



docker

# INTRODUCTION DOCKER



ANTHONY DI PERSIO

## DECOUVERTE DOCKER

Comprendre le concept de conteneurs, avantages et inconvénients

01

## INSTALLATION DE DOCKER

Installation des outils de Docker, découverte de l'écosystème

02

## LES IMAGES & CONTENEURS

Comprendre le concept d'image et de conteneur Docker

03

## LE DOCKERFILE

Créer une image et l'instancier en un conteneur Docker

04

## TABLE DES MATIÈRES

05

## DOCKER VOLUME

Comprendre la persistance des données dans Docker

06

## DOCKER-COMPOSE

Comprendre le fonctionnement d'une application multi conteneurs

07

## LE NETWORK DOCKER

Appréhender la communication réseau avec Docker

08

## LES ORCHESTRATEURS

Introduction à la notion d'orchestrateur de conteneurs

01

# DECOUVERTE DOCKER

Comprendre le concept de conteneurs, avantages et inconvénients



# DECOUVERTE DE DOCKER

- Qu'est-ce que Docker ?
  - Le logiciel « Docker » est une technologie de conteneurisation qui permet la création et l'utilisation de conteneurs Linux.
- Le terme « Docker » désigne plusieurs choses
  - le projet d'une communauté Open Source
  - les outils issus de ce projet Open Source
  - l'entreprise Docker Inc. qui constitue le principal soutien de ce projet
  - les outils que l'entreprise prend officiellement en charge

# DECOUVERTE DE DOCKER

- Donc...? Qu'est-ce que **Docker** ?
  - Le logiciel « Docker » est une technologie de conteneurisation qui permet la création et l'utilisation de conteneurs Linux®
  - La communauté Open Source Docker travaille à l'amélioration de cette technologie disponible gratuitement pour tout le monde.
  - L'entreprise **Docker Inc.** s'appuie sur le travail de la communauté Docker, sécurise sa technologie et partage ses avancées avec tous les utilisateurs. Elle prend ensuite en charge les technologies améliorées et sécurisées pour ses clients professionnels.

# DECOUVERTE DE DOCKER

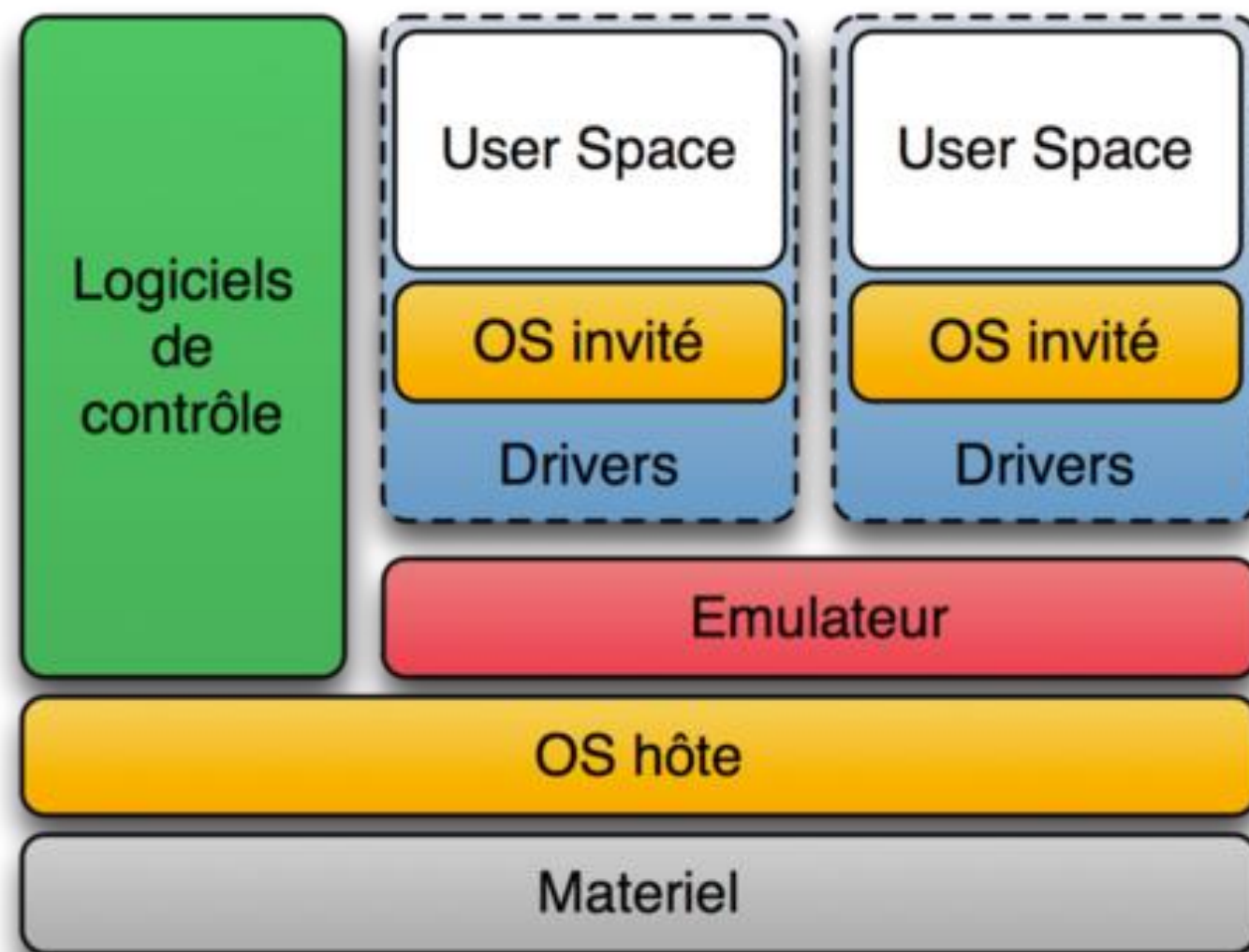
- À quoi sert **Docker** ?
  - Avec la technologie **Docker**, vous pouvez traiter les conteneurs comme des machines virtuelles très légères et modulaires
  - Ces conteneurs vous offrent une grande flexibilité :
    - ✓ Vous pouvez les créer, déployer des environnements très rapidement
    - ✓ Vous pouvez les copier et déplacer d'un environnement à un autre très facilement
    - ✓ Il vous permet d'optimiser vos applications pour le cloud

# DECOUVERTE DE DOCKER

- Virtualisation vs Conteneurisation?
  - La virtualisation est la capacité de faire tourner un, ou plusieurs serveur virtuel sur une seule machine physique grâce à un hyperviseur
  - L'hyperviseur permet d'émuler les différentes ressources matérielles d'un serveur physique et permet à des machines virtuelles de les partager
  - Une machine virtuelle possède ses propres ressources matérielles et son propre système d'exploitation

# DECOUVERTE DE DOCKER

- Détails d'une Machine Virtuelle (VM)





# DECOUVERTE DE DOCKER

- La **virtualisation** apporte des **avantages** :
  - Ressources adaptées aux besoins de l'application.
  - Faciliter de manipulation (Sauvegarde, bascule,...)
  - Réduction des dépenses et réduction d'équipements nécessaires
  - Facilités pour l'administration
- La virtualisation a des inconvénients également :
  - Réduction des performances
  - Multiplication des couches OS

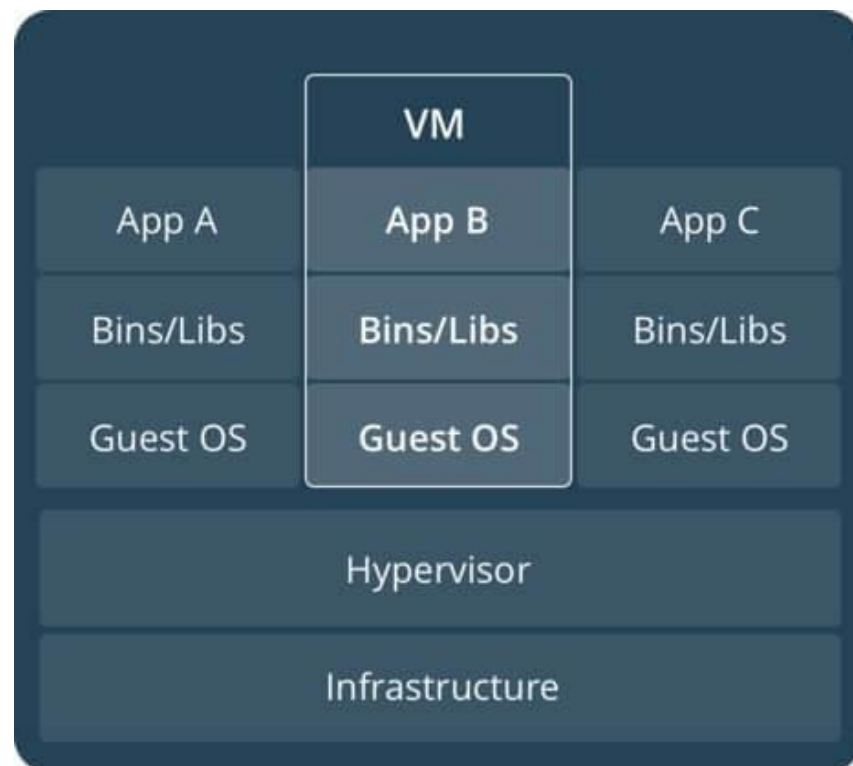
# DECOUVERTE DE DOCKER

- Virtualisation vs Conteneurisation?
  - Isolation en VM
    - ✓ Se fait au niveau matérielle
    - ✓ Accès virtuel aux ressources via l'hôte à l'aide de l'hyperviseur
  - Isolation Dans la conteneurisation
    - ✓ Se fait au niveau du système d'exploitation
    - ✓ Exécution native sur linux et partage du noyau hôte avec les conteneurs

# DECOUVERTE DE DOCKER

- Virtualisation vs Conteneurisation?

Machines Virtuelles



Conteneurs Dockers



- Avantages de Docker

- ✓ Flexibilité et légèreté grâce au partage du noyau de l'hôte
- ✓ Scalabilité ou Extensibilité ...

