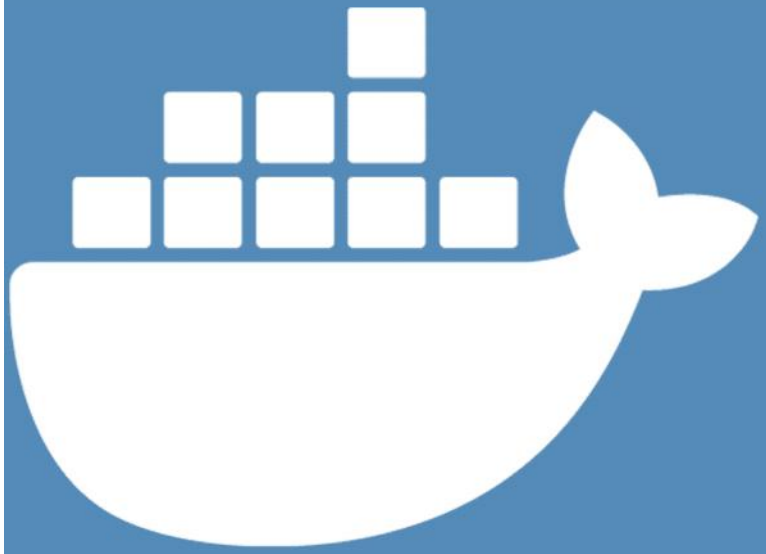


DOCKER – LIVRAISON CONTINUE

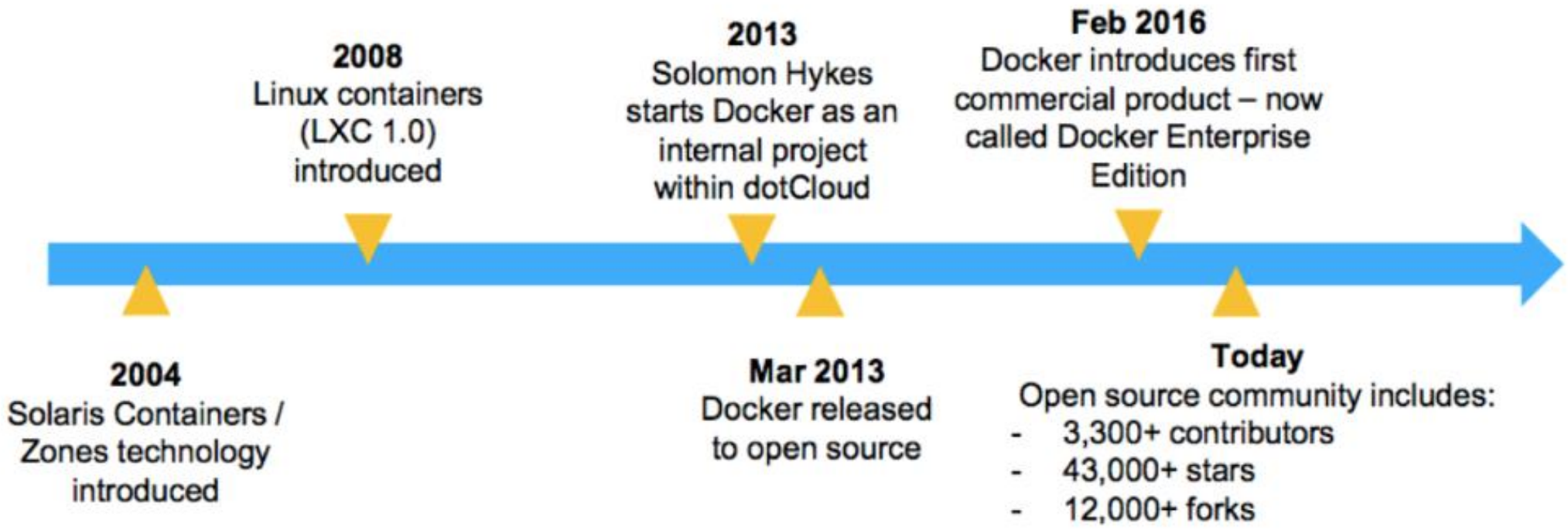
2021-2022

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)

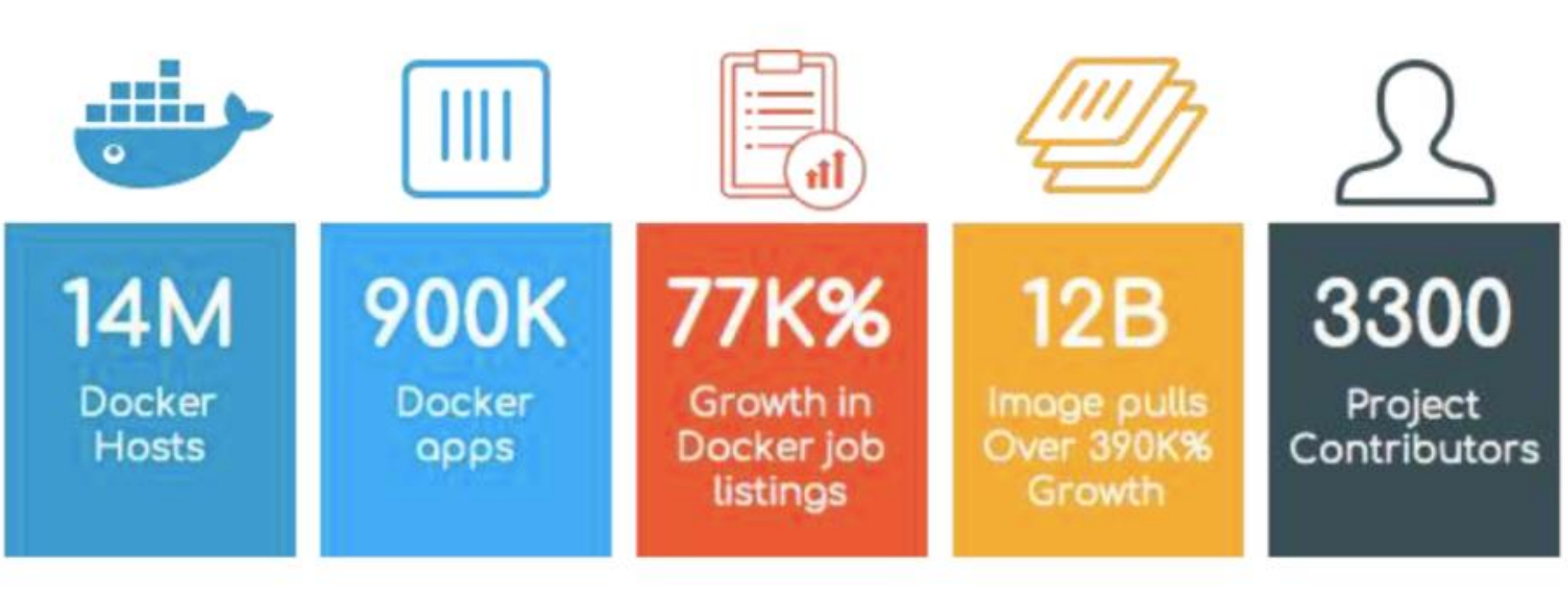
Plan

- Séquence : Docker
  - Charge : 3 heures
    - Concepts Docker
    - Conteneurs et machines virtuelles – Docker Historique, Adoption incroyable en 4 ans, Projets Docker, Evolution des SI, Qu'est ce qu'un Conteneur ?
    - Docker Engine
    - Images et conteneurs (Conteneur vs VMs),
    - Architecture Docker
    - Les images Docker
    - Différence entre image et conteneur
    - Layers
    - Composition des conteneurs
    - **TP1:** Créer et exécuter un conteneur (docker run détails), docker containers instances, docker container isolation [LAB 1]
    - **TP2 :** Docker avec Jenkins [LAB 2]

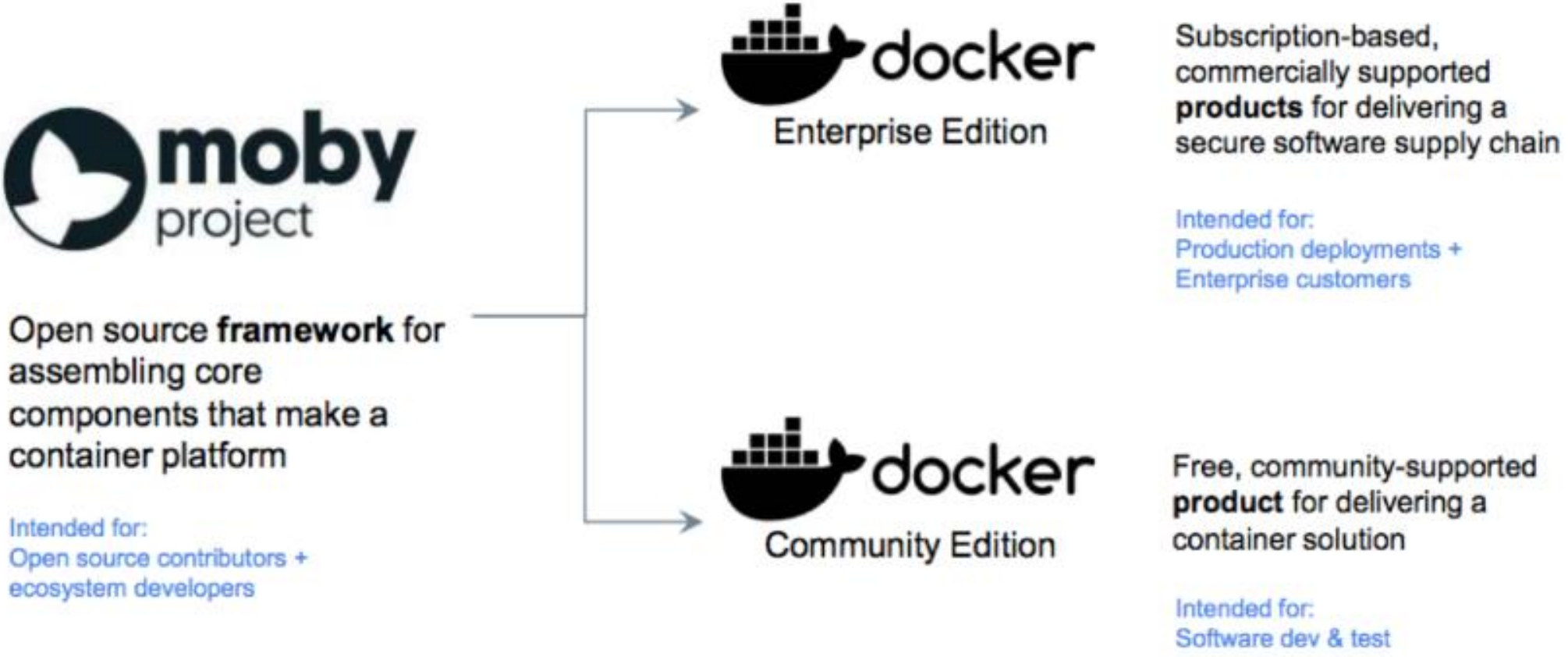
Historique


2004
Solaris Containers / Zones technology introduced
2008
Linux containers (LXC 1.0) introduced
Mar 2013
Docker released to open source
2013
Solomon Hykes starts Docker as an internal project within dotCloud
Feb 2016
Docker introduces first commercial product – now called Docker Enterprise Edition
Today
Open source community includes:
- 3,300+ contributors
- 43,000+ stars
- 12,000+ forks

