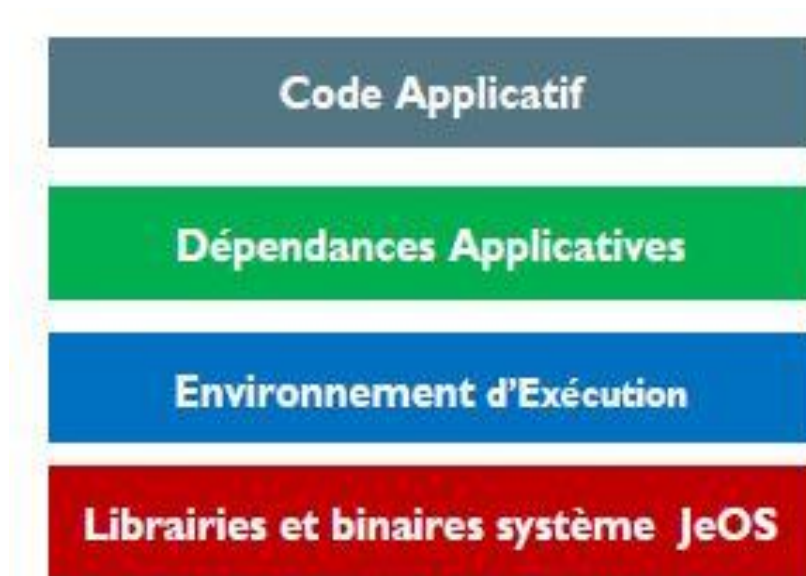
Evolution des serveurs



Conteneur (jolie métaphore), c'est quoi ?

- Les mêmes idées que la virtualisation, mais sans virtualisation :
 - ✓ Isolation et automatisation
 - ✓ Agnostique sur le contenu et le transporteur
 - ✓ Principe d'infrastructure consistante et répétable
 - ✓ Peu de surcharge (overhead) par rapport à une VM.
- Les conteneurs permettent d'exécuter plusieurs parties d'une application dans des micro-services, indépendamment les uns des autres, sur le même matériel, avec un niveau de contrôle bien plus élevé sur leurs éléments et cycles de vie.

Conteneur (jolie métaphore), c'est quoi ?



Un conteneur (jolie métaphore), c'est quoi ?

- Encapsule une application
- Fournit à l'application un système de fichiers complet (/ , /usr/, /bin, /opt, /etc)
- Fournit les binaires nécessaires à son exécution (bash, sh, python, etc.)
- Possède sa propre interface réseau
- Possède ses propres utilisateurs Linux (ex : root)
- Facile à créer et à supprimer
- Léger en terme de ressource demandée.

Avantages des conteneurs

- **Réduire des coûts**

Les conteneurs permettent de réduire les coûts, d'augmenter la densité de l'infrastructure, tout en améliorant le cycle de déploiement.

- **Se limiter aux ressources nécessaires**

Conteneur ne réserve pas la quantité de CPU, RAM et disque attribuée auprès du système hôte. Ainsi, nous pouvons allouer 16 Go de RAM à notre conteneur, mais si celui-ci n'utilise que 2 Go, le reste ne sera pas verrouillé (à la différence de la VM).

Avantages des conteneurs

- **Démarrer rapidement vos conteneurs**

Les conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation, ils peuvent démarrer beaucoup plus rapidement et plus fréquemment qu'une machine virtuelle sur nos serveurs hôtes, et ainsi réduire encore un peu les frais de l'infrastructure.

- **Donner plus d'autonomie à vos développeurs**

Possibilité de faire tourner des conteneurs sur le poste des développeurs, et ainsi de réduire les différences entre la "sainte" production, et l'environnement local sur le poste des développeurs.

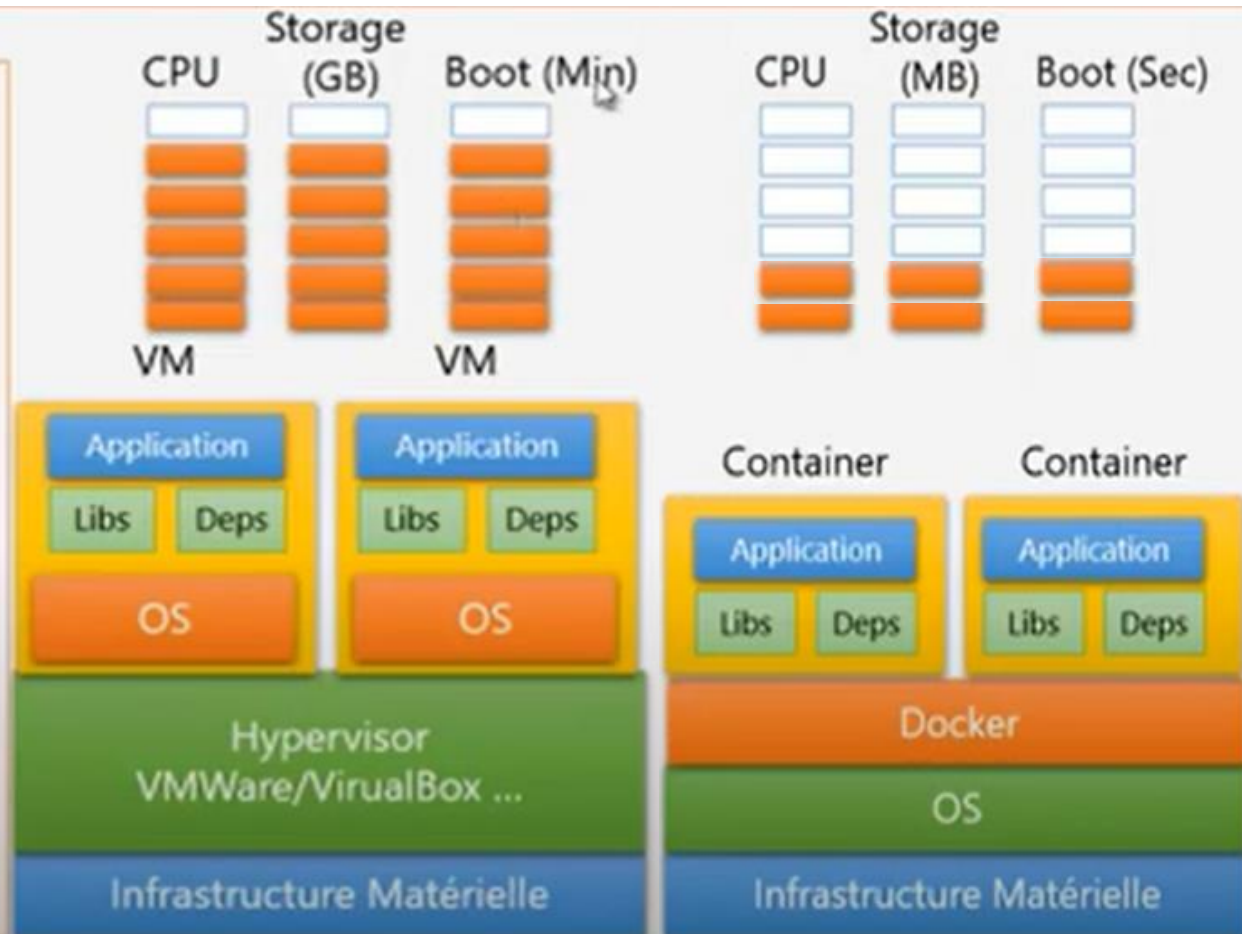
Conteneurs VS machines virtuelles

- Machine Virtuelle :

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

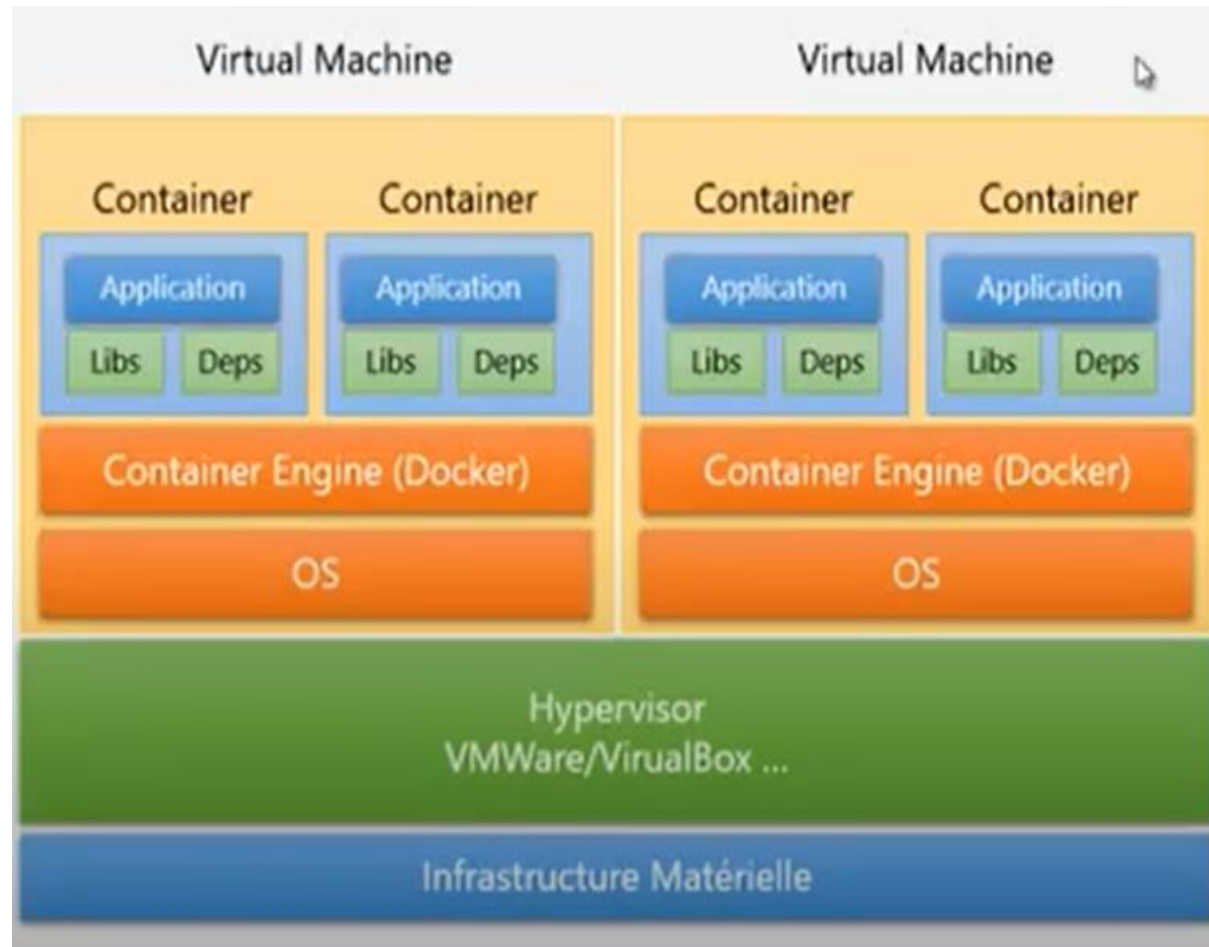
- Conteneur :

- Permet de créer un environnement d'exécution des applications
- Les conteneurs utilisent le même OS
- Tous les conteneurs utilisent le même kernel OS (Linux), Consomme peu de ressources, boot rapide (qq secondes)