# DECOUVERTE DE DOCKER

- Comment fonctionne la technologie **Docker** ?
  - La technologie Docker utilise le noyau Linux et des fonctions de ce noyau pour :
    - ✓ Séparer les processus afin qu'ils puissent s'exécuter de façon indépendante
  - Elle utilise des fonctionnalités nativement disponibles sur Linux
    - ✓ La création de groupes de contrôle « **cgroups** »
    - ✓ La création des espaces de noms « **namespaces** »

# DECOUVERTE DE DOCKER

- Un **conteneurs** et un **processus linux isolé** à l'aide de...
  - **Namespaces linux** qui sont un mécanisme permettant de limiter l'accès d'un processus
  - **Les cgroups linux** qui sont des mécanismes permettant de limiter l'accès aux ressources d'un processus
- Quelques exemples de **namespaces**:
  - Namespace **PID**
  - Namespace **USER**
- Quelques exemples de **cgroups**:
  - cgroup **cpuset**
  - cgroup **memory**

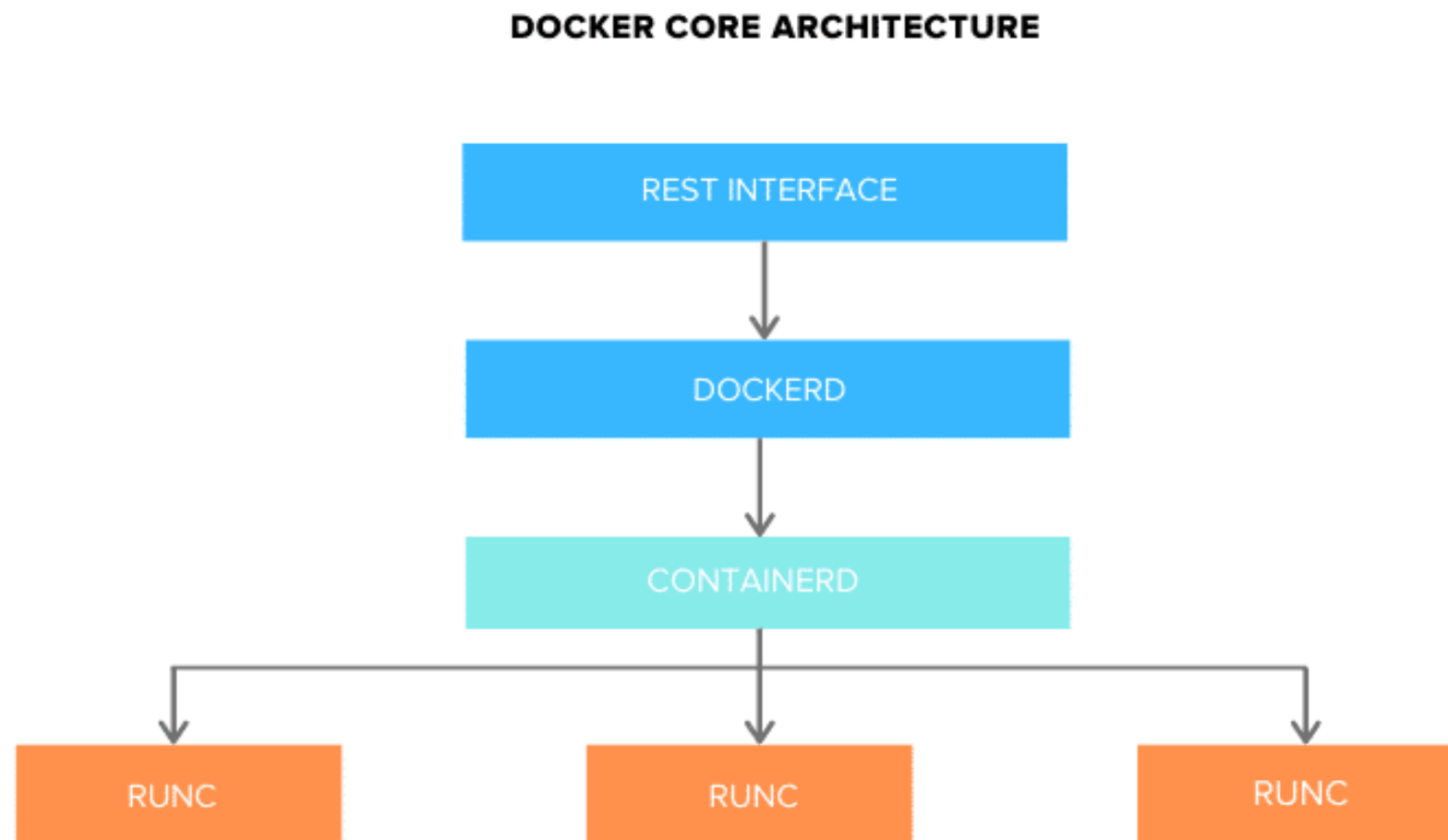


# DECOUVERTE DE DOCKER

- L'utilisation de conteneur n'est pas une technologie récente
  - Il y a de nombreuses applications qui utilisent ce concept
    - ✓ Chroot sur Unix (1982)
    - ✓ Jail sur BSD (2000)
    - ✓ Conteneurs sur Solaris (2004)
    - ✓ LXC (Linux conteneurs) sur Linux (2008)
- Les conteneurs ne sont pas nouveaux, mais leur utilisation pour déployer facilement des applications l'est.
  - La notoriété de docker vient du fait qu'il a su permettre aux utilisateurs de gérer facilement leurs conteneurs avec une interface en ligne de commande (CLI) très simple

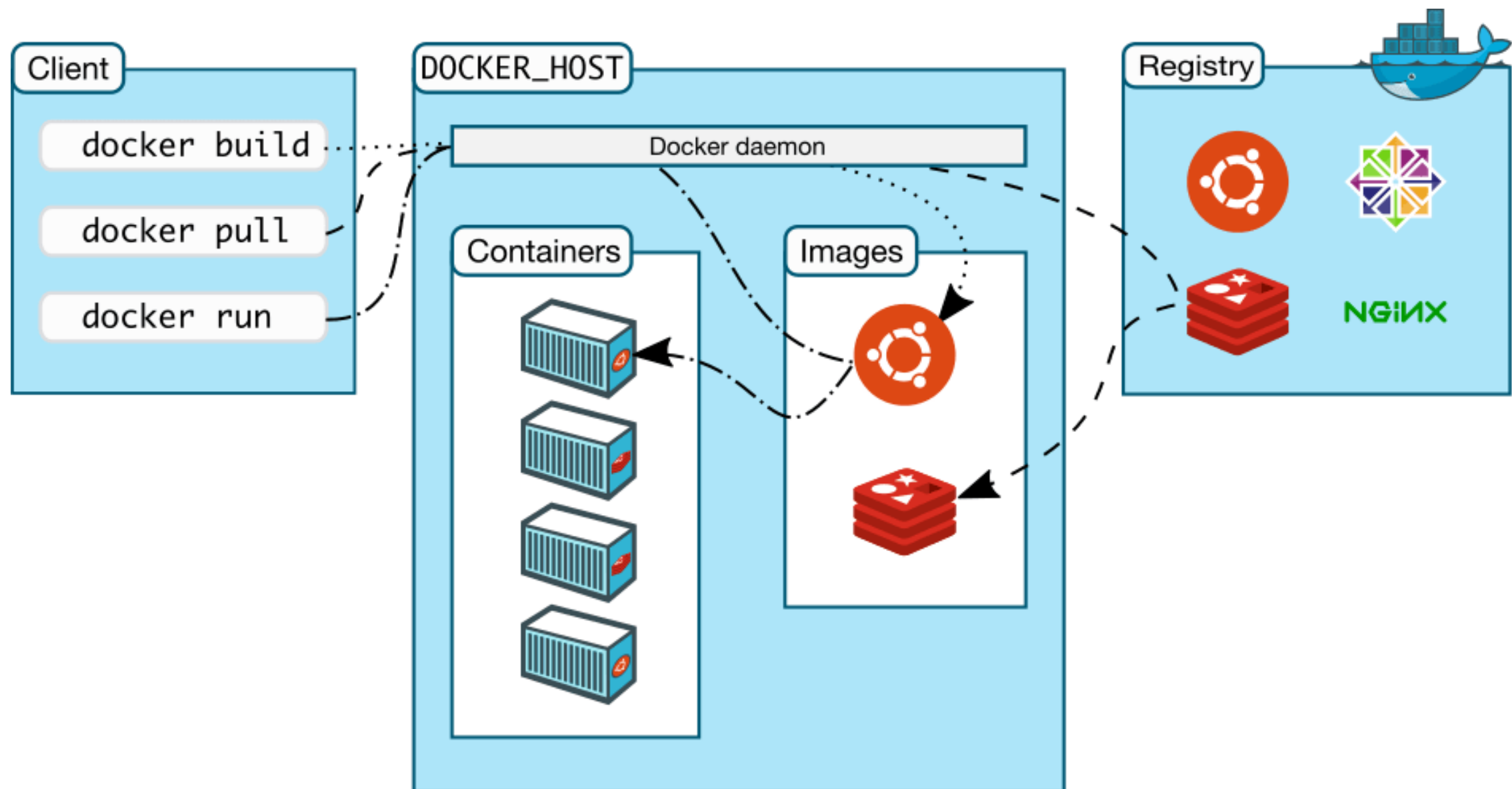
# DECOUVERTE DE DOCKER

- Docker est composé de :
  - Docker engine
  - Docker-containerd
  - Docker-runc



# DECOUVERTE DE DOCKER

- Le fonctionnement de Docker



02

# INSTALLATION DE DOCKER

Installation des outils de Docker, découverte de l'écosystème

# INSTALLATION DE DOCKER

- Docker peut être installer sur toute type de distribution
  - Windows
  - Linux
  - MacOS
- Docker est disponible en deux éditions
  - **Docker Community Edition (CE)**
    - ✓ Idéale pour les développeurs individuels et les petites équipes cherchant à se familiariser avec Docker
  - **Docker Enterprise Edition (EE)**
    - ✓ Conçue pour les équipes de développement d'entreprise et les équipes système.