Adoption incroyable en 4 ans

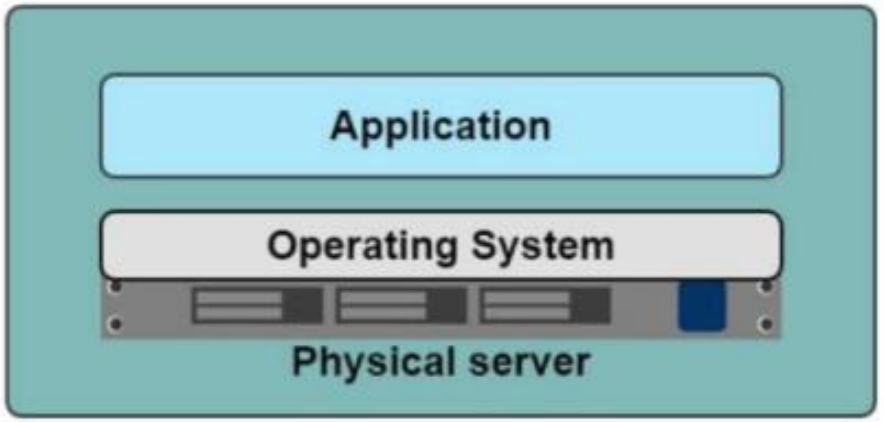


Projets Docker



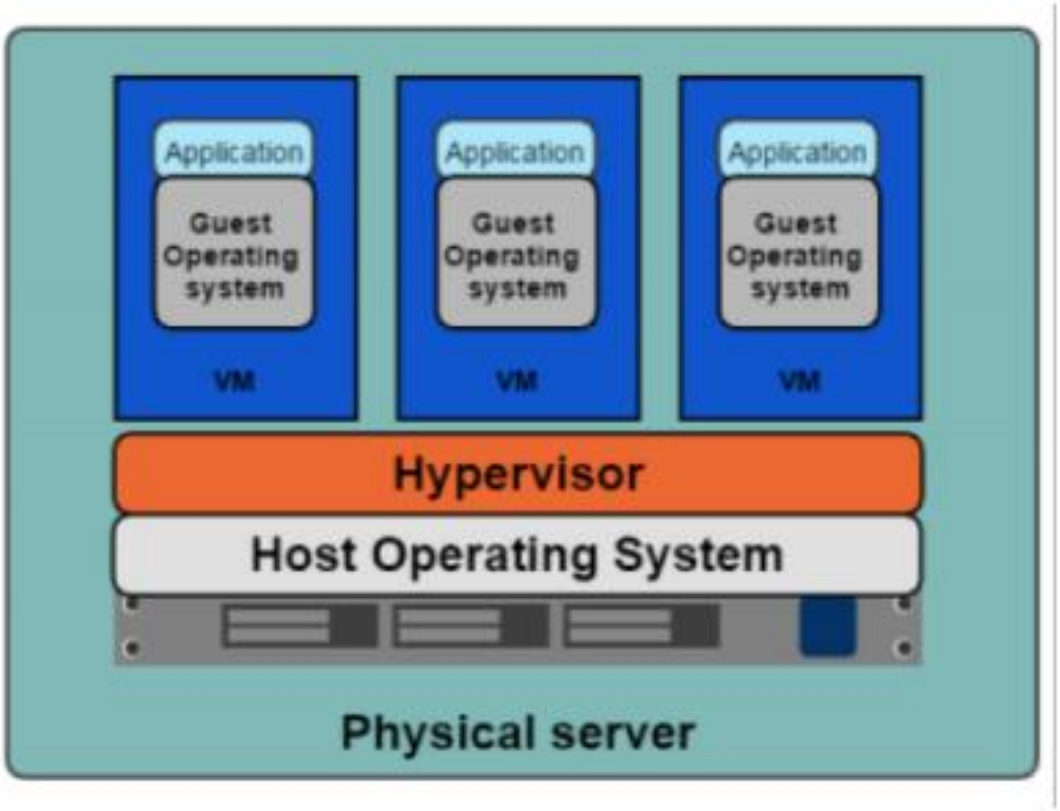
Évolution des SI

- Une seule application dans un seul serveur physique
- Limites
  - Temps de déploiement important
  - Gaspillage de ressources
  - Passage à l'échelle
  - Migration
  - Vendor lock-in



Évolution des SI

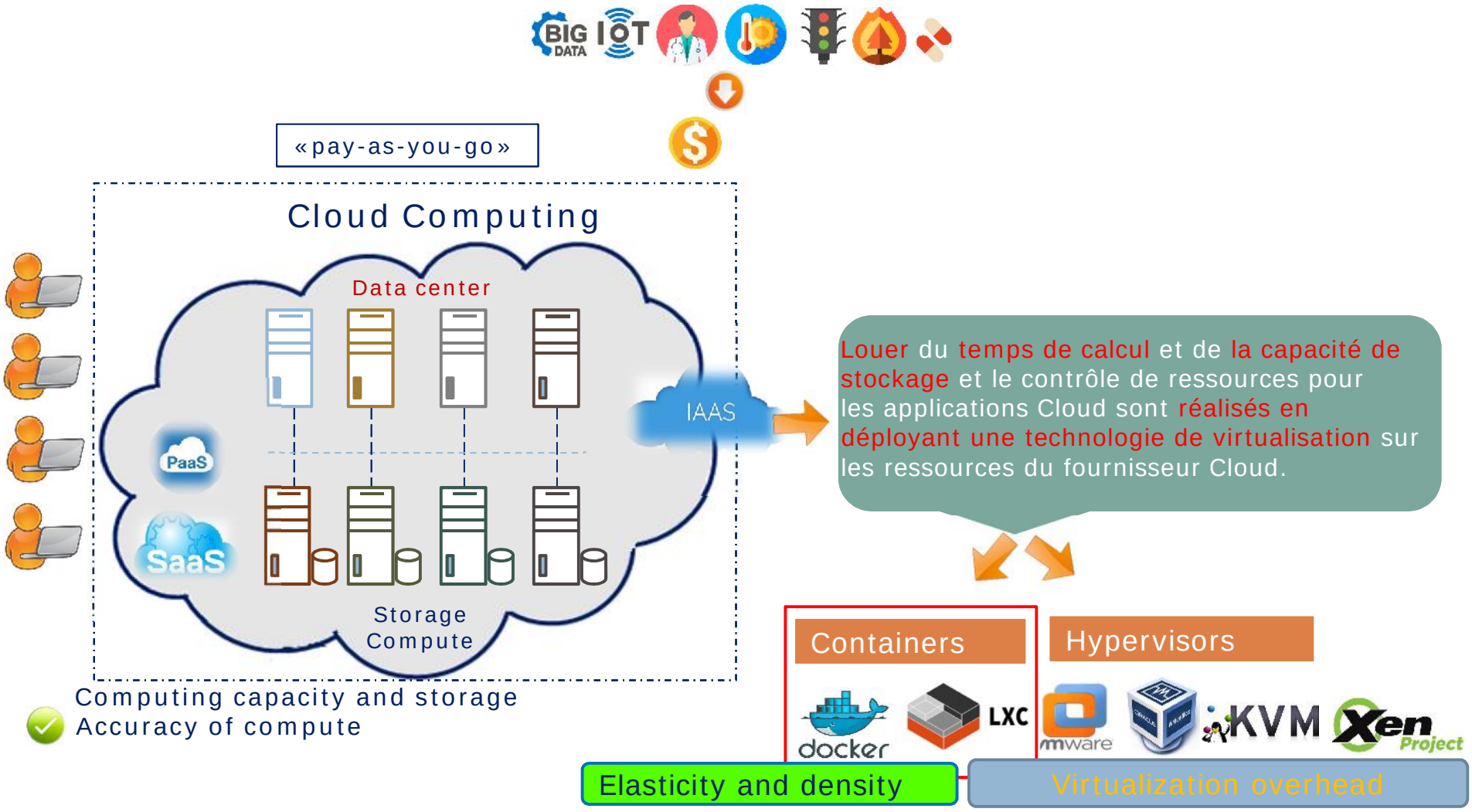
- Virtualisation à base d'hyperviseur.
  - Un serveur physique peut héberger plusieurs applications.
  - Chaque application est déployée dans une machine virtuelle.



Évolution des SI

- Avantage des VMs
  - Meilleur exploitation des ressources
  - Une machine physique est divisée en plusieurs machines virtuelles
  - Plus facile de passer à l'échelle
  - VMs dans le Cloud (Azure, Amazon, Vmware)
  - Elasticité rapide
  - Modèle Pay as You Go

Évolution des SI



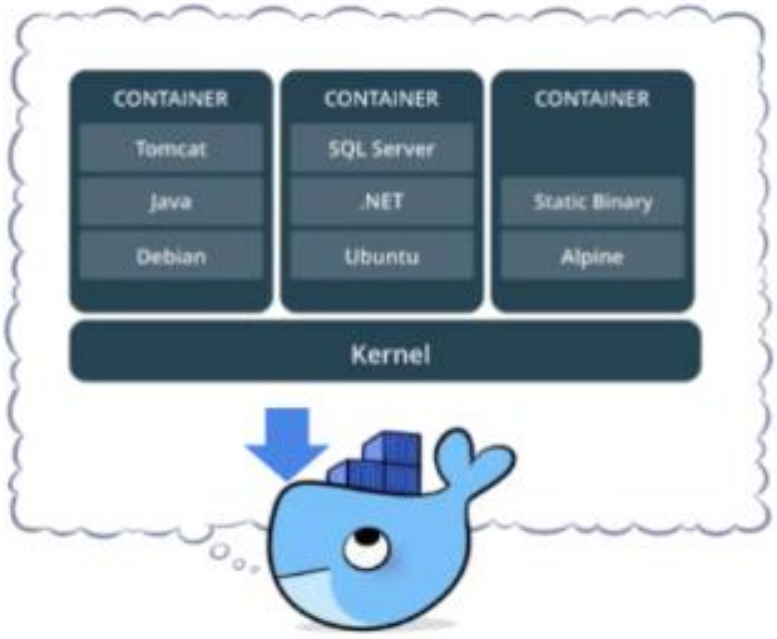
Évolution

- Limites des VMs
  - Chaque VM a besoin
    - CPU
    - Stockage (Disque)
    - RAM
    - Un OS invité


  - En augmentant les VMs, on demande plus de ressources
  - Chaque OS invité alloue ses propres ressources
    - Gaspillage
  - Portabilité d'application n'est pas garantie

Qu'est ce qu'un Conteneur ?

