

# FORMATION DOCKER

**Créer et administrer vos conteneurs virtuels  
d'applications  
2 jours**

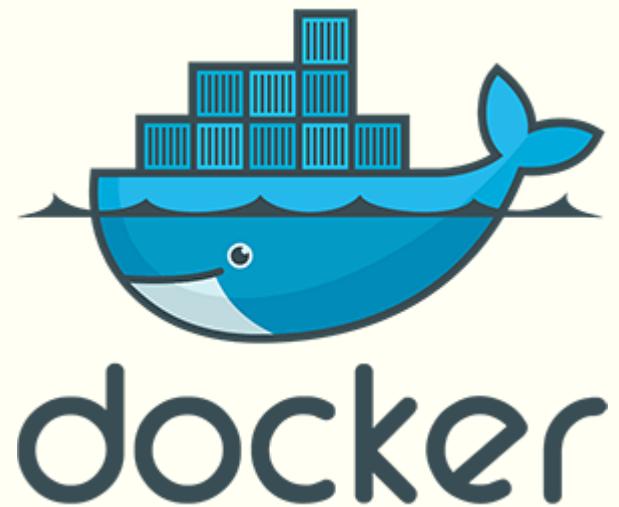


# Disposition de titre et de contenu avec liste

---

---

- **Module 1: Introduction à Docker**
- **Module 2: Installation et prise en main**
- **Module 3: Cycle de vie d'un container**
- **Module 4: Création d'images**
- **Module 5: Volume**
- **Module 7: Docker Compose**



# MODULE 1

**Introduction à Docker**

# Introduction à Docker

---

---

- De la virtualisation à Docker
- Les différents types de virtualisation.
- La conteneurisation : LXC, namespaces, control-groups.
- Docker versus virtualisation.
- L'architecture de Docker
- Présentation des différents composants

# De la virtualisation à Docker

---

*Build, ship and run any app, anywhere*

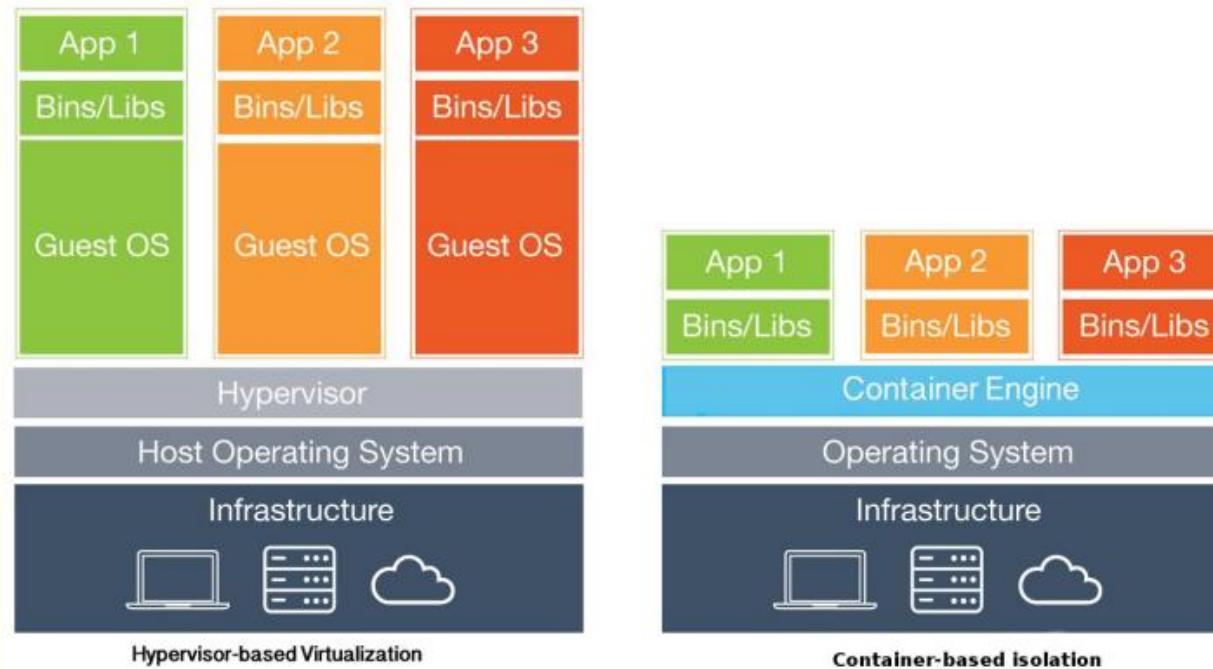
sous cette devise, la plateforme de conteneurs (environnements) open source Docker promeut une alternative flexible et économique en ressources à la création de composants matériels basés sur des machines virtuelles (VM).

# Les différents types de virtualisation

---

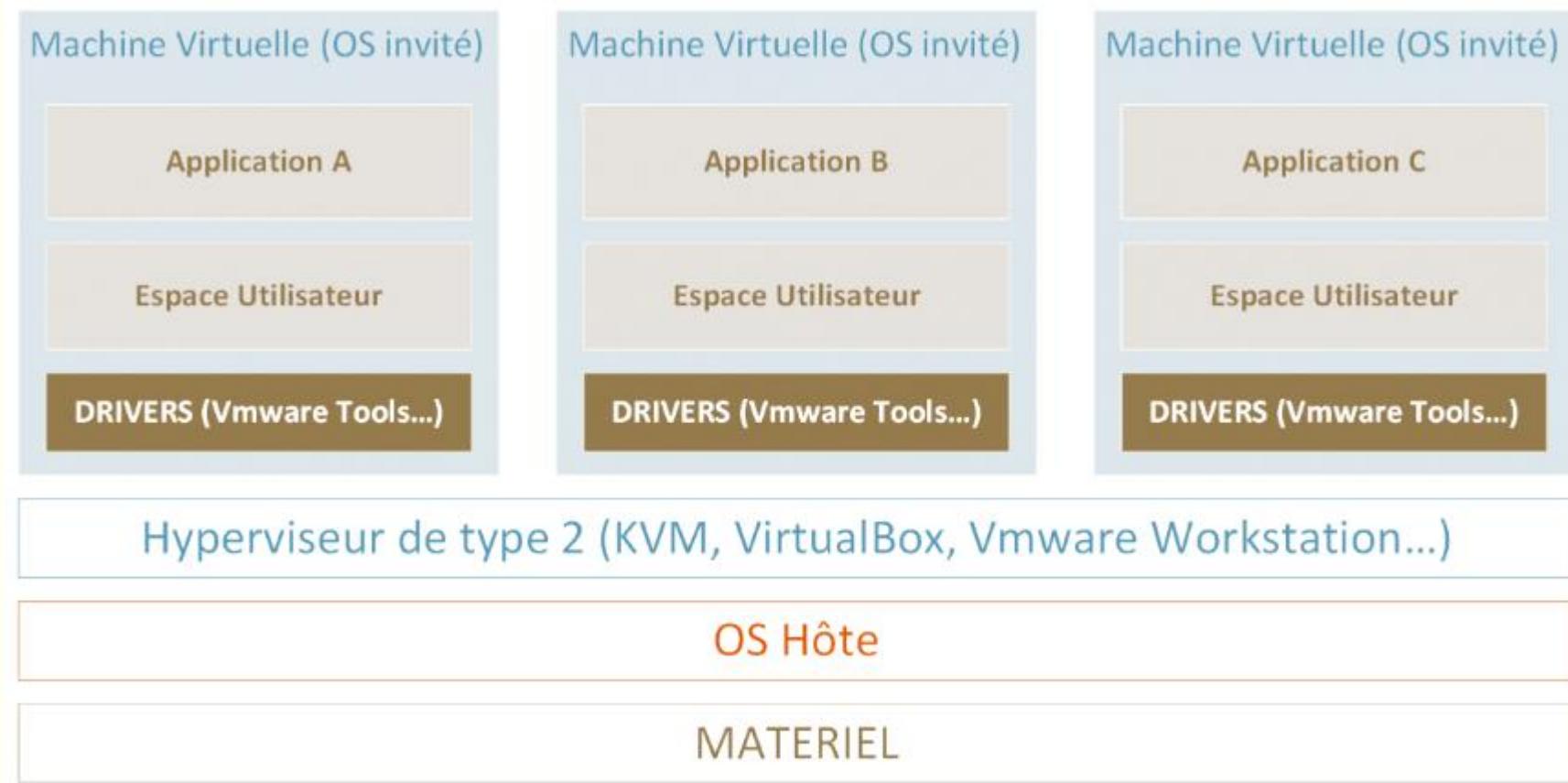
Deux techniques de virtualisation

- la virtualisation matérielle traditionnelle
- La virtualisation basée sur les conteneurs



# La virtualisation par hyperviseur

---



# La virtualisation par hyperviseur

---

- Avec la virtualisation par hyperviseur de type 2 :
  - ✓ VMware Workstation,
  - ✓ VirtualBox
  - ✓ ...
- chaque hôte invité virtualise un environnement complet, ce qui permet notamment de pouvoir exécuter un système virtualisé différent du système hôte (typiquement du Windows sur une Red Hat par exemple).
- la virtualisation matérielle traditionnelle repose sur le démarrage de plusieurs systèmes invités sur un système hôte commun,

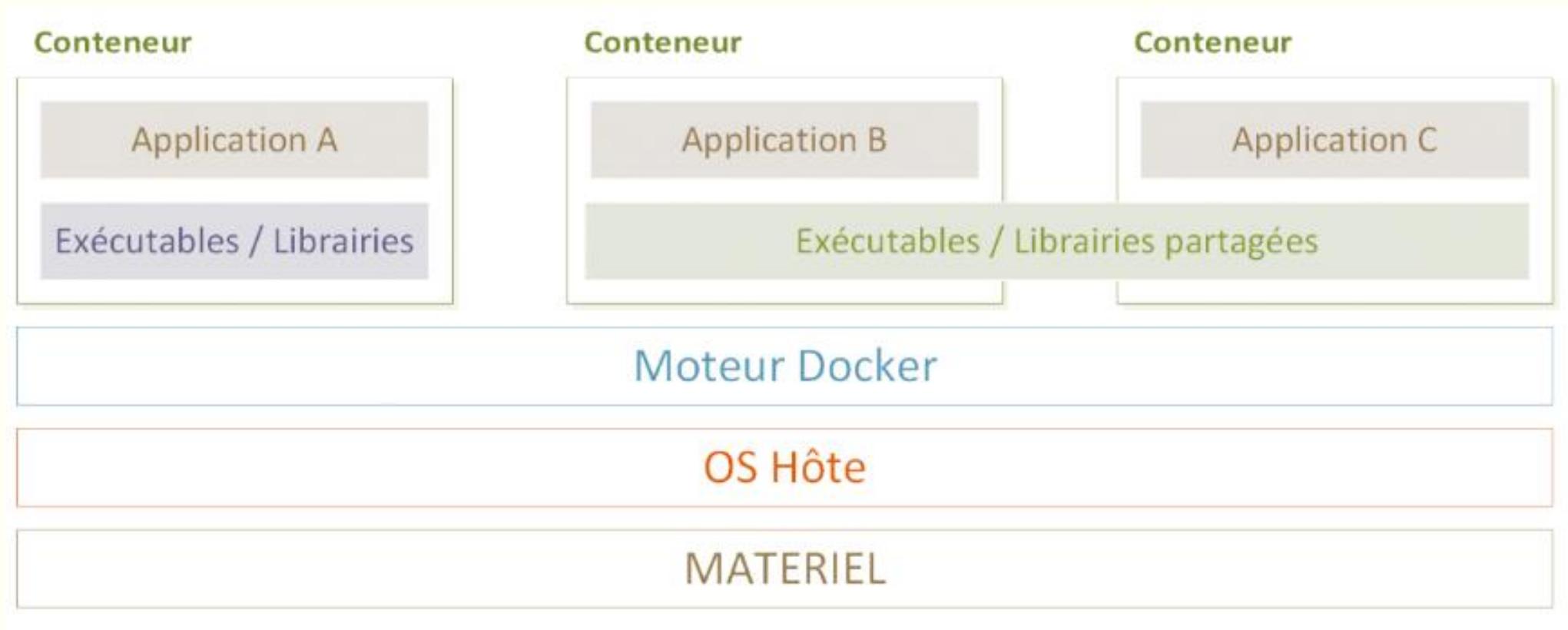
# La virtualisation par hyperviseur

---

- Si les applications sont encapsulées dans le contexte de la virtualisation matérielle classique, cela se fait à l'aide d'un hyperviseur.
- Il agit comme une couche abstraite entre le système hôte et les systèmes invités virtuels.
- Chaque système invité est implémenté comme une **machine complète avec un noyau de système d'exploitation séparé**.
- Les ressources matérielles du système hôte (CPU, mémoire, espace disque, périphériques disponibles) sont allouées proportionnellement par **l'hyperviseur**.