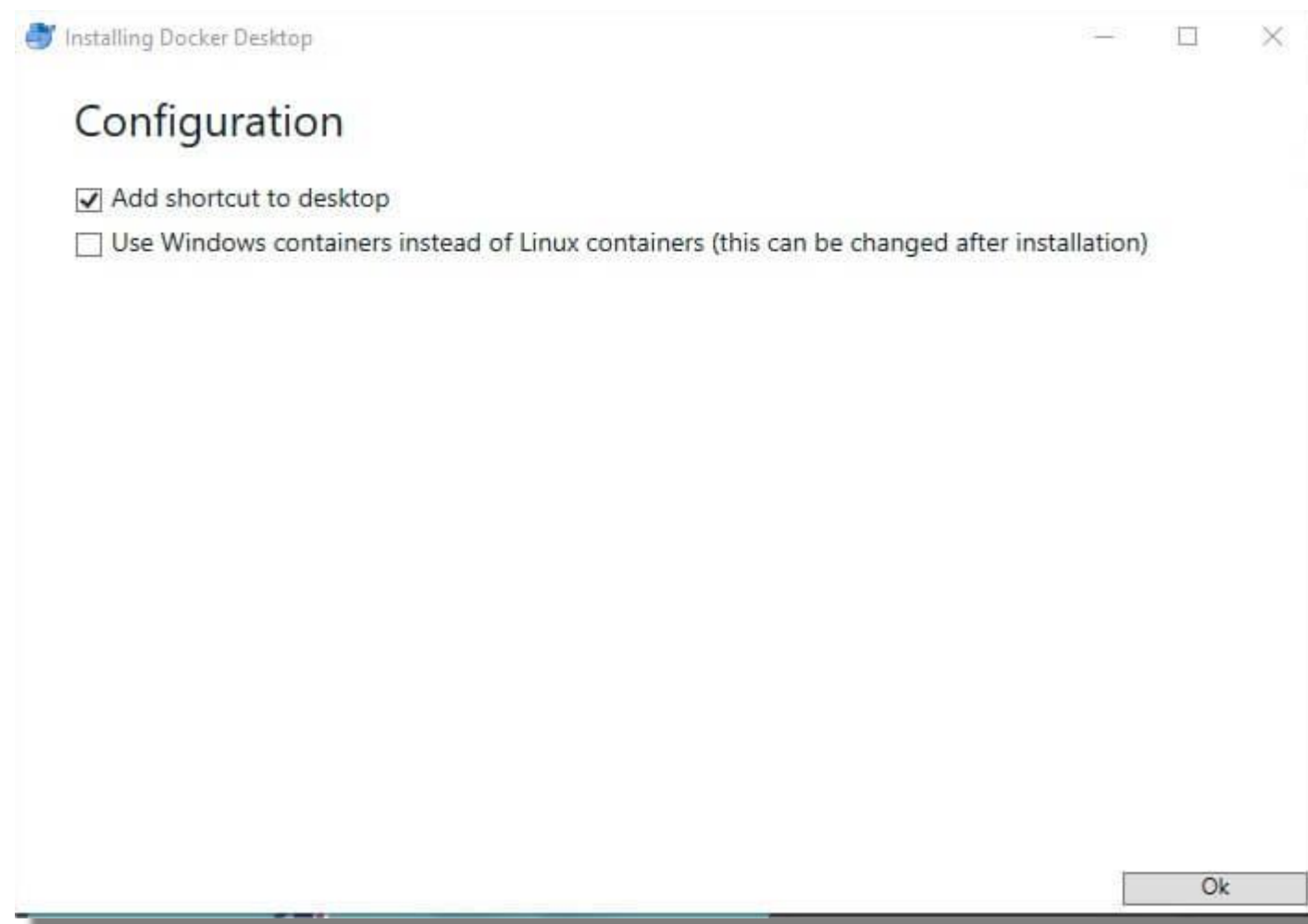


# INSTALLATION DE DOCKER

- Installation de **Docker** sur Windows
  - [hub.docker.com/](https://hub.docker.com/)
- Configuration minimale requise pour une installation Windows
  - **Windows 10 64 bits**: Pro, Entreprise ou Education (version 15063 ou ultérieure)
  - La **virtualisation** est **activée** dans le **BIOS** (normalement elle est activée par défaut, sinon activer **HyperV**)
  - Au moins **4Go** de **RAM**
- Pour utiliser **Docker** il vous faut un **compte Docker-Hub**
  - Une fois inscrit et connecté, Cliquer sur « **get Docker** » (CE)

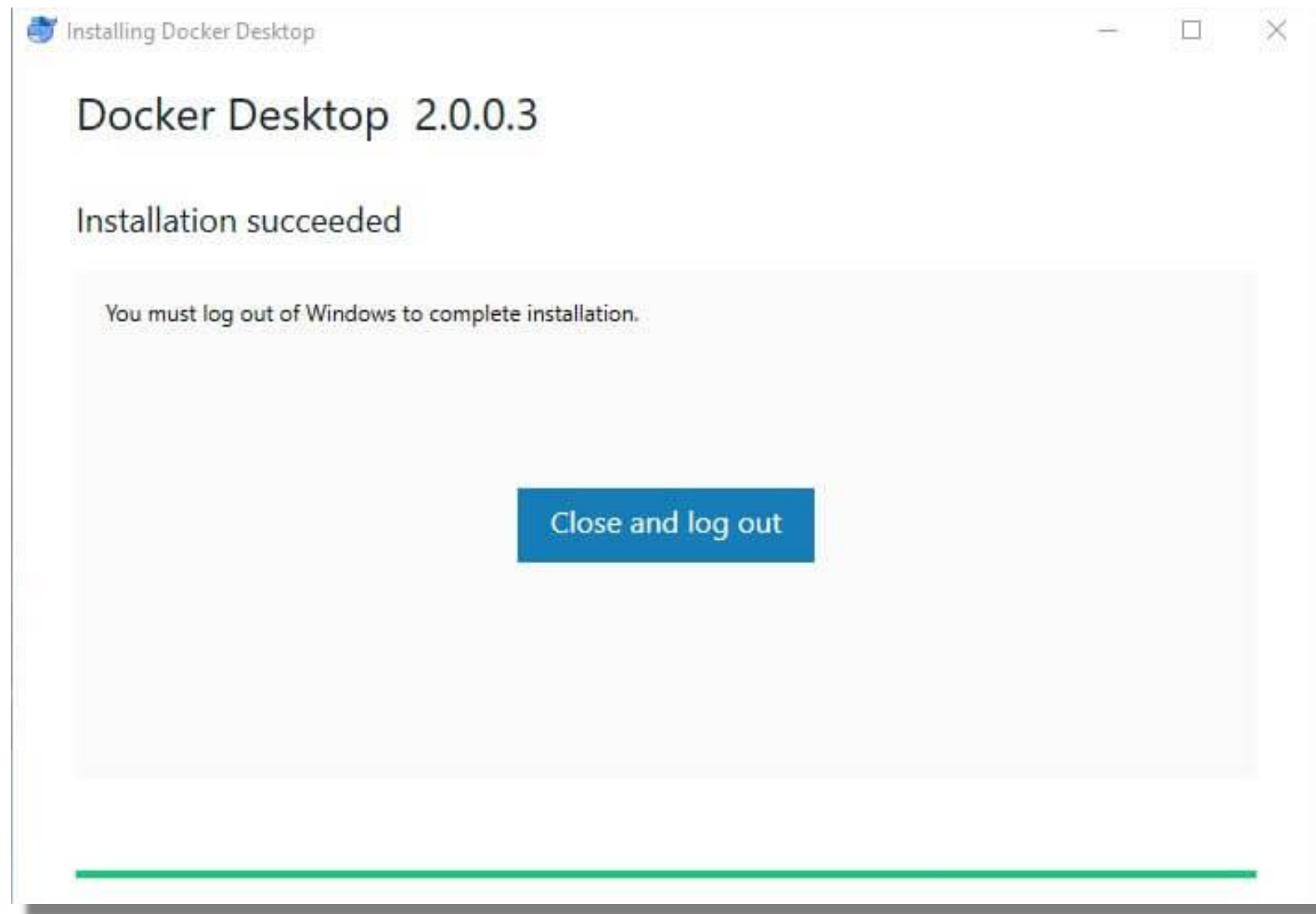
# INSTALLATION DE DOCKER

- Une fois Docker CE téléchargé, procéder à son installation



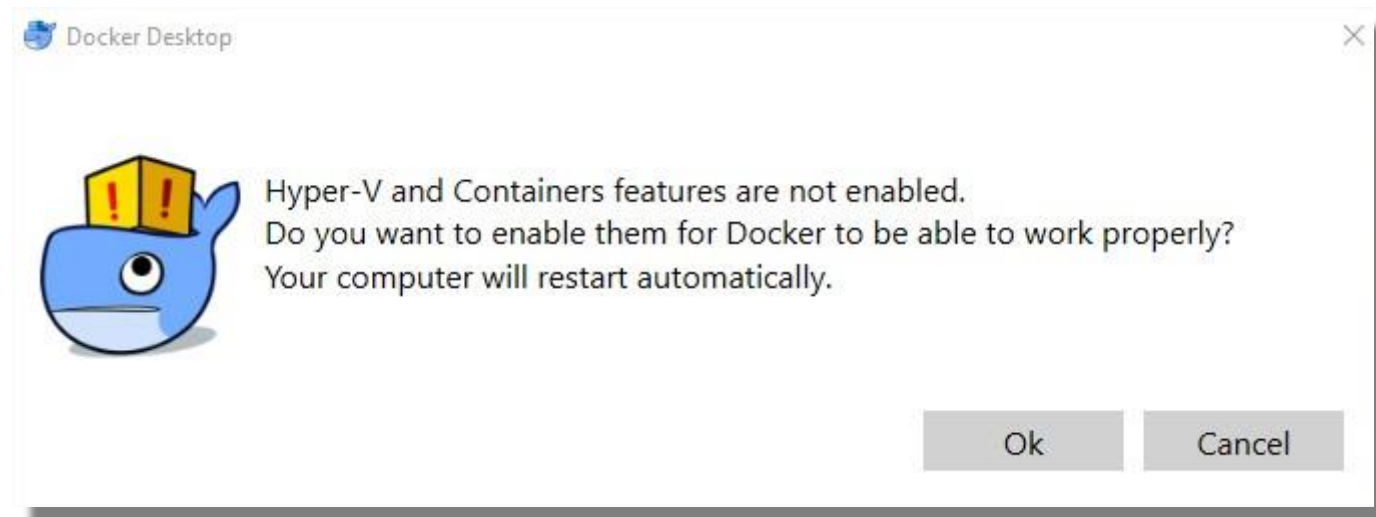
# INSTALLATION DE DOCKER

- Une fois terminée, vous recevrez le message suivant :



# INSTALLATION DE DOCKER

- Si vous n'avez pas activé Hyper-V, alors Docker s'en chargera



- Cliquez sur "OK" pour activer **Hyper-V**. Par la suite votre machine va **automatiquement se redémarrer** à la fin de l'activation d'Hyper-V

# INSTALLATION DE DOCKER

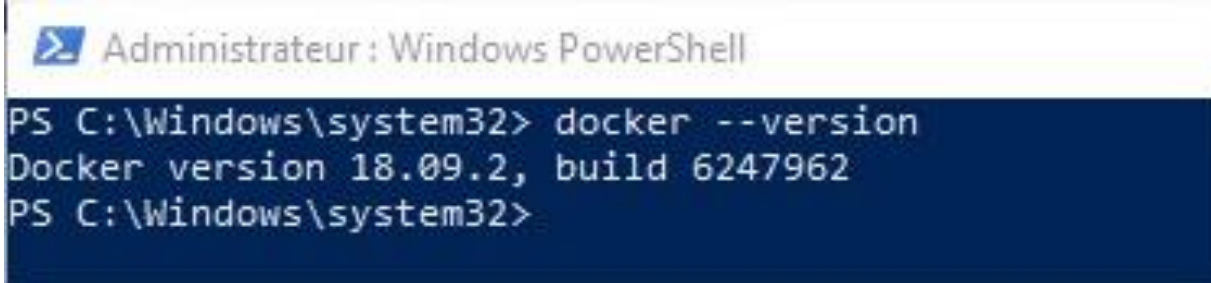
- Après votre redémarrage, Vous verrez la fenêtre suivante indiquant que le moteur Docker est bien installé





