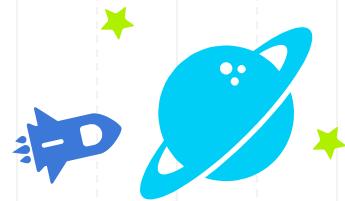


# Introduction Docker

## Conteneurisation et déploiement avec python





# Présentation du cours

- **Durée :** 3 jours (8h) - 24h
- **Objectifs :**
  - Apprendre la CLI Docker
  - Savoir construire et manipuler des container docker
  - Déployer un application python sous forme de container docker
  - Comprendre les principes de l'orchestration de container
- **Notations**
  - TP & QCM





## Docker overview (1/2)

- **Les conteneurs:**
  - principes, objectifs et solutions.
- **Docker en pratique :**
  - Installation de Docker, Conteneurs et images, Lancer son premier conteneur, le cycle de vie des conteneurs, accéder au conteneur et construire une image Docker : le Dockerfile et ses instructions
- **Prise en main du client Docker :**
  - Introduction à la CLI Docker, interactions avec un conteneur démarré, commandes relatives aux images, interactions avec le registry, réseau et volumes.
- **Bonnes pratiques :**
  - Variables d'environnement et conteneurs: ENV, méta-information et images: LABEL. Paramétriser le build d'une image, modifier le contexte système au cours du build et Auto-guérison (self healing).



## Docker overview (2/2)

- **Docker Compose :**
  - Introduction à Docker Compose
  - Etude de cas
- **Orchestration de conteneurs Docker:**
  - Docker Swarm: clustering avec Docker.
  - Gestion des configurations et des secrets.
  - Kubernetes pour le clustering avancé : Environnement et prise en main.
  - Déploiement d'une l'application exemple.





# Les conteneurs: principes, objectifs et solutions

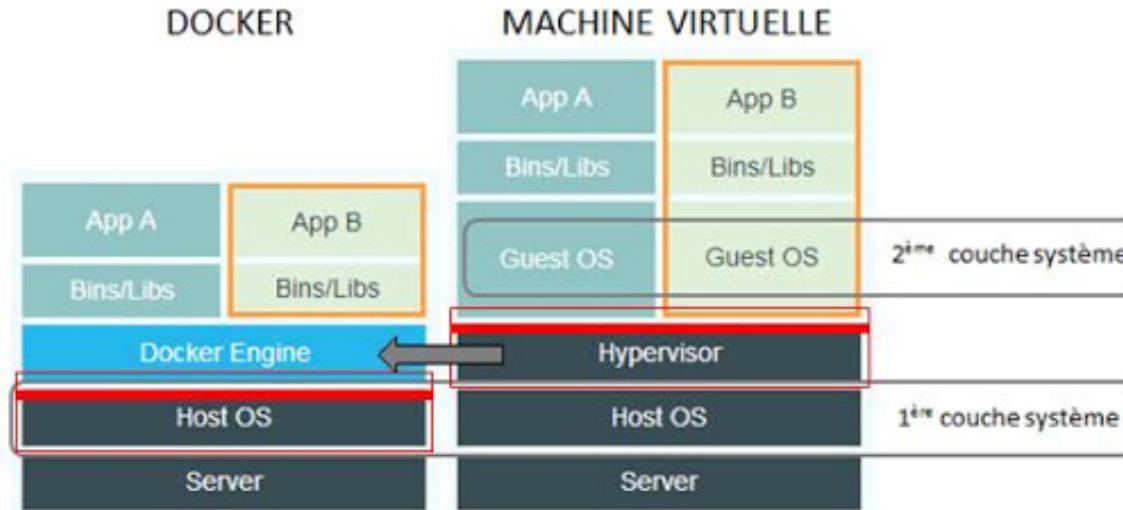
- Les conteneurs logiciels **cherchent à répondre à la même problématique que les conteneurs physiques** aussi appelés conteneurs intermodaux, (les grandes boites de métal standardisées) **qui transportent de nos jours l'essentiel des biens matériels** par camion, train, avion ou bateau.
- Dans une architecture à base de conteneurs:
  - **Le contenu du conteneur**, c'est-à-dire le code et ses dépendances (jusqu'au niveau de l'OS), est de la **responsabilité du développeur**.
  - **La gestion du conteneur et des dépendances** que celui ci va entretenir avec l'infrastructure (stockage, réseau ou calcul) sont de la **responsabilité de l'exploitant**.
- Par ailleurs, **il existent aussi des conteneurs autre que Docker** pour d'autres systèmes, comme FreeBSD ou Solaris. Nous ne les aborderons pas dans le cadre de ce cours.