# La virtualisation par conteneur

---



# La virtualisation par conteneur

---

- La virtualisation par conteneur telle que Docker la propose permet à un système Linux de contenir un ou plusieurs processus dans un environnement d'exécution indépendant :
  - ✓ indépendant du système hôte
  - ✓ et des conteneurs entre eux.
- la virtualisation par conteneur est plus rapide que la virtualisation par hyperviseur car les conteneurs ont directement accès aux ressources matérielles de l'hôte.
- La virtualisation basée sur les conteneurs est donc également appelée **virtualisation au niveau du système d'exploitation**.

# La virtualisation par conteneur

---

- Avec la virtualisation par conteneurs, cependant, aucun système invité complet n'est simulé.
- À la place, les applications sont démarrées dans des conteneurs.
- Même s'ils partagent le même noyau, celui du système hôte, ils fonctionnent comme des processus isolés dans l'espace utilisateur (User Space).

# La virtualisation par conteneur

---

- Les systèmes d'exploitation modernes divisent la mémoire virtuelle en deux zones distinctes :
  - ✓ **espace noyau**
  - ✓ **et espace utilisateur.**
- Alors que **l'espace noyau est réservé exclusivement à l'exploitation** du noyau et d'autres composants centraux du système d'exploitation, **l'espace utilisateur** correspond à la zone mémoire **disponible pour les applications.**
- La stricte séparation entre le noyau et l'espace utilisateur est principalement utilisée pour protéger le système contre les applications malveillantes ou défectueuses.

# La conteneurisation : LXC, namespaces, control-groups

---

---

- Le grand avantage de la virtualisation basée sur des conteneurs est que les applications ayant des exigences différentes peuvent être exécutées indépendamment les unes des autres sans que l'overhead d'un système invité séparé doive être accepté.
- La technologie des conteneurs utilise deux fonctions de base du noyau Linux : **les groupes de contrôle (Cgroupes)** et **les espaces de noms de noyau**.

# La conteneurisation : Namespaces

---

- Les **namespaces** (espaces de noms) limitent un processus et ses processus fils à une section spécifique du système sous-jacent.
- Pour encapsuler les processus, Docker utilise des espaces de noms dans cinq zones différentes :
  - ✓ Système d'identification (UTS)
  - ✓ Identifiant de processus (PID)
  - ✓ Communication inter-processus (IPC) .
  - ✓ Ressources réseau (NET)
  - ✓ Point de montage des fichiers système (MNT)

# La conteneurisation : Namespaces

---

- **Système d'identification (UTS)** : dans la virtualisation basée sur les conteneurs, les espaces de noms UTS sont utilisés pour attribuer des conteneurs à leurs propres noms d'hôte et de domaine.
- **Identifiant de processus (PID)** : chaque conteneur Docker utilise son propre espace de noms pour les ID de processus. Les processus qui se déroulent à l'extérieur d'un conteneur ne sont pas visibles à l'intérieur du conteneur. Cela signifie que les processus encapsulés dans des conteneurs sur le même système hôte peuvent avoir le même PID sans conflit.
- **Communication inter-processus (IPC)** : les espaces de noms IPC isolent les processus dans un conteneur de sorte que la communication avec les processus à l'extérieur du conteneur soit empêchée.

# La conteneurisation : Namespaces

---

- **Ressources réseau (NET)** : les espaces de noms de réseau permettent d'attribuer à chaque conteneur des ressources réseau distinctes, telles que des adresses IP ou des tables de routage.
- **Point de montage des fichiers système (MNT)** : grâce aux espaces de noms de montage, un processus isolé ne voit jamais le système de fichiers entier de l'hôte, mais seulement une petite partie de celui-ci, généralement une image créée spécialement pour ce conteneur.

# La conteneurisation : control-groups

---

- **Les Cgroups** limitent l'accès aux ressources mémoire, CPU et E/S, ce qui empêche les besoins en ressources d'un processus d'affecter d'autres processus en cours d'exécution.

# La conteneurisation : LXC

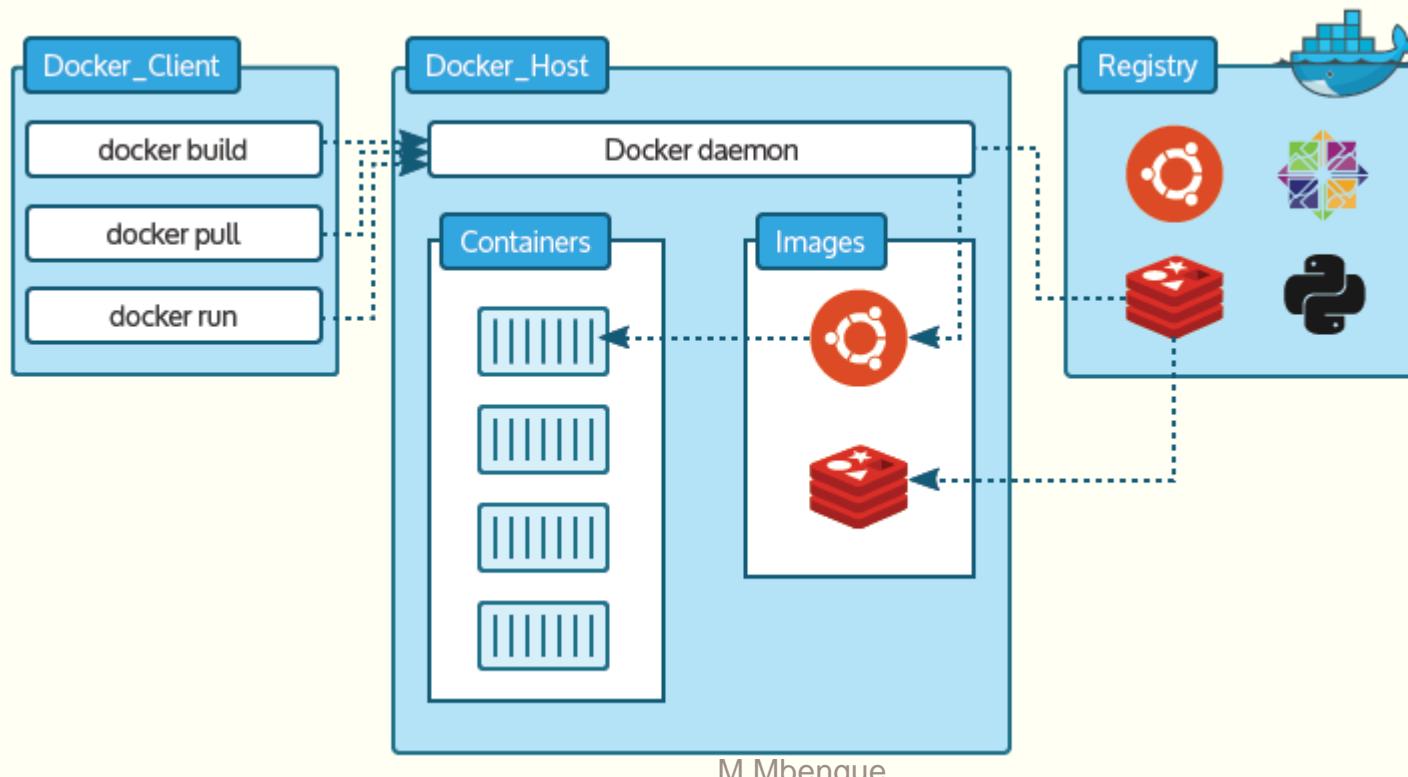
---

- Jusqu'à la version 0.8.1, l'isolation du processus Docker était basée sur des conteneurs Linux (LXC).
- Depuis la version 0.9, le format de conteneur Libcontainer est disponible pour les utilisateurs.
- Ceci permet à Docker d'être utilisé sur plusieurs plateformes et d'exécuter le même conteneur sur différents systèmes hôtes, mais a également permis d'offrir une version Docker pour Windows et MacOs.

# L'architecture de Docker

---

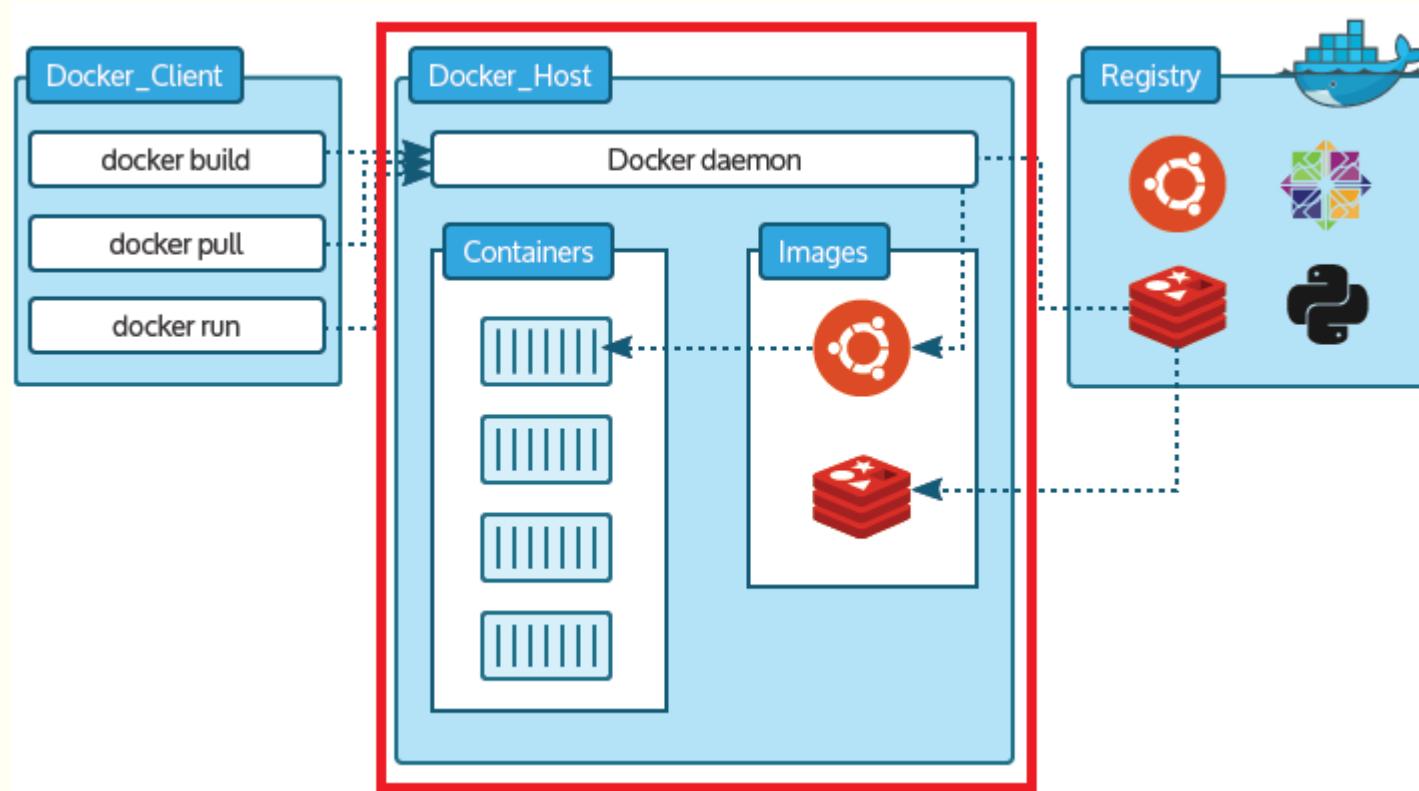
- Docker est une technologie utilisée pour le développement, l'exécution et le déploiement des applications packagées dans des containers.
- Elle utilise une architecture **client-serveur**.