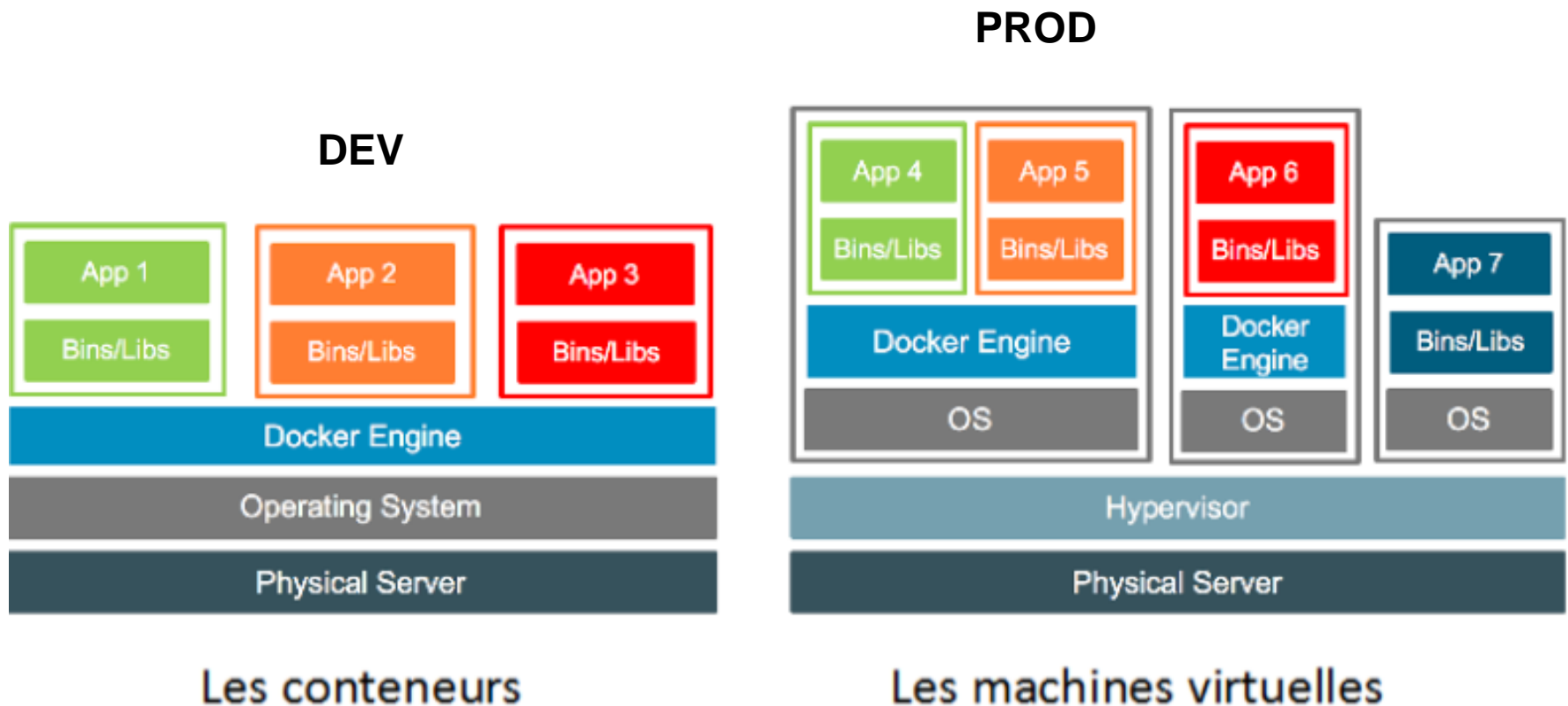


Des conteneurs dans les machines virtuelles ?

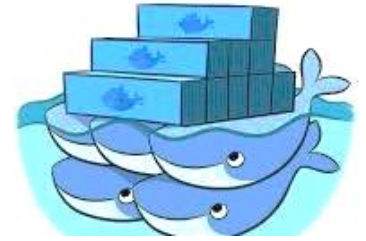
- Docker ne vient pas pour remplacer les machines virtuelles.
- Dans la pratique on utilise les deux :
 - Les machines virtuelles pour virtualiser les machines
 - Utiliser Docker pour isoler les environnements d'exécution des applications dans des machines virtuelles.
- Ceci pour tirer le bénéfice des technologies



Des conteneurs dans les machines virtuelles ?



CaaS (Container-as-a-Service)



- Le service de conteneur (CaaS) est un modèle métier dans lequel les fournisseurs de Cloud Computing proposent des services liés à la virtualisation basée sur les conteneurs en tant que services évolutifs en ligne.
- Ceci permet aux utilisateurs finaux d'utiliser des services de conteneurs sans avoir à fournir l'infrastructure dont ils auraient besoin.
- Il s'agit d'un terme marketing qui se réfère à des modèles de services Cloud existants tels que les services IaaS (Infrastructure as a Service), PaaS (Platform as a Service) et SaaS (Software as a Service).

CaaS (Container-as-a-Service)

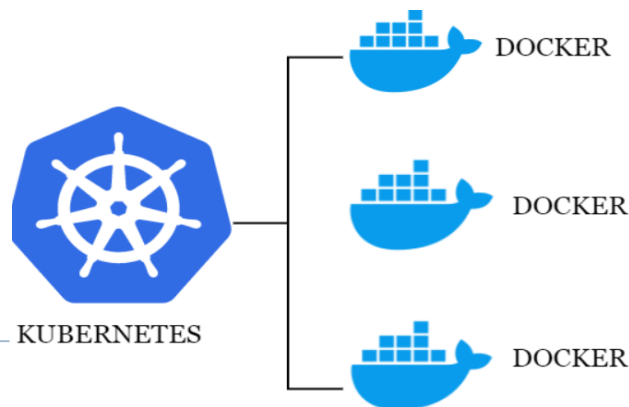
- Le service de conteneur permet aux utilisateurs de charger, d'organiser, d'exécuter, de faire évoluer, d'administrer et d'arrêter des conteneurs à l'aide des appels API ou de l'interface de portail Web d'un fournisseur.
- Comme pour la plupart des services de cloud, les utilisateurs ne payent que les ressources CaaS qu'ils utilisent (instances de traitement, équilibrage et planification de charge).
- Les trois plateformes CaaS les plus populaires avec Google Container Engine (GKE), Amazon EC2 Container Service (ECS) et Microsoft Azure Container Service (ACS).

CaaS (Conteneurs as-a-Service)

- Imaginons une application Web de vente en ligne organisée en une architecture de micro-services, où la propriété du domaine d'activité structure le système de services.
- Les domaines des services pourraient être : l'authentification, le panier d'achat, le paiement, etc. Chaque solution a sa propre base de code et demeure conteneurisée.
- Grâce au CaaS, ces services de conteneurs peuvent se déployer instantanément dans un système.

Comment fonctionne CaaS ?

- L'interaction avec l'environnement de conteneur basé sur le Cloud se fait via une interface utilisateur graphique (GUI) ou sous forme d'appels d'API.
- Les technologies de conteneurs disponibles pour les utilisateurs diffèrent selon le fournisseur. Cependant, le noyau de chaque plateforme CaaS est un outil d'orchestration (aussi appelé orchestrator), qui permet la gestion d'architectures de conteneurs complexes.
- Le marché de la virtualisation en conteneur est dominé par trois outils d'orchestration (Docker Swarm, Kubernetes, etc)



Inconvénients des conteneurs

- Les conteneurs doivent être compatibles avec le système sous-jacent. → **Ils ne sont pas cross-platform.**
 - ✓ Les OS Linux ARM exécutent des conteneurs Linux ARM,
 - ✓ les OS Linux x86 exécutent des conteneurs Linux x86
 - ✓ les OS Windows x86 exécutent des conteneurs Windows x86.

Environnements d'exécution

- Des exemples courants d'environnements d'exécution de conteneur sont :

- runC



- Containerd



- Docker



- Podman



- L'outil « Docker » est l'outil le plus populaire !!



docker.

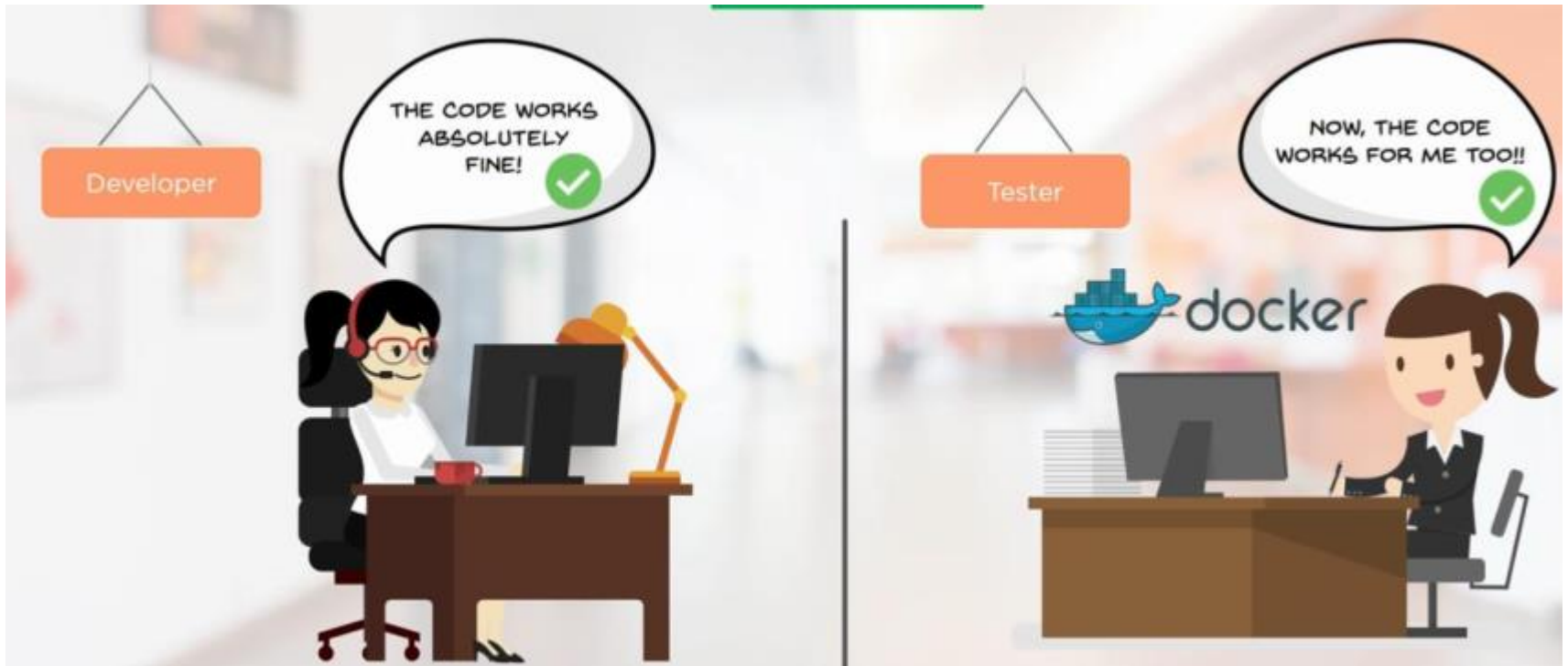
Avant Docker

- **Dev** : Cela fonctionne bien dans mon système
- **Testeur** : Cela ne fonctionne pas dans mon système !!



Après Docker

Le testeur et le développeur exécutent tous les deux la même application dans l'environnement Docker sans avoir à faire face aux différences de dépendances comme auparavant.



Arrivé du Docker



- Docker a été créé pour les besoins d'une société de Platform as a Service (PaaS) appelée **DotCloud**.
- Arrivé sur le marché en 2013.
- Open Source → gratuit.
- Permet de procurer une API de haut niveau pour encapsuler des applications.
- Basé initialement sur LXC.
- Repose maintenant sur leur propre technologie libcontainer.



Docker



- Docker est un outil permettant d'**empaqueter une application et ses dépendances** dans un conteneur virtuel, qui pourra être exécuté sur n'importe quel serveur Linux.
- Docker **étend** le format de **conteneur** Linux standard, **LXC**, avec une **API de haut niveau** fournissant une solution de virtualisation qui exécute les processus de façon isolée.
- Cela permet de garantir la fiabilité d'exécution et la stabilité d'une application.

Docker, pour quoi faire ?