# Comment les conteneurs autorisent la réalisation de nouvelles architectures informatiques

- La première chose à comprendre c'est la différence entre la conteneurisation et la virtualisation.





# Comment les conteneurs autorisent la réalisation de nouvelles architectures informatiques

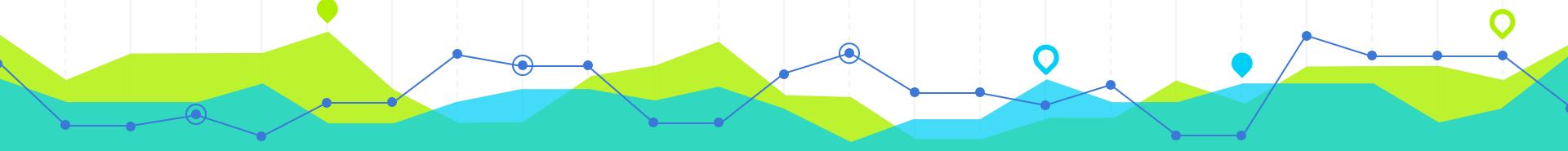
- Une architecture à base de conteneurs offre une **solution de compromis**.
- Le conteneur offre **l'isolation** permettant à un développeur d'embarquer l'ensemble des dépendances dont il a besoin **y compris les dépendances de niveau OS**.
- De plus, un conteneur s'appuie sur le noyau du système hôte. Il est donc **très léger et démarre presque aussi vite que le processus qu'il encapsule**.
- L'idée est donc de **simplifier au maximum le système d'exploitation** en éliminant les services non utilisés par les applicatifs.
- **Le nombre de conteneurs qu'un même hôte peut exécuter est donc nettement plus élevé que son équivalent en machines virtuelles.**





## Qu'est-ce qu'un conteneur finalement ?

- Un conteneur est tout simplement **un système de fichiers sur lequel s'exécutent des processus** (type application) de manière:
  - **Contrainte** : grâce à CGroups (Control Groups en référence à la commande chroot qui permet de partitionner les ressources d'un hôte comme le processeur, la mémoire les accès au réseau ou à d'autres terminaux) qui spécifie les limites en termes de ressources.
  - **Isolée**: grâce notamment au fait que les conteneurs ne se voient pas les uns les autres si ils ne partagent pas un même réseau.





# Docker en pratique : Installation

- **Sur Ubuntu**

Aller sur la documentation officielle et installer docker en ligne de commande

- **Sur Mac OS**

Aller sur la documentation officielle et télécharger l'application Docker Desktop