# L'architecture de Docker

---

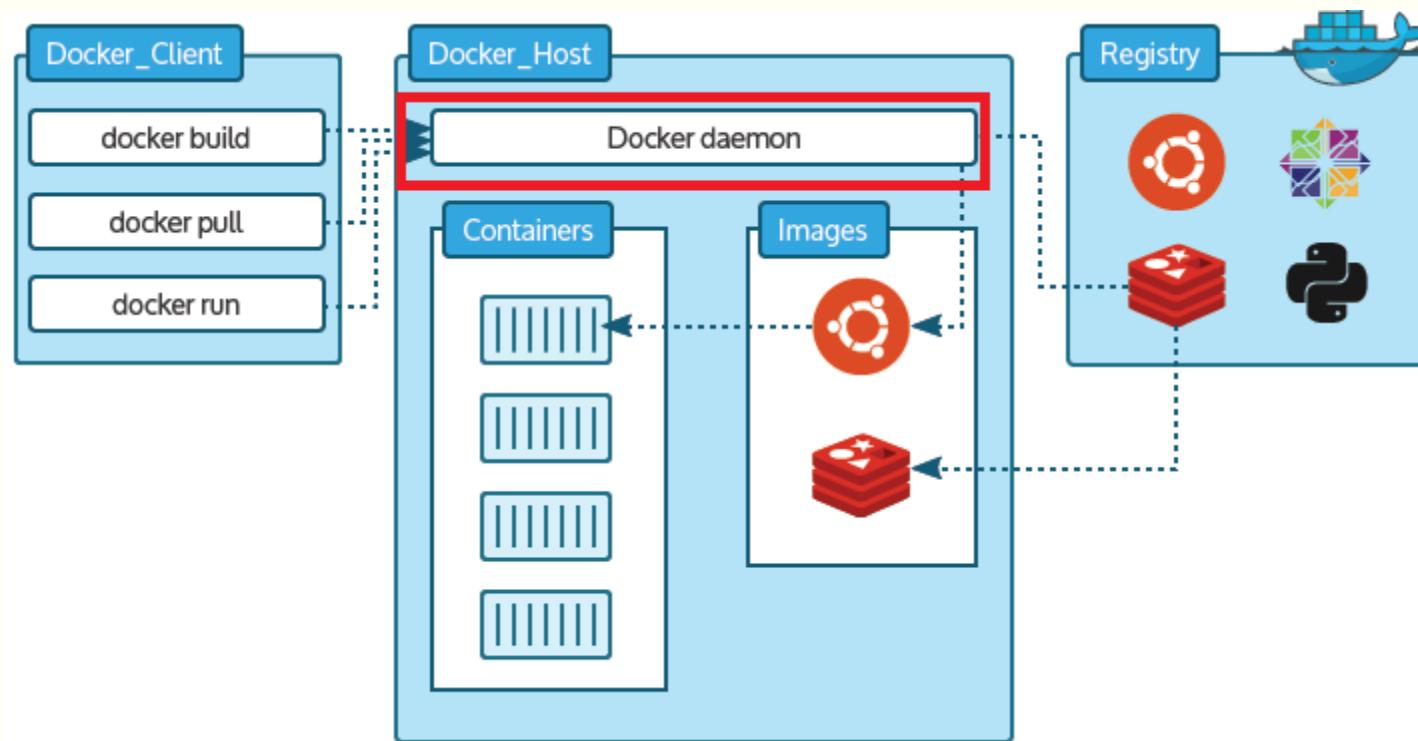
- Au centre, Le **Docker Host** représente la machine physique ou la machine virtuelle dans laquelle Docker Daemon et les containers sont déployés.



# L'architecture de Docker

---

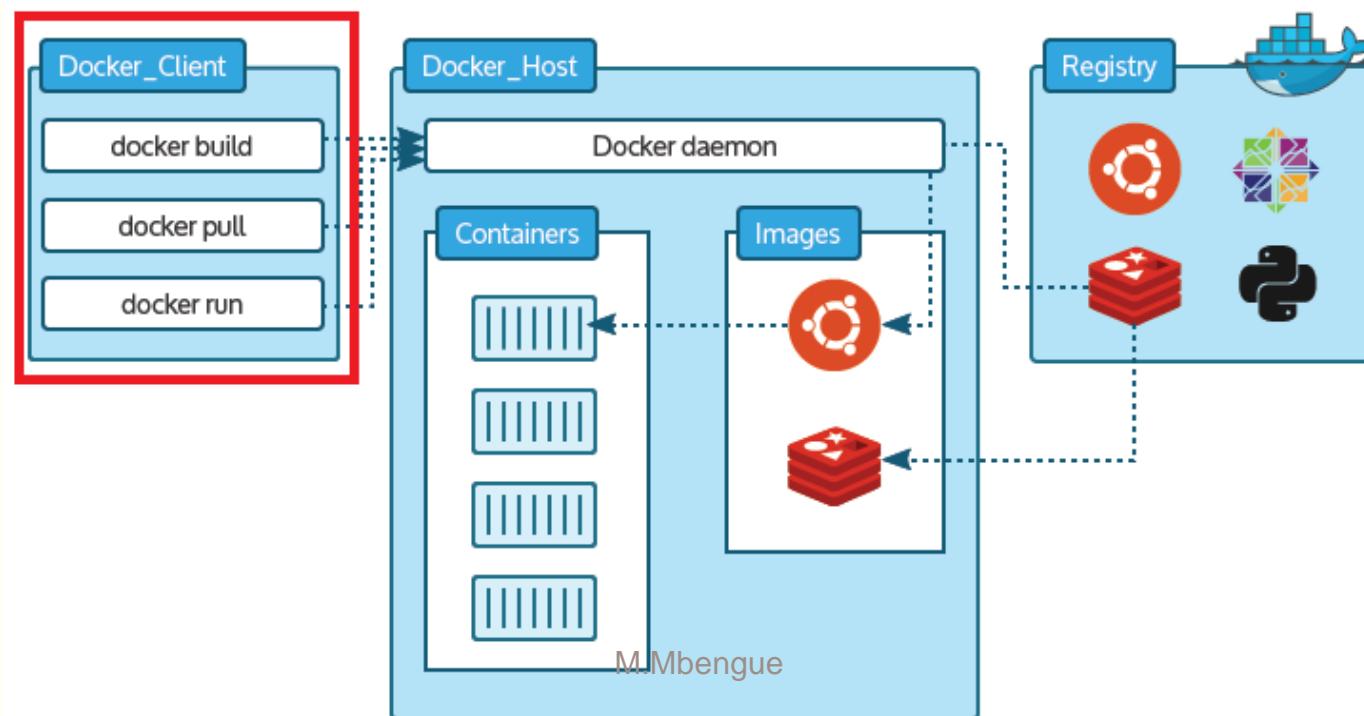
- Le **Docker Daemon** est à la fois responsable de la création, du démarrage et du monitoring des containers, mais aussi de la construction et du stockage des images. Le système d'exploitation a la charge de démarrer le **Docker Daemon**.



# L'architecture de Docker

---

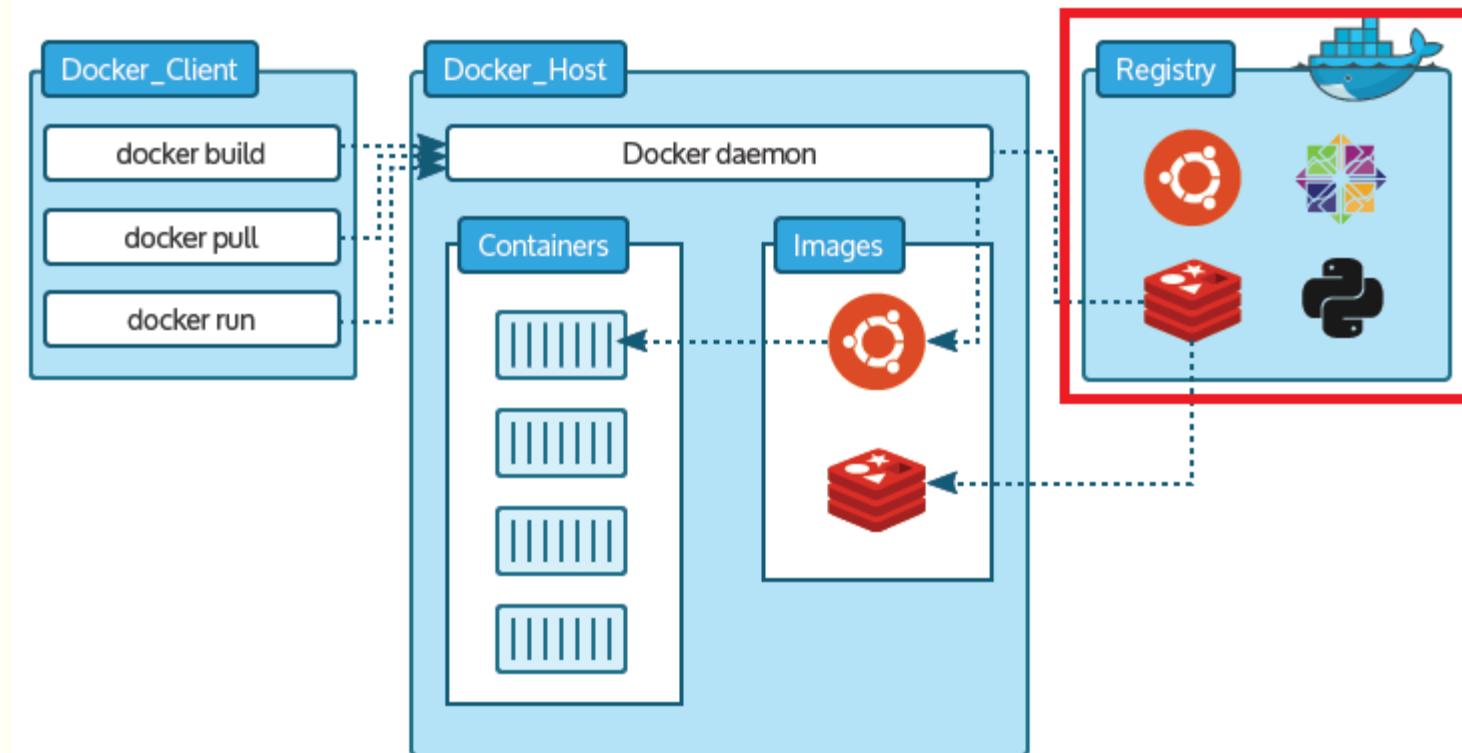
- Le **Docker Client** se trouve à gauche de l'image. Il communique avec le Docker Daemon via sockets par une API RESTful. L'objectif du **Docker Client** est de contrôler l'hôte, créer des images, publier, exécuter et gérer les containers correspondants à linstanciation de ces images. La communication via HTTP facilite le contrôle des connexions des Docker Daemons. **La combinaison des Docker Clients et des Docker Daemons est appelée Docker Engine**



# L'architecture de Docker

---

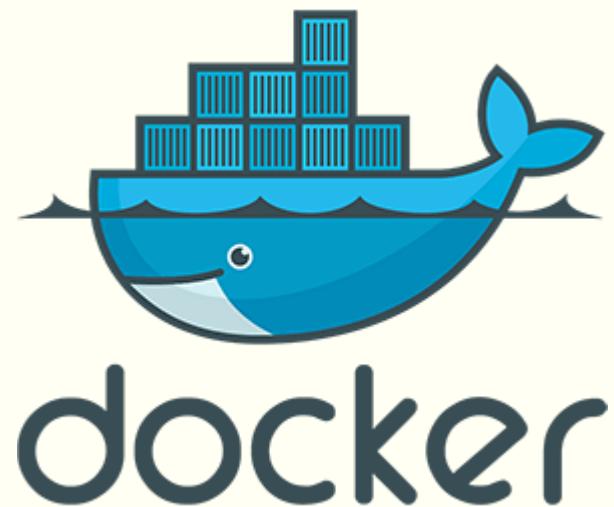
- Le **Docker Registry** à droite de l'image, permet de stocker et distribuer des images. Par défaut, le Registry est le Docker Hub, qui détient des milliers d'images publiques. les containers Docker sont créés en utilisant ces images de base.