# INSTALLATION DE DOCKER

- Dernière étape, lancez votre powershell (ou terminal, CMD) en tant qu'administrateur et exécutez la commande suivante afin de vérifier que votre Docker CE c'est correctement installé
  - `$ docker --version`



```
Administrateur : Windows PowerShell
PS C:\Windows\system32> docker --version
Docker version 18.09.2, build 6247962
PS C:\Windows\system32>
```

- Docker est maintenant installé sur votre machine

03

# LES IMAGES & CONTENEURS

Comprendre le concept d'image et de conteneur Docker

# LES IMAGES DOCKER

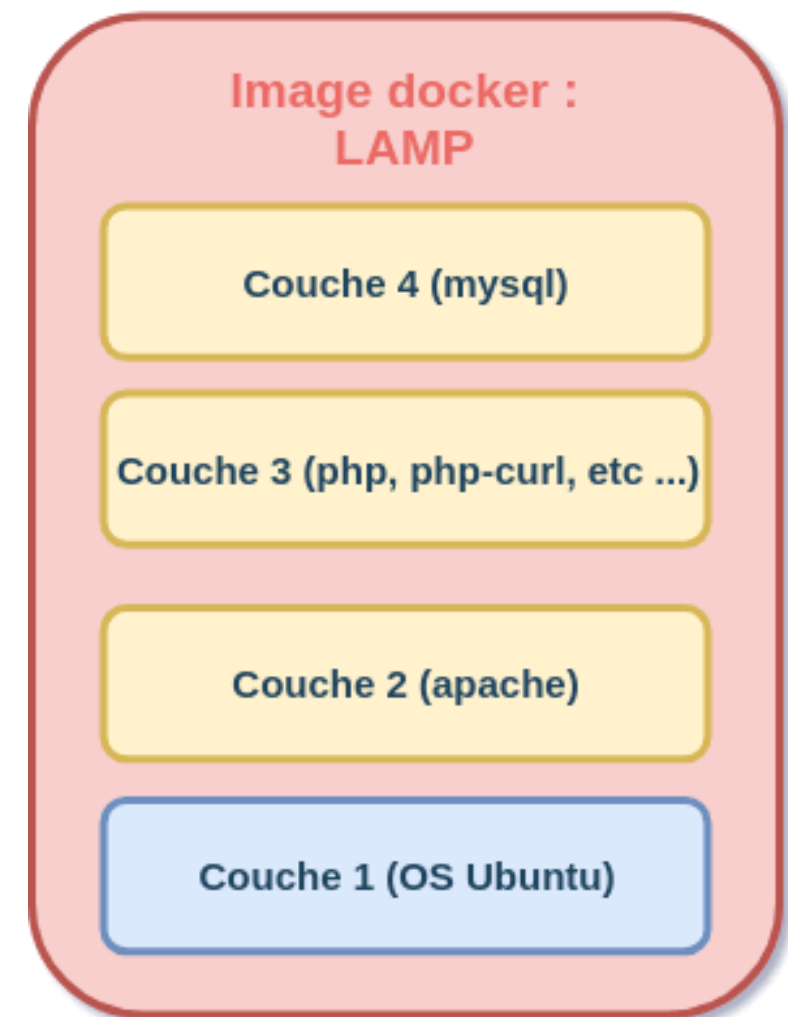
- Qu'est qu'une image Docker ?
  - Sur Docker, un conteneur est lancé en exécutant une image.
- Une image est un package qui inclut tout ce qui est nécessaire à l'exécution d'une application
  - Le code
  - L'exécution
  - Les variables d'environnement
  - Les bibliothèques
  - Les fichiers de configuration

# LES IMAGES DOCKER

- Une **image** est un modèle composé de plusieurs couches
  - Ces **couches** contiennent notre **application** ainsi que les **fichiers binaires** et les **bibliothèques requises**
- Lorsqu'une **image** est **instanciée**, son nom est un **conteneur**
  - Un **conteneur** est donc une **image** en **cours d'exécution**
- Pour mieux **comprendre** le système de **couche**, imaginons par exemple qu'on souhaite **déployer** notre **application web** dans un **serveur LAMP** (*Linux Apache MySQL PHP*)
  - Cette **image** sera **composé** de **4 couches**
    - ✓ Ces couches sont des layers (calques) en lecture seule

# LES IMAGES DOCKER

- Exemple de **serveur LAMP** en détail
  - Une couche **OS**
    - ✓ Pour exécuter Apache, MySQL...
  - Une couche **Apache**
    - ✓ Pour exécuter le serveur Web
  - Une couche **PHP**
    - ✓ Interpréteur et Library PHP
  - Une couche **MySQL**
    - ✓ Contiendra notre SGBD Mysql





# LES IMAGES DOCKER

- Premières commandes Docker
  - Pour commencer on va d'abord récupérer la liste des commandes possibles
    - ✓ \$ docker help
- Sur l'**output** de la commande **help**, nous avons une information d'une grande utilité et vous permettra de gagner beaucoup de temps
  - Run 'docker COMMAND --help' *for more information on a command*. Exemple la commande docker volume

✓ \$ docker volume

```
Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

# LES IMAGES DOCKER

- Commandes Docker pour de l'information

- Petit rappel de la commande exécutée précédemment pour vérifier le fonctionnement

- ✓ \$ docker --version

```
Docker version 20.10.5, build 55c4c88
```

- La commande info permet d'afficher encore plus de détails sur votre installation de Docker

- ✓ \$ docker info

```
Client:
Context:    default
Debug Mode: false
Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.5.1-docker)
  scan: Docker Scan (Docker Inc., v0.6.0)