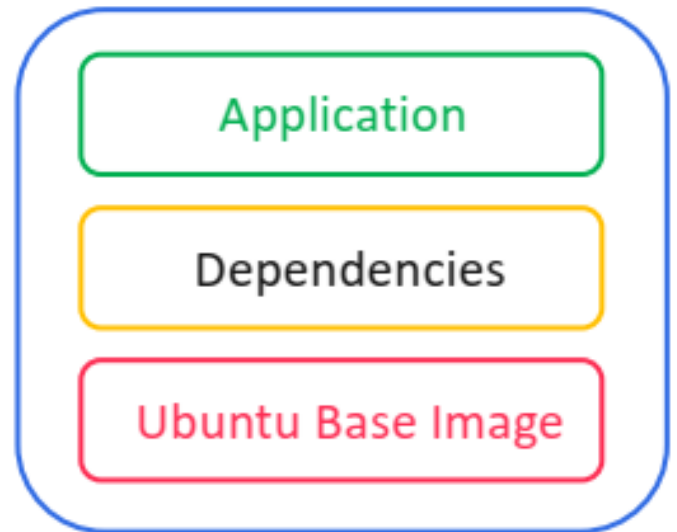
- **Le développement** : cela permet de facilement avoir le même environnement de développement qu'en **production**. Cela permet également de pouvoir sur la même machine, tester avec plusieurs versions d'un même logiciel.
- **Le déploiement** : puisque Docker a pour vocation de conteneuriser des applications, il devient simple de créer un conteneur pour une application, et la dispatcher. Un conteneur qui fonctionne sur une machine avec une distribution X, fonctionnera sur une autre machine avec une distribution Y.
- **L'installation des applications** : étant donné que Docker propose une multitude d'outils, il est facile et rapide d'installer une application : bien souvent une seule ligne de commande suffit pour avoir par une application fonctionnelle.

Docker

- Les points forts :
 - Installation simple (Linux, OSX, Windows)
 - Ligne de commande très sympathique (docker help)
 - Langage de description des images (avec notion de parent)
 - Communauté très active
 - API pour le pilotage:
 - GUI, Orchestration, hébergement cloud, intégration continue, OS, ...

Exemple

Ubuntu + Python + Dependencies



Qu'apporte la nouvelle version de docker?

Docker existe maintenant sous deux versions :

- Pour faciliter la gestion des architectures complexes, Docker a construit une plateforme de Containers-as-a-Service. Baptisée Docker Enterprise Edition (Docker EE)
- Docker possède également développé une version destinée aux développeurs et à tous ceux qui veulent apprendre Linux, Docker Community Edition baptisée (Docker CE)

Evidement, il faut vivre : CE vs EE



Qui utilise Docker?



Installation Docker

- **Docker for Linux**
- **Docker for Windows** : Win10 Pro/Ent only Uses Hyper-V with tiny Linux VM for Linux Containers
- **Docker Toolbox** : Win7/8/8.1 or Win10 Home Runs a tiny Linux VM in VirtualBox
- **Docker for MAC**
- **Online Emulator** : <https://labs.play-with-docker.com/>

Docker

- Terminologie :
 - Client/server : outil utilisant l'API du serveur/Daemon
 - Image : conteneur en lecture seule
 - Docker hub : répertoire (dépôts) public (<https://index.docker.io/>)
 - Conteneur : élément manipulable
- Analogie avec le développement objet :
 - Images équivalentes aux classes
 - Les couches sont équivalentes à l'héritage
 - Les conteneurs sont des instances

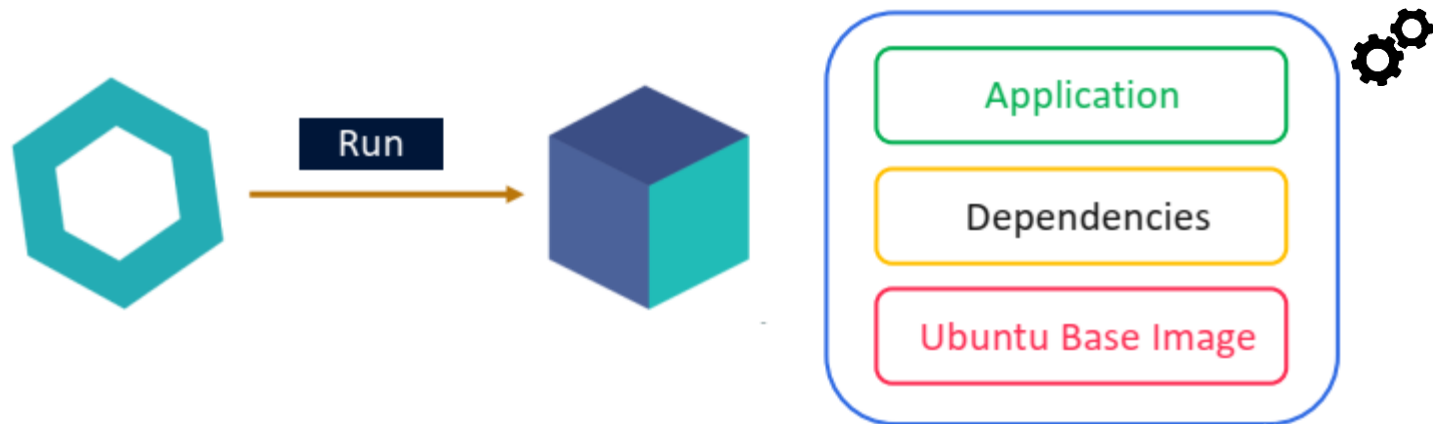


Image Docker

- Tout comme les machines virtuelles, les conteneurs Docker sont basés sur des images.
- Une image est un modèle en **lecture seule** qui contient toutes les instructions dont le moteur Docker a besoin afin de créer un conteneur Docker.
- Une image portable d'un conteneur est décrite comme image Docker sous la forme d'un fichier texte, on parle alors d'un « **Dockerfile** ».
- Si un conteneur doit être démarré sur un système, un paquet avec l'image correspondante est **chargé en premier**, si elle n'existe pas localement. Donc, l'image chargée fournit le système de fichiers requis pour l'exécution, y compris tous les paramètres.

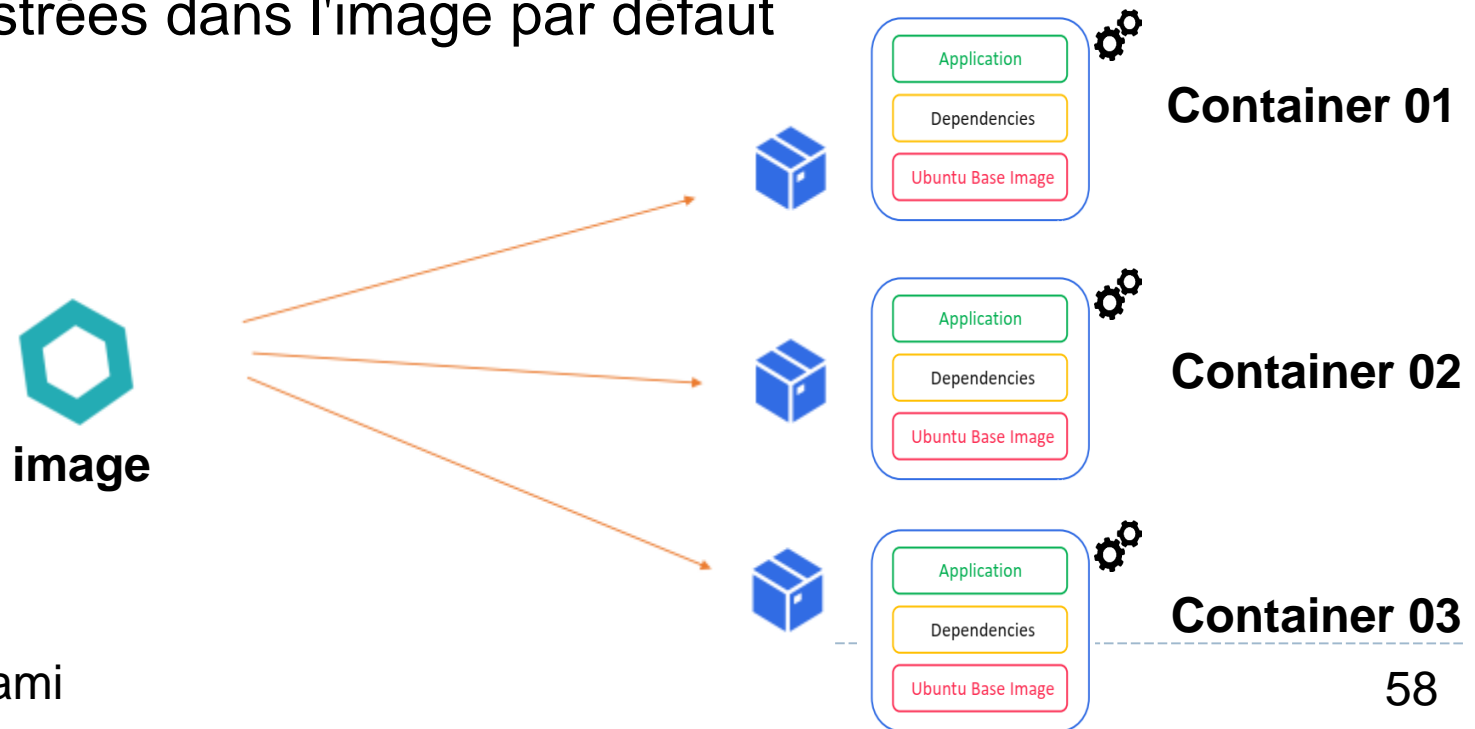
Conteneur Docker

- Les conteneurs sont **une instance exécutable** d'images ou d'applications prêtes créées à partir d'images Docker. Grâce à l'API Docker ou à la CLI, nous pouvons créer ou supprimer un conteneur.
- Les conteneurs peuvent être connectés à un ou plusieurs réseaux, même créer une nouvelle image ou attacher un stockage à son état actuel. Les conteneurs sont par défaut isolés les uns des autres et de leur machine hôte.

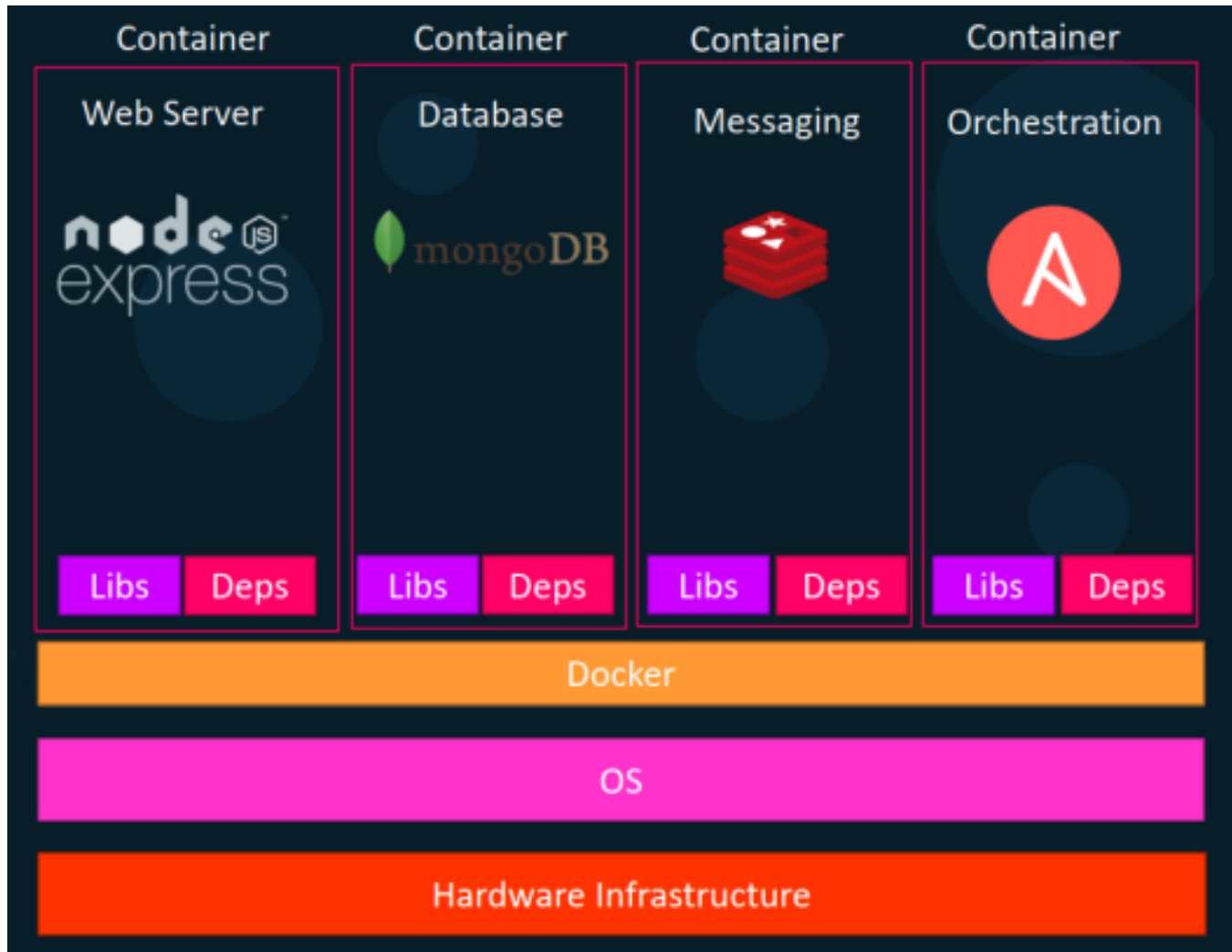


Conteneur Docker

- Nous pouvons exécuter n'importe quel nombre de conteneurs basés sur une image et Docker s'assure que chaque conteneur créé a un nom unique dans l'espace de noms.
- L'image Docker est un modèle en lecture seule. Les modifications apportées aux conteneurs ne seront pas enregistrées dans l'image par défaut