# Disposition de titre et de contenu avec liste

---

**Question ?**



# MODULE 2

Installation et prise en main

# Présentation

---

---

- Installation sur les différentes plateformes
- Présentation de Docker for Mac et Docker for Windows
- Architecture client / daemon
- Lancement d'un container

# Editions Docker

---

---

Il y a deux éditions de Docker :

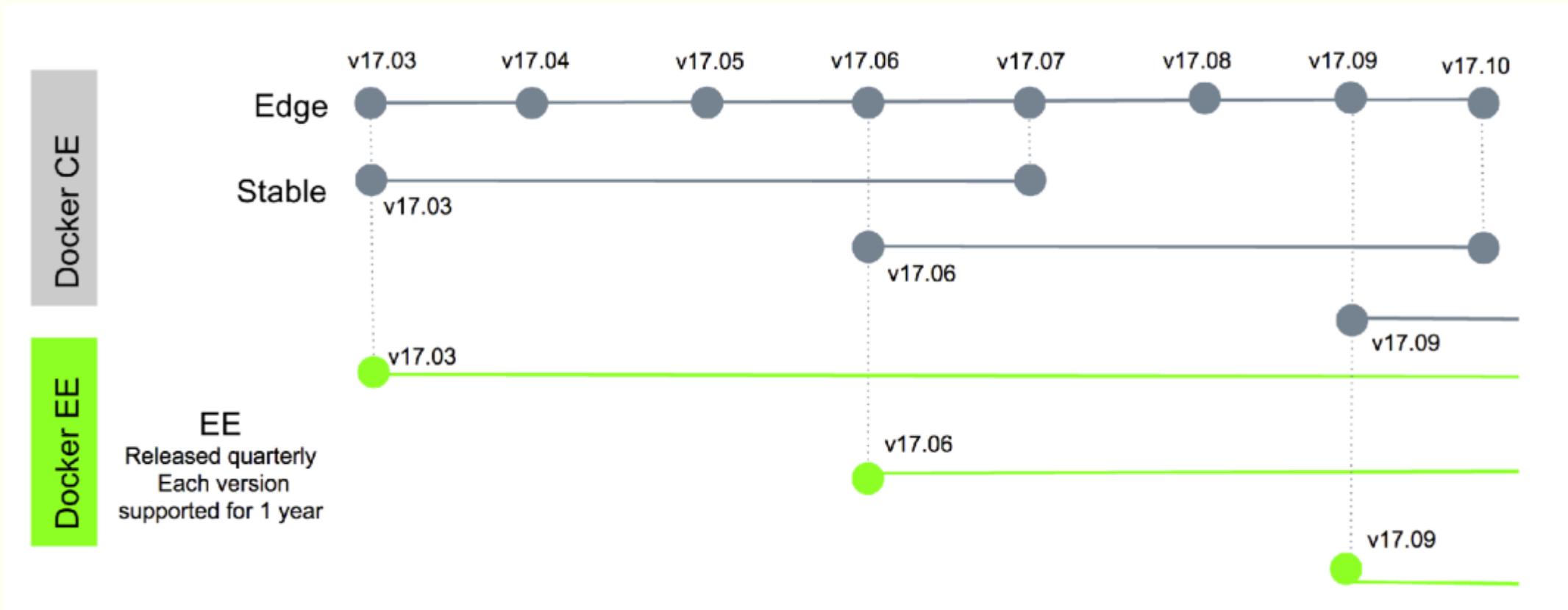
- **Docker CE** : Community Edition
- **Docker EE** : Enterprise Edition



# Version en MM.YY

---

---



# Installation de Docker Ubuntu (Linux)

---

Il existe trois façons différentes d'installer la plateforme de conteneurs Docker sur votre système Ubuntu en fonction des conditions générales et préalables :

- **Installation manuelle via le package DEB**
- **Installation à partir du dépôt Docker**
- **Installation aus dem Ubuntu-Repository**

# Installation de Docker Ubuntu (Linux)

---

## Configuration système

Pour installer la version actuelle de Docker sur votre distribution Ubuntu, vous avez besoin de la variante 64 bits de l'une des versions suivantes d'Ubuntu :

- Yakkety 16.10
- Xenial 16.04 (LTS)
- Trusty 14.04 (LTS)

# Installation de Docker Ubuntu (Linux)

---

---

- **Désinstaller Docker**

```
apt-get remove docker docker-engine docker.io
```

- **Mettre à jour les listes de paquets**

```
sudo apt-get update
```

- **Installation des paquets supplémentaires**

- **apt-get install \**

```
apt-transport-https \
```

```
ca-certificates \
```

```
curl \
```

```
software-properties-common
```

# Installation de Docker Ubuntu (Linux)

---

---

- **Ajoutez une clef GPG** : ajouter la clef GPG officielle de Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
```

- **Vérifier la clef GPG**

```
apt-key fingerprint 0EBFCD88
```

- **Configurer le dépôt Docker** : pour garantir un accès sécurisé au dépôt Docker, entrez la commande suivante :

```
sudo add-apt-repository \
```

```
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
```

```
$(lsb_release -cs) \
```

```
stable"
```

# Installation de Docker Ubuntu (Linux)

---

---

- Mettre à jour l'index des paquets

`sudo apt-get update`

- Installation du Docker à partir du référentiel

`sudo apt-get install docker-ce`

- Il ne nous reste plus qu'à lancer Docker :

`systemctl start docker`

`systemctl enable docker`

- Verify that Docker CE is installed correctly by running the hello-world image

`sudo docker run hello-world`

# Compatibilité avec Windows

---

- Windows 10 Pro
- Windows Server 2016 : version 1709 +

Microsoft docs (FR) :

- Moteur Docker sur Windows : [manage-docker/configure-docker-daemon](#)
- Conteneurs Windows 10 : [windowscontainers/quick-start/quick-start-windows-10](#)
- Conteneurs Windows Server : [windowscontainers/quick-start/quick-start-windows-server](#)

# Composants Docker (Windows)

---

- **Docker for Windows** comprend :
- **Docker Engine** : daemon, le moteur de base qui gérer les conteneurs.
- **Docker CLI client** : Command Line Interface client pour Windows
- **Docker Compose** : permets de définir (en YAML) et d'exécuter des applications docker multi-conteneurs
- **Docker Machine** : permets de provisionner plusieurs hôtes Docker distants sur différentes plateformes.
- **Kitematic** : racheté par Docker, permet de faciliter l'utilisation de Docker (Win/Mac) via une interface graphique.

# Activer la Virtualisation (Windows)

---

- Vérifiez que la virtualisation est activée sur votre PC :
- **Windows 10** : Ouvrir le **Gestionnaire des tâches > Onglet Performance > Processeur**

Utilisation	Vitesse	Vitesse de base :	3,40 GHz
10%	1,88 GHz	Sockets :	1
Processus	Threads	Cœurs :	4
193	2681	Processeurs logiques :	4
Durée de fonctionnement		Virtualisation :	Activé
0:08:31:06		Cache de niveau 1 :	256 Ko
		Cache de niveau 2 :	1,0 Mo
		Cache de niveau 3 :	6,0 Mo

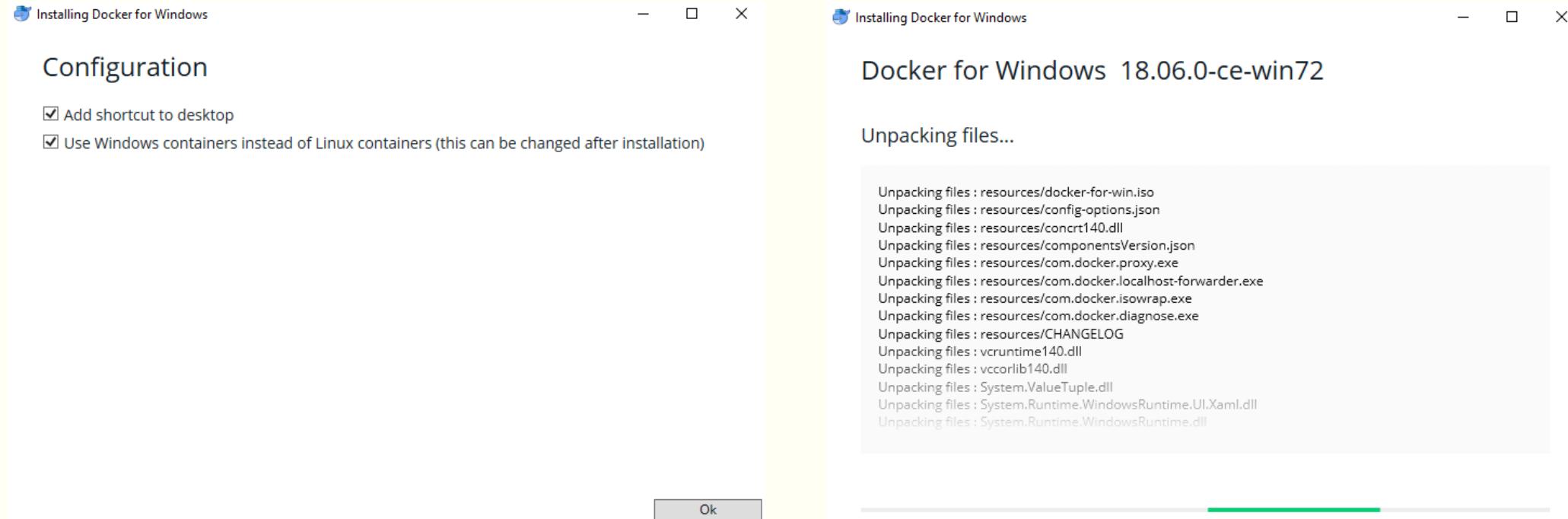
- Sinon, l'activer depuis le BIOS.

# Installation de Docker (Windows)

---

---

Lancez l'installation de Docker.

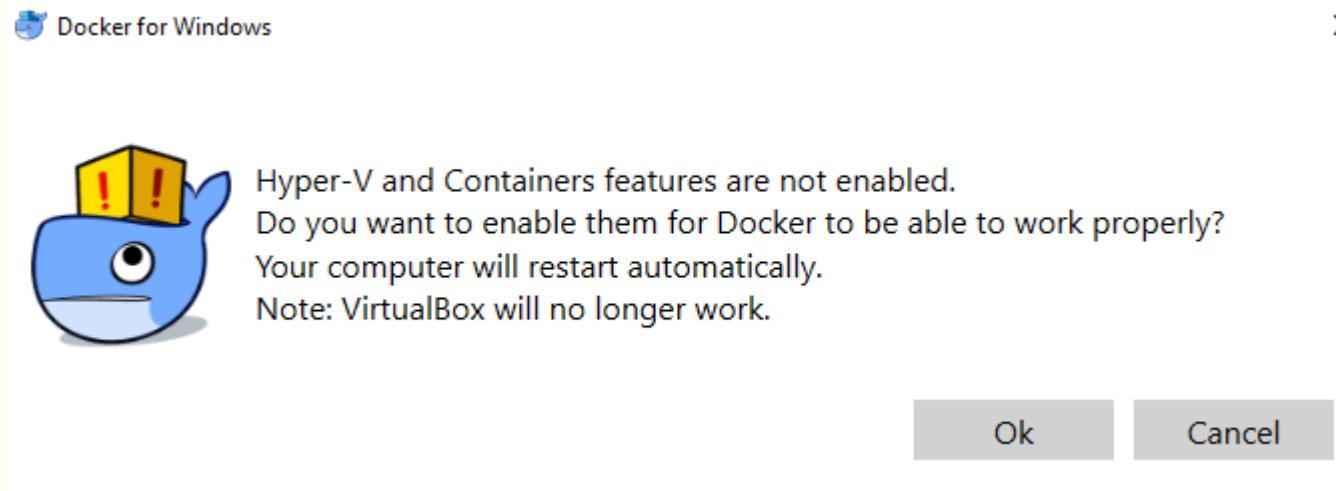


- Une fois terminé, vous devez redémarrer votre ordinateur.
- Cliquez sur **Close and log out**

# Installation de Docker (Windows)

---

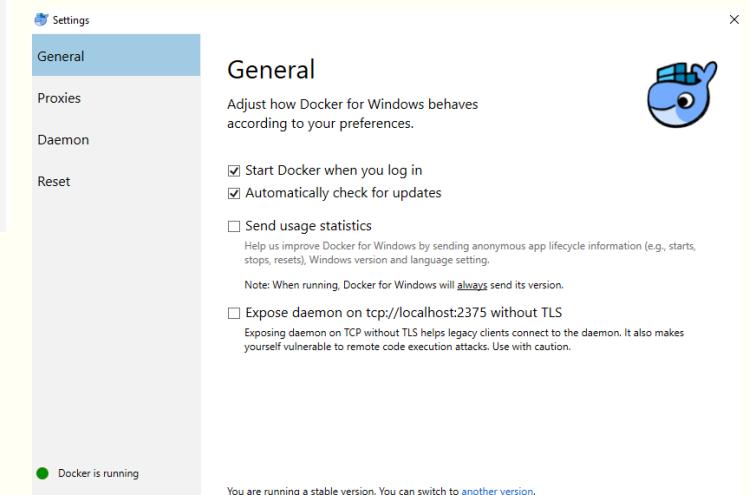
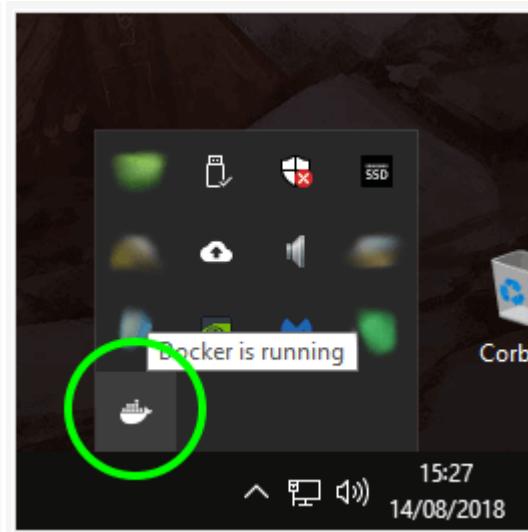
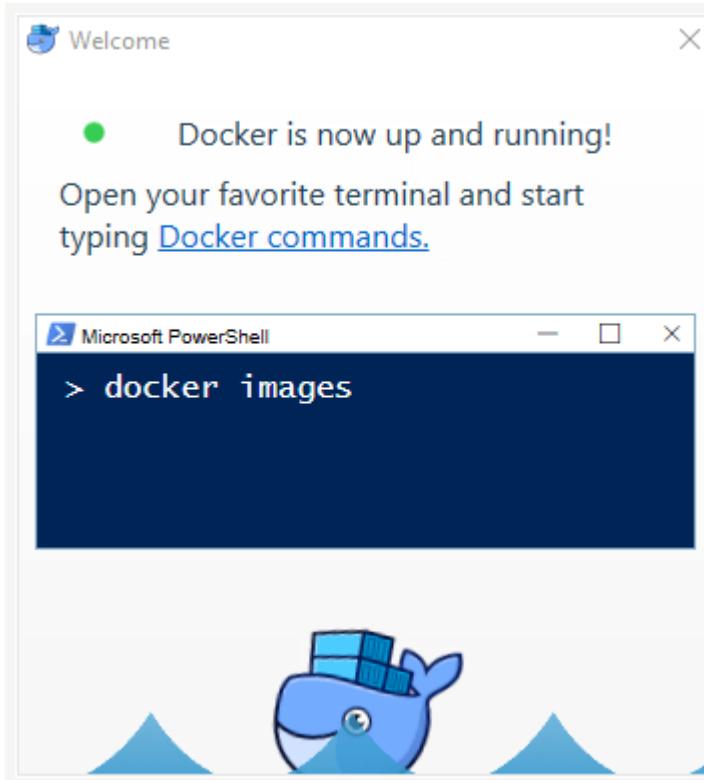
- Après le redémarrage de l'ordinateur, Docker va se lancer automatiquement.
- Ensuite, **un message s'affiche concernant les fonctionnalités Hyper-V**



- **Cliquez Ok pour activer les fonctionnalités Hyper-V**
- **Votre ordinateur va redémarrer automatiquement pour appliquer la mise à jour (Hyper-V)**

# Installation de Docker (Windows)

- Une fois l'ordinateur redémarré, une fenêtre Docker s'affiche et indique que **Docker est en cours et prêt.**



- Effectuez un **clic-droit** sur l'icône Docker
- Cliquez sur **Settings**

# Docker Windows CLI

---

---

- Testons le bon fonctionnement de Docker en ligne de commande. Lancez la console **Windows PowerShell**.
- Afficher la version client/serveur :