Server:
Containers: 9
  Running: 1
  Paused: 0
  Stopped: 8
Images: 13
Server Version: 20.10.5
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Cgroup Version: 1
Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentr
  ies splunk syslog
Swarm: inactive
Runtimes: runc io.containerd.runc.v2 io.containerd.runtime.v1.linux
```

# LES IMAGES DOCKER

- Commandes **Docker** pour la gestion des images
  - Pour lister l'ensemble des images présentes sur votre repos local

✓ \$ docker image ls

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myapp	latest	c13d22c29096	3 days ago	415MB
my_lamp	latest	8b2d8ac6b226	3 days ago	552MB
volume_test	latest	6df9cc60aaf6	3 days ago	72.7MB

✓ \$ docker images

- Elle nous donne différentes informations

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
Le titre REPOSITORY peut porter à confusion, c'est essentiellement le nom de l'image.	un tag ici est une façon de faire référence à votre image, ils sont utilisés principalement pour affecter une version à une image	L'identifiant de l'image (unique pour chaque image téléchargée)	Date de la dernière modification de l'image	Taille de l'image

# LES IMAGES DOCKER

- Commandes **Docker** pour la gestion des images
  - Supprimer une image (par nom ou id)
    - ✓ `$ docker rmi <nom_image ou id_image>`
  - Avec l'option **-f** pour **forcer** la suppression
    - ✓ `$ docker rmi -f <nom_image ou id_image>`
  - Supprimer toutes les images
    - ✓ `$ docker rmi -f $(docker images -q)`

# LES IMAGES DOCKER

- Commandes **Docker** pour la gestion des images
  - Rechercher une image sur le hub registry
    - ✓ `$ docker search <nom_image>`
  - Avec l'option **--filter** pour **trier** les images officielles
    - ✓ `$ docker search <nom_image> --filter "is-official=true"`

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	12334	[OK]	
websphere-liberty	WebSphere Liberty multi-architecture images ...	273	[OK]	
ubuntu-upstart	Upstart is an event-based replacement for th...	110	[OK]	
open-liberty	Open Liberty multi-architecture images based...	46	[OK]	
ubuntu-debootstrap	debootstrap --variant=minbase --components=m...	44	[OK]	

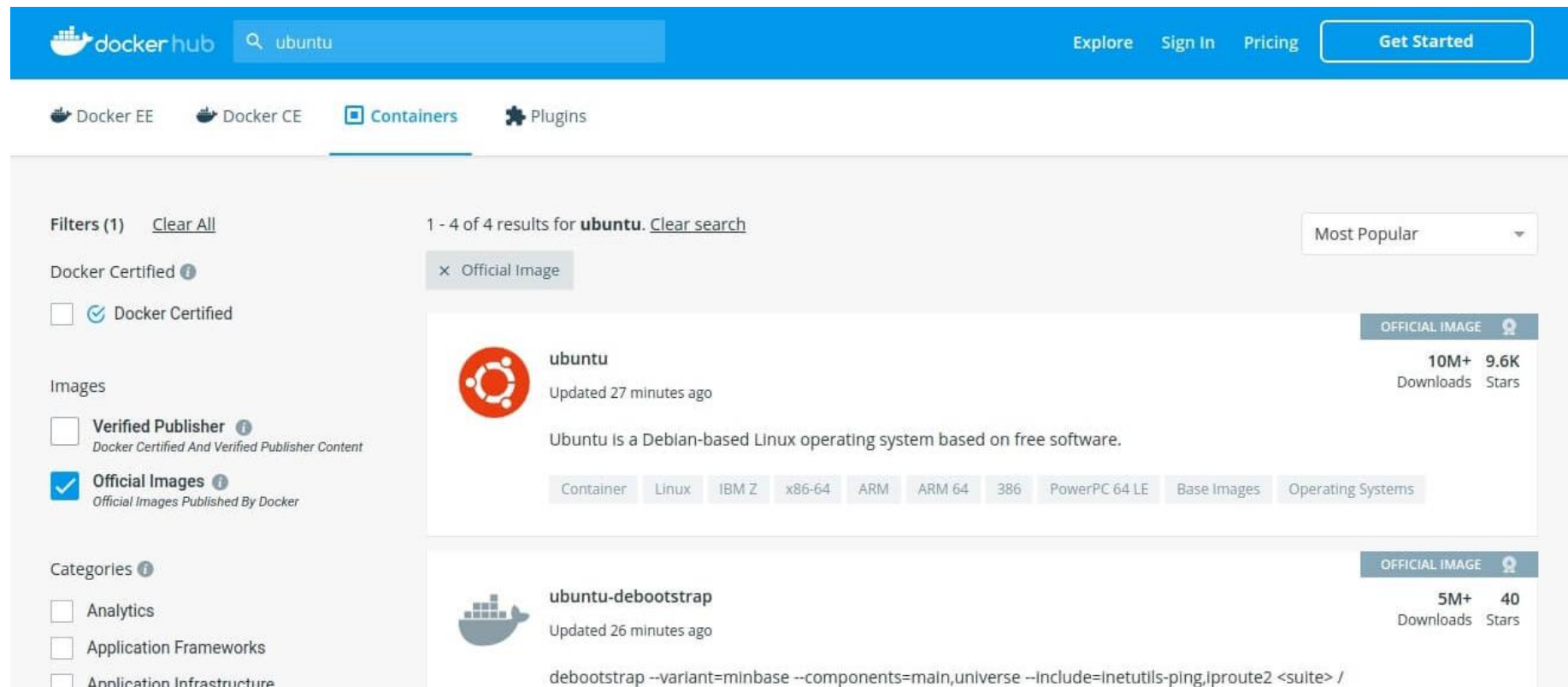
# LES IMAGES DOCKER

- Commandes **Docker** pour la gestion des images
  - Pour télécharger une image à partir du hub
    - ✓ `$ docker pull <nom_image>`
  - Télécharger une version précise (tag)
    - ✓ `$ docker pull ubuntu:16.04`
  - Télécharger la dernière version
    - ✓ `$ docker pull ubuntu:latest`



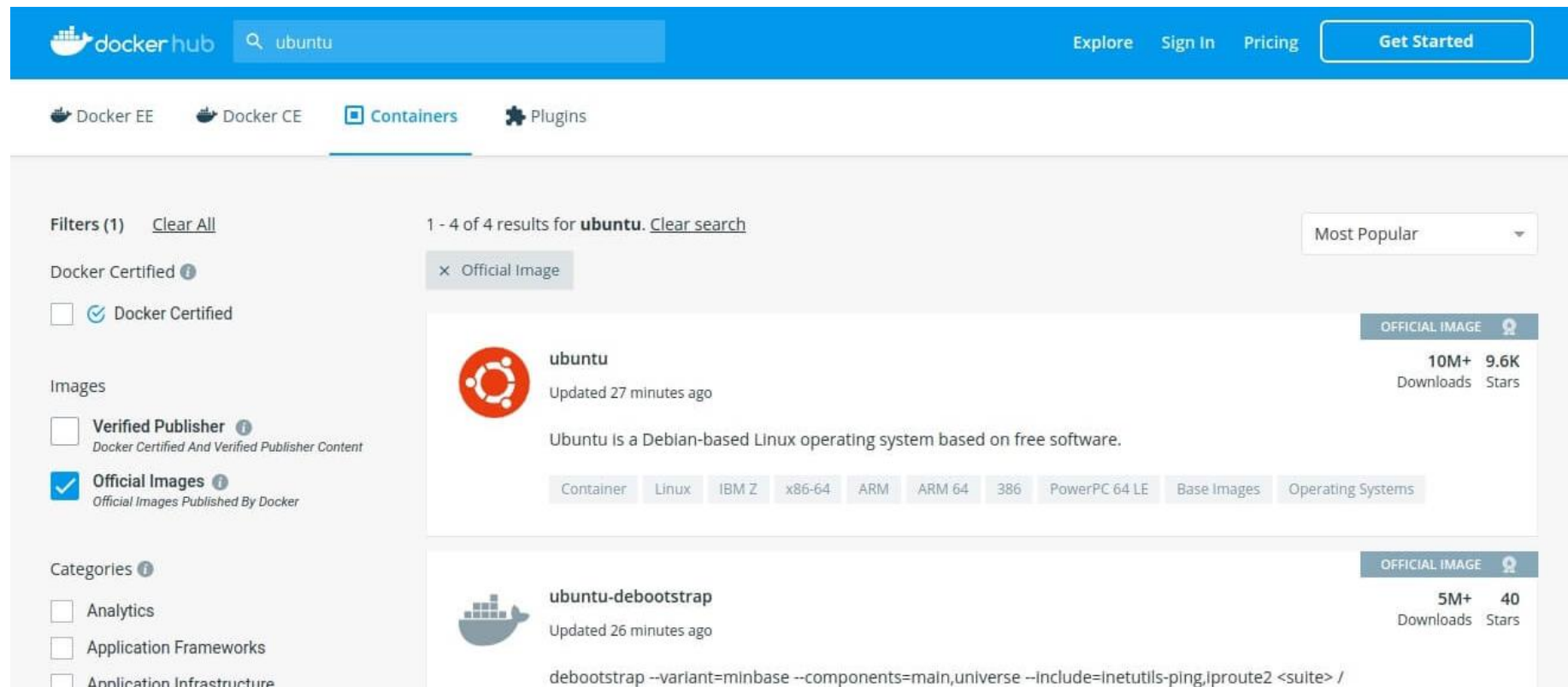
# LES IMAGES DOCKER

- Le Hub Registry Docker
  - Où est-ce que je peux retrouver la liste des images disponibles ?
  - ✓ Le hub Registry



# LES IMAGES DOCKER

- Le Hub Registry Docker
  - Où est-ce que je peux retrouver la liste des images disponibles ?
  - ✓ Le [hub Registry](#)



# LES IMAGES DOCKER

- Le Hub Registry Docker

The screenshot displays the Docker Hub interface for the 'ubuntu' image. At the top, the Docker Hub logo and search bar are visible. The main section features the Ubuntu logo, a star rating, and the text 'Docker Official Images'. Below this, a list of tags includes 'Container', 'Linux', '386', 'PowerPC 64 LE', 'IBM Z', 'x86-64', 'ARM', 'ARM 64', 'Base Images', and 'Operating Systems'. A red box highlights the 'Linux - x86-64 (latest)' dropdown menu and the 'docker pull ubuntu' command button. Below the red box, the 'DESCRIPTION' tab is active, showing a list of supported tags and their respective Dockerfile links. The 'Quick reference' section provides links to help, issue tracking, and maintenance information. A login prompt for reviews is also visible.

**Supported tags and respective Dockerfile links**

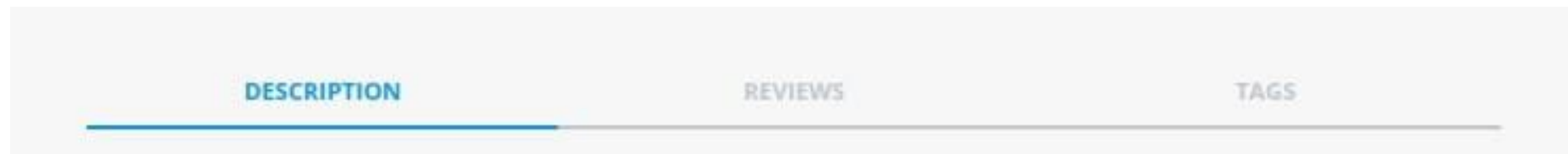
- 18.04, bionic-20190515, bionic, latest ([bionic/Dockerfile](#))
- 18.10, cosmic-20190515, cosmic ([cosmic/Dockerfile](#))
- 19.04, disco-20190515, disco, rolling ([disco/Dockerfile](#))
- 19.10, eoan-20190508, eoan, devel ([eoan/Dockerfile](#))
- 14.04, trusty-20190515, trusty ([trusty/Dockerfile](#))
- 16.04, xenial-20190515, xenial ([xenial/Dockerfile](#))

**Quick reference**

- **Where to get help:**  
the Docker Community Forums, the Docker Community Slack, or Stack Overflow
- **Where to file issues:**  
the cloud-images bug tracker (include the `docker` tag)
- **Maintained by:**  
Canonical and Tianon (Debian Developer)
- **Supported architectures:** (more info)  
`amd64`, `arm32v7`, `arm64v8`, `i386`, `ppc64le`, `s390x`

# LES IMAGES DOCKER

- Le Hub Registry Docker



- **DESCRIPTION** : Description de l'image, souvent on retrouve quelques tags, la configuration de votre conteneur (par exemple la config de votre base de données pour une image basé sur du mysql) et les liens github vers les sources du projet.
- **REVIEWS** : l'avis des utilisateurs
- **TAGS** : les différents tags disponible pour cette image

# LES CONTENEURS DOCKER

## Différence entre image et conteneur dans Docker

- Rappel lorsque vous utilisez des **fonctionnalités** permettant une **isolation** du **processus**
  - namespaces
  - cgroups
    - ✓ On appelle cela des **conteneurs**
- Un **conteneur** est une **instance d'exécution** d'une **image**
  - Plus précisément un **conteneur** est ce que l'**image** **devient** en **mémoire** lorsqu'elle est **exécutée**
    - ✓ Avec un état, un processus utilisateur...

# LES CONTENEURS DOCKER

## Créer un conteneur

- Créer une instance de notre image avec **docker run**
  - `$ docker run [OPTIONS] <Image_Name ou ID>`
    - ✓ Exemple: `$ docker run debian:latest`
- Cette commande peut être complétées par des options
  - `$ docker run -tid ubuntu:latest`
  - ✓ `$ docker run --help` // [Documentation](#)

# LES CONTENEURS DOCKER

## Options de la commande **run**

- **-t** : Allouer un terminal tty (terminal virtuel)
- **-i** : Garder un STDIN ouvert (l'entrée standard, plus précisément l'entrée clavier)
- **-d** : Exécuter le conteneur en arrière-plan
- **-p** : Exposer un ou plusieurs ports (mapping)
- **--name** : donner un nom au container
- **--expose** : Exposer un port ou une plage de ports
  - on demande au firewall du conteneur de nous ouvrir un port ou une plage de port
- **-p** ou **-publish** : Mapper un port déjà exposé, plus simplement ça permet de faire une redirection de port



# LES CONTENEURS DOCKER

## Commandes pour administrer un conteneur

- Pour inspecter un conteneur
  - `$ docker inspect <nom_conteneur>`
- Pour stopper un conteneur Actif
  - `$ docker stop <nom_conteneur>`
- Pour supprimer un conteneur
  - `$ docker rm <nom_conteneur>`

# LES CONTENEURS DOCKER

## Commandes pour administrer un conteneur

- Pour **exécuter** une **commande** dans un **conteneur**
  - `$ docker exec [OPTIONS] <Id ou name> command`
- **Docker exec** peut être également complétée par des options
  - `$ docker exec -ti <nom_conteneur> bash`
- **Lister les conteneurs actifs**
  - `$ docker container ls`    *ou*    `$ docker ps`
- **Lister tous les conteneurs**
  - `$ docker container ls -a`    *ou*    `$ docker ps -a`

04

# LE DOCKERFILE

Créer une image et l'instancier en un conteneur Docker

# LE FICHIER DOCKERFILE

- Le fichier **dockerfile** permet de **créer** des images Docker
  - Rappel, une image est un modèle composé de plusieurs couches...
    - ✓ Elles contiennent notre application