Conteneur Docker

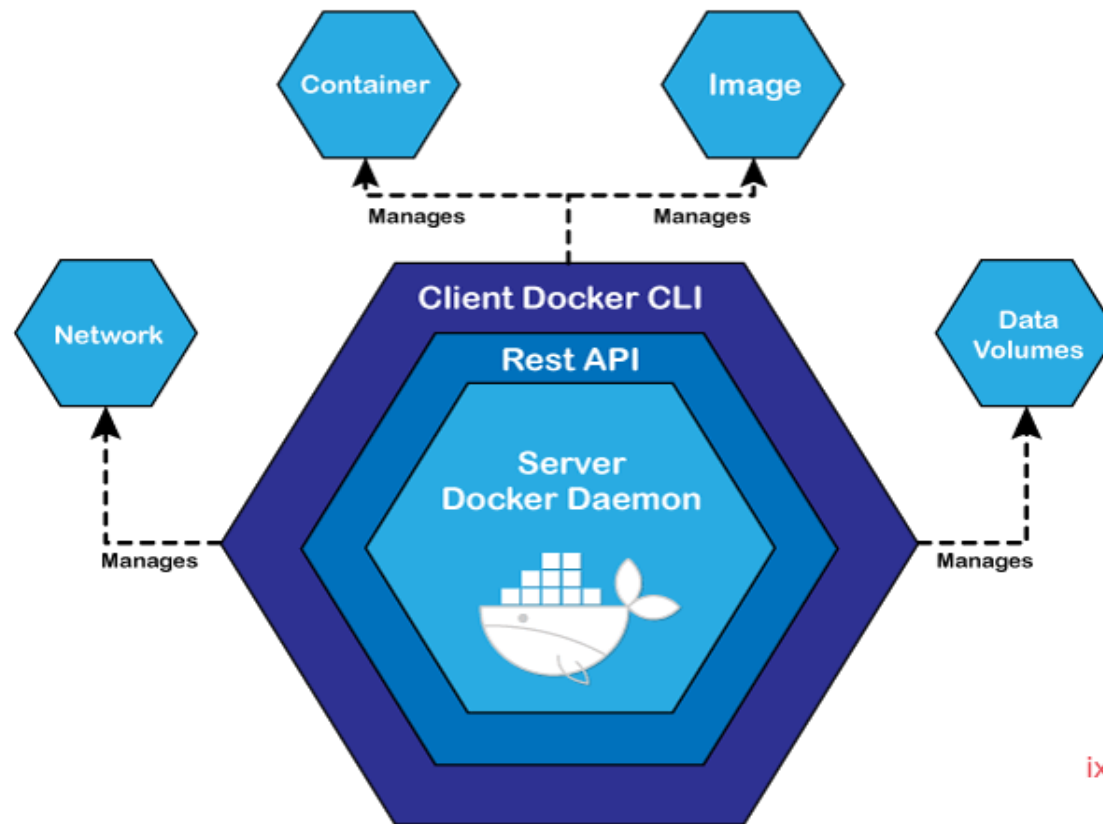


Docker Architecture

- Docker est considéré comme une application **client-serveur** constituant la base de la plateforme de conteneur.
- Docker se compose de deux composants de base à savoir :
 - ✓ Moteur Docker (Docker-Engine)
 - ✓ Registre Docker (Docker hub)

Le moteur Docker (Docker-Engine)

- L'architecture du « moteur Docker » est divisée en trois composants :



ix

Le moteur Docker (Docker-Engine)

- Les trois composants sont :
 - ✓ Terminal du système d'exploitation (Command-Line Interface, CLI) qui interagit avec le daemon docker via l'API REST et permet aux utilisateurs de le contrôler grâce aux scripts ou aux entrées utilisateur.
 - ✓ Interface de programmation REST (API REST) basée sur le paradigme de programmation REST en spécifiant un ensemble d'interfaces qui permettent à d'autres programmes de communiquer et de donner des instructions au daemon Docker.
 - ✓ Le daemon docker : Il s'exécute en arrière-plan sur le système hôte et sert à contrôler le moteur Docker de manière centralisée. Dans cette fonction, il crée et gère toutes les images, conteneurs ou réseaux.
- Le Terminal du système d'exploitation (CLI) utilise l'API REST pour interagir ou contrôler le démon Docker via des commandes CLI directes ou des scripts.


Registre Docker (Docker hub)



- Le registre Docker est basé sur le Cloud pour les référentiels (dépôts) de logiciels, en d'autres termes une sorte de **bibliothèque** pour stocker les images.
- Plusieurs images sont disponibles (des images de base et des images préconfigurées).
- Le service en ligne est divisé en un espace public et un espace privé.
 - L'espace public offre aux utilisateurs la possibilité de télécharger leurs propres images et de les partager avec la communauté.
 - L'espace privé présente une zone restreinte où les images téléchargées du registre ne sont pas accessibles au public et peuvent, par exemple, être partagées au sein de l'entreprise ou avec des amis et des connaissances.







Registre Docker (Docker hub)

- Il est accessible via : <https://hub.docker.com> :