- Emballage (Packaging) standardisé des applications et leurs dépendances
  - Isolation des applications (les unes des autres)
  - Partage du même noyau du système d'exploitation
  - Supporté par tous les systèmes Linux et Windows Server



Conteneur vs VMs

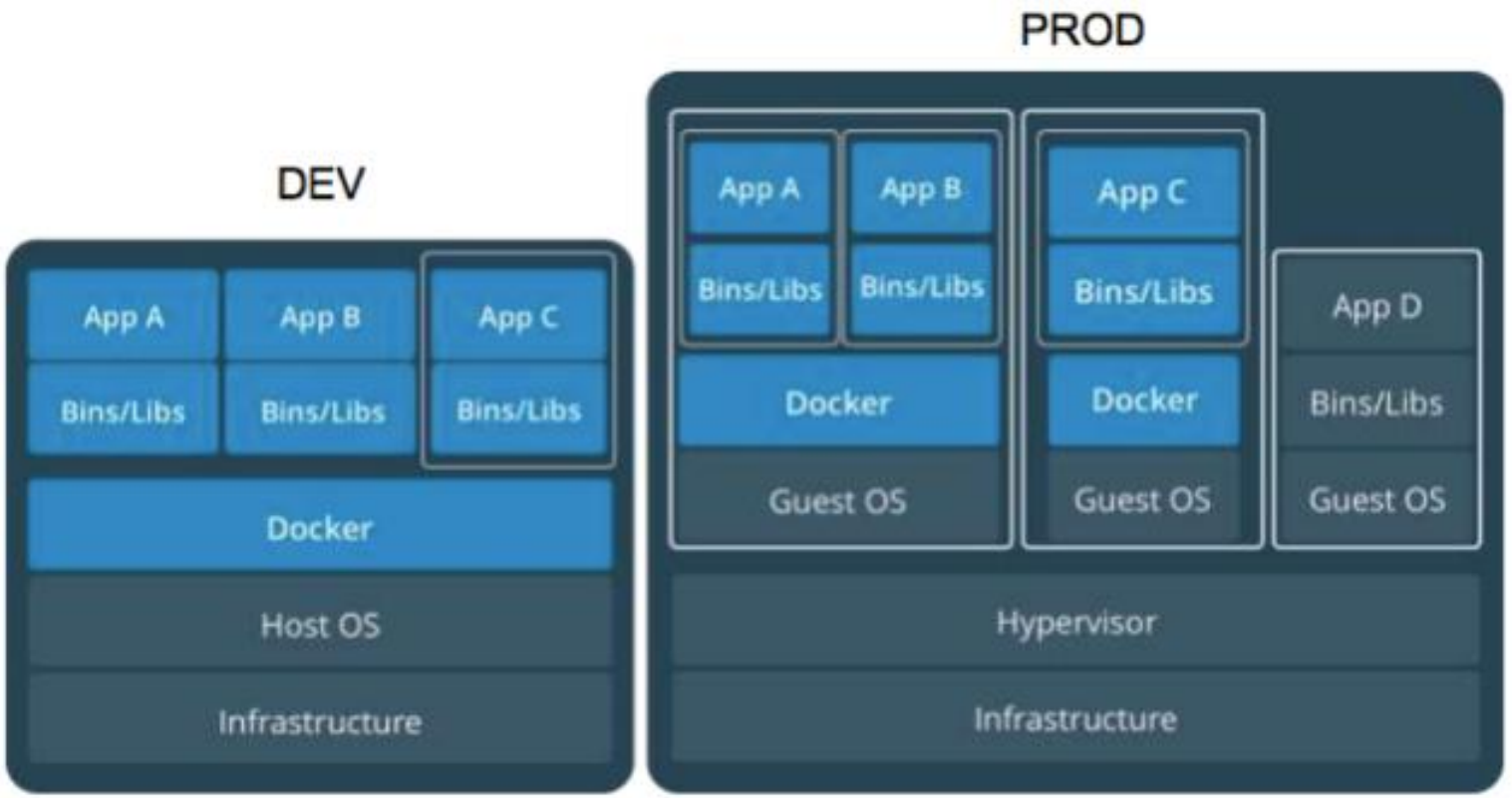
- Les **conteneurs** sont construits au niveau de l'application (App Level)
- Les **VMs** sont au niveau de l'infrastucture (Infra.Level) permettant de transformer la machine en plusieurs serveurs.

Conteneur vs VMs

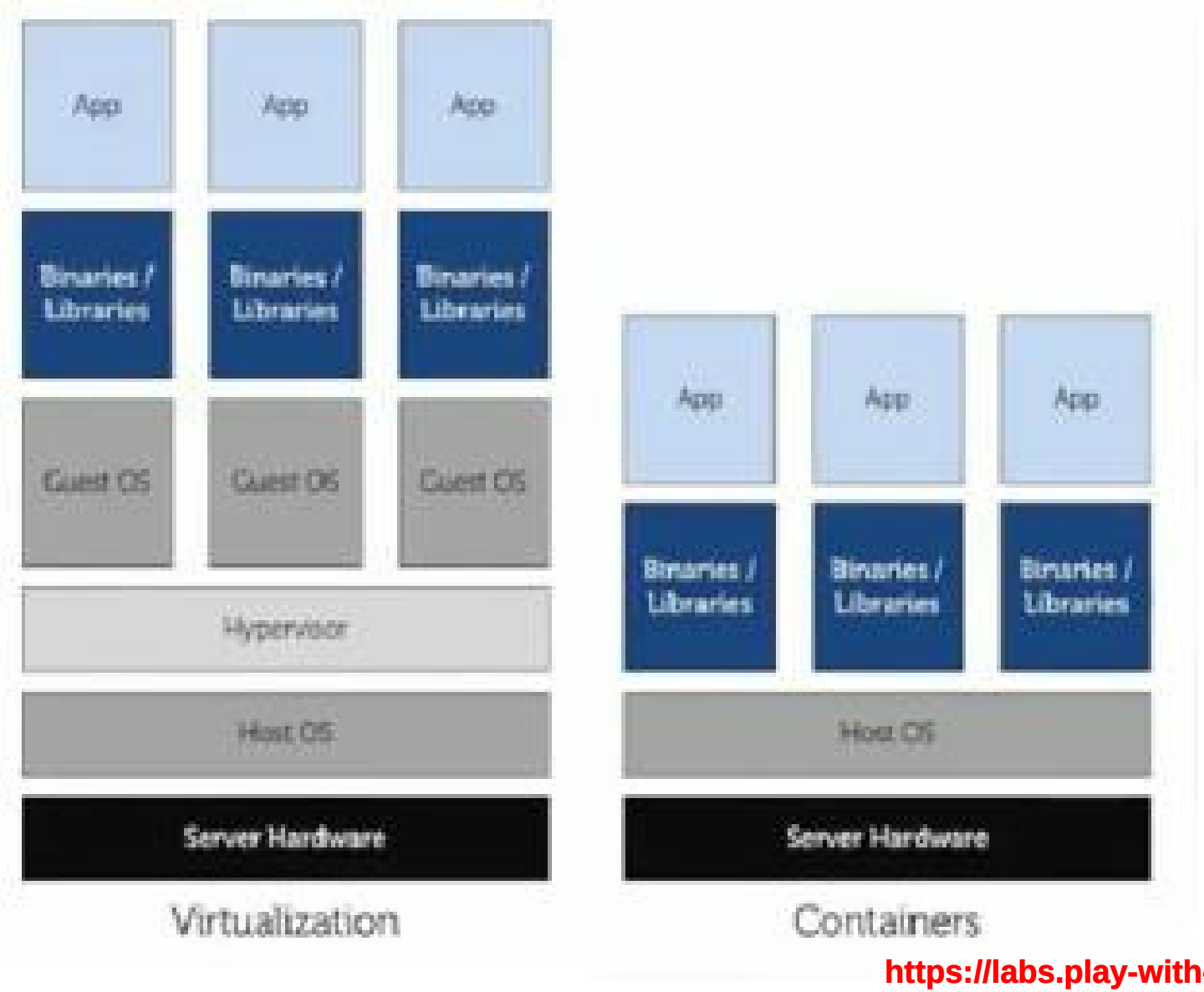
- Un **conteneur** s'exécute nativement sur Linux et partage le noyau de la machine hôte avec d'autres conteneurs.
  - Il exécute un processus discret, ne prenant pas plus de mémoire que tout autre exécutable, ce qui le rend léger.
- Une **machine virtuelle** (VM) exécute un système d'exploitation « invité » à part entière avec un accès virtuel aux ressources hôte via un hyperviseur.
- En général, les machines virtuelles entraînent des surcoûts en termes de performances au-delà de ce qui est consommé par la logique de votre application.

Conteneurs et VMs ensemble

- Les conteneurs et VMs offrent ensemble une énorme flexibilité permettant aux IT de déployer et de gérer de manière optimale les applications.



Conteneurisation

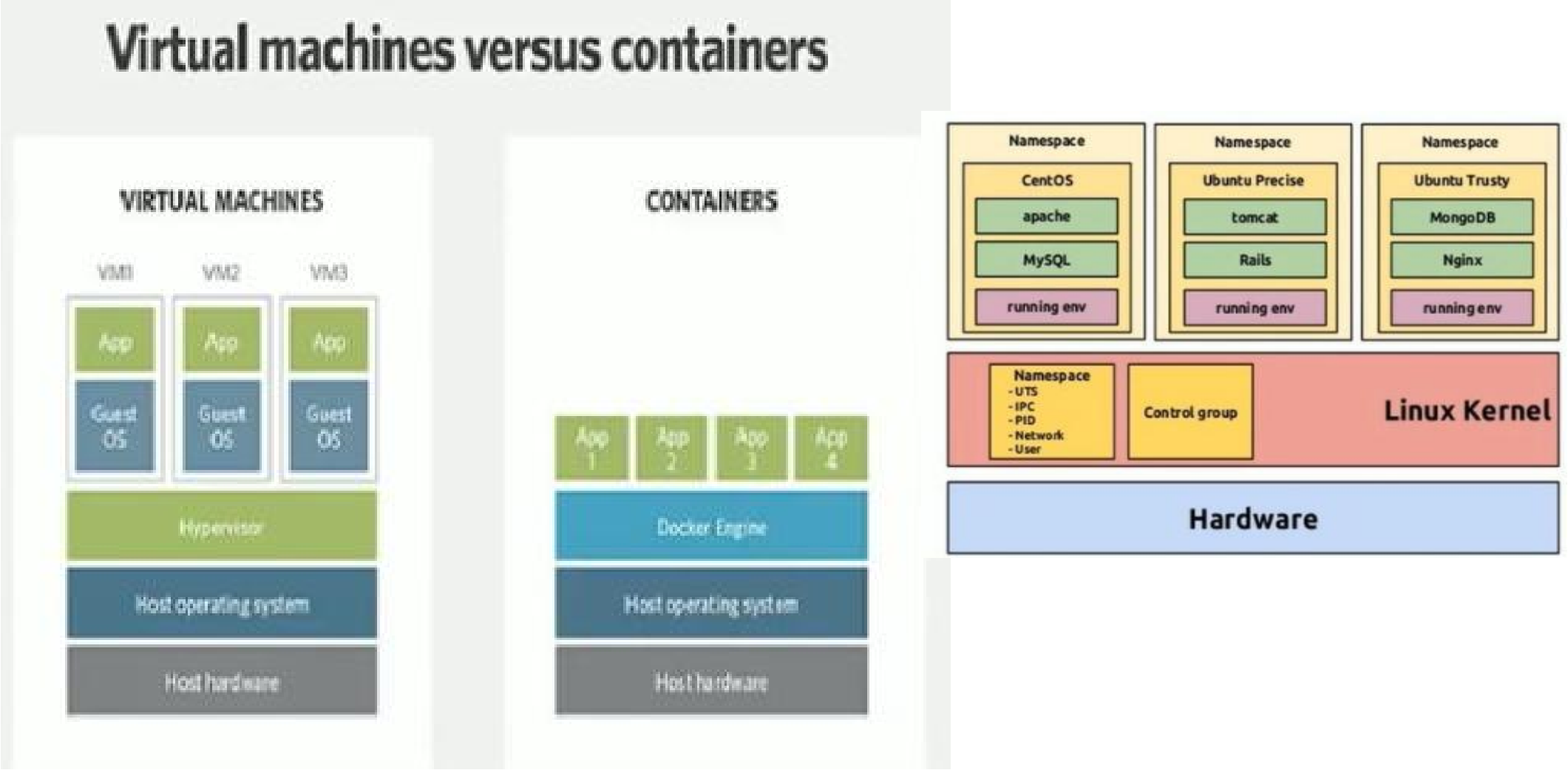


https://labs.play-with-docker.com/

Conteneurisation

Machines virtuelles (VM)	Conteneurs
Représente la virtualisation au niveau matériel	Représente la virtualisation du système d'exploitation
Poids lourd	Poids léger
Performances limitées	Performances natives
Démarrage plus lent	Démarrage en quelques secondes
Entièrement isolé et donc plus sécurisé	Isolation au niveau du processus et donc moins sécurisée