```
01. PS C:\WINDOWS\system32> docker version
02.
03. Client:
04.   Version:      18.06.0-ce
05.   API version: 1.38
06.   Go version:   go1.10.3
07.   Git commit:   Offa825
08.   Built:        Wed Jul 18 19:05:28 2018
09.   OS/Arch:      windows/amd64
10.   Experimental: false
11.
12. Server:
13.   Engine:
14.     Version:      18.06.0-ce
15.     API version: 1.38 (minimum version 1.24)
16.     Go version:   go1.10.3
17.     Git commit:   Offa825
18.     Built:        Wed Jul 18 19:23:19 2018
19.     OS/Arch:      windows/amd64
20.     Experimental: false
21.
22. PS C:\WINDOWS\system32>
```

# Installation de Docker (Mac)

---

---

- Docker CE pour Mac : <https://store.docker.com/editions/community/docker-ce-desktop-mac?tab=description>
- L'installation est réellement très simple :



# Installation de Docker (Mac)

---

---



# Installation de Docker (Mac)

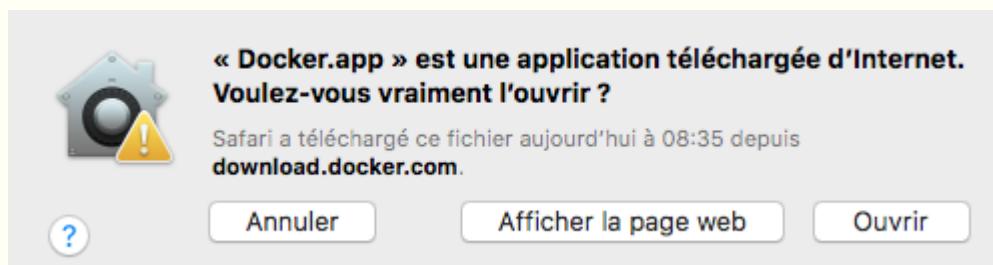
---

---

- La version de mon Mac :



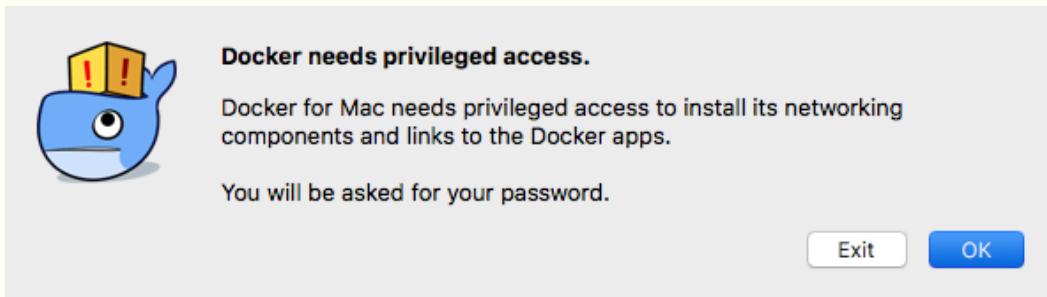
- Vu que l'application n'est pas sur l'Apple Store on a ce message :



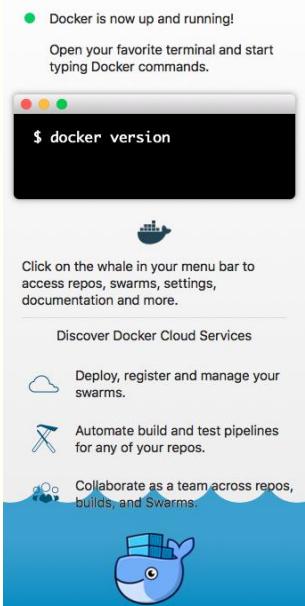
# Installation de Docker (Mac)

---

- Ensuite on doit donner les droits :



- Et c'est fini :



```
MacBook-Pro-de-XXXX:~ XXXXX$ docker version
Client:
  Version: 17.03.0-ce
  API version: 1.26
  Go version: go1.7.5
  Git commit: 60ccb22
  Built: Thu Feb 23 10:40:59 2017
  OS/Arch: darwin/amd64

Server:
  Version: 17.03.0-ce
  API version: 1.26 (minimum version 1.12)
  Go version: go1.7.5
  Git commit: 3a232c8
  Built: Tue Feb 28 07:52:04 2017
  OS/Arch: linux/amd64
  Experimental: true
```

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

TP

Installation de VMware

TP

Installation de Ubuntu

TP

Installation de docker-ce



# MODULE 3

**Cycle de vie d'un container**

# Présentation

---

---

- Les commandes de bases
- Les principales commandes pour la gestion d'un container
- Création de containers
- Création de containers: mode interactif
- Création de containers: foreground / background
- Création de containers: mapping de port
- Les commandes de base: inspect
- Les commandes de base: logs
- Les commandes de base: exec
- Les commandes de base: stop / rm

# Unification et amélioration de la CLI

---

---

- Avant Docker 1.13
  - **docker ps** : pour afficher la liste des conteneurs
  - **docker images** : la liste des images
  - **docker volume ls** : liste des volumes
- Avec Docker 1.13
  - **docker container ls**
  - **docker container run**

# Les commandes de bases

---

---

- En exécutant la commande `docker help`

```
Management Commands:
checkpoint  Manage checkpoints
container   Manage containers
image       Manage images
network     Manage networks
node        Manage Swarm nodes
plugin      Manage plugins
secret      Manage Docker secrets
service     Manage services
stack       Manage Docker stacks
swarm       Manage Swarm
system      Manage Docker
volume      Manage volumes
```

Vous pourrez voir les différentes  
“management commands” mais  
également les “commands” historiques.

```
Commands:
attach      Attach local standard input, output, and error streams to a running container
build       Build an image from a Dockerfile
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local filesystem
create      Create a new container
diff        Inspect changes to files or directories on a container's filesystem
events     Get real time events from the server
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
history    Show the history of an image
images     List images
import      Import the contents from a tarball to create a filesystem image
info        Display system-wide information
inspect    Return low-level information on Docker objects
kill        Kill one or more running containers
load        Load an image from a tar archive or STDIN
login      Log in to a Docker registry
logout     Log out from a Docker registry
logs       Fetch the logs of a container
pause      Pause all processes within one or more containers
port       List port mappings or a specific mapping for the container
ps          List containers
pull       Pull an image or a repository from a registry
push       Push an image or a repository to a registry
rename    Rename a container
restart   Restart one or more containers
rm         Remove one or more containers
rmi       Remove one or more images
run        Run a command in a new container
save       Save one or more images to a tar archive (streamed to STDOUT by default)
search    Search the Docker Hub for images
start     Start one or more stopped containers
stats     Display a live stream of container(s) resource usage statistics
stop      Stop one or more running containers
tag        Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top        Display the running processes of a container
unpause   Unpause all processes within one or more containers
update   Update configuration of one or more containers
version  Show the Docker version information
wait      Block until one or more containers stop, then print their exit codes
```

# Les commandes : docker container

---

```
master@ubuntu:~$ docker container --help

Usage: docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect    Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs       Fetch the logs of a container
  ls          List containers
  pause      Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune      Remove all stopped containers
  rename     Rename a container
  restart    Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start      Start one or more stopped containers
  stats      Display a live stream of container(s) resource usage statistics
  stop       Stop one or more running containers
  top        Display the running processes of a container
  unpause   Unpause all processes within one or more containers
  update     Update configuration of one or more containers
  wait       Block until one or more containers stop, then print their exit codes
```

# Les commandes : docker container les plus utiles

---

- run: création d'un container
- ls: liste des containers
- inspect: détails d'un container
- logs: visualisation des logs d'un container
- exec: lancement d'un processus
- stop: arrêt d'un container
- rm: suppression d'un container

# Création de containers

---

## docker container run hello-world

```
master@ubuntu:~$ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adb7777e9aacf18357296e799f81cabcfde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
master@ubuntu:~$
```

# Disposition de titre et de contenu avec liste

---

TP

Création de containers

# Création de containers: mode interactif

---

- Permet d'avoir accès à un shell interactif
- 2 option à utiliser
  - -t pour l'allocation d'un pseudo TTY
  - -i pour garder STDIN ouvert

**docker container run -it ubuntu**

```
master@ubuntu:~$ docker container run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
dal315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db9489744ecc9b4f50c7fa671f23c49
Status: Downloaded newer image for ubuntu:latest
root@274f08f0669d:/# █
```

# Disposition de titre et de contenu avec liste

---

---

TP

Création de containers:  
mode interactif

# Création de containers: foreground / background

---

- En foreground par default
- Option –d pour lancer un container en background
- Retourne l'identifiant du container

Création d'un container basé sur  
l'image nginx en foreground