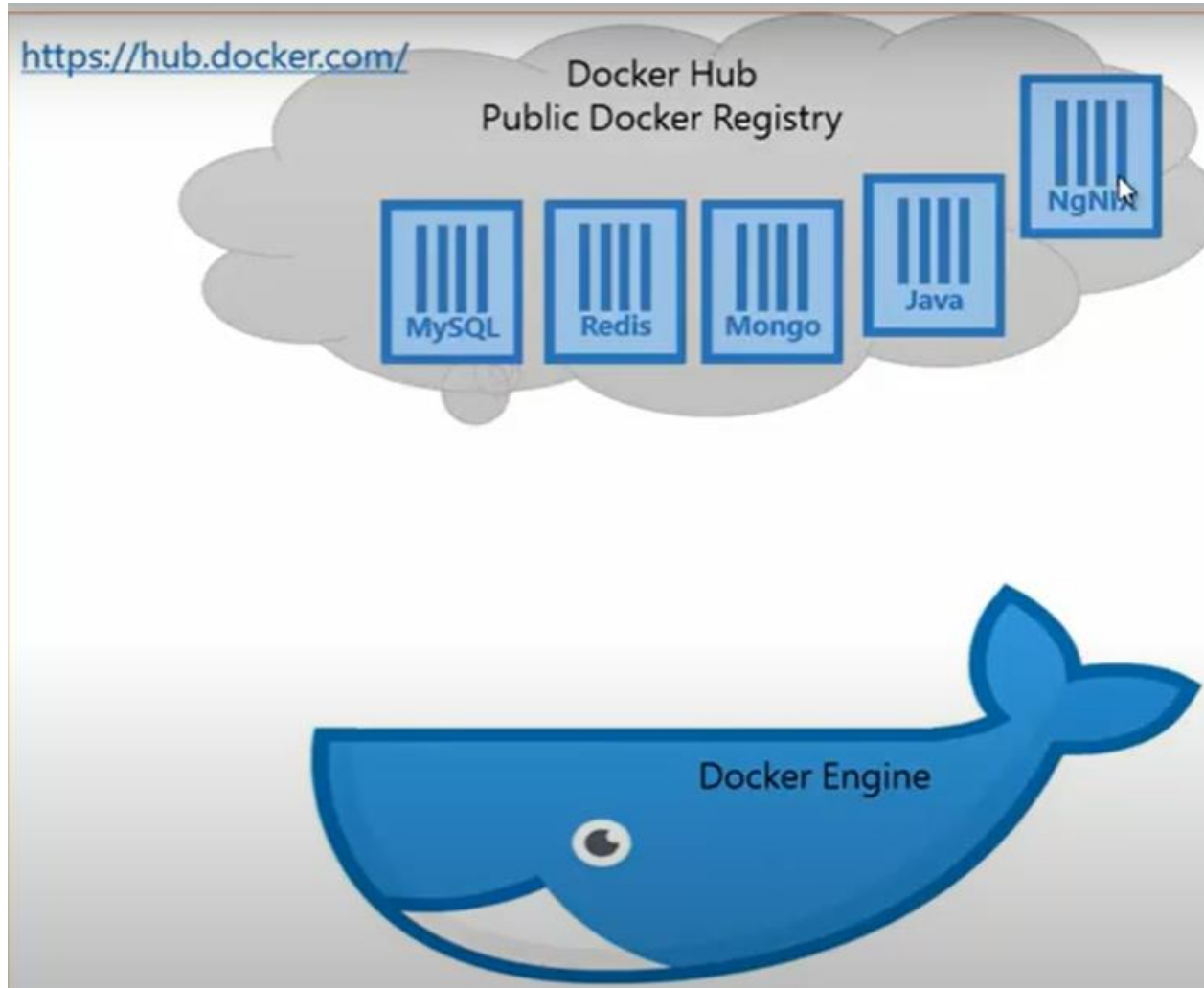


DashboardExploreOrganizationsCreate▼myfreedockerid▼

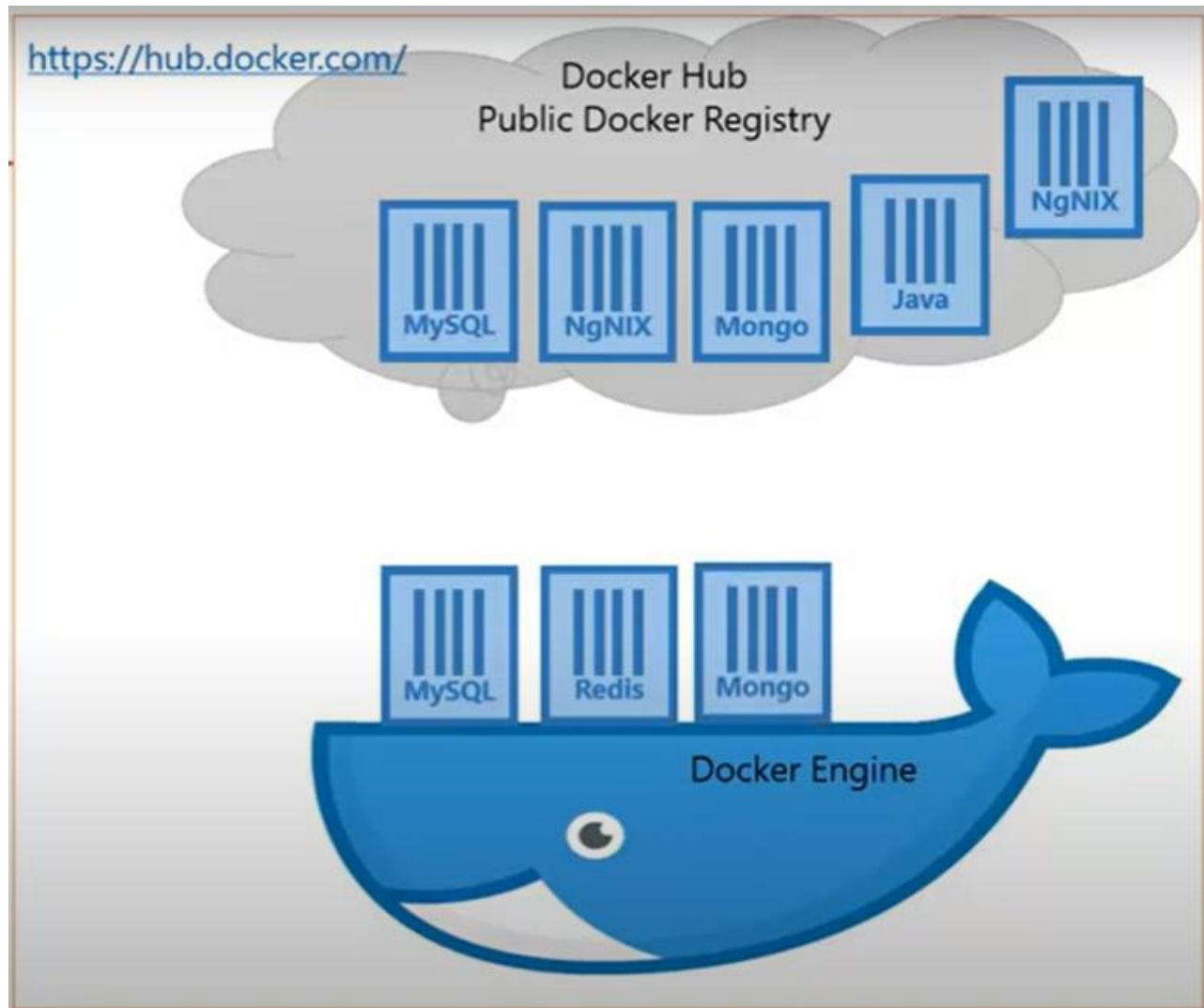
Explore Official Repositories

 nginx official	5.5K STARS	10M+ PULLS	> DETAILS
 redis official	3.5K STARS	10M+ PULLS	> DETAILS
 busybox official	945 STARS	10M+ PULLS	> DETAILS
 ubuntu official	5.6K STARS	10M+ PULLS	> DETAILS
 alpine official	2.0K STARS	10M+ PULLS	> DETAILS
 registry official	1.4K STARS	10M+ PULLS	> DETAILS

Registre Docker (Docker hub)

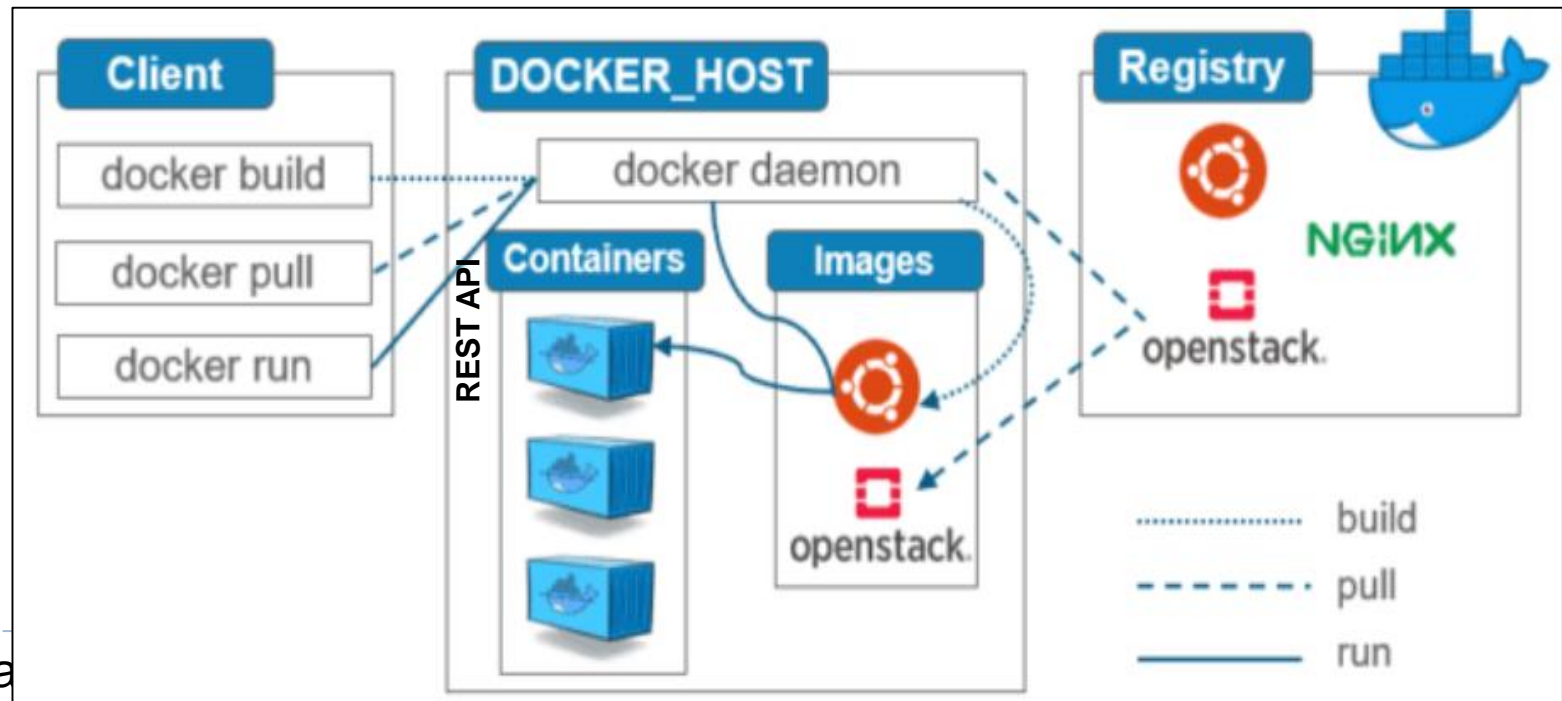


Registre Docker (Docker hub)



Docker : structure et fonctions

- Docker fonctionne sur une architecture **client-serveur**. Il comprend :
 - Le **client Docker** est utilisé pour déclencher les commandes Docker comme : build (créer), pull (télécharger) ou run (démarrer).
 - L'**hôte Docker** permet d'exécuter le démon Docker afin de gérer les images, les conteneurs, les réseaux, les volumes, les plugins, etc.
 - Le **registre Docker** permet de stocker les images Docker.



Les commandes de base

- **Connaitre la version du docker**

\$ docker --version

- **Accéder au menu d'aide du docker**

\$ docker --help

- **Avoir l'aide sur une commande spécifique**

\$ docker pull --help

- **Rechercher une image dans le hub de docker**

\$ docker search ubuntu

- **Afficher l'historique d'une image**

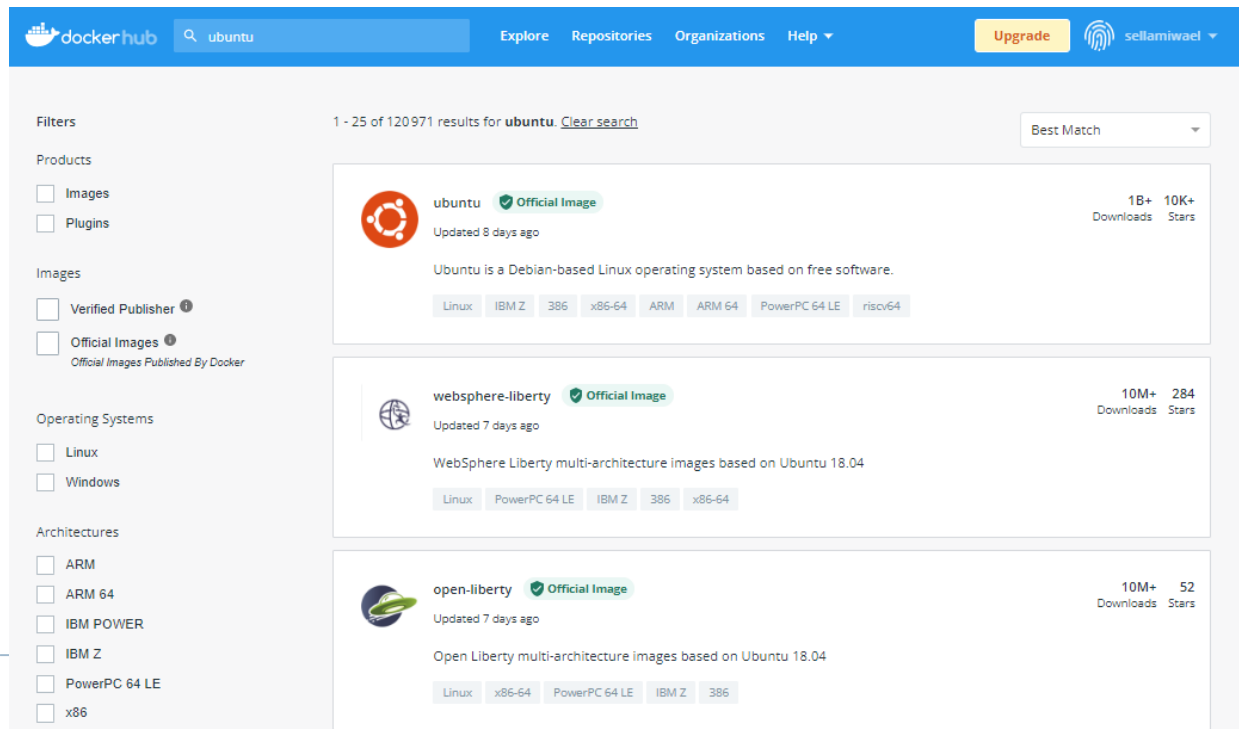
\$ docker history ubuntu

Gestion des images et des conteneurs



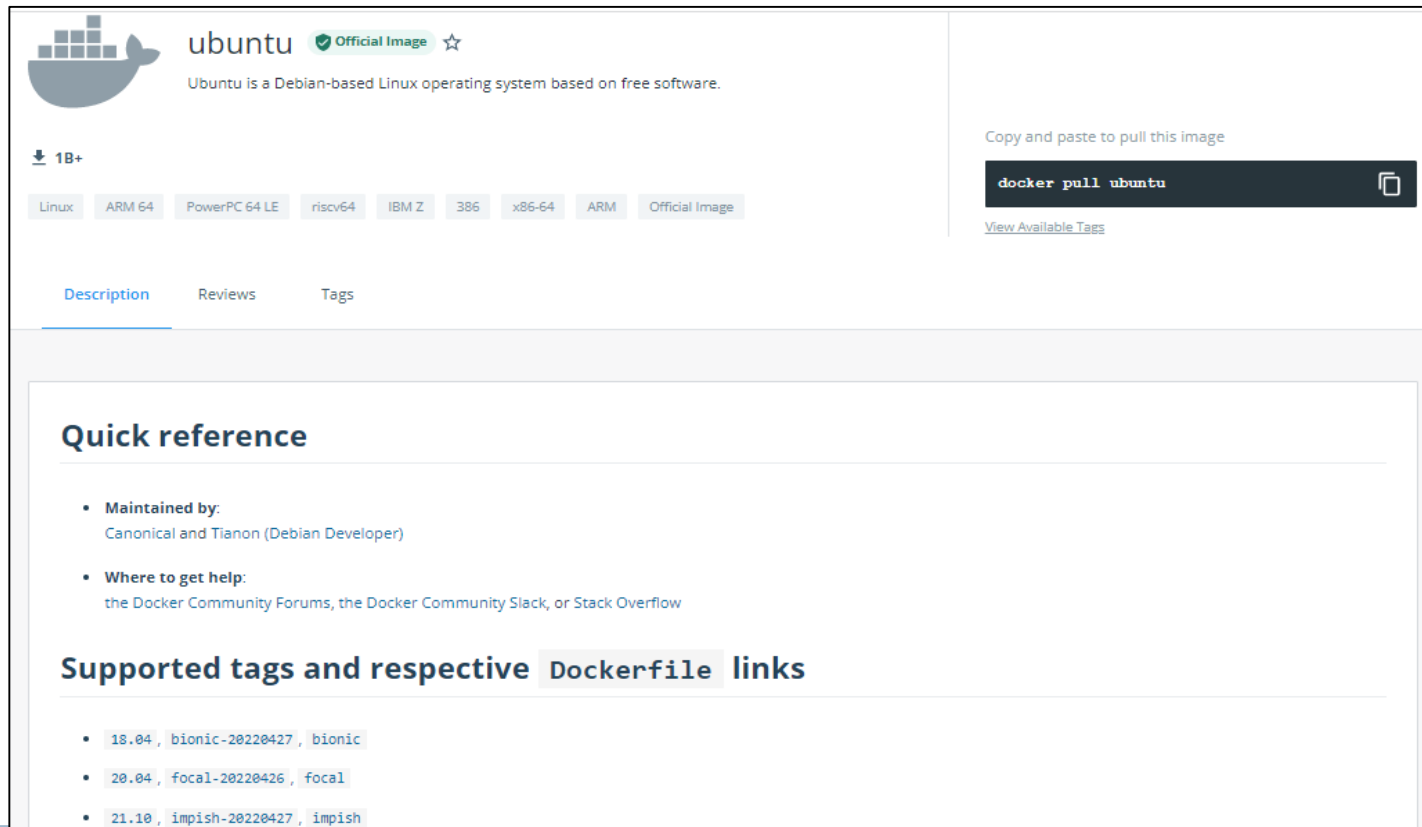
Télécharger les images Docker

- **Exemple :** afin de trouver l'image «ubuntu», il faut accéder à la page d'accueil de «Docker-hub» et en tapant «ubuntu» dans la barre de recherche à droite du logo de Docker.
- Dans les résultats de la recherche, cliquer sur la ressource pour accéder au dépôt public de cette image.