- **Sur Windows :: Installer une distribution linux sur VM (conseiller)**

Vous pouvez utiliser pour cela VirtualBox ou VMWare

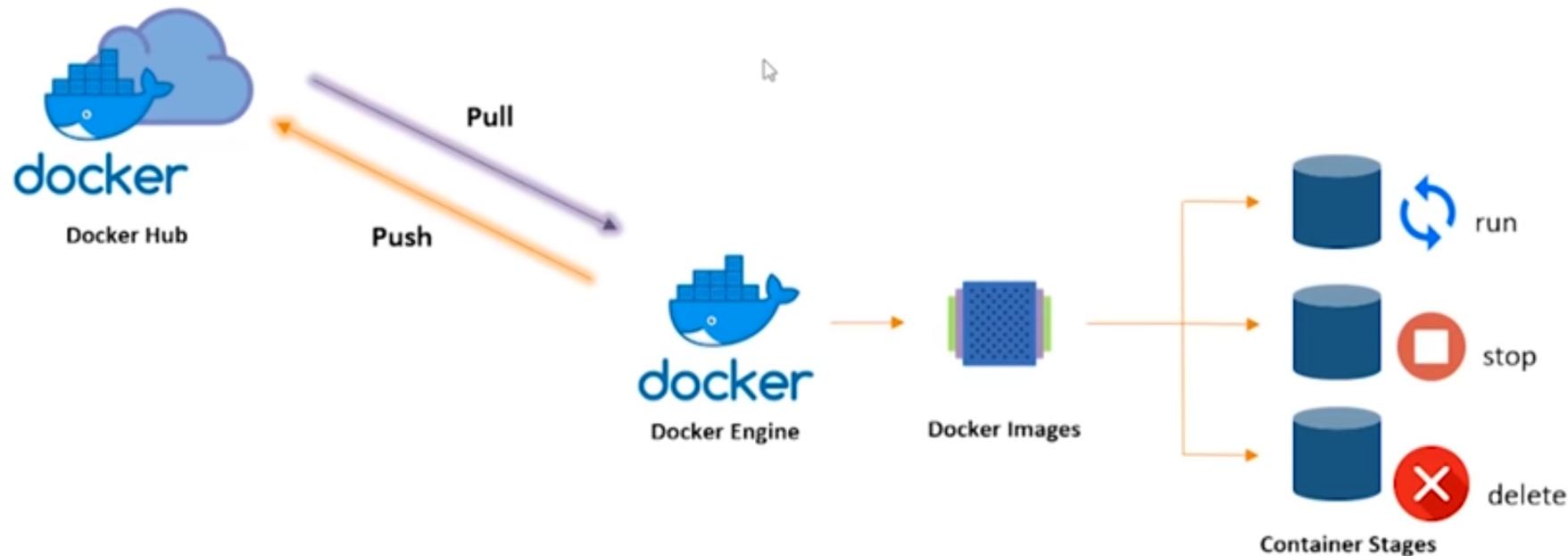
Ou télécharger l'application Docker Desktop