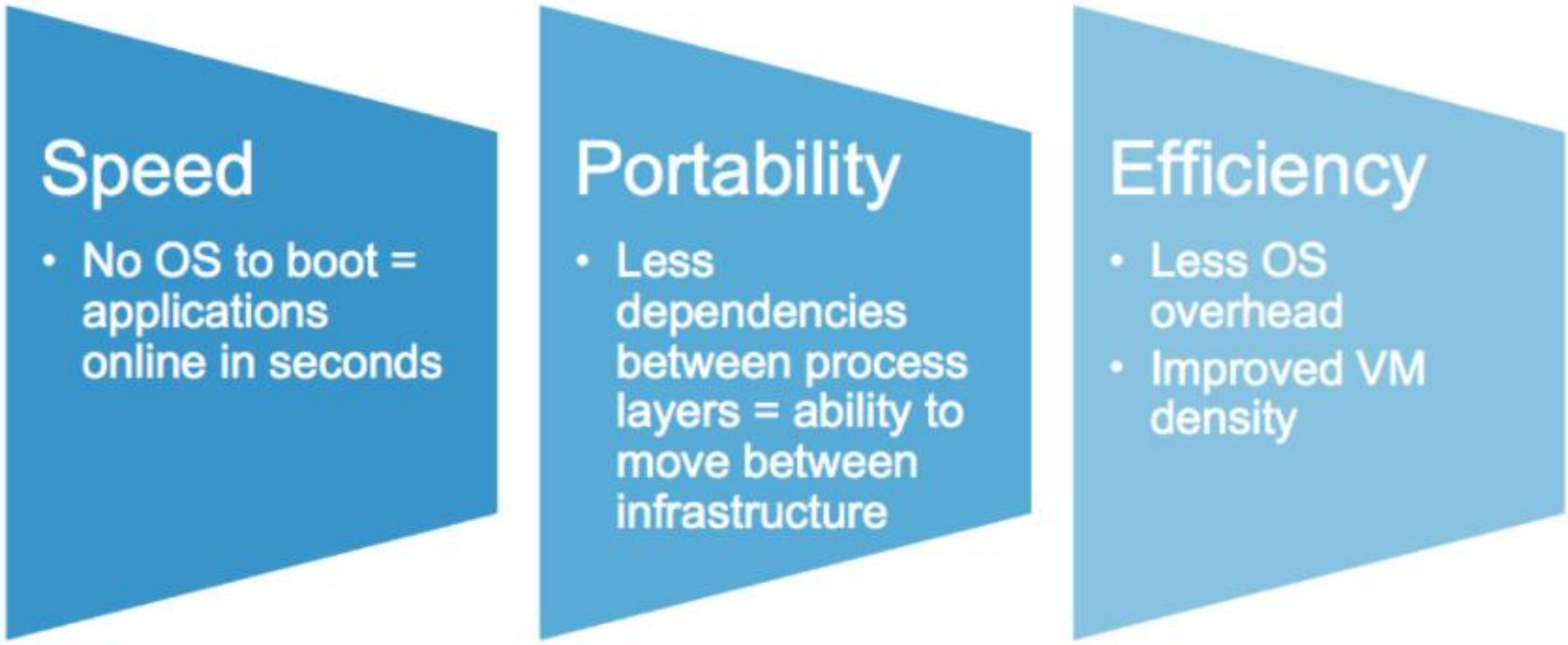
Conteneurisation



Concepts Docker

- **Flexible:** même les applications les plus complexes peuvent être conteneurisées.
- **Léger:** les conteneurs exploitent et partagent le noyau hôte, ce qui les renc beaucoup plus efficaces en termes de ressources système que les machines virtuelles.
- **Portable:** Vous pouvez créer localement, déployer sur le Cloud et exécuter n'importe où.
- **Faiblement couplé:** les conteneurs sont hautement autonomes et encapsulés, ce qui vous permet de remplacer ou de mettre à niveau l'un sans en perturber les autres.
- **Évolutif:** vous pouvez augmenter et distribuer automatiquement les répliques de conteneurs dans un centre de données.

Principaux avantages de Docker Container



Docker Basics

Image

The basis of a Docker container. The content at rest.

Container

The image when it is 'running.' The standard unit for app service

Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

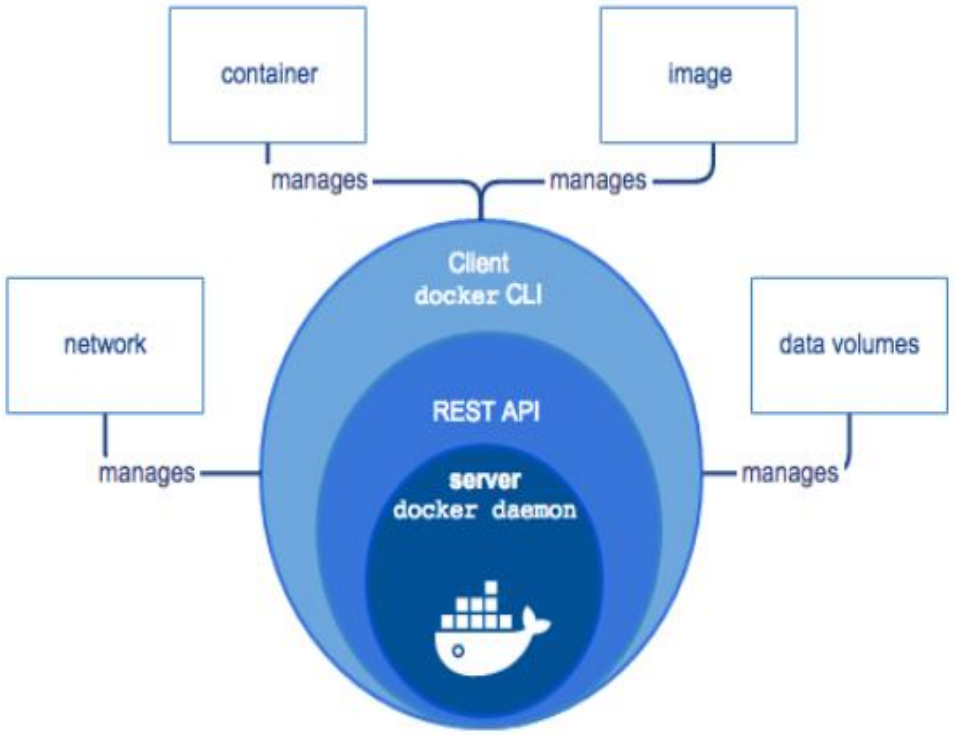
Registry

Stores, distributes and manages Docker images

Control Plane

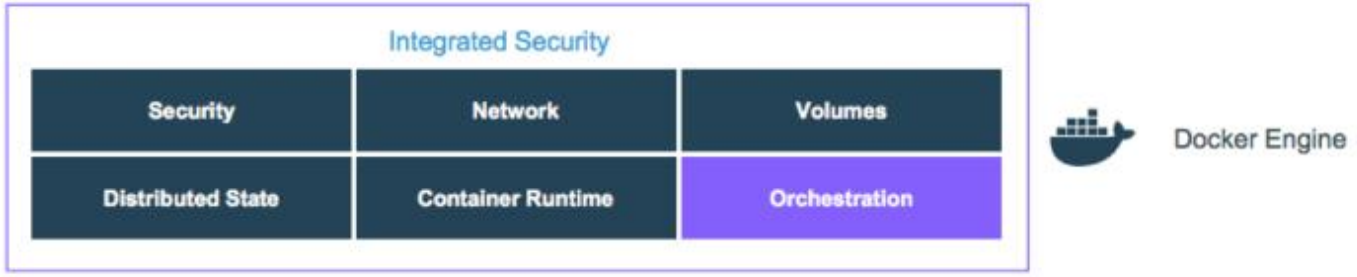
Management plane for container and cluster orchestration

Docker Engine

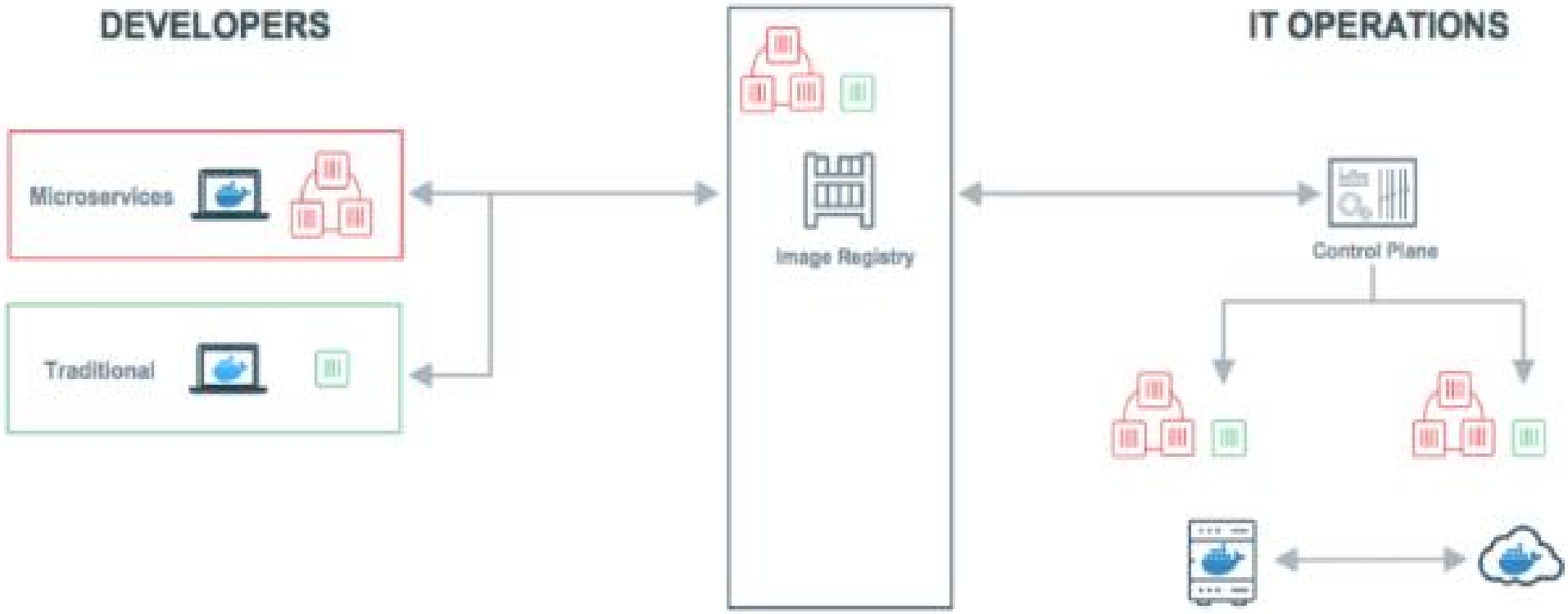


Docker Engine est une application client-serveur avec ces composants majeurs:

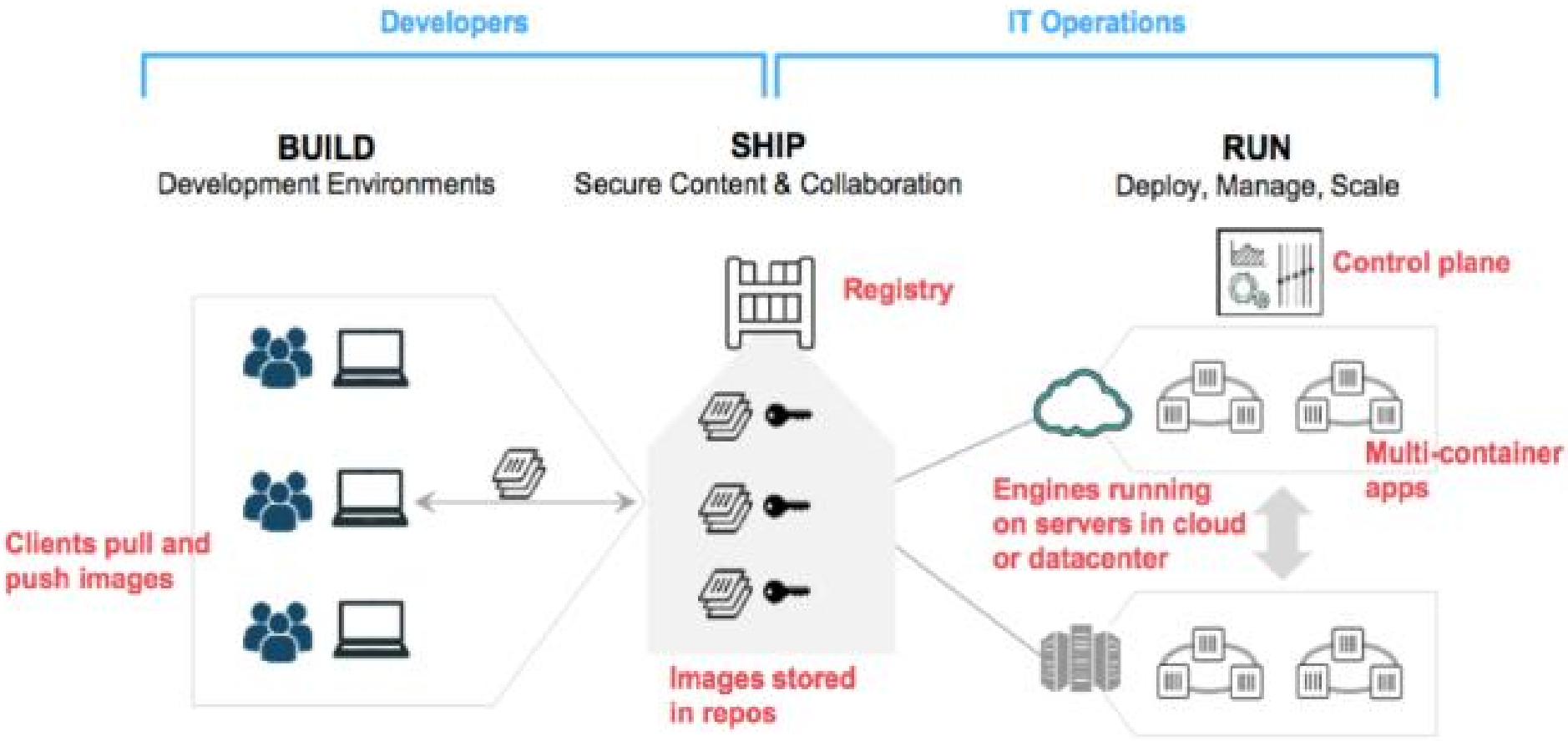
- Un serveur de type longue durée d'exécution appelé processus démon (dockerd).
- Une API REST qui spécifie les interfaces que les programmes peuvent utiliser pour parler au démon et lui indiquer quoi faire.
- Un client d'interface de ligne de commande (CLI) (docker).



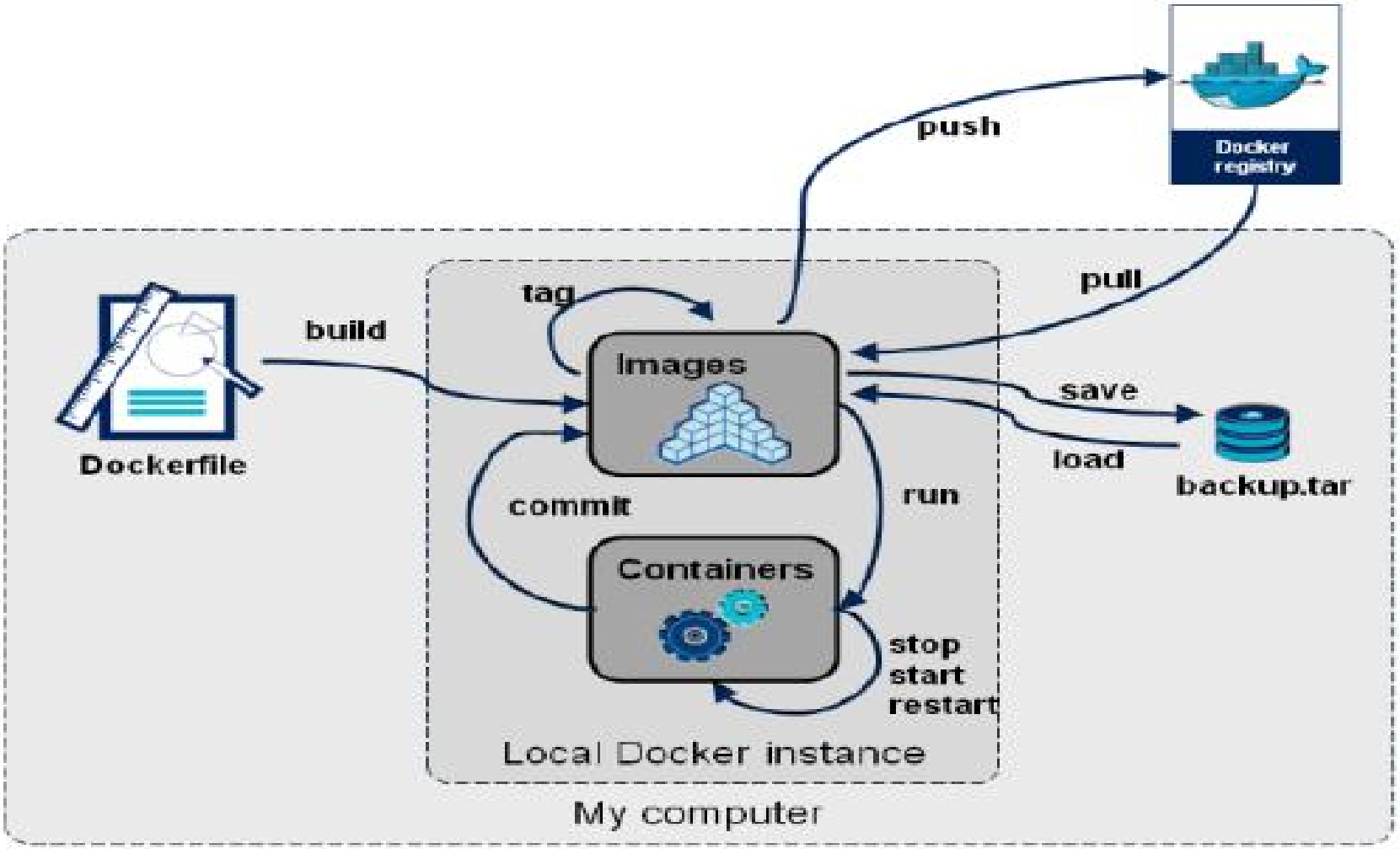
Building a Software Supply Chain



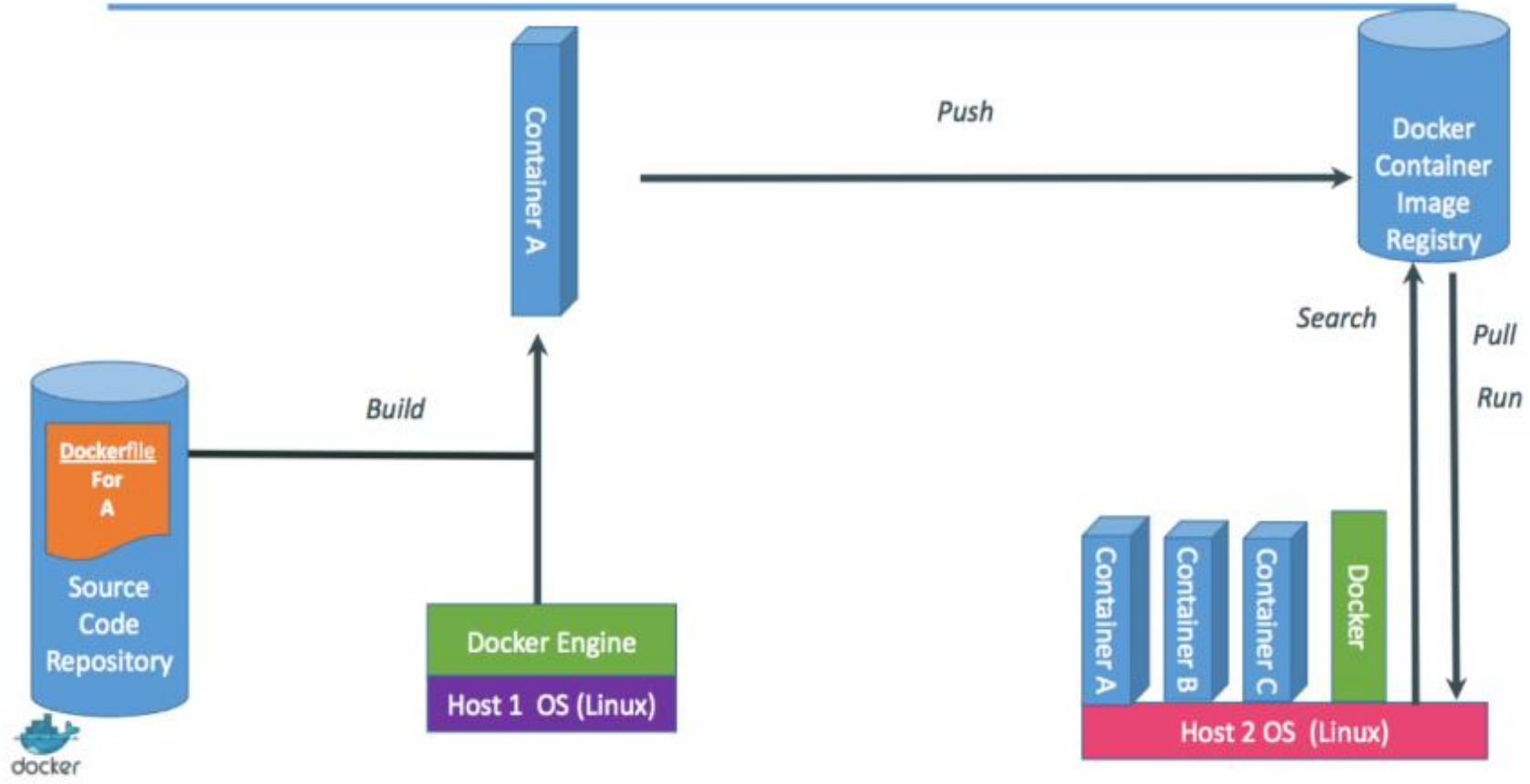
Containers as a Service (CaaS)