```
$ docker container run nginx  
<process hangs on>
```

Création d'un container basé sur  
l'image nginx en background

```
$ docker container run -d nginx  
6f6cac745aa19c01015906480586294f9cc23cd7d1733552a50999a93aef5555
```

Création d'un container basé sur  
l'image alpine en tâche en  
background

```
$ docker container run -d alpine ping 8.8.8.8  
6df4108911d37c357ee6b4928b678f1c996628d336c92617448ffe98a8d0b146
```

# Disposition de titre et de contenu avec liste

---

---

TP

Création de containers en  
foreground / background

# Création de containers: mapping de port

---

- Pour être accessible depuis l'extérieur, un container peut publier un port sur la machine hôte
- Port statique: *-p HOST\_PORT:CONTAINER\_PORT*
- Port dynamique: *-P CONTAINER\_PORT*
- Conflit si plusieurs containers utilisent le même port de la machine hôte

# Création de containers: mapping de port

---

```
// Le port 80 de l'instance Nginx tournant dans le container est publié sur le port 8080 de l'hôte  
$ docker container run -d -p 8080:80 nginx
```



# Disposition de titre et de contenu avec liste

---

---

TP

**Création d'un container en exposant un port sur la machine hôte**

# Les commandes de base: ls

---

```
$ docker container ls      # Liste les containers actifs
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS               NAMES
5dbae351233f        nginx      "nginx -g 'daemon ..."   10 seconds ago    Up 9 seconds      0.0.0.0:8080->80/tcp   goofy_mestorf
6df4108911d3        alpine     "ping 8.8.8.8"          54 seconds ago    Up 54 seconds
6f6cac745aa1        nginx      "nginx -g 'daemon ..."   About a minute ago Up About a minute 80/tcp      priceless_turing

$ docker container ls -a    # Liste les containers actifs et stoppés
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS               NAMES
5dbae351233f        nginx      "nginx -g 'daemon ..."   2 minutes ago     Up 2 minutes      0.0.0.0:8080->80/tcp   goofy_mestorf
6df4108911d3        alpine     "ping 8.8.8.8"          3 minutes ago     Up 3 minutes
6f6cac745aa1        nginx      "nginx -g 'daemon ..."   3 minutes ago     Up 3 minutes      80/tcp               priceless_turing
47016eee384e        nginx      "nginx -g 'daemon ..."   3 minutes ago     Exited (0) 3 minutes ago  fervent_visvesvaraya
8653e0b2dd45        ubuntu     "/bin/bash"            4 minutes ago     Exited (0) 4 minutes ago  condescending_rosalind
16c68c2264c1        alpine     "echo hello"           4 minutes ago     Exited (0) 4 minutes ago  nostalgic_banach
3c1c4b0b474c        alpine     "echo hello"           4 minutes ago     Exited (0) 4 minutes ago  elastic_varahamihira

$ docker container ls -q    # Liste les IDs des containers actifs
5dbae351233f
6df4108911d3
6f6cac745aa1
```

# Disposition de titre et de contenu avec liste

---

---

TP

Les commandes de base : ls

# Les commandes de base: inspect

---

- Vue détaillée d'un container
- Disponible sur chacune des primitives Docker

```
$ docker container inspect 6df4108911d3
[
  {
    "Id": "6df4108911d37c357ee6b4928b678f1c996628d336c92617448ffe98a8d0b146",
    "Path": "ping",
    "Args": [
      "8.8.8.8"
    ],
    "State": {...},
    "Image": "sha256:02674b9cb179d57c68b526733adf38b458bd31ba0abff0c2bf5ceca5bad72cd9",
    "HostConfig": {...},
    "GraphDriver": {...},
    "Config": {...},
    "NetworkSettings": {...},
    ...
  }
]
```

# Les commandes de base: inspect

---

- Go templates <https://docs.docker.com/engine/reference/commandline/inspect/>

```
$ docker container inspect --format '{{ .Id }}' 58039df1aa2f
58039df1aa2fa93df7cbd9fb0d8bdb44206cb7d9f76ab50271a3c5b6b1dc8303

$ docker container inspect --format '{{ .NetworkSettings.IPAddress }}' 6df4108911d3
172.17.0.3

$ docker container inspect --format '{{json .State }}' 58039df1aa2f | jq
{
  "Status": "running",
  "Running": true,
  "Paused": false,
  "Restarting": false,
  "OOMKilled": false,
  "Dead": false,
  "Pid": 9224,
  "ExitCode": 0,
  "Error": "",
  "StartedAt": "2017-05-22T13:49:18.425075377Z",
  "FinishedAt": "0001-01-01T00:00:00Z"
}
```

# Disposition de titre et de contenu avec liste

---

TP

Inspection d'un container

# Les commandes de base: logs

---

- Visualisation des logs d'un container
- Option -f pour la mise à jour automatique
- Pas de fichiers de logs dans un container
- Ecriture des logs sur la sortie / erreur standard

```
$ docker container logs -f 6df4108911d3
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=402 ttl=60 time=1.881 ms
64 bytes from 8.8.8.8: seq=403 ttl=60 time=1.939 ms
64 bytes from 8.8.8.8: seq=404 ttl=60 time=1.987 ms
64 bytes from 8.8.8.8: seq=405 ttl=60 time=1.821 ms
64 bytes from 8.8.8.8: seq=406 ttl=60 time=1.857 ms
64 bytes from 8.8.8.8: seq=407 ttl=60 time=2.482 ms
...
```

# Les commandes de base: exec

---

- Permet de lancer un processus dans un container existant
- Souvent utilisée avec les options `-t` et `-i` pour avoir un shell interactif
- Très utile pour faire du debug

```
# Exécution d'un shell dans un container actif
$ docker container exec -ti 6df4108911d3 sh
/ # ps aux
PID  USER      TIME  COMMAND
 1 root      0:00 ping 8.8.8.8
 5 root      0:00 sh
 9 root      0:00 ps aux
/ #
```

# Disposition de titre et de contenu avec liste

---

---

TP

Lancement d'un processus  
dans un container existant

# Les commandes de base: stop

---

- Stoppe un ou plusieurs containers
  - \$ docker container stop ID
  - \$ docker container stop \$(docker container ls -q)
- Les containers stoppés existent toujours
  - \$ docker container ls -a

```
$ docker container stop 6df4108911d3
6df4108911d3
```

```
$ docker container stop $(docker container ls -q)
5dbae351233f
6f6cac745aa1
```

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
```

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
5dbae351233f        nginx              "nginx -g 'daemon ..."   32 minutes ago   Exited (0) 21 seconds ago
6df4108911d3        alpine              "ping 8.8.8.8"       33 minutes ago   Exited (137) 57 seconds ago
6f6cac745aa1        nginx              "nginx -g 'daemon ..."   33 minutes ago   Exited (0) 21 seconds ago
...                  M.Mbengue
```

# Les commandes de base: rm

---

- Supprime définitivement un ou plusieurs container
  - \$ docker container rm ID
  - \$ docker container rm \$(docker container ls -aq)
- Option -f si le container n'est pas stoppé

```
$ docker container rm 6df4108911d3  
6df4108911d3
```

```
$ docker container rm -f $(docker container ls -aq)  
5dbae351233f  
6f6cac745aa1  
47016eeee384e  
8653e0b2dd45  
16c68c2264c1  
3c1c4b0b474c
```

```
$ docker container ls -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

# Disposition de titre et de contenu avec liste

---

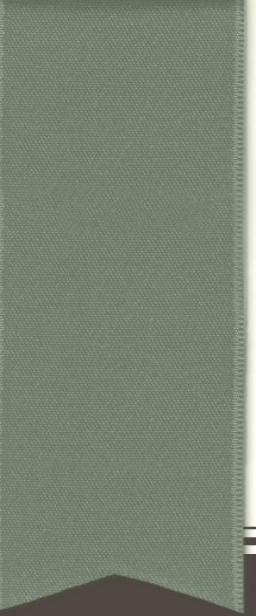
TP

Arrêt et suppression de  
tous les containers existant

# Disposition de titre et de contenu avec liste

---

**Question ?**



# MODULE 4

Création d'images

# Présentation

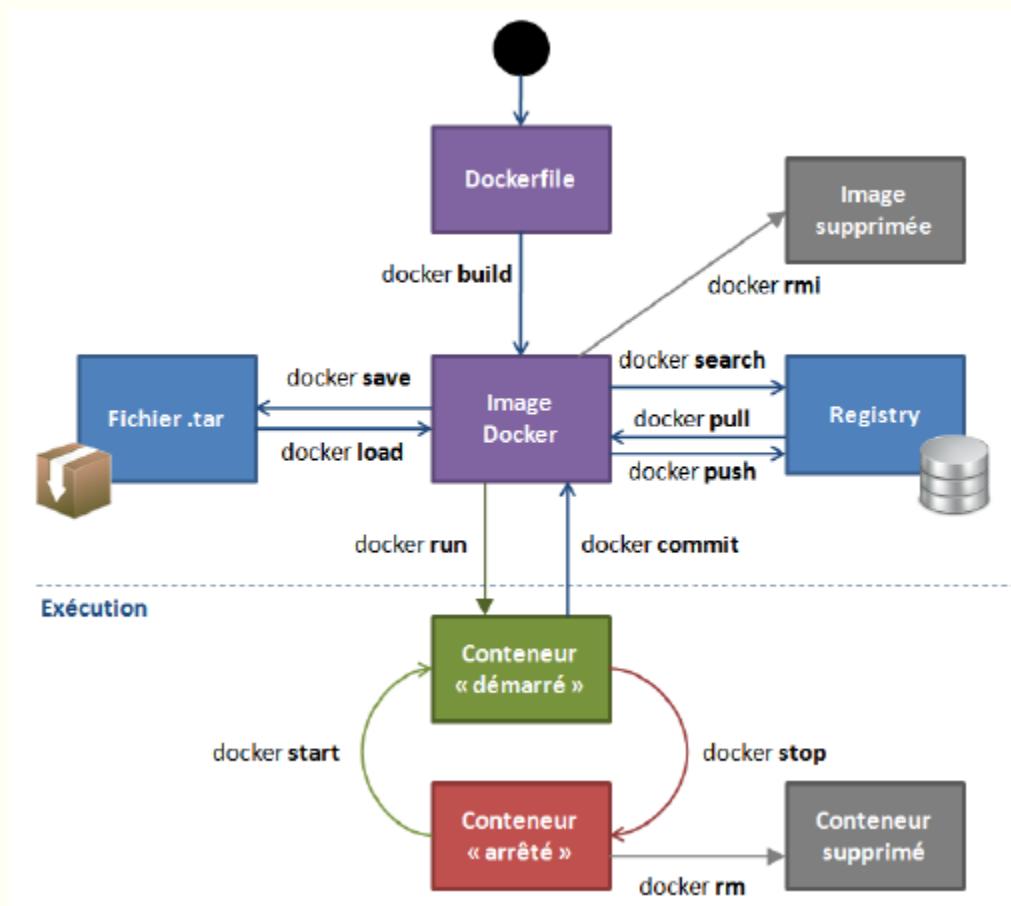
---

---

- Anatomie d'une image
- Le Docker Hub
- Pointer sur un registre donné
- **Méthodes de création d'une image**
- Création d'images personnalisées
- Le fichier Dockerfile en détails
- Importance de la prise en compte du cache

# Docker: Vue Globale

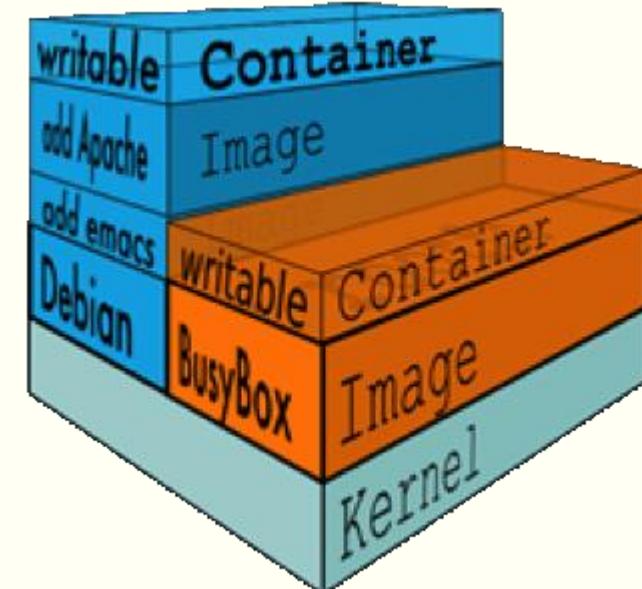
---



# Définition : Image

---

- Un Template pour instancier des containers
- Système de fichiers d'un container (rootfs)
- Composée d'une ou de plusieurs layers en lecture seule
- Chaque layer
  - Contient un système de fichiers et des métadonnées
  - Référence une autre layer
  - Est indépendante
  - Est réutilisable par d'autres images
- Une couche read-write créée pour chaque container