- **Vérifier pour installation avec la commande : *docker info***





# Le cycle de vie des conteneurs



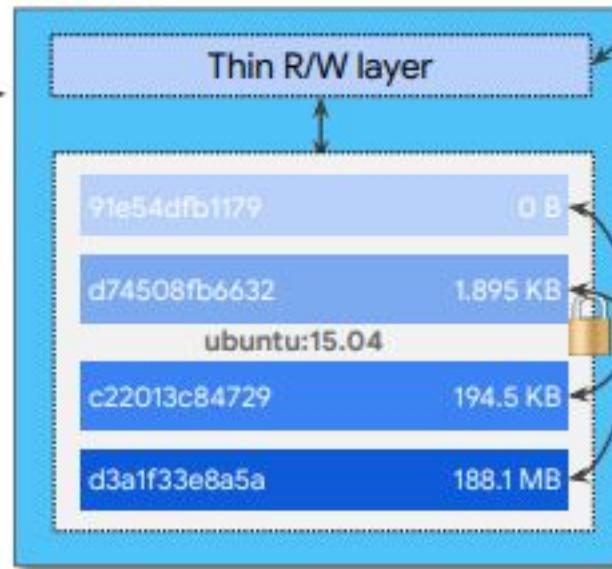


# Le concept de image layers

Dockerfile

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Container

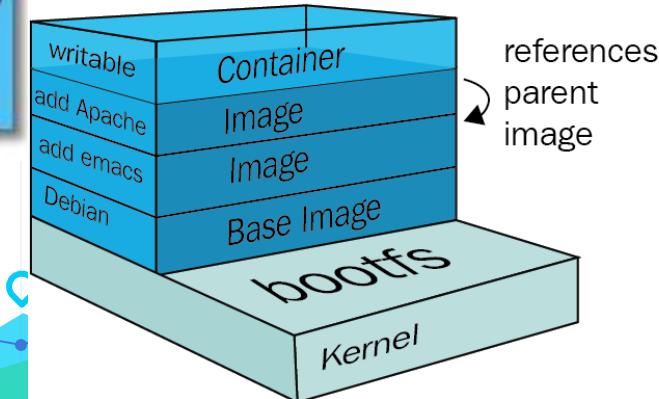


Container  
layer

docker commands

```
$> docker build -t py-web-server .
$> docker run -d py-web-server
$> docker images
$> docker ps
$> docker logs <container id>
$> docker stop py-web-server
```

Base  
Image  
Layers  
(R/O)



references  
parent  
image



# Construire une image Docker : Dockerfile et ses instructions

- Repose sur peu d'instruction mais il est important de bien les comprendre.  
Ci-dessous, les principales instructions à connaître :

**FROM** : Specify the base image

**MAINTAINER** : Specify the image maintainer

**RUN** : Run a command

**ADD** : Add a file or directory

**EXPOSE** : expose ports to be accessed

**ENV** : Create an environment variable

**CMD** : What process to run when launching a container from this image.





# Introduction à la CLI Docker : vocabulaire



## Docker Engine

- ★ Virtualizes OS, not hardware
- ★ Infrastructure agnostic
- ★ Best for Microservices

## Container

- ★ Runs only what you need
- ★ Runs minimal OS, app runtime, file system, etc.
- ★ Runs in its own process