Télécharger les images Docker

- Dans la zone d'en-tête de la page, nous trouvons le nom de l'image, la catégorie du dépôt et l'heure du dernier téléchargement (last pushed).



Télécharger les images Docker

- Structure d'une commande en Docker :

\$ docker <command> <options> <image>

- Pour télécharger une image à partir d'un dépôt:

\$ docker pull [OPTIONS] NAME [:TAG|@DIGEST]

- Déterminer l'image en spécifiant le nom de l'image (NAME).
- Spécifier les tags (:TAG) et les numéros d'identification uniques (@DIGEST) afin de télécharger une version spécifique d'une image.
- **Exemple** : \$ docker pull hello-world

Télécharger les images Docker

- Afficher une vue d'ensemble de toutes les images sur votre système :
\$ docker images

```
osboxes@osboxes: ~  
osboxes@osboxes:~$ sudo docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
hello-world          latest              48b5124b2768        6 weeks ago        1.84 kB  
docker/whalesay      latest              6b362a9f73eb        21 months ago      247 MB
```

- Lorsqu'on démarre un conteneur, l'image sous-jacente est téléchargée sous forme de copie à partir du dépôt et stockée de façon permanente sur votre ordinateur.
- Un nouveau téléchargement n'est lancé que si la source de l'image change, par exemple si une version plus récente est disponible dans le dépôt.

Supprimer les images Docker

Pour supprimer une image :

\$ docker image rm [OPTIONS] IMAGE [IMAGE...]

ou

\$ docker rmi [OPTIONS] IMAGE [IMAGE...]

Démarrer un conteneur

- Pour démarrer une image du docker, utiliser la commande :

```
$ docker run [OPTIONS] IMAGE [:TAG|@DIGEST] [CMD] [ARG...]
```

N.B :

- ✓ Des configurations optionnelles peuvent être définies par des arguments supplémentaires (--name, -network, -d, -it,).
- ✓ La seule partie obligatoire est le **nom de l'image** du docker désiré.
- ✓ Si l'image n'existe pas, elle sera d'abord téléchargée à partir du « docker hub »

Démarrer un conteneur

- **\$ docker run image_name:tag**
 - ✓ Pour spécifier la version (:tag)
 - ✓ Exemple : \$ docker run redis:4.0
- **\$ docker run -d image_name**
 - ✓ -d : pour **détacher** le conteneur du processus principal de la console. Il vous permet de continuer à utiliser la console pendant que votre conteneur tourne sur un autre processus;
 - ✓ La valeur par défaut de l'image est la dernière version
- **\$ docker ps**
 - ✓ Permet de lister les conteneur qui sont en cours d'exécution
 - ✓ Chaque conteneur créé dispose d'un identifiant unique et d'un nom du conteneur.
- **\$ docker ps -a**
 - ✓ Permet de lister tous les conteneur avec leurs status (Up, Exited, Created)

Démarrer un conteneur

- Exemple :

\$ docker run --name cont1 hello-world

- Démarrer/arrêter un conteneur :

- **\$ docker start container1**

- **\$ docker stop container1**

- Lien de 2 conteneurs

- *\$ docker run -it --link <name-of-container> -d <image-name>*

- **\$ docker run -it --name container2 --link container1 -d ubuntu**

Démarrer un conteneur

Exemple :

\$ docker run -it --name docker1 ubuntu /bin/bash

- run : on veut lancer le conteneur
- -it : on veut un terminal et être interactif avec lui
- ubuntu : l'image à utiliser pour ce conteneur
- /bin/bash : on lance « bash »

```
$ docker run -it --name docker1 ubuntu /bin/bash  
root@vm :/# cat /etc/issue  
Ubuntu 20.04.3 LTS  
root@vm :/# exit
```

- le daemon recherche d'abord l'image désirée dans le répertoire du fichier local.
- Comme aucun paquet du même nom n'est trouvé ici, une extraction du dépôt docker est initiée. Ensuite, le démon démarre le programme de « bash ».

Démarrer un conteneur

- Pour afficher une vue d'ensemble de tous les conteneurs exécutés sur le système:

\$ sudo docker ps -a

```
osboxes@osboxes: ~  
osboxes@osboxes:~$ sudo docker ps -a  
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS          NAMES  
f4b2a3131480   docker/whalesay  "cowsay What did t..." 27 minutes ago Exited (0) 27 minutes ago           kickass_bartik  
07e4a970f186   docker/whalesay  "cowsay What did t..." 28 minutes ago Exited (0) 28 minutes ago           serene_bohr  
a5fd4f05a980   docker/whalesay  "cowsay boo"             40 minutes ago Exited (0) 40 minutes ago           zen_fermat  
98344a7a2a9a   docker/whalesay  "cowsay"                 17 hours ago   Exited (0) 17 hours ago           wizardly_payne  
277225cdcf03   docker/whalesay  "cowsay boo"             17 hours ago   Exited (0) 17 hours ago           distracted_euclid  
74ffb508094b   docker/whalesay  "/bin/bash"              17 hours ago   Exited (0) 17 hours ago           sad_sammet  
6f88a421cd40   docker/whalesay  "cowsay boo"             18 hours ago   Exited (0) 18 hours ago           quirky_visvesvaraya  
fdccbb23913a   hello-world     "/hello"                 4 days ago     Exited (0) 4 days ago           quirky_turing  
1b75142bdc01   hello-world     "/hello"                 5 days ago     Exited (0) 5 days ago           heuristic_montalcini  
d22ed3918a8a   hello-world     "/hello"                 5 days ago     Exited (0) 5 days ago           silly_khorana  
osboxes@osboxes:~$
```

- La sortie du terminal comprend des informations telles que l'**ID** du conteneur respectif, l'**image** sous-jacente, la **commande exécutée** lorsque le conteneur a été démarré, l'**heure** à laquelle le conteneur respectif a été démarré et l'**état** actuel.

RUN – Port Mapping

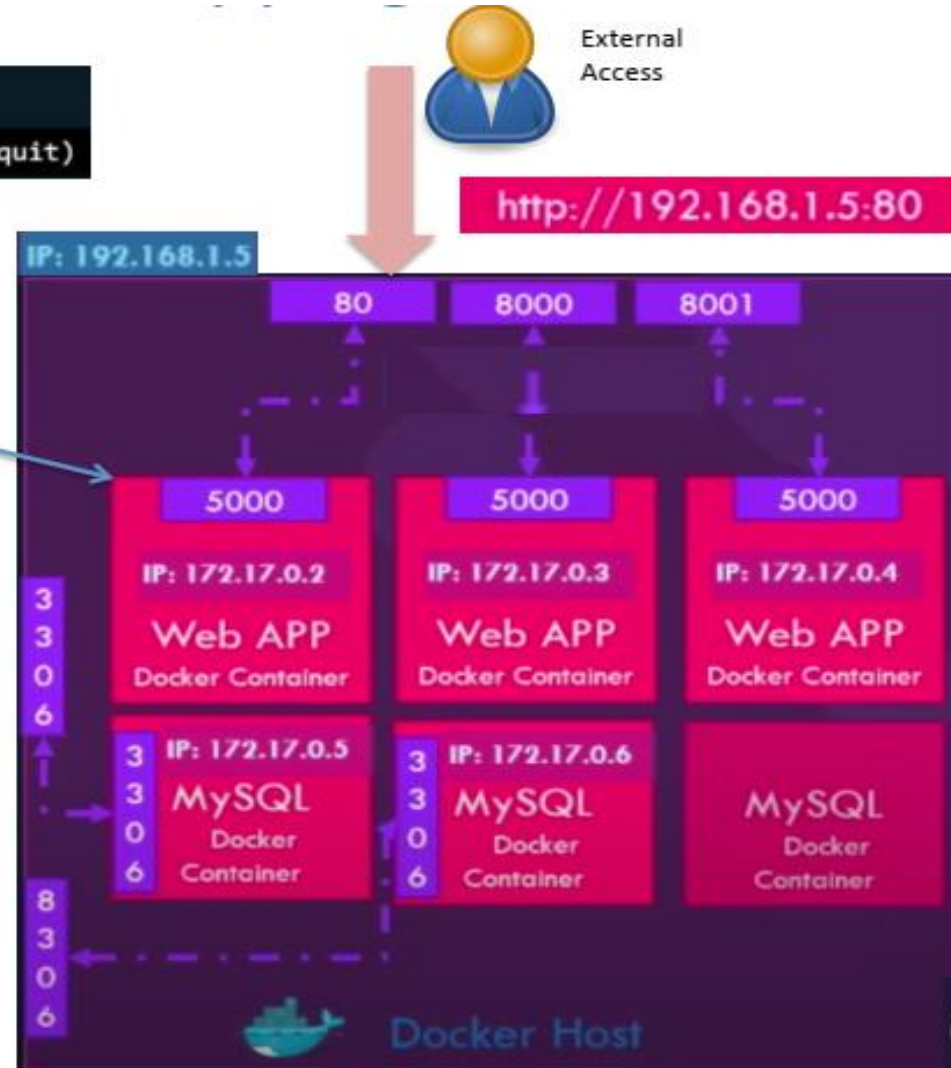
```
docker run kodecloud/webapp
```

* Running on <http://0.0.0.0:5000/> (Press CTRL+C to quit)