Cycle de vie Docker



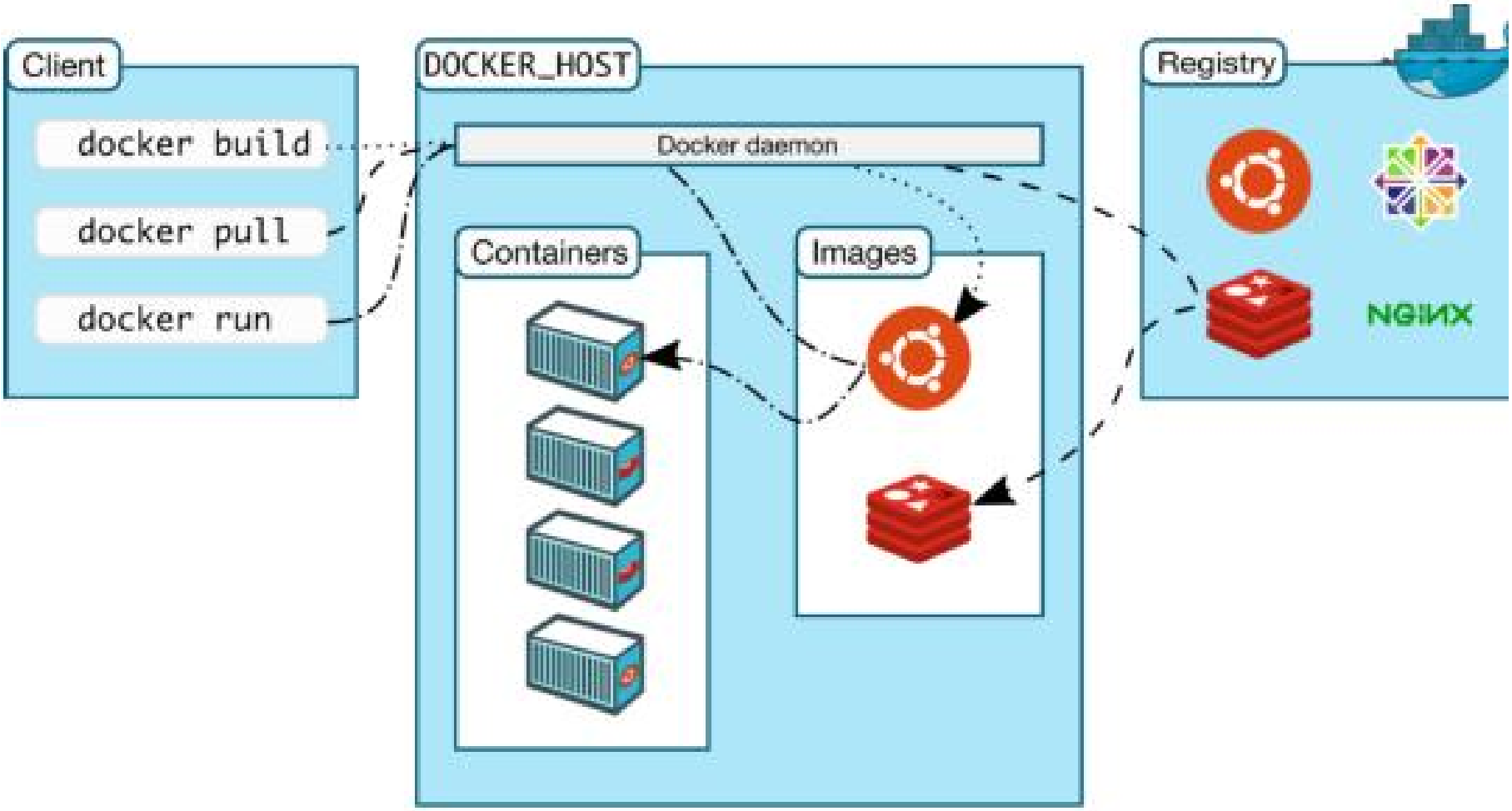
Cycle de vie Docker



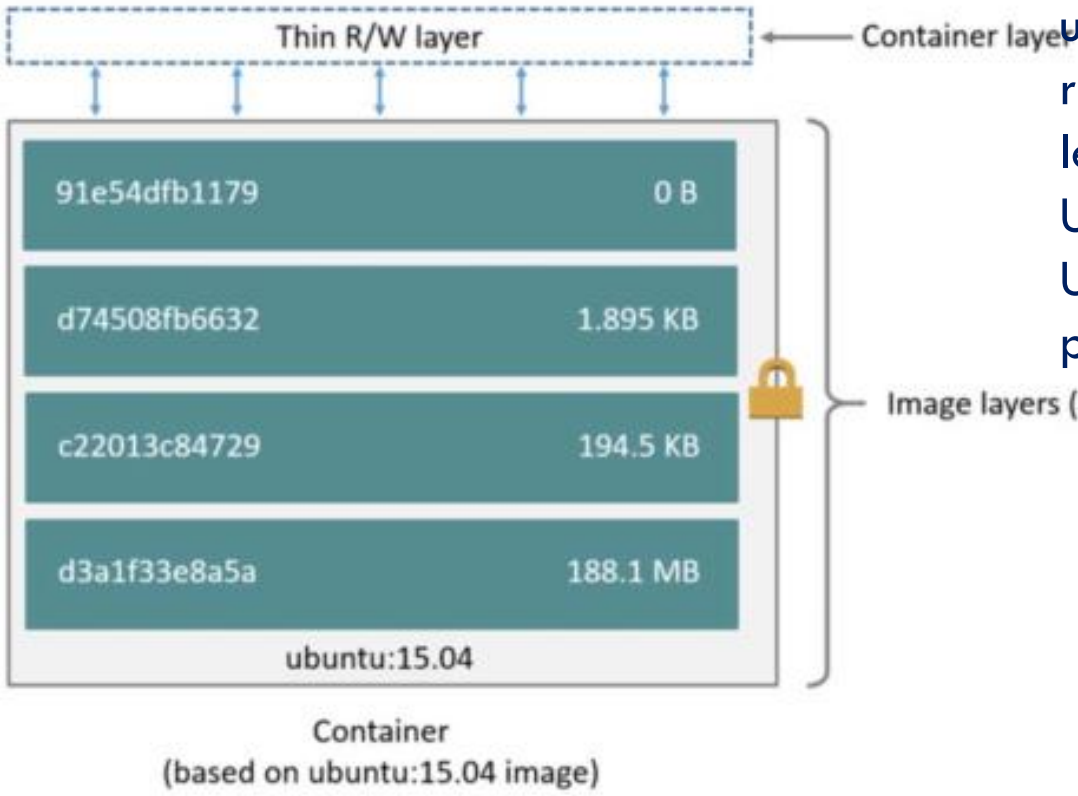
Images et conteneurs

- Un conteneur est un processus en cours d'exécution, avec quelques fonctionnalités d'encapsulation supplémentaires afin de le maintenir isolé de l'hôte et des autres conteneurs.
- Un aspect important de l'isolation est que chaque conteneur interagit avec son propre système de fichiers privé.
- Ce système de fichiers est fourni par une image Docker
- Une image comprend tout ce qui nécessaire pour exécuter une application
  - le code ou le binaire, les environnements d'exécution, les dépendances et tout autre objet du système de fichiers requis.

Architecture Docker

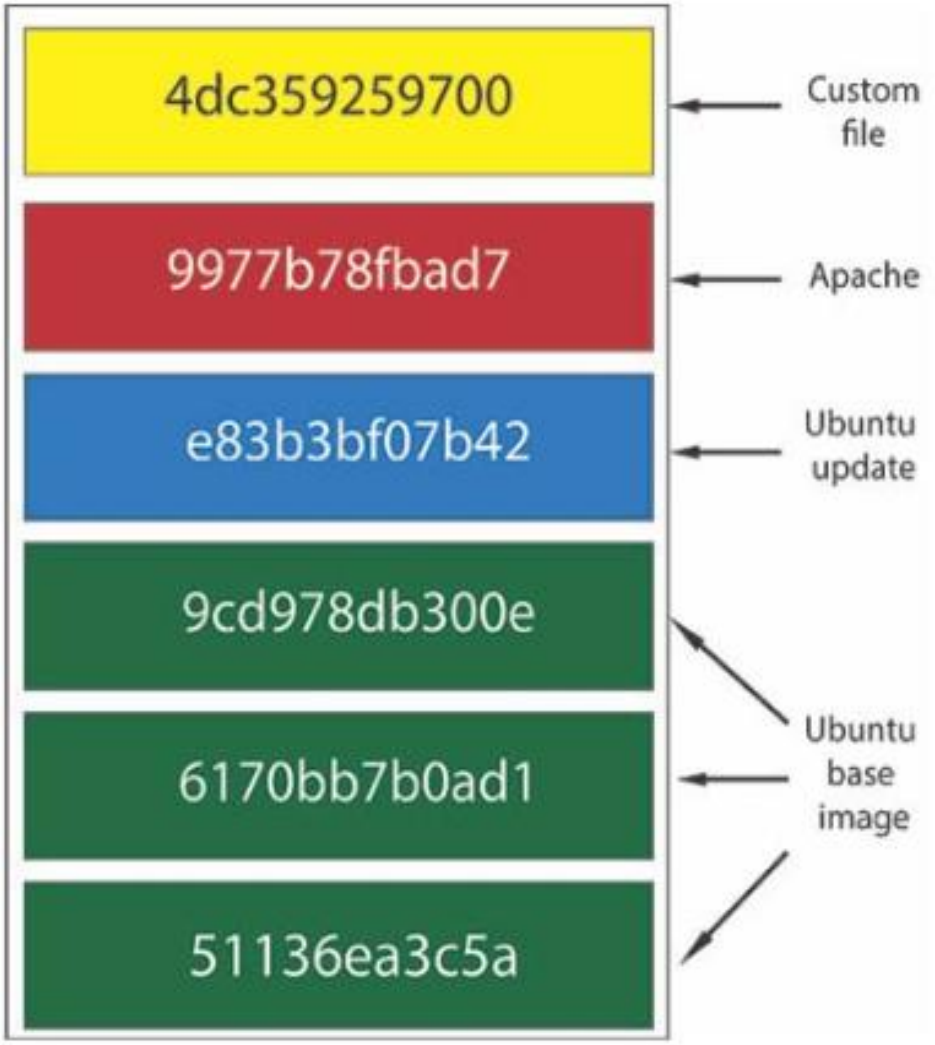


Les images Docker



Une image se décompose en layers
Une image Docker fait référence à une liste de couches en lecture seule qui représentent les différences dans le système de fichiers.
Un conteneur = une image + un layer r/w
Une image, peut servir de base pour plusieurs conteneurs.