## Application

- ★ Application Runtime
- ★ Application Dependencies
- ★ Environment Setting

**Docker Engine** - Manages and creates Docker images and containers

**Container** - Isolated process running application, runtime and dependencies

**Dockerfile** - Contains instructions to assemble the image

**Image** - Read-only executable for creating containers

**Application** - Binaries, libraries, and dependencies needed to run application

**Docker Hub** - Hosts public repositories of Docker images

**Local image repository** - Stores Docker images locally



# Exemple : Hello world

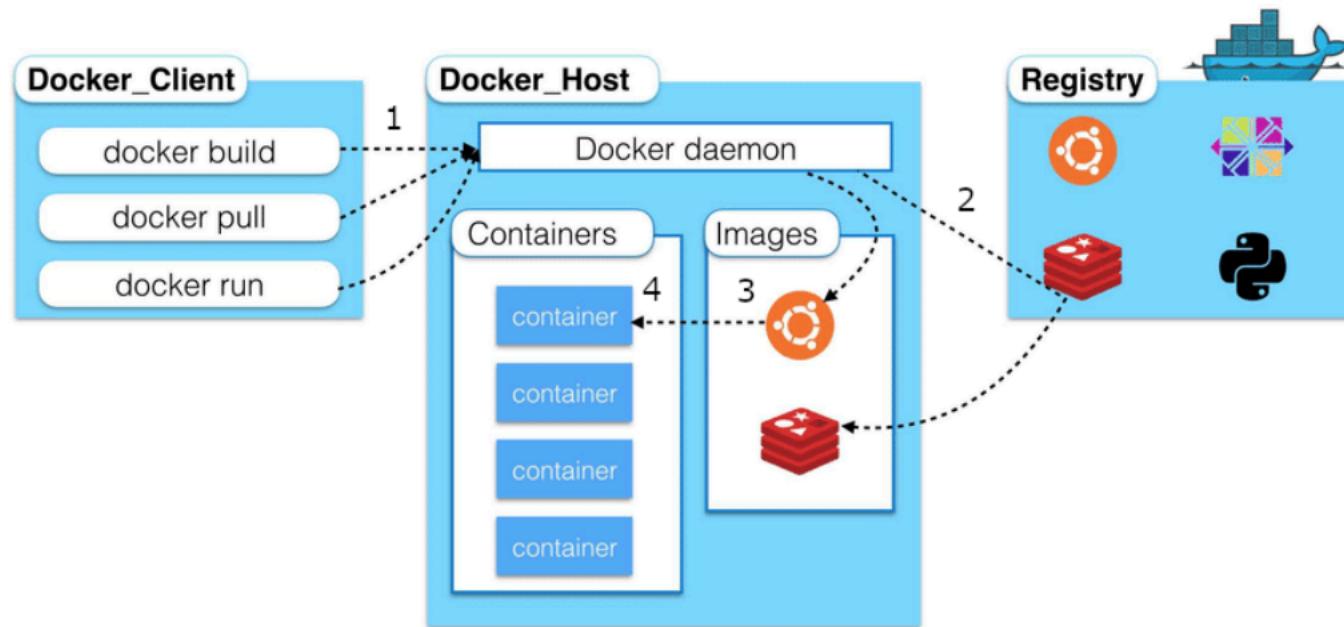
● Source : isatorres

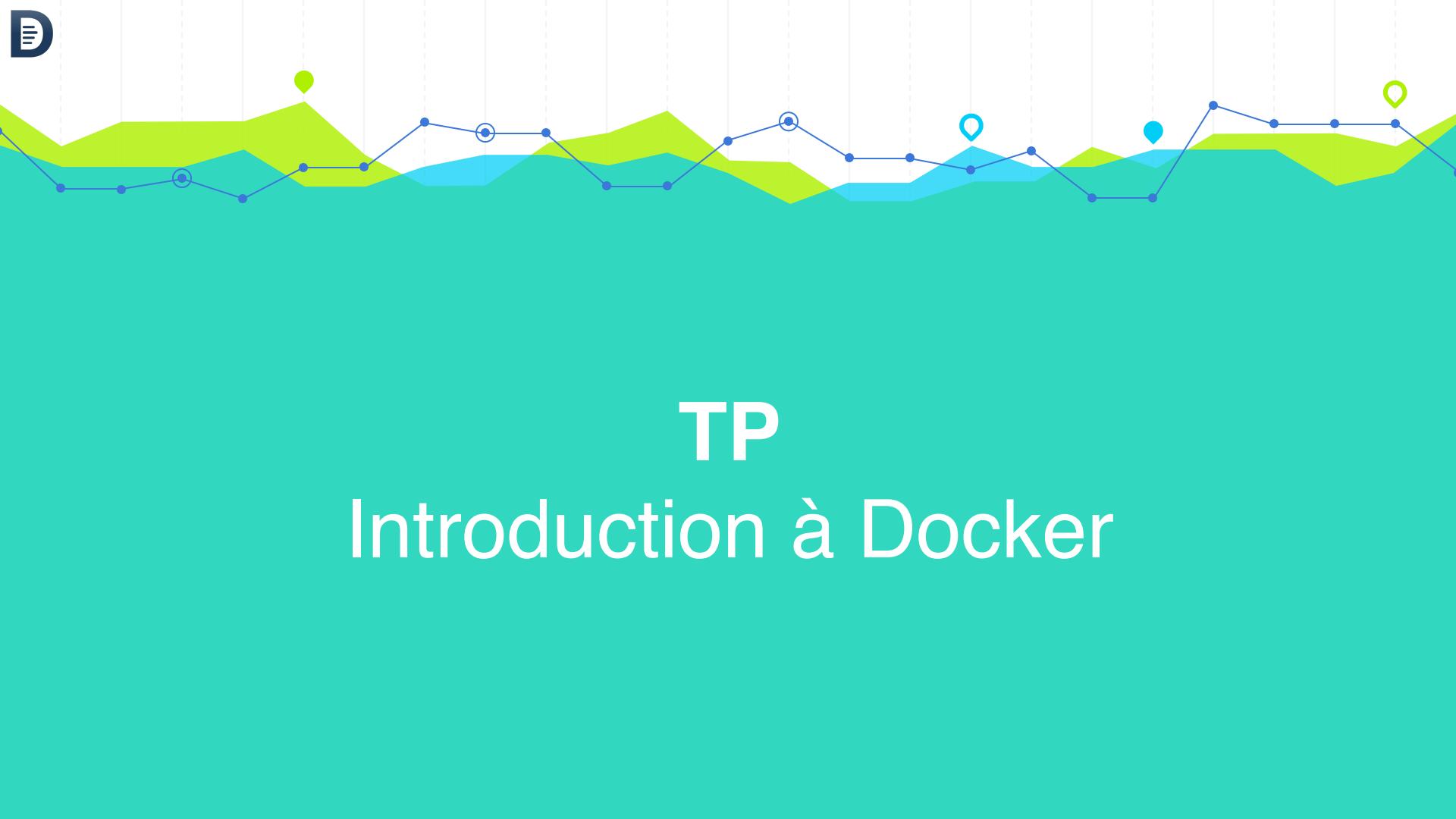
Running a Docker container  
Scenario: Containerized Hello World!