<http://172.17.0.2:5000>

Internal Access

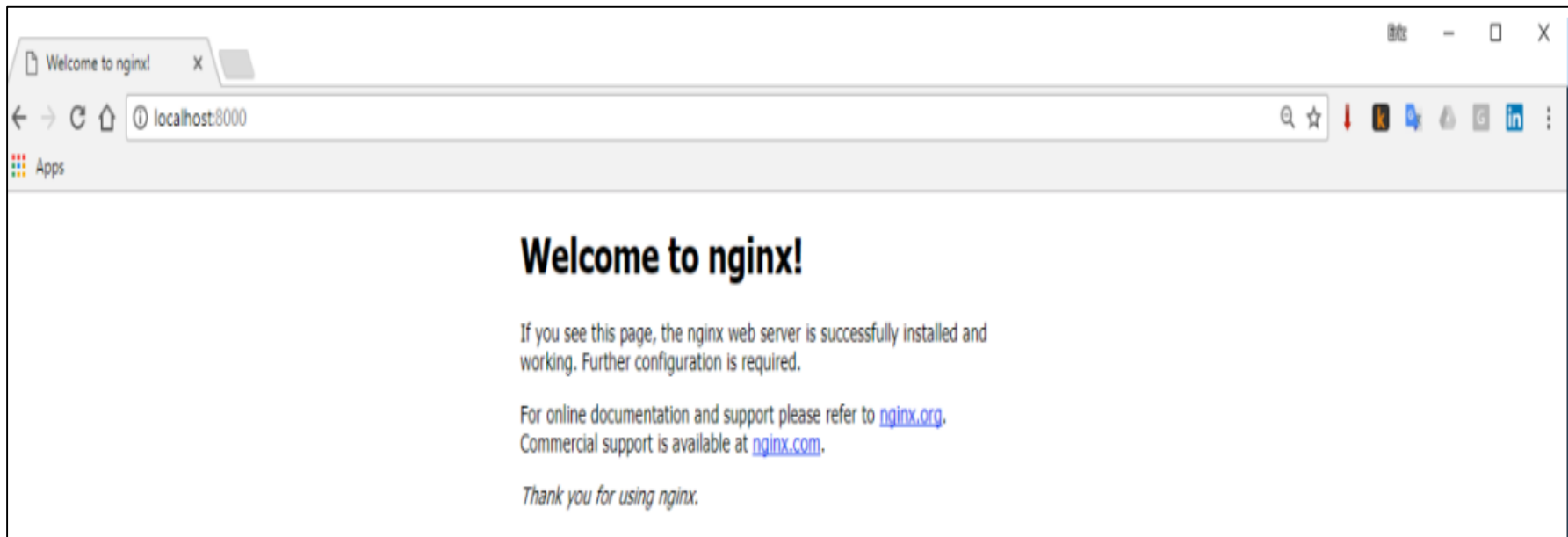
```
docker run -p 80:5000 kodecloud/simple-webapp
docker run -p 8000:5000 kodecloud/simple-webapp
docker run -p 8001:5000 kodecloud/simple-webapp
docker run -p 3306:3306 mysql
docker run -p 8306:3306 mysql
```



Manipulation des conteneurs

Démarrons **Nginx** (serveur HTTP + reverse proxy) en container et exposons le port 80 sur le port 8000 de notre machine :

```
$ docker run -p 8000:80 --name nginx_name -d nginx
```



```
$ docker ps
```

```
$ docker stop nginx_name
```


EXEC Command

- Permet d'exécuter une commande dans un conteneur démarré.
- Exemple : Démarrer un conteneur MySQL

```
$ docker run --name mysqlCont1 -d mysql -e MYSQL_ROOT_PASSWORD=root
```

EXEC Command

- Démarrage le client mysql dans le conteneur démarré my-mysql
- Pour le faire en mode interactif, il faut utiliser les options -it (input terminal)

```
$ sudo docker exec -it my-mysql mysql --password
```

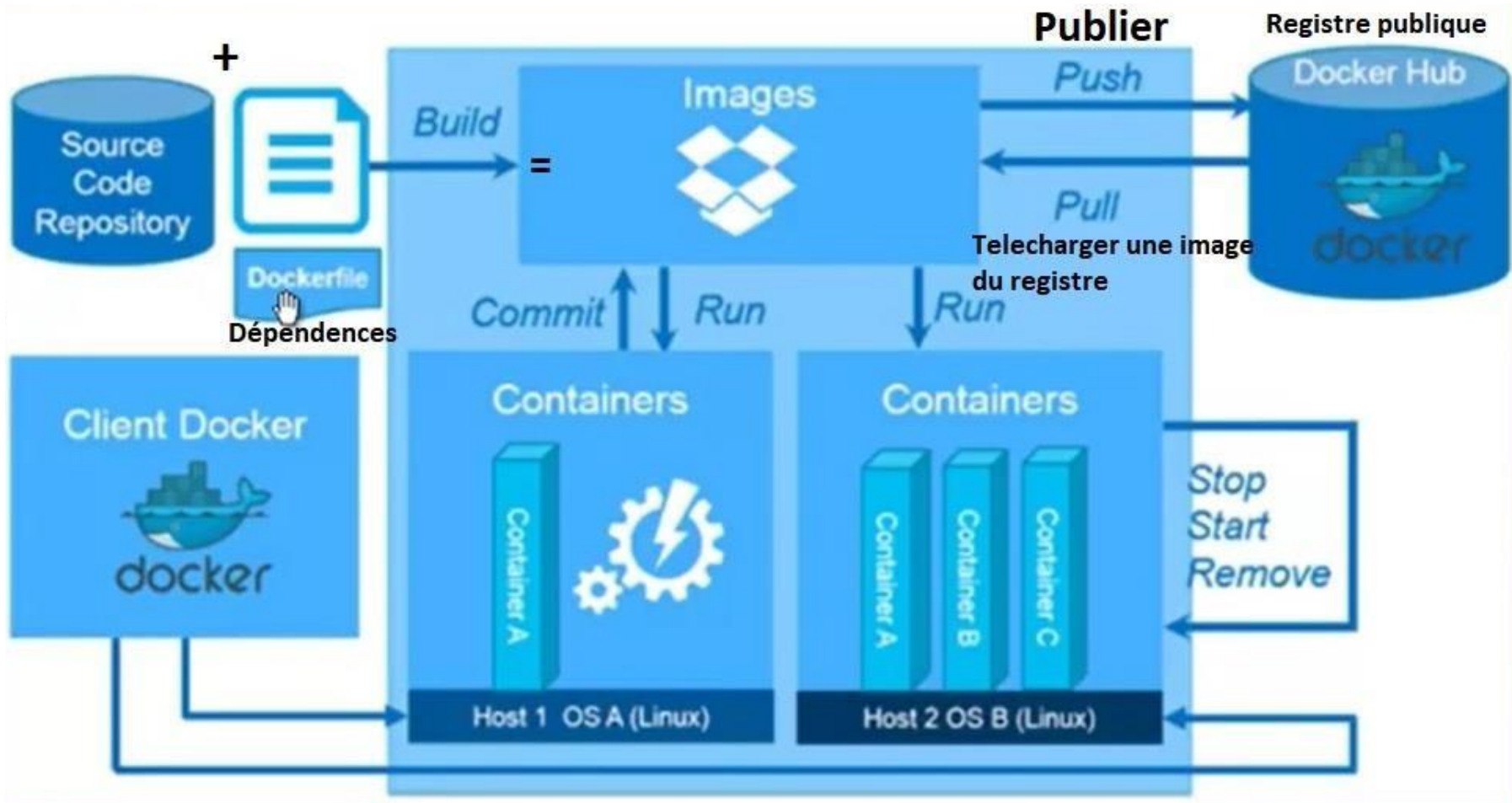
```
youssfi@youssfi-VirtualBox: ~  
File Edit View Search Terminal Help  
youssfi@youssfi-VirtualBox:~$ sudo docker exec -it my-mysql mysql --password  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 16  
Server version: 8.0.17 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.07 sec)  
  
mysql> █
```

Démarrage d'un conteneur

En quoi consiste le démarrage du container ?

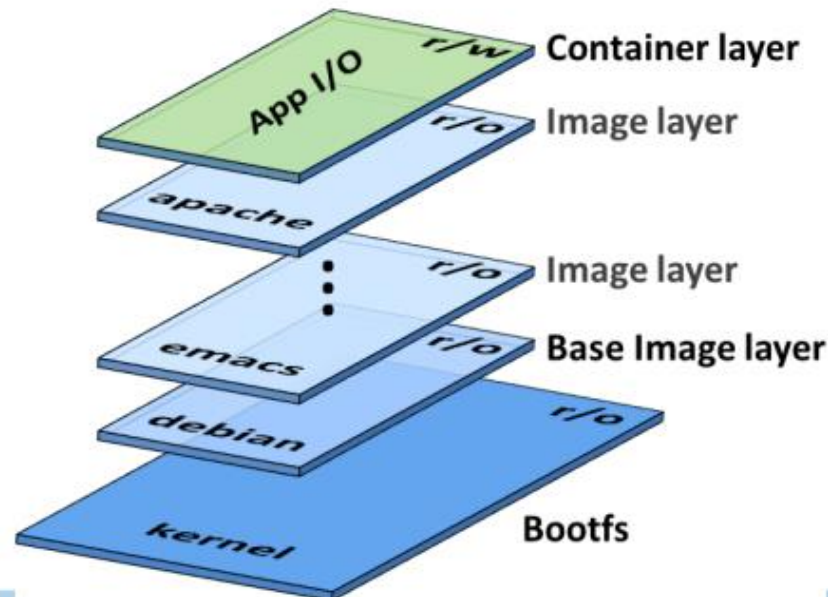
1. Rechercher l'image ➔ Si l'image n'existe pas en local, alors téléchargement via le hub.
2. Créer un nouveau conteneur basé sur cette image et se préparer à démarrer
3. Attacher le conteneur au réseau privé et obtention d'une adresse IP.
4. Ouvrir les ports pour répondre aux requêtes
5. Capturer des messages entrées-sorties

Création d'une image



Création d'une image

- Une image docker est une succession de couches (layers) qui contiennent une liste de modifications du système de fichiers. On peut faire une analogie avec GIT où chaque layer serait un commit.
- Au moment de la création d'un conteneur, une couche est ajouté au-dessus de ceux de l'image.

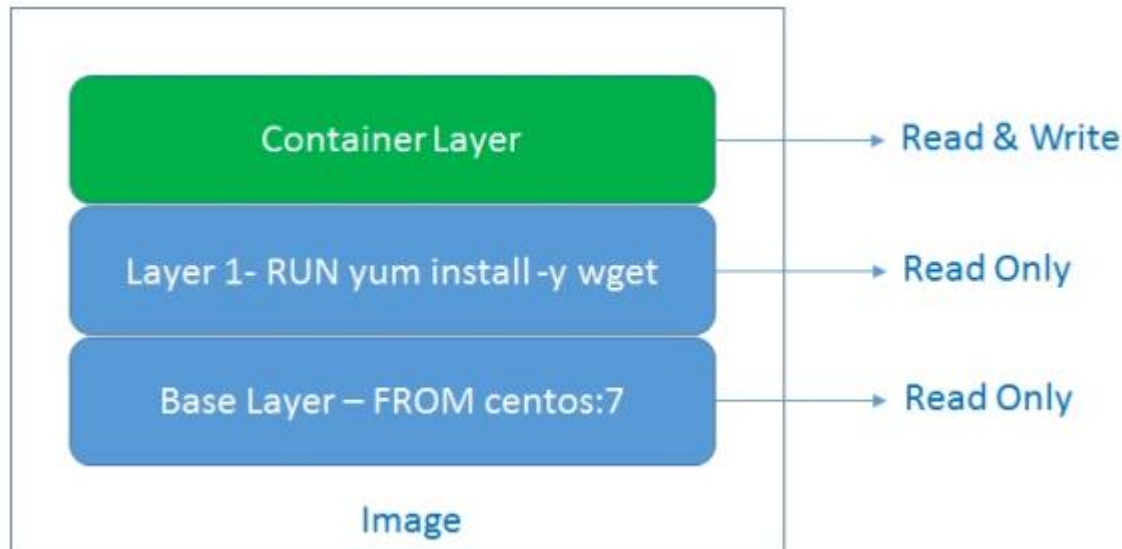


Création d'une image

- Le développeur crée un fichier **Dockerfile** contenant les commandes que docker va exécuter pour construire une image docker de cette application.
- L'image docker contient tout ce dont l'application a besoin pour s'exécuter correctement.
 - **\$ docker build**
- Les images docker peuvent être publiées dans un registre publique (Docker hub) ou privé.
 - **\$ docker push image_name**

Dockerfile

- Le Dockerfile est un fichier qui contient toutes les instructions et arguments pour créer une image comme des métadonnées (mainteneur, label, etc.), ou même les commandes à exécuter pour installer un logiciel.
- Chaque instruction on va créer une nouvelle layer correspondant à chaque étape de la construction de l'image, ou de la recette.



Instructions du Dockerfile

Les instructions de Dockerfile :

- **FROM** : définir l'image **source**; Elle n'est utilisable qu'une seule fois dans un Dockerfile.
- **CMD** : définir la commande qui doit être exécutée lors de démarrage du conteneur
- **RUN** : exécuter des **commandes** dans le conteneur ;
- **WORKDIR** : définir votre **répertoire de travail** ;
- **ADD** : ajouter des fichiers dans le conteneur ;
- **COPY** : copier des fichiers dans le conteneur ;
- **EXPOSE** : définir les **ports d'écoute** par défaut ;
- **VOLUME** : définir le répertoire à partager avec la machine hôte.