    - ✓ Mais aussi les fichiers binaires, les bibliothèques...
  - Le **dockerfile** contient toutes les couches de notre conteneur :
    - ✓ Il contient toutes les **instructions** pour la **création de notre conteneur**
    - ✓ Il est composé de clé / valeur le rendant ainsi très léger.

# LE FICHIER DOCKERFILE

- Description des clés le plus souvent utilisées
  - FROM : Définit l'image de base qui sera utilisée par les instructions suivantes
  - LABEL : Ajoute des métadonnées à l'image avec un système de clés-valeurs, permet par exemple d'indiquer à l'utilisateur l'auteur du Dockerfile
  - ARG : Variables temporaires qu'on peut utiliser dans un Dockerfile
  - ENV : Variables d'environnements utilisables dans votre Dockerfile et conteneur
  - RUN : Exécute des commandes Linux ou Windows lors de la création de l'image

# LE FICHIER DOCKERFILE

- Description des clés le plus souvent utilisées
  - **COPY** : Permet de **copier** des **fichiers** depuis notre **machine locale** vers le conteneur **Docker**
  - **ADD** : Même chose que **COPY** mais **prend en charge** des **liens** ou des **archives** (si le format est reconnu, alors il sera décompressé à la volée)
  - **ENTRYPOINT** : C'est le **point d'entrée** de votre **conteneur**, en d'autres termes, c'est la **commande** qui sera **toujours exécutée** au **démarrage** du **conteneur**. Il prend la forme de tableau **JSON** (ex : CMD ["cmd1","cmd1"]) ou de texte
  - **CMD** : Spécifie les arguments qui seront envoyés au **ENTRYPOINT**

# LE FICHIER DOCKERFILE

- Description des **clés** le plus souvent utilisées
  - **CMD** : Spécifie les arguments qui seront envoyés au **ENTRYPOINT**
    - ✓ Si il est utilisé pour fournir des arguments par défaut pour l'instruction **ENTRYPOINT**, alors les instructions **CMD** et **ENTRYPOINT** doivent être spécifiées au format de tableau **JSON**
  - **WORKDIR** : Définit le répertoire de travail qui sera utilisé pour le lancement des commandes **CMD** et/ou **ENTRYPOINT** et ça sera aussi le dossier courant lors du démarrage du conteneur
  - **EXPOSE** : Expose un port



# LE FICHIER DOCKERFILE

- Description des **clés** le plus souvent utilisées
  - **VOLUMES** : Crée un **point de montage** qui permettra de **persister les données**
    - ✓ Nous y reviendrons plus tard...
  - **USER** : Désigne quel est l'utilisateur qui lancera les prochaines instructions **RUN, CMD** ou **ENTRYPOINT**
    - ✓ Par **défaut** c'est l'utilisateur **root**
- Une fois rédigé, le **dockerfile** peut être **exécuté**

# LE FICHIER DOCKERFILE

- Créer une image à partir d'un Dockerfile
  - Voici la commande pour qui nous permet de construire une image docker depuis un Dockerfile
    - ✓ `$ docker build -t <image_name> .`
  - Une fois construite votre image docker est stockée sur la machine hôte.
    - ✓ `$ docker images // pour les lister`
  - Ensuite, exécutez votre image personnalisée
    - ✓ `$ docker run [OPT] <Image_Name>`

# LE FICHIER DOCKERFILE

- Publier ses images sur Docker Hub
  - Si vous souhaitez partager votre image avec d'autres utilisateurs, vous pouvez utiliser le [Hub Docker](https://hub.docker.com/).

The screenshot shows the 'Create Repository' page on Docker Hub. The header includes the Docker Hub logo, a search bar, and navigation links: Explore, Repositories, Organizations, Get Help, and a user profile for 'petitvulcan'. Below the header, there's a breadcrumb 'Repositories > Create' and a status 'Using 0 of 1 private repositories. [Get more](#)'. The main form is titled 'Create Repository' and includes a dropdown menu with 'petitvulcan' selected, a 'Name' field, and a 'Description' field. Under the 'Visibility' section, 'Public' is selected with the note 'Appears in Docker Hub search results', while 'Private' is unselected with the note 'Only visible to you'. The 'Build Settings (optional)' section mentions 'Autobuild triggers a new build with every git push to your source code repository. [Learn More](#)'. A message at the bottom says 'Please re-link a GitHub or Bitbucket account' with a note about updated connection methods and a 'Learn More' link. At the very bottom, there are icons for GitHub and Bitbucket, both labeled 'Disconnected', and three buttons: 'Cancel', 'Create', and 'Create & Build'.

# LE FICHIER DOCKERFILE

- Publier ses images sur Docker Hub
  - L'étape suivante est de se connecter au hub Docker à partir de la ligne de commande
    - ✓ `$ docker login`
  - Récupérer ensuite l'id ou le nom de votre image
    - ✓ `$ docker images // pour les lister`
  - Ensuite il faut **rajouter** un **tag** à l'id ou le nom de l'image récupérée
- ✓ `docker tag <img_NameOrId> <Hub-User>/<RepoName>[:<tag>]`

05

# DOCKER VOLUME

Comprendre la persistance des données dans Docker

# DOCKER VOLUME

- Afin de **comprendre** ce qu'est un **volume Docker**, nous devons d'abord **préciser** le **fonctionnement normal** du **système de fichiers** dans **Docker**
  - En effet, une **image Docker** se **compose** d'un **ensemble de layers (calques)** en **lecture seule**
    - ✓ **Docker** ajoute au **sommet** de cette **pile de layers** un **nouveau layer en lecture-écriture**
    - ✓ **Docker** appelle cette **combinaison de couches** un **"Union File System"**

# DOCKER VOLUME

- Voyons **comment** le moteur **Docker** gère les **modifications** de vos **fichiers** au sein de votre **conteneur**
  - Lors d'une **modification** de **fichier**, **Docker** crée une **copie** depuis les **couches** en **lecture seule** vers le **layer** en **lecture-écriture**
  - Lors d'une **création** de **fichier**, **Docker** crée le **fichier** que sur le **layer** en **lecture-écriture**, et ne **touche pas** au **layer** en **lecture seule**
  - Lors d'une **suppression** de **fichier**, **Docker** **supprime** le **fichier** que sur le **layer** en **lecture-écriture**, et si il est **déjà existant** dans le **layer** en **lecture seule** alors il le **garde**



# DOCKER VOLUME

- Les données dans le **layer** de **base** sont en **lecture seule**
  - Elles sont donc **protégées** et **intactes** par **toutes modifications**, seul le **layer** en **lecture-écriture** est **impacté** lors de **modifications de données**
  - Lorsqu'un **conteneur** est **supprimé**, le **layer** en **lecture-écriture** est **supprimé** avec...
  - ✓ Cela signifie que **toutes les modifications** apportées **après le lancement** du **conteneur** auront **disparus** avec

# DOCKER VOLUME

## La création des volumes

- Les volumes Docker ont une **double** fonction
  - Faire **persister** les données
  - **Echanger** des données entres conteneurs
- Les volumes sont des **répertoires** (ou des fichiers) qui **ne font pas partie** du **système** de fichiers Union
  - Ils **existent** sur le **système** de fichiers hôte
- les volumes constituent souvent le **meilleur choix** pour **persistance** des **données** pour le layer en **lecture-écriture**, car un volume n'augmente pas la **taille** des conteneurs

# DOCKER VOLUME

## Créer et gérer des volumes

- Pour **créer un volume**, nous utiliserons la commande suivante
  - `$ docker volume create <NomVolume>`
- Pour **lister les volumes**
  - `$ docker volume ls`
- Pour **recupérer les informations d'un volume**
  - `$ docker volume inspect <NomVolume>`