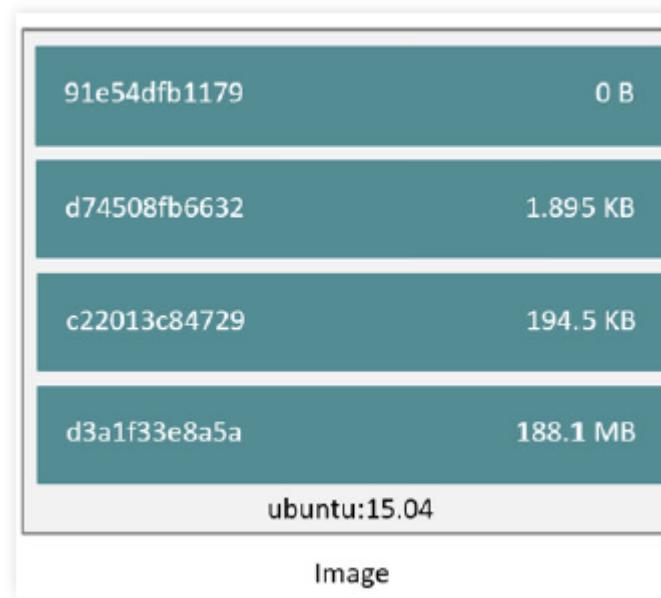


# LAYERS

---

---

- Les conteneurs et leurs images sont décomposés en couches (layers)
- Les layers peuvent être réutilisés entre différents conteneurs
- Gestion optimisée de l'espace disque.

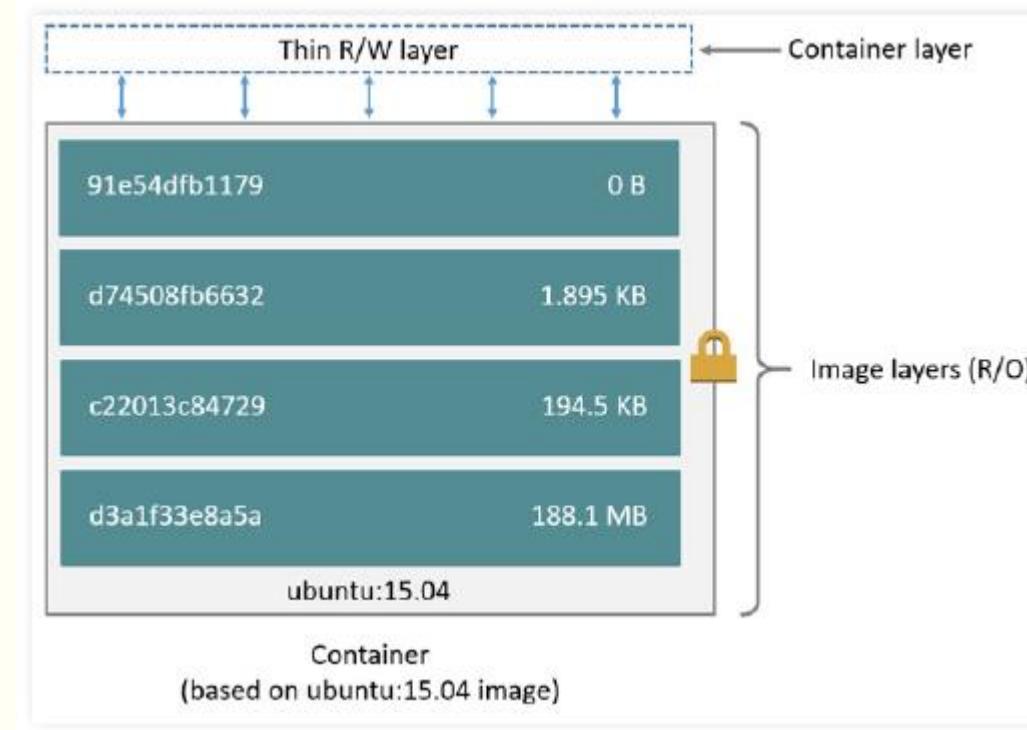


Une image se décompose en layers

M.Mbengue

# LAYERS : UN CONTENEUR

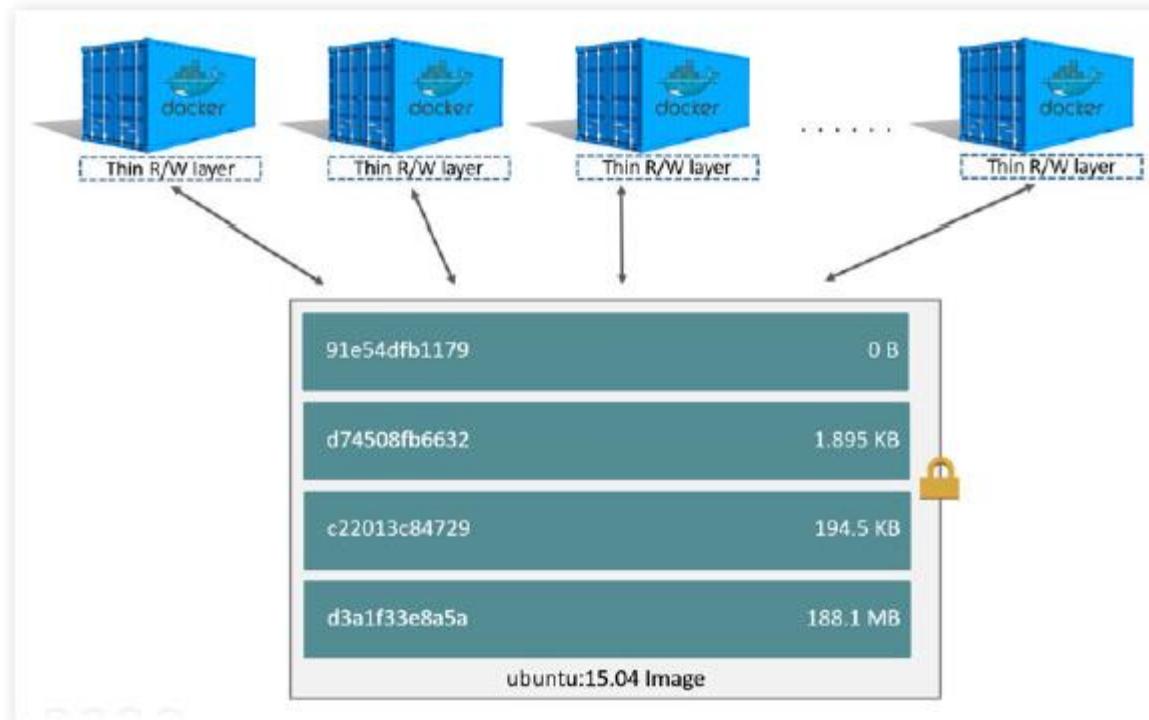
---



**Une conteneur = une image + un layer r/w**

# LAYERS : PLUSIEURS CONTENEURS

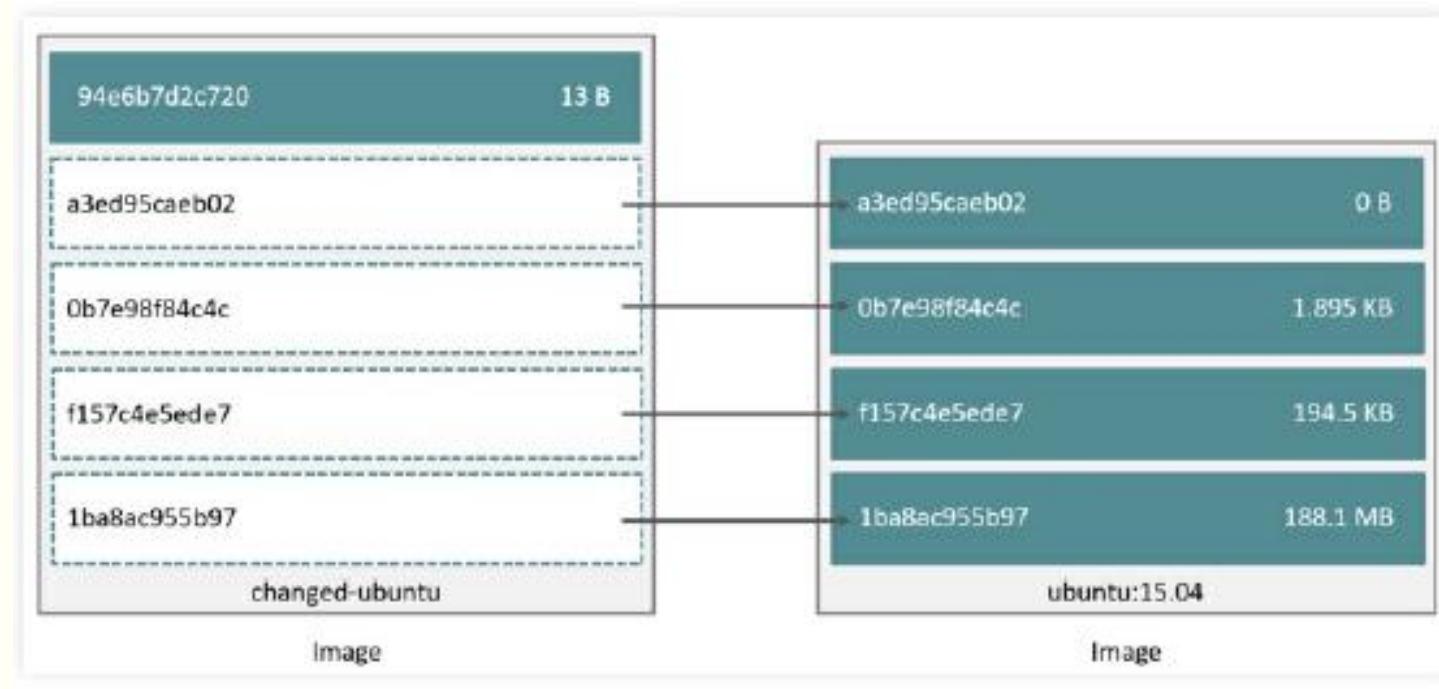
---



Une image, plusieurs conteneurs

# LAYERS : RÉPÉTITION DES LAYERS

---



Les layers sont indépendants de l'image

# Union filesystem & Copy-On-Write

---

- Une image est représentée comme un graph de layers
- Utilisation d'un storage driver pour
  - Unifier l'ensemble des layer en un seul filesystem
  - Gérer le layer read-write liée au container
- Différents drivers disponible en fonction du cas d'usage
  - Aufs, devicemapper, btrfs, overlay /overlays2 / zfs
- Par défaut les images sont dans **/var/lib/docker** sur la machine hôte

# STOCKAGE

---

- STOCKAGE : AUFS
  - ✓ A unification filesystem
  - ✓ Stable : performance écriture moyenne
  - ✓ Non intégré dans le Kernel Linux (mainline)
- STOCKAGE : DEVICE MAPPER
  - Basé sur LVM
  - Thin Provisionning et snapshot
  - Intégré dans le Kernel Linux (mainline)
  - Stable : performance écriture moyenne
- STOCKAGE : OVERLAYFS
  - Successeur de AUFS
  - Performances accrues
  - Relativement stable mais moins éprouvé que AUFS ou Device Mapper

# Exemple

---

---

- Exemple d'image d'une application Node.js

Code applicatif

Dépendances applicatives (node\_modules)

Node.js runtime

Ubuntu

- Exemple d'image d'une application Java

Code applicatif (.war)

Dépenances applicatives (jars)