Steps	What happens?	Image Repository	Container Runtime	Good things to know
Create Dockerfile to assemble image <code>touch Dockerfile</code>	Nothing yet, it's just an empty text file	Empty	Empty	Sample Dockerfile: <code>FROM alpine CMD [ "echo", "hello world!" ]</code>
Build image <code>docker build dockerfilePath -t imageName</code>	Image is created and added to the local repo		Empty	Instructions in the Dockerfile tells Docker where to place files and how to start the application
Create container from image <code>docker run imageName</code>  Try it! <code>docker run hello-world</code>	A container process is created from the image		○ ↓ Docker	Docker downloads an image named <i>imageName</i> from Docker Hub if not found in the local image repo. A random container name is generated. Provide your own name: <code>docker run --name containerName imageName</code>
 Application is running in an isolated process! The container runs as long as the contained process with PID 1 is running (enter <code>pid 1</code> in a container terminal). Note: Short-running PID 1 processes = short running containers				List all containers (running and exited) <code>docker ps -a</code>
Start a stopped container <code>docker start containerName</code>	The application is started.		○ → Docker	Get a terminal to the container <code>docker exec -it containerName sh</code>
Stop the container process <code>docker stop containerName</code>	The application is terminated.		○ Docker	Show container logs <code>docker logs containerName</code>
Remove stopped container <code>docker rm containerName</code>	Container is removed. The image stays in the local repo.	Empty	Empty	List local images <code>docker images</code>
Remove local image from local repo <code>docker rmi imageName</code>	Image is removed. Dockerfile persists in the file system.	Empty	Empty	All containers created from an image must be removed before removing an image



# Docker pull





# TP

# Introduction à Docker



## Écrire une image Docker avec les spécifications suivantes :

- une image alpine de base (se renseigner sur les images alpines, quels sont leur avantage par rapport à une image normale?)
- python3.x, pip3 et vim
- une installation automatique du fichier *requirements.txt* (qui contiendra entre autre la librairie virtualenv) que vous écrirez à la racine de votre application. Renseignez vous sur ce fichier *requirements.txt*, que fait il, pourquoi est il utile?
- un répertoire *app* (dans lequel se trouvera l'application *app.py*)
- une exposition du port de l'application
- pour finir builder un container qui lancera l'application lors de sa création
- (**BONUS**) se renseigner sur le micro-framework web *Flask* et personnaliser l'application





# Merci à vous

++