```
[
  {
    "CreatedAt": "2019-07-03T10:28:55+02:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/data-test/_data",
    "Name": "data-test",
    "Options": null,
    "Scope": "local"
  }
]
```

# DOCKER VOLUME

## Relier un conteneur à un volume

- D'abord il faut **définir** le **répertoire** du **conteneur** à **sauvegarder**
  - **Exemple:** /data
- Ensuite il faut **créer** un **volumes**
  - `$ docker volume create Test_v`
- Pour **indiquer** un **volume** lors de la **création** d'un **conteneur**
  - `$ docker run --name <NomConte.> -v Test_v:/data <img>`
- Pour **supprimer** un **volume**
  - `$ docker volume rm <NomVolume>`

06

# DOCKER-COMPOSE

Comprendre le fonctionnement d'une application multi conteneurs

# DOCKER-COMPOSE

- Docker Compose est un outil permettant de définir le comportement de vos conteneurs et d'exécuter des applications Docker à conteneurs multiples
  - La configuration se fait à partir d'un fichier YAML
- Ensuite, avec une seule commande, vous créez et démarrez tous vos conteneurs de votre configuration
  - Il permet d'exécuter plusieurs commandes dans un ordre bien précis
- Le fichier **docker-compose.yml** est composé de nombreux paramètres, nous allons détailler les principaux

# DOCKER-COMPOSE

- Détails du fichier `docker-compose.yml`
  - `version: '3.7'` : Voir liste des versions de [Docker Compose](#)
  - `services` : Les services ne sont en réalité que des conteneurs
    - ✓ On déclare ici l'ensemble de nos conteneurs / services
  - `volumes` : On déclare les volumes qui seront créés par docker
- Le fichier `docker-compose` doit se trouver à la racine du projet
  - Au même niveau que le fichier `dockerfile`