Layers



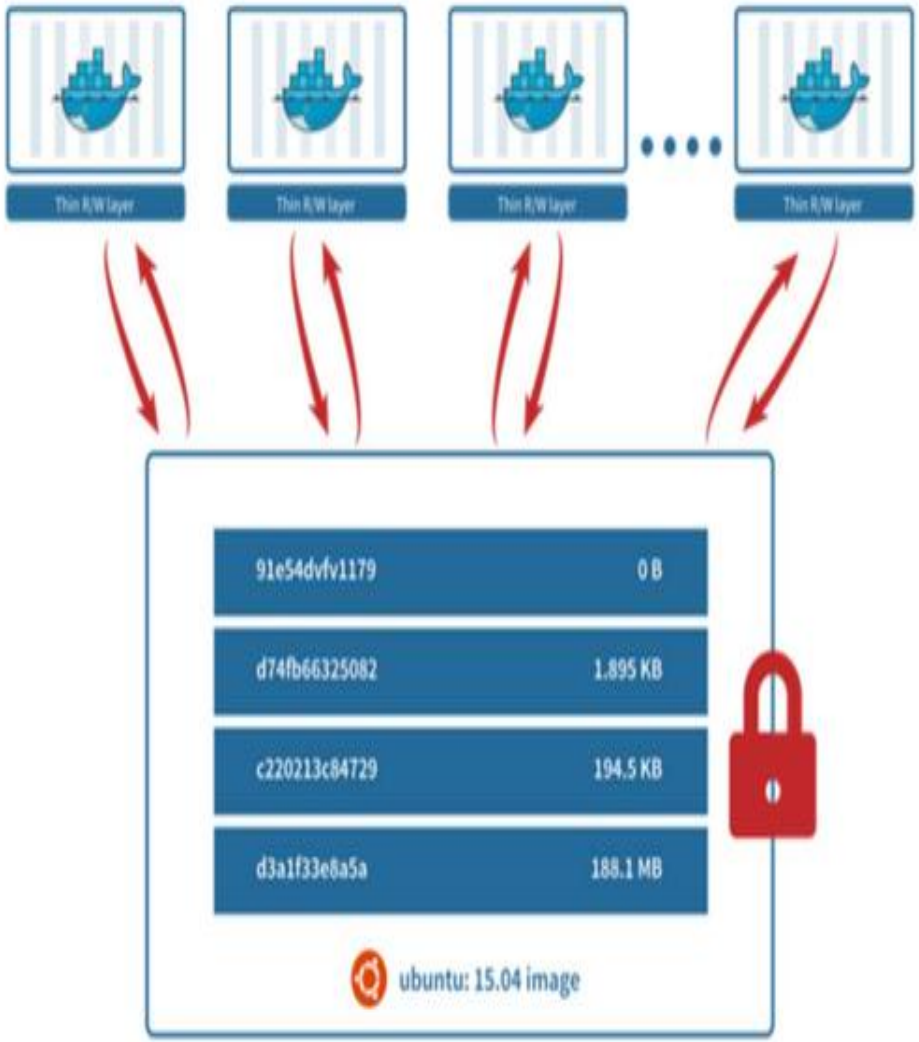
La création de conteneurs d'applications à utiliser implique la création d'une image de base Docker sur laquelle certains ou tous les conteneurs d'applications sont basés.

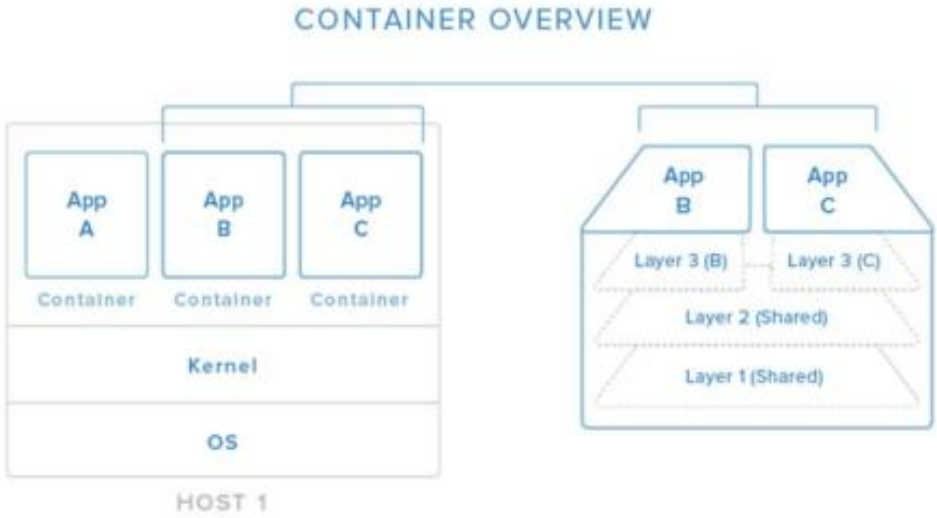
Les layers peuvent être réutilisés entre différents conteneurs

Gestion optimisée de l'espace disque

L'utilisation d'une image de base permet de réutiliser les configurations d'image car de nombreuses applications partageront des dépendances, des bibliothèque et une configuration.

Plusieurs conteneurs





Composition de conteneurs

En général, le workflow de développement ressemble à ceci :

Créez et testez des conteneurs individuels pour chaque composant de votre application en créant d'abord des images Docker.

Assemblez vos conteneurs et votre infrastructure de support dans une application complète.

Testez, partagez et déployez votre application conteneurisée complète.

Définitions

Terminologie	Définition
Docker Host	Une machine physique ou virtuelle qui exécute un démon Docker et contient des images mises en cache ainsi que des conteneurs exécutables créés à partir d'images.
Docker Images	Un modèle (template) pour créer des conteneurs Docker.
Docker Registry	Un référentiel d'images Docker pouvant être utilisé pour créer des conteneurs Docker. Docker Hub https://hub.docker.com/ est l'exemple le plus populaire de référentiel (repository) Docker.
Docker Machine	Un utilitaire de gestion de plusieurs hôtes Docker, qui peut s'exécuter localement dans VirtualBox ou à distance dans un service d'hébergement cloud tel qu'Amazon Web Services, Microsoft Azure ou Digital Ocean

Démo

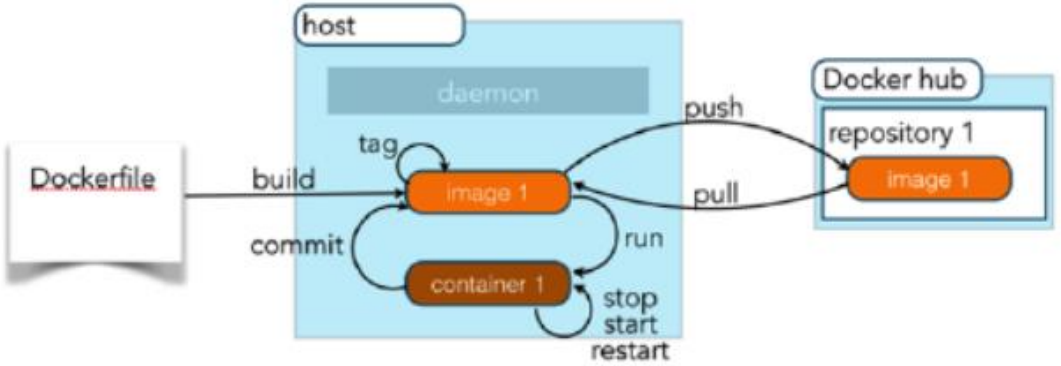
Démarrage d'un conteneur

- Avec Docker installé, vous pouvez commencer à exécuter des conteneurs. Si vous n'avez pas, déjà, le conteneur que vous souhaitez exécuter, Docker téléchargera l'image nécessaire pour créer le conteneur à partir du Docker Hub, puis le créera et l'exécutera.
- Pour exécuter le simple conteneur **hello-world** afin de vous assurer que tout est correctement configuré, exécutez les commandes suivantes :

```
docker run hello-world
```

Workflow local

- Docker a un flux de travail typique qui vous permet de **créer** des images, d'**extraire** des images, de **publier** des images et d'**exécuter** des conteneurs.



- Le flux de travail Docker typique consiste à créer une image à partir d'un **Dockerfile**, qui contient des instructions sur la façon de configurer un conteneur ou d'extraire une image d'un registre Docker tel que Docker Hub.

Pull Image From Docker Registry

- Le moyen le plus simple d'obtenir une image est de visiter https://hub. docker.com et recherchez une image déjà préparée à partir de laquelle créer un conteneur.

```
docker pull mysql
```

- Si vous souhaitez une version spécifique, vous pouvez également ajouter une balise pour identifier la version souhaitée.

```
docker pull mysql:5.5.45
```

Building Image From A Dockerfile

- Vous pouvez toujours créer vos propres images en créant un **Dockerfile**.
- Les Dockerfiles contiennent des instructions pour hériter d'une image existante, où vous pouvez ensuite ajouter un logiciel ou personnaliser des configurations.
- Voici des exemples simples de ce que vous pourriez trouver dans un fichier nommé Dockerfile :

```
FROM mysql:5.5.45
RUN echo America/New_York | tee /etc/timezone &&
↳dpkg-reconfigure --frontend noninteractive tzdata
```

Dockerfile 147 Bytes

```
1  FROM openjdk:8-jdk-alpine
2  EXPOSE 8083
3  ADD target/docker-spring-boot.war docker-spring-boot.war
4  ENTRYPOINT ["java","-jar","/docker-spring-boot.war"]
```

Building Image From A Dockerfile

- Pour créer (build) cette image à partir du répertoire contenant le Dockerfile, exécutez la commande suivante :

```
docker build .
```

- Cette commande créera une image sans nom. Vous pouvez le voir en exécutant la commande pour lister les images :

```
docker images
```

Building Image From A Dockerfile

- Cela affiche toutes les images mises en cache localement, y compris celles créées à l'aide de la commande **build**.

```
REPOSITORY    TAG         IMAGE ID        VIRTUAL SIZE
<none>        <none>      4b9b8b27fb42    214.4 MB
mysql         5.5.45      0da0b10c6fd8    213.5 MB
```