Dockerfile

Structure de la commande :

INSTRUCTION

ARGUMENT

Dockerfile

```
FROM Ubuntu

RUN apt-get update
RUN apt-get install python

RUN pip install flask
RUN pip install flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

1. OS - Ubuntu

2. Update apt repo

3. Install dependencies using apt

4. Install Python dependencies using pip

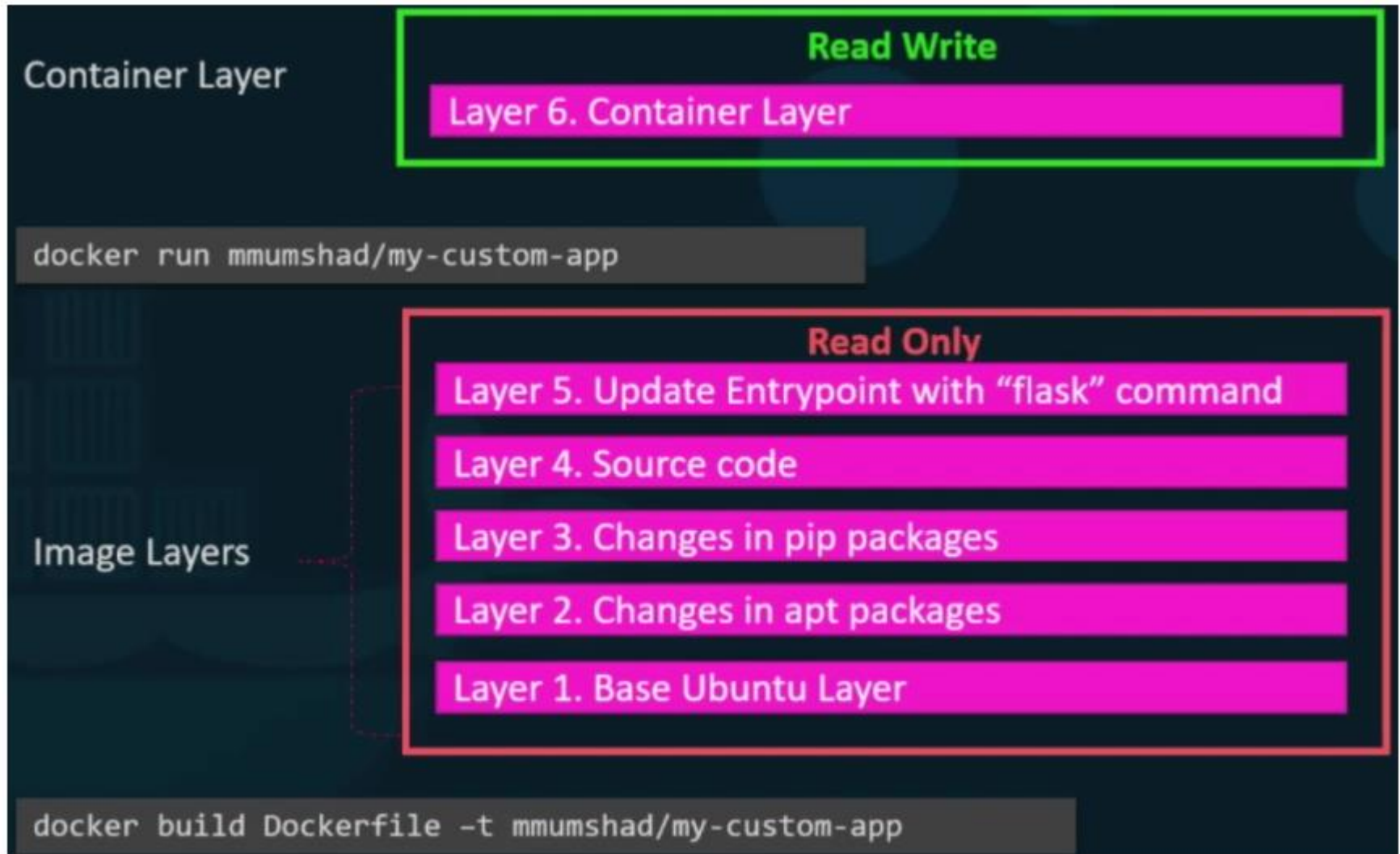
5. Copy source code to /opt folder

6. Run the web server using "flask" command

- Créer une image nommée « docker-image-simple » à partir du fichier Dockerfile (y compris le point à la fin):

```
$ docker build Dockerfile -t polytech/docker-image-simple .
```

Architecture en couche



Exemple d'un Dockerfile

- FROM debian:9
- RUN apt-get update -yq \
- && apt-get install curl gnupg -yq \
- && curl -sL https://deb.nodesource.com/setup_10.x | bash \
- && apt-get install nodejs -yq \
- && apt-get clean -y
- ADD . /app/
- WORKDIR /app
- RUN npm install
- EXPOSE 2368
- VOLUME /app/logs
- CMD npm run start

La persistance des données

Volume

- Sauvegarder les données d'un conteneur

- **Créer un volume :**

`$ docker volume create <VOLUMENAME>`

Exemple :

`$ docker volume create volume-test`

- **Pour lister les volumes :**

`$ docker volume ls`

La persistance des données

Volume

- Pour récolter des informations sur un volume, il faut utiliser la commande suivante :

```
$ docker volume inspect volume-test
```

Résultat sous format JSON:

```
[
  {
    "CreatedAt": "2019-07-03T10:03:20+02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/volume-test/_data",
    "Name": "volume-test",
    "Options": {},
    "Scope": "local"
  }
]
```

La persistance des données

Volume

- Créer et monter le volume « dataVolume » dans le dossier « **/data** » du conteneur.

```
$ docker run -it -v dataVolume:/data ubuntu /bin/bash
```

- **Pour supprimer un volume :**

```
$ docker volume rm dataVolume
```

Tag et espace de nom des images

Les images peuvent avoir des tags :

- Les tags symbolisent des différences de version d'une image
- C'est le tag:latest qui est utilisé par défaut

Les images disposent de trois espaces de nom :

- Racine : ubuntu
- Utilisateur et organisations : polytech/ts-node
- Auto-hébergées (le serveur) : localhost:5000/myapache

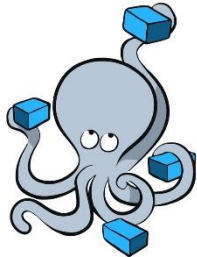
Exercice

En utilisant l'image php:7.0-apache

Afficher l'index.php contenant :

```
<?php echo "Hello tout le monde";?>
```

Orchestration de conteneurs



docker
Compose



kubernetes

Orchestration des conteneurs

- L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs. Les entreprises qui ont besoin de déployer et de gérer des centaines ou des milliers de conteneurs Linux® et d'hôtes peuvent tirer parti de l'orchestration des conteneurs.
- Applications multi-conteneurs : une application Docker complexe qui inclut plusieurs conteneurs (par exemple, un serveur Web et une base de données s'exécutant dans des conteneurs distincts),
- La construction, l'exécution et la connexion des conteneurs à partir de fichiers Docker distincts sont fastidieuses et chronophages.

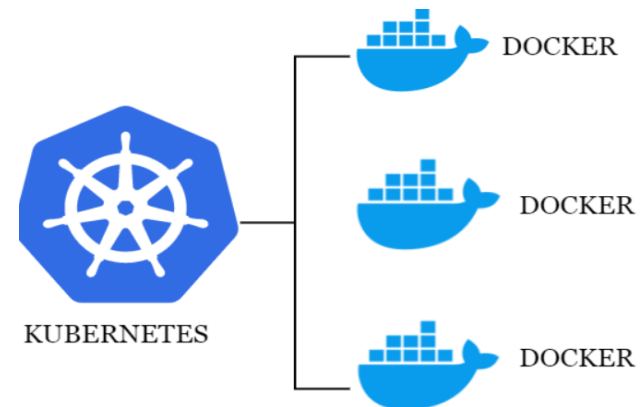
À quoi sert l'orchestration des conteneurs ?

L'orchestration des conteneurs pour automatiser et gérer les tâches suivantes :

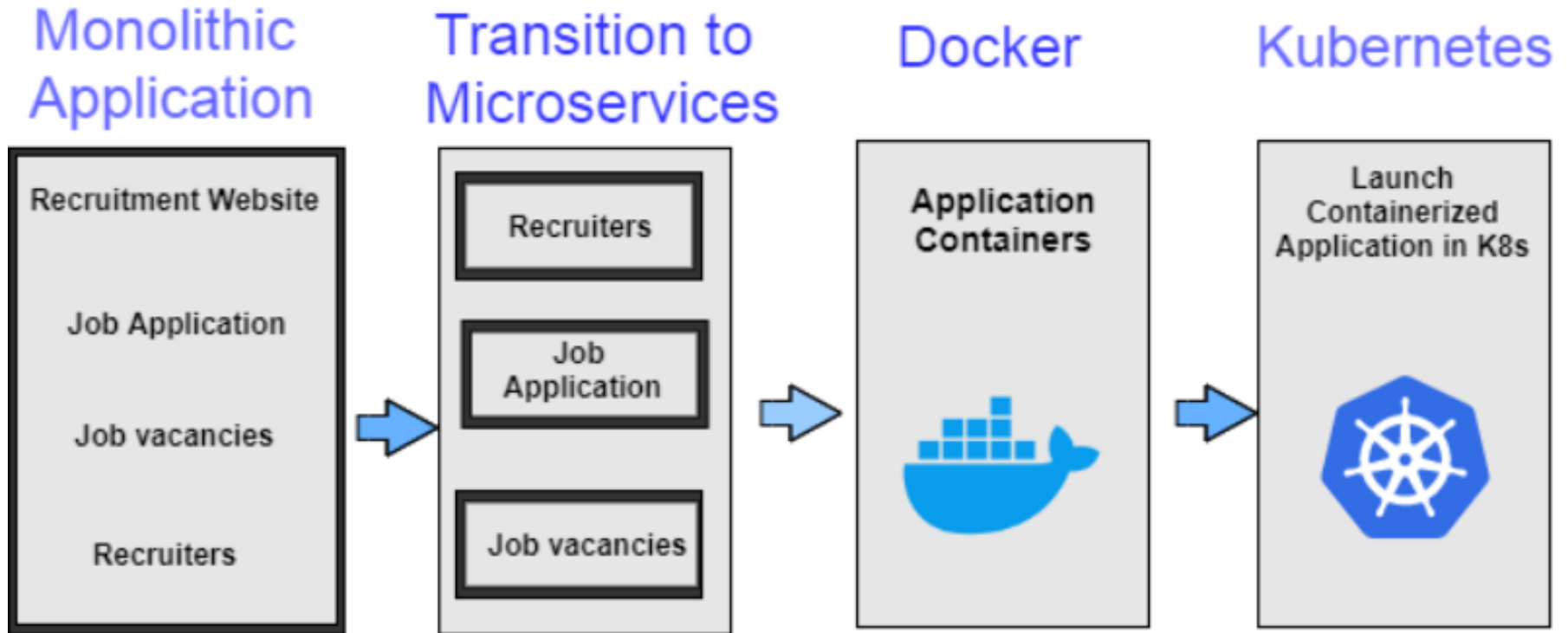
- Provisionnement et déploiement
- Configuration et planification
- Disponibilité des conteneurs
- Mise à l'échelle ou suppression de conteneurs en fonction des charges de travail dans l'infrastructure
- Équilibrage de la charge et routage du trafic
- Surveillance de l'intégrité des conteneurs
- Configuration des applications en fonction du conteneur sur lequel elles vont s'exécuter
- Sécurisation des interactions entre les conteneurs

Outils d'orchestration des conteneurs

- Les outils d'orchestration des conteneurs fournissent un cadre pour la gestion à grande échelle de l'architecture de conteneurs et de microservices.
- Beaucoup de solutions sont disponibles sur le marché pour aider à gérer le cycle de vie des conteneurs. Parmi les plus connues, on peut citer :
 - ✓ Kubernetes,
 - ✓ Docker Compose
 - ✓ Docker Swarm
 - ✓ Apache Mesos.



Outils d'orchestration des conteneurs





Fin de ce chapitre

Avez – vous des questions ?