```
|— app
|   |— db-config.php
|   |— index.php
|   └— validation.php
|— articles.sql
|— docker-compose.yml
└— Dockerfile
```

# DOCKER-COMPOSE

Lancer l'application depuis `docker-compose.yml`

- Placez vous au niveau du dossier qui contient le fichier `docker-compose.yml`
  - Ensuite lancez la commande suivante pour **exécuter les services du `docker-compose.yml`**
    - ✓ `$ docker-compose up -d`
  - Pour seulement **lister les conteneurs du `docker-compose.yml`**
    - ✓ `$ docker-compose ls`
  - **Vérifiez les logs des services de votre Docker Compose**
    - ✓ `$ docker-compose logs`



# DOCKER-COMPOSE

Arrêter / Redemarrer une application docker-compose.yml

- Voici les principales commandes pour :
  - Tuer les conteneurs du Docker Compose
    - ✓ `$ docker-compose kill`
  - Stopper les conteneurs du Docker Compose
    - ✓ `$ docker-compose stop // -t pour indiquer un timeout`
  - Démarrer les conteneurs du Docker Compose
    - ✓ `$ docker-compose start`

# DOCKER-COMPOSE

## Supprimer une application docker-compose.yml

- Voici les principales commandes pour :
  - Arrêtez les conteneurs et supprimer les conteneurs, réseaux, volumes, et les images
    - ✓ `$ docker-compose down // -t pour un timeout`
  - Supprimer des conteneurs stoppés du Docker Compose
    - ✓ `$ docker-compose rm // -f pour forcer la suppression`
  - Lister les images utilisées dans le docker-compose.yml
    - ✓ `$ docker-compose images`

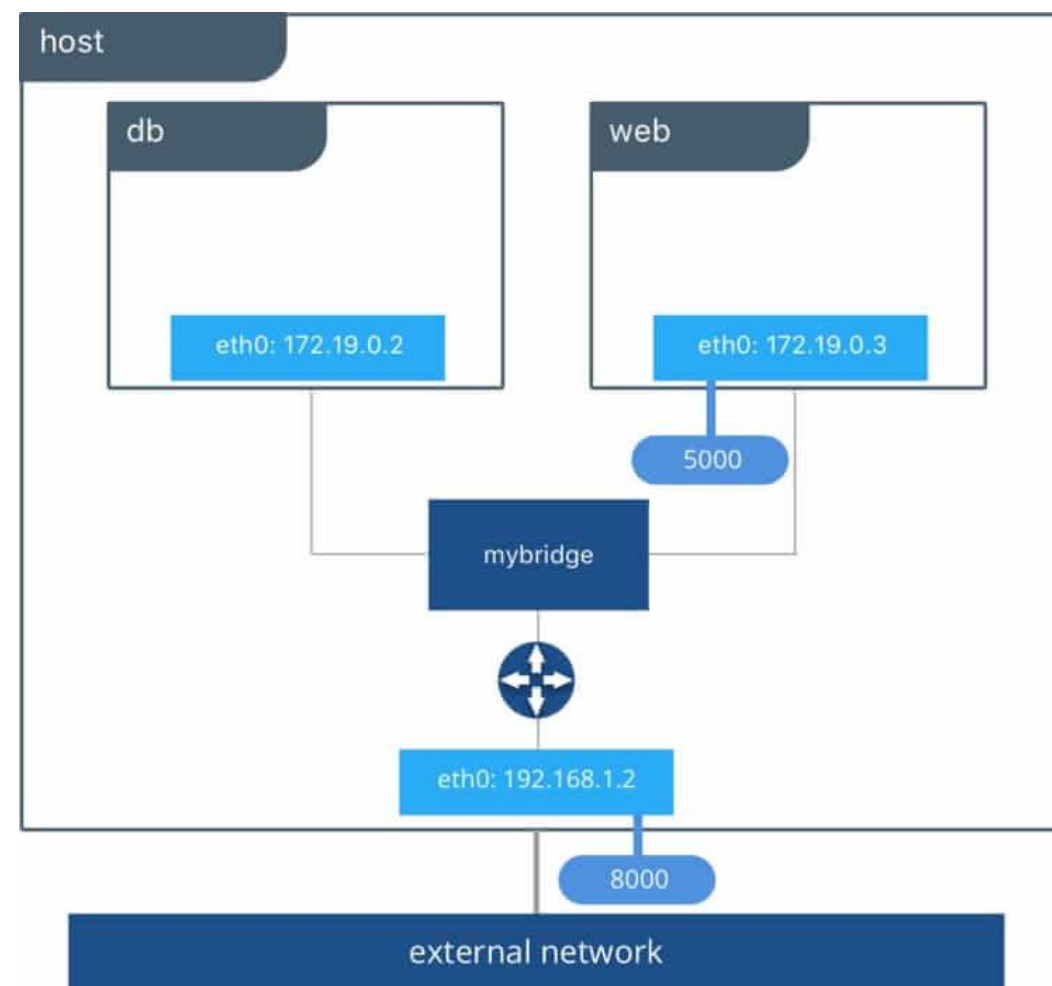
07

# LE NETWORK DOCKER

Appréhender la communication réseau avec Docker

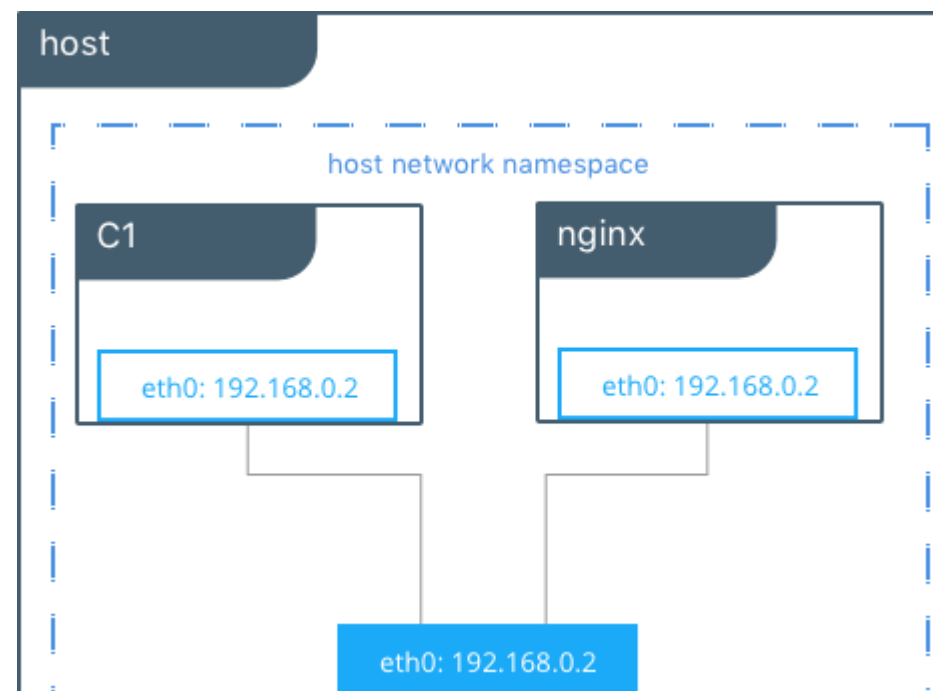
# LE NETWORK DOCKER

- Présentation des différents types de réseau Docker
  - Le driver Bridge



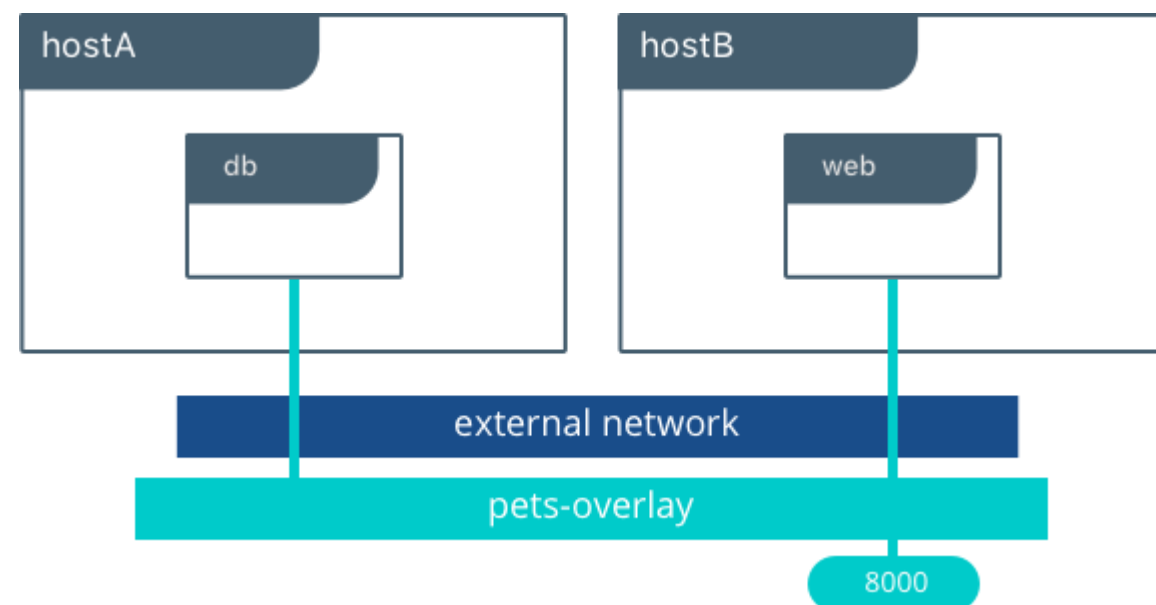
# LE NETWORK DOCKER

- Présentation des différents types de réseau Docker
  - Le driver host



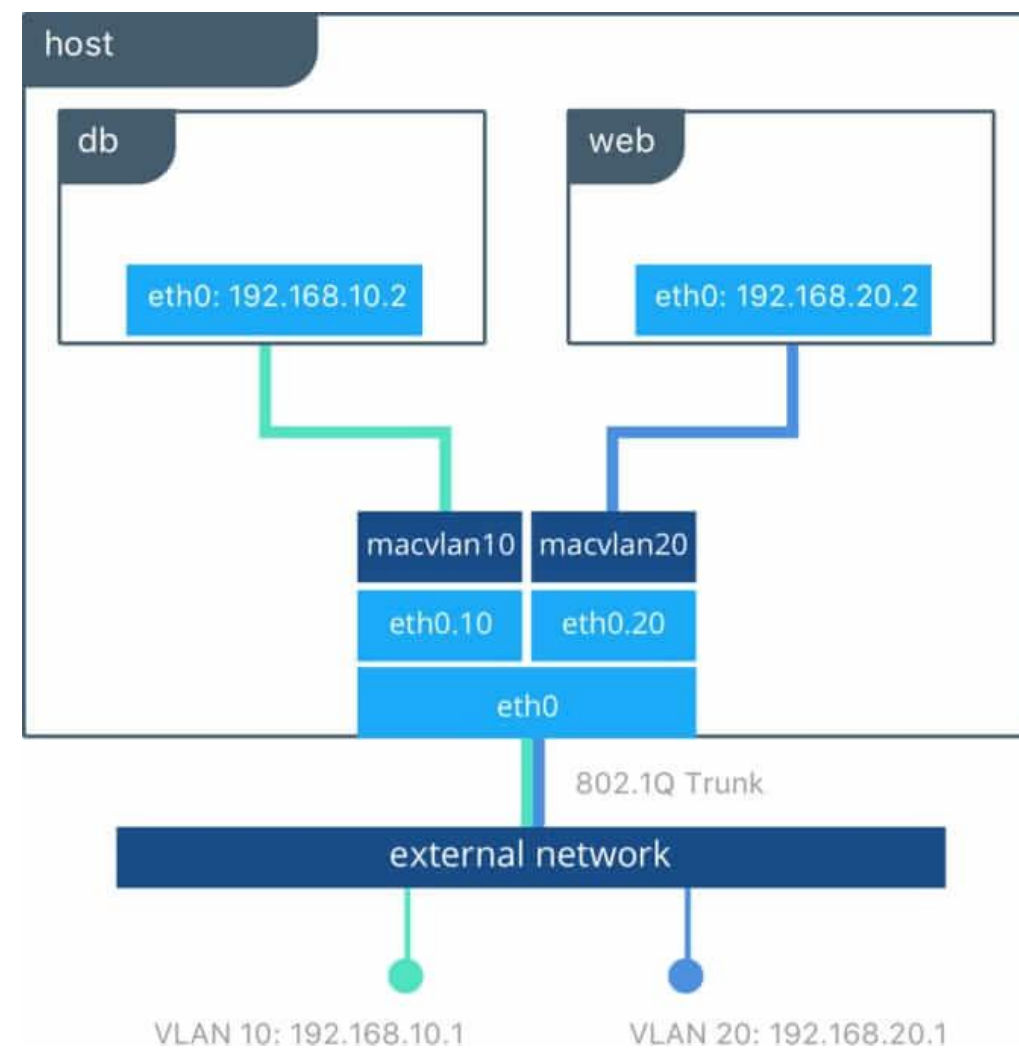
# LE NETWORK DOCKER

- Présentation des différents types de réseau Docker
  - Le driver overlay



# LE NETWORK DOCKER

- Présentation des différents types de réseau Docker
  - Le driver macvlan



# LE NETWORK DOCKER

## Manipulation du réseau dans Docker

- Voici les principales commandes pour :
  - Créer un réseau Docker
    - ✓ `$ docker network create --driver <driver> <network>`
  - Lister l'ensemble des réseaux Docker
    - ✓ `$ docker network ls`
  - Récupérer des informations sur le réseau docker
    - ✓ `$ docker network inspect mon-bridge`



# LE NETWORK DOCKER

## Manipulation du réseau dans Docker

- Voici les principales commandes pour :
  - **Connecter** un conteneur à un **réseau docker**
    - ✓ `$ docker network connect <network> <container>`
  - **Déconnecter** un conteneur du **réseau docker**
    - ✓ `$ docker network disconnect <network> <container>`
  - **Démarrer** un conteneur et le **connecter** à un **réseau docker**
    - ✓ `$ docker run --network <network> <image>`

# LE NETWORK DOCKER

## Manipulation du réseau dans Docker

- Voici les principales commandes pour :
  - Créer une requête de ping
    - ✓ `$ docker exec <conteneur> ping -c 1 <ip>`
  - Supprimer un ou plusieurs réseau(x) docker
    - ✓ `$ docker network rm <network>`
  - Supprimer tous les réseaux docker non inutilisés
    - ✓ `$ docker network prune // -f pour forcer`

08

# LES ORCHESTRATEURS

Introduction à la notion d'orchestrateur de conteneurs

# LES ORCHESTRATEURS

Qu'est-ce qu'un orchestrateur de conteneurs?

- Il permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs
  - Compatible avec tous les environnements qui exécutent des conteneurs
    - ✓ Elle permet de déployer la même application dans différents environnements sans modifier sa conception
  - les microservices stockés dans les conteneurs simplifient l'orchestration des services (stockage, réseau, sécurité...)

# LES ORCHESTRATEURS

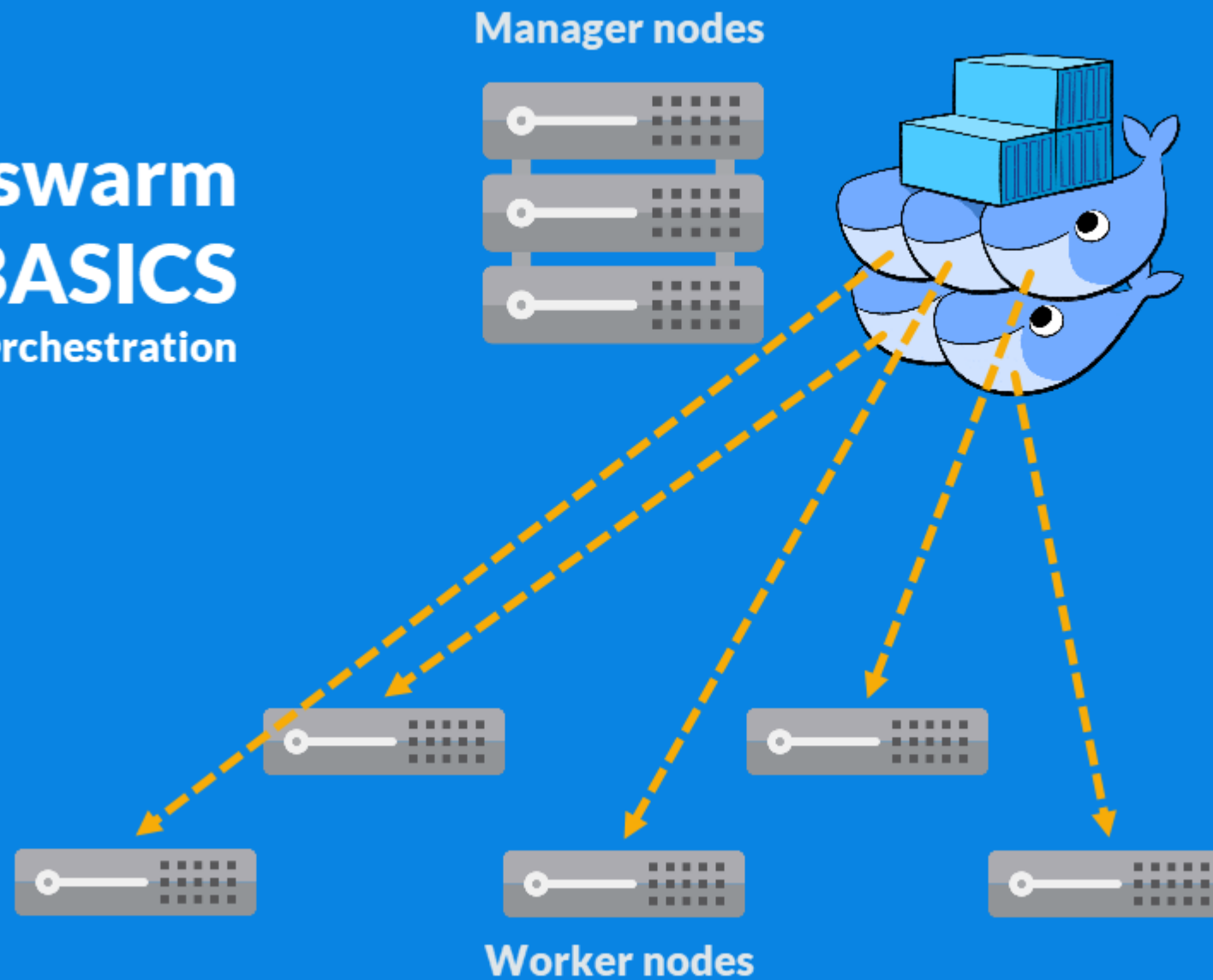
À quoi sert l'orchestration des conteneurs ?

- Vous pouvez utiliser l'orchestration des conteneurs pour automatiser et gérer les tâches suivantes
  - Approvisionnement et déploiement
  - Configuration et planification
  - Allocation des ressources
  - Mise à disposition des conteneurs
  - Mise à l'échelle ou suppression de conteneurs
  - Équilibrage de la charge et routage du trafic
  - Surveillance de l'intégrité des conteneurs
  - Configuration des applications en fonction du conteneur...

# LES ORCHESTRATEURS

## docker swarm BASICS

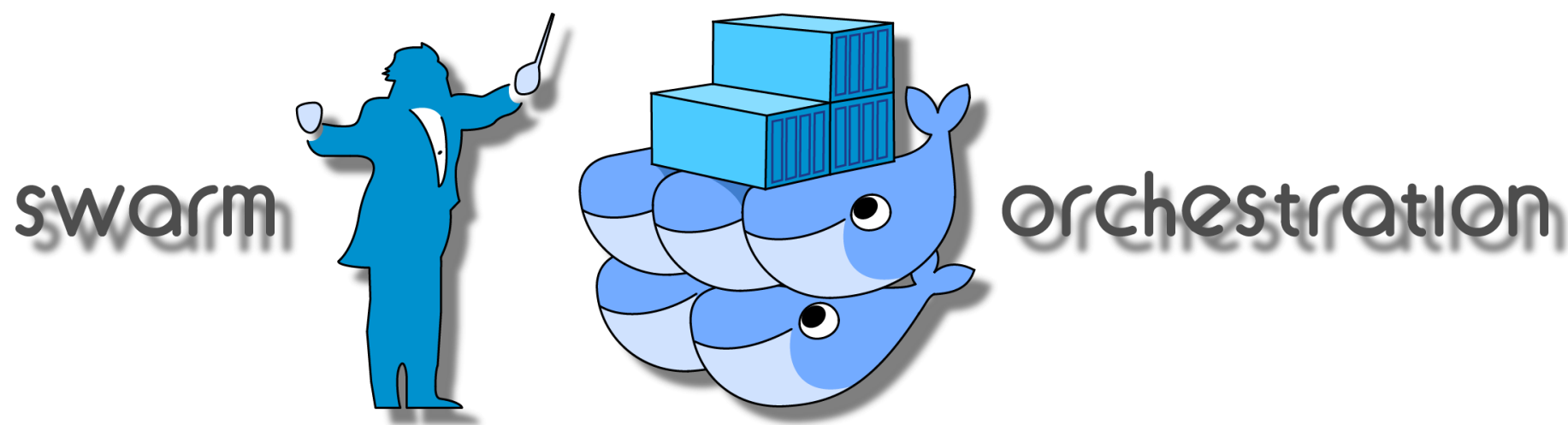
Container Orchestration



# LES ORCHESTRATEURS

Swarm est un orchestrateur dédié à Docker

- Swarm est un **orchestrateur** créé par et dédié à Docker
  - Il prend en charge les **dockerfiles** et **docker-compose**
    - ✓ Plus facile à prendre en main que **Kubernetes**



# LES ORCHESTRATEURS

Kubernetes est le leader de orchestrateurs

- Il est l'outils de déploiement le plus performant actuellement
  - Fonction autoscalling pour les déploiement
    - ✓ Plus difficile à prendre en main et à maitriser





# MERCI DE VOTRE ATTENTION

Des questions...?