- Comme vous pouvez le voir, la commande build a créé une image avec un nom de référentiel et un nom de balise <none>. Cela a tendance à ne pas être très utile, vous pouvez donc utiliser une option -t pour nommer l'image pour une utilisation plus facile :

```
docker build -t est-mysql .
```

Building Image From A Dockerfile

- En affichant les images à nouveau, vous pouvez voir que l'image est beaucoup plus claire.

```
REPOSITORY    TAG       IMAGE ID        VIRTUAL SIZE
est-mysql     latest    4b9b8b27fb42    214.4 MB
mysql         5.5.45    0da0b10c6fd8    213.5 MB
```

Démarrage d'une image

- Pour exécuter une image Docker, il vous suffit d'utiliser la commande **run** suivie d'un nom d'image local ou trouvé dans Docker Hub.
- Généralement, une image Docker nécessitera des variables d'environnement supplémentaires, qui peuvent être spécifiées avec l'option -e. Pour les processus de longue durée comme les démons, vous devez également utiliser une option -d.

```
docker run -e MYSQL_ROOT_PASSWORD=root+1 -d est-
↳mysql
```

Démarrage d'une image

- Pour démarrer l'image **est-mysql**, vous devez exécuter la commande suivante pour configurer le mot de passe de l'utilisateur root MySQL, comme indiqué dans la documentation du référentiel mysql de Docker Hub :

```
docker run -e MYSQL_ROOT_PASSWORD=root+1 -d est-
↳mysql
```

Démarrage d'une image

- Pour voir le conteneur en cours d'exécution, vous pouvez utiliser la commande Docker **ps** :

```
docker ps
```

- La commande **ps** affiche tous les processus en cours d'exécution, le nom de l'image à partir de laquelle ils ont été créés, la commande exécutée, tous les ports sur lesquels le logiciel écoute et le nom du conteneur.

```
CONTAINER ID   IMAGE       COMMAND
30645f307114   est-mysql   “/entrypoint.sh mysql”
PORTS          NAMES
3306/tcp       serene_brahmagupta
```

Démarrage d'une image

- Comme vous pouvez le voir dans les processus en cours ci-dessus, le nom du conteneur est serene_brahmagupta. Il s'agit d'un nom généré automatiquement et il peut être difficile à maintenir.
- Il est donc recommandé de nommer explicitement le conteneur à l'aide de l'option **-name** pour fournir votre nom au démarrage du conteneur :

```
docker run --name my-est-mysql -e MYSQL_ROOT_
↳PASSWORD=root+1 -d est-mysql
```

Démarrage d'une image

- Vous remarquerez à partir de la sortie **ps** que le conteneur écoute le port 3306, mais cela ne signifie pas que vous pouvez utiliser la ligne de commande MySQL ou MySQL Workbench localement pour interagir avec la base de données, car ce port n'est accessible que dans le Docker sécurisé environnement dans lequel il a été lancé.
- Pour le rendre disponible en dehors de cet environnement, vous devez mapper les ports à l'aide de l'option -p.

```
docker run --name my-est-mysql -e MYSQL_ROOT_
↳PASSWORD=root+1 -p 3306:3306 -d est-mysql
```

Démarrage d'une image

- Maintenant, mysql écoute sur un port auquel vous pouvez vous connecter. Mais vous devez toujours savoir quelle est l'adresse IP pour vous connecter. Pour déterminer l'adresse IP, vous pouvez utiliser la commande **dockermachine ip** pour la découvrir.

```
docker-machine ip default
```

- En utilisant **default** comme nom de machine, qui est la machine par défaut installée avec Docker Toolbox, vous recevrez l'adresse IP de la machine hébergeant votre conteneur Docker. Avec l'adresse IP, vous pouvez maintenant vous connecter à MySQL en utilisant votre ligne de commande MySQL locale.

```
mysql -h 192.168.99.100 -u root -proot+1
```

Démarrer et arrêter les conteneurs

- Maintenant que vous avez un conteneur Docker en cours d'exécution, vous pouvez l'arrêter en utilisant la commande d'arrêt Docker et nom du conteneur :

```
docker stop my-est-mysql
```

- L'état complet du conteneur est écrit sur le disque, donc si vous souhaitez le ré-exécuter dans l'état dans lequel il se trouvait lorsque vous l'avez arrêté, vous pouvez utiliser la commande start :

```
docker start my-est-mysql
```

Tagging

- Maintenant que vous avez une image que vous avez exécuté et validé, c'est une bonne idée de la marquer avec un nom d'utilisateur, un nom d'image et un numéro de version avant de la transférer dans le référentiel (docker-hub).
- Vous pouvez y parvenir en utilisant la commande tag de Docker :

```
docker tag est-mysql javajudd/est-mysql:1.0
```

Push Image

- Enfin, vous êtes prêt à envoyer votre image vers Docker Hub pour que votre équipe l'utilise.
- Vous devez vous connecter à l'aide de la commande login. Lorsque vous y êtes invité, saisissez le nom d'utilisateur, le mot de passe et l'adresse e-mail avec lesquels vous vous êtes inscrit.

```
docker login
```

- Maintenant, poussez votre image à l'aide de la commande push, en spécifiant votre nom d'utilisateur, le nom de l'image et le numéro de version.

```
docker push javajudd/est-mysql:1.0
```

Affichage des conteneurs

- Vous avez déjà vu comment la commande **ps** peut lister les conteneurs en cours d'exécution, mais qu'en est-il de tous les conteneurs, quel que soit leur état ?

- En ajoutant l'option -a, vous pouvez tous les voir. Avec une liste de tous les conteneurs, vous pouvez décider lequel démarrer ou supprimer.

```
docker ps -a
```

Supprimer les conteneurs

- Lorsque vous avez fini d'utiliser un conteneur, plutôt que de l'avoir autour, vous voudrez le supprimer pour récupérer de l'espace disque.

- Pour supprimer un conteneur, vous pouvez utiliser la commande **rm**

```
docker rm my-est-mysql
```

Supprimer les images

- Les images peuvent prendre des quantités importantes d'espace, allant d'un mégaoctet à plusieurs centaines de mégaoctets, vous devrez donc purger les images indésirables à l'aide de la commande **rmi**

```
docker rmi est-mysql
```

- Pendant le cycle de débogage de la création d'une nouvelle image, vous pouvez générer une grande quantité d'images indésirables et sans nom, qui sont désignées par le nom <none>. Vous pouvez facilement supprimer toutes les images pendantes à l'aide de la commande suivante :

```
docker rmi $(docker images -q -f dangling=true)
```

Ports

- Il est souvent utile de savoir quels ports sont exposés par un conteneur, comme le port 3306 pour accéder à une base de données MySQL ou le port 80 pour accéder à un serveur Web. La commande port peut être utilisée pour afficher les ports exposés.

```
docker port my-est-mysql
```

Processus

- Si vous avez besoin de voir les processus en cours d'exécution dans un conteneur, vous pouvez utiliser la commande top :

```
docker top my-est-mysql
```

Exécuter des commandes

- Vous pouvez exécuter des commandes dans un conteneur en cours d'exécution à l'aide de la commande **exec**. Pour lister le contenu de la racine du disque dur, vous pouvez par exemple procéder comme suit :

```
docker exec my-est-mysql ls /
```

DOCKERFILE

- Comme vous l'avez déjà vu, le **Dockerfile** est le principal moyen de créer une image Docker. Il contient des instructions telles que des commandes Linux pour l'installation et la configuration du logiciel.

- La commande **build** peut faire référence à un Dockerfile sur votre PATH ou à une URL, telle qu'un référentiel GitHub. Avec le Dockerfile, tous les fichiers du même répertoire ou de ses sous-répertoires seront également inclus dans le cadre du processus de génération.

Instructions DOCKERFILE

- Les instructions sont exécutées dans l'ordre dans lequel elles se trouvent dans le Dockerfile. Le fichier Docker peut également contenir des commentaires de ligne commençant par le caractère #.
- Ce tableau contient la liste des commandes disponibles.

INSTRUCTION	DESCRIPTION
FROM	This must be the first instruction in the Dockerfile and identifies the image to inherit from
MAINTAINER	Provides visibility and credit to the author of the image

Dockerfile 147 Bytes

```
1  FROM openjdk:8-jdk-alpine
2  EXPOSE 8083
3  ADD target/docker-spring-boot.war docker-spring-boot.war
4  ENTRYPOINT ["java","-jar","/docker-spring-boot.war"]
```

Instructions DOCKERFILE

INSTRUCTION	DESCRIPTION
RUN	Executes a Linux command for configuring and installing
ENTRYPOINT	The final script or application used to bootstrap the container, making it an executable application
CMD	Provide default arguments to the ENTRYPOINT using a JSON array format
LABEL	Name/value metadata about the image
ENV	Sets environment variables
COPY	Copies files into the container
ADD	Alternative to copy
WORKDIR	Sets working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions
EXPOSE	Ports the container will listen on
VOLUME	Creates a mount point
USER	User to run RUN, CMD, and/or ENTRYPOINT instructions

Dockerfile

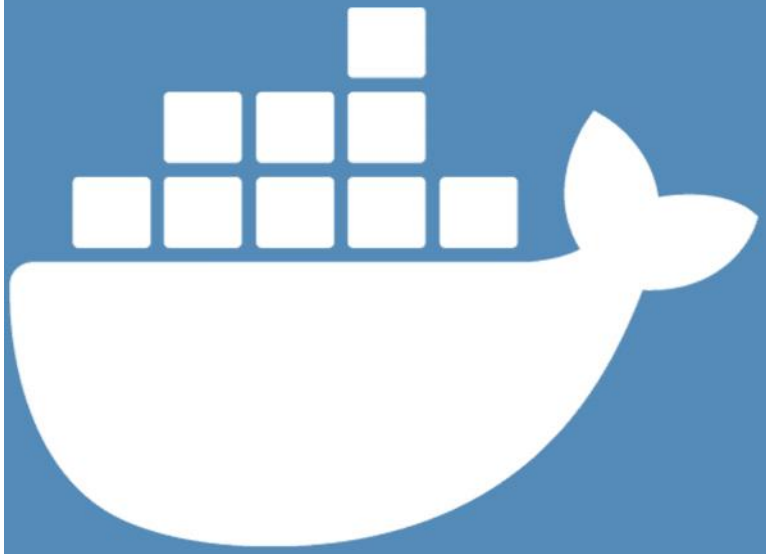
- FROM ubuntu:16.04RUN
- apt-get update && apt-get install -y openssh-server git apache2 python vim
- RUN mkdir /var/run/sshd
- RUN echo 'root:root' | chpasswd
- RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/'/etc/ssh/sshd_config
- git clone https://github.com/walidsaad/MyApp-v2 /var/www/html/MyApp
- ADD MyApp /var/www/html/MyApp
- WORKDIR /var/www/html/MyAppEXPOSE 22 80

IaC Infrastructure en tant que code

- Terraform windows
  - https://learn.hashicorp.com/tutorials/terraform/install-cli

Références

- Cours, Heithem Abbes, maître assistant à la FST
- Cours, Elyes JEBRI, Expert DevOps
- Slides, Formation DevOps, Wevioo



DOCKER - LIVRAISON CONTINUE

2021-2022

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)