JBoss

Java JRE

Debian

# Méthodes de création d'une image

---

- Workflow
  - Effectuer des modifications dans un container
    - Installation de packages
    - ...
  - Commiter les changements dans une nouvelle image
    - \$ docker container commit ID NAME
    - Union des layers read-only de l'image initiale et de layer read-write du container
- Approche non recommandée

# Méthodes de création d'une image

---

- Utilisation de la commande *commit*

```
$ docker container run -ti ubuntu
root@fc663b785b07:/# ping
bash: ping: command not found

root@fc663b785b07:/# apt-get update && apt-get install -y iputils-ping && ping -c3 8.8.8.8
...
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=27.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=27.6 ms

C-P / C-Q

$ docker container commit fc663b785b07 myping
sha256:c0d582cf43da9339bde5882879d8f0087158149687bad09838852d20338ace6

$ docker container run -ti myping ping -c3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=27.2 ms
```

# Méthodes de création d'une image : avec Dockerfile

---

---

- Fichier texte
- Série d'instructions pour construire le système de fichier
- \$ docker image build .
- Approche recommandée

# Disposition de titre et de contenu avec liste

---

---

TP

Création d'une image à  
partir d'un container

# Dockerfile

---

- Permet la création d'image personnalisées
- Flow standard
  - utilisation d'une image de base
    - distribution linux: Alpine, Ubuntu, CentOS, ...
    - serveur applicatif
    - runtime: node, java, ...
    - base de données
  - ajout de librairies et utilitaires
  - ajout et compilation du code applicatif
- *\$ docker image build [OPTIONS] DOCKERFILE\_PATH*

# Dockerfile : instructions principales

---

- FROM: indique l'image de base à utiliser
- MAINTAINER: auteur / mainteneur de l'image créée
- RUN: exécution d'une commande dans l'image
- EXPOSE: exposition des ports de l'application
- COPY: copie des ressources depuis la machine locale vers l'image
- ENTRYPOINT: "wrapper" autour de l'application
- CMD: commande exécutée lors de linstanciation de limage
- VOLUME: définition d'un volume (données gérées en dehors de lunion filesystem)
- WORKDIR: répertoire de travail

<https://docs.docker.com/engine/reference/builder/#parser-directives>

# Dockerfile : exemple serveur web

---

- Création d'une image contenant un serveur web
  - basé sur nginx
  - Configuration spécifique

```
# Image de base
FROM nginx:1.11.5

# Copie d'un fichier de configuration
COPY nginx.conf /etc/nginx/nginx.conf
```

- \$ docker image build -t www .

# Dockerfile : exemple application node.js

---

```
# Image de base
FROM node:6.10.3

# Copie de la liste des dependances
COPY package.json /app/package.json

# Installation / compilation des dependances
RUN cd /app && npm install

# Copie du code applicatif
COPY . /app/

# Exposition du port HTTP
EXPOSE 80

# Positionnement du répertoire de travail
WORKDIR /app

# Commande executee au lancement d'un container
CMD [ "npm", "start" ]
```

udom

# Dockerfile : ENTRYPOINT / CMD

---

- Définition de la commande à exécuter lorsqu'un container est créé
- ENTRYPOINT: "wrapper" autour de l'application
- CMD: commande par défaut
- Concaténation de ENTRYPOINT et CMD
- 2 formats possible
  - Shell, ex: /bin/ping localhost
  - Exec, ex: ["ping", "localhost"] - Format recommandé

# Dockerfile : ENTRYPOINT / CMD

---

- ENTRYPOINT non spécifié
- CMD spécifiée en ligne de commande écrase celle définie dans le Dockerfile

```
FROM alpine
CMD ["ping", "localhost"]
```



```
$ docker image build -t entry:v1.0 .
```



```
$ docker container run entry:v1.0
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.093 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.152 ms
...
$ docker container run entry:v1.0 echo "hello"
hello
```

# Dockerfile : ENTRYPOINT / CMD

---

- ENTRYPOINT spécifie une commande de base
- CMD est utilisé comme un paramètre de ENTRYPOINT
- CMD spécifiée en ligne de commande écrase celle définie dans le Dockerfile

```
FROM alpine
ENTRYPOINT ["ping"]
CMD ["localhost"]
```



```
$ docker image build -t entry:v2.0 .
```



```
$ docker container run entry:v2.0
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.089 ms
...

```

```
$ docker container run entry:v2.0 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=37 time=0.440 ms
64 bytes from 8.8.8.8: seq=1 ttl=37 time=0.424 ms
...

```

# Disposition de titre et de contenu avec liste

---

---

TP

Création d'une image à  
partir d'un Dockerfile

# Exemple : Application java (1/2)

---

---

```
public class Main
{
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Main.java

```
FROM openjdk:7
COPY . /usr/src/myapp
WORKDIR /usr/src/myapp
RUN javac Main.java
CMD ["java", "Main"]
```

Dockerfile

## Exemple : Application java (2/2)

```
$ docker image build -t hellojava:v1.0 .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM openjdk:7
7: Pulling from library/openjdk
...
Status: Downloaded newer image for openjdk:7
--> 18e25fe931b5
Step 2 : COPY . /usr/src/myapp
--> a3392b5f0f5c
Removing intermediate container a52f5bc66513
Step 3 : WORKDIR /usr/src/myapp
--> Running in 0296cb78aff9
--> 0b39a33a6d46
Removing intermediate container 0296cb78aff9
Step 4 : RUN javac Main.java
--> Running in 8aed4e60f335
--> 9d64d1e1b699
Removing intermediate container 8aed4e60f335
Step 5 : CMD java Main
--> Running in 62b64f86536a
--> fae0940e69aa
Removing intermediate container 62b64f86536a
Successfully built fae0940e69aa
```

Contenu du répertoire courant envoyé au daemon

Une layer est créée pour chaque instruction du Dockerfile

Création d'un container à partir de l'image créée

```
$ docker container run hellojava:v1.0
Hello, World
```

# Exemple : Application Python (1/2)

---

```
# https://github.com/mmulqueen/pyStrich  
  
from pystrich.datamatrix import  
DataMatrixEncoder  
  
encoder = DataMatrixEncoder('Hello')  
print(encoder.get_ascii())
```

barcode.py

```
FROM python:3  
  
ADD barcode.py /  
  
RUN pip install pystrich  
  
CMD [ "python", "/barcode.py" ]
```

Dockerfile

# Exemple : Application Python (2/2)

---

```
$ docker image build -t barcode .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM python:3
--> 175259937daf
Step 2 : ADD barcode.py /
--> 4bb590da4ba1
Removing intermediate container 8bcc214fb2dd
Step 3 : RUN pip install pystrich
--> Running in ed48c05e8bcd
...
--> 25e8a17abb50
Removing intermediate container ed48c05e8bcd
Step 4 : CMD python /barcode.py
--> Running in afc4d6dccc7d
--> 3432b53e2391
Removing intermediate container afc4d6dccc7d
Successfully built 3432b53e2391
```

```
$ docker run barcode
Sending to Step 1: FFFFFFFF
--> f7525
Step 2: ADD barcode.py /
XXXXXXXXXX
XX XX XX XXXX XX XX XX XX XX XX XX XX XX
XXXX XX XX XXXX XXXX XXXXXX XX
XX XXXX XXXX XXXX XX XX
XX XX XXXX XX XXXX XXXXXX XXXX
XXXX XX XXXX XXXX XX XX
XXXX XX XXXXXX XXXX XXXXXX XXXX
XX XX XX XXXXXX XX XX XX XX
XX XX XXXX XX XXXX XXXXXX XX
XXXX XXXXXX XXXX XXXXXXXXXXXX XX
XX XXXXXX XX XX XXXXXXXXX
XX XX XX XX XX XXXXXXXXXXXX XX
XX XXXXXXXX XX XXXX XX XX XX
XX XX XXXX XX XXXX XX XX
XXXX XX XXXX XXXX XXXX XXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```



# Exemple : Application Node (1/2)

---

```
var express = require('express');
var util    = require('util');
var app = express();
app.get('/', function(req, res) {
  res.setHeader('Content-Type', 'text/plain');
  res.end(util.format("%s - %s", new Date(), 'Got
HTTP Get Request'));
});
app.listen(process.env.PORT || 80);
```

index.js

```
{
  "name": "testnode",
  "version": "0.0.1",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": { "express": "^4.14.0" }
}
```

package.json

```
# Image de base
FROM node:6.10.3

MAINTAINER Luc Juggery <luc.juggery@gmail.com>
ENV LAST_UPDATED 20161209T075500

# Copie de la liste des dependances
COPY package.json /app/package.json

# Installation / compilation des dependances
RUN cd /app && npm install

# Copie du code applicatif
COPY . /app/

WORKDIR /app

EXPOSE 80

CMD ["npm", "start"]
```

## Exemple : Application Node (2/2)

```
$ docker image build -t nodewww .
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM node:4.6.0
--> e8428963b85a
Step 2 : MAINTAINER luc.juggery@gmail.com
--> 14c334ec4cd2
Step 3 : ENV LAST_UPDATED 20160719T115500
--> c782340d6227
Step 4 : COPY package.json /tmp/package.json
--> bda526528eb1
Removing intermediate container c911301ee6f4
Step 5 : RUN cd /tmp && npm install
...
Step 9 : EXPOSE 80
--> Running in 942ad93daf5c
--> e1d1fc083a7e
Removing intermediate container 942ad93daf5c
Step 10 : CMD npm start
--> Running in 55c88ab159c4
--> 9fc84bc63648
Removing intermediate container 55c88ab159c4
Successfully built 9fc84bc63648
```



# Disposition de titre et de contenu avec liste

---

---

TP

Création d'un serveur pong  
en NodeJS

# Cache

---

---

- Mécanisme d'utilisation des layers déjà créées
  - accélère la création d'image
  - ex: permet d'éviter la recompilation des dépendances suite à une typo dans le source
- Invalidation du cache
  - modification d'une instruction dans le Dockerfile
    - ex: valeur d'une variable d'environnement
  - modifications de fichiers copiés dans l'image (instructions ADD / COPY)
  - utile pour forcer la création d'une nouvelle image

# Cache

---

- Chaque instruction utilise le cache créé lors du build précédent
- Création de l'image quasi immédiate

```
$ docker image build -t test .
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM node:4.6.0
---> e8428963b85a
Step 2 : MAINTAINER luc.juggery@gmail.com
---> Using cache
---> 14c334ec4cd2
Step 3 : COPY package.json /app/package.json
---> Using cache
---> 467ea8990046
Step 4: RUN cd /app && npm install
...
Step 8: EXPOSE 80
---> Using cache
---> e1d1fc083a7e
Step 9: CMD npm start
---> Using cache
---> 9fc84bc63648
Successfully built 9fc84bc63648
```

# Disposition de titre et de contenu avec liste

---

---

TP

**Observation de la prise en  
compte du cache**

# Image Cli

---

---

```
$ docker image --help

Usage:docker image COMMAND

Manage images

Options:
  --help  Print usage

Commands:
  build      Build an image from a Dockerfile
  history    Show the history of an image
  import     Import the contents from a tarball to create a filesystem image
  inspect    Display detailed information on one or more images
  load       Load an image from a tar archive or STDIN
  ls         List images
  prune     Remove unused images
  pull       Pull an image or a repository from a registry
  push       Push an image or a repository to a registry
  rm        Remove one or more images
  save      Save one or more images to a tar archive (streamed to STDOUT by default)
  tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

# Image Cli : build

---

- Création d'une image depuis un Dockerfile
- Principales options
  - -t : spécifie le tag de l'image
  - -f : spécifie le fichier à utiliser pour la construction (Dockerfile par défaut)
  - --no-cache : invalide le cache
- Le contexte du build (répertoire courant) est envoyé au daemon
  - Le contenu du fichier .dockerignore est ignoré du contexte (eg: node\_modules)

```
$ docker image build -t hellojava:v1.0 .
Sending build context to Docker daemon 3.072 kB
...
Successfully built fae0940e69aa
```

# Image Cli:pull

---

- Download une image depuis un registry (Docker Hub par défaut)
- Chaque layer de l'image est downloadée
- Image automatiquement downloadée lors de la création d'un container
- Format de nommage: USER/IMAGE:VERSION
  - \$ docker image pull mongo
  - \$ docker image pull mhart/alpine-node:6.9.4

```
$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
$ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
0a8490d0dfd3: Pull complete
Digest: sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Status: Downloaded newer image for alpine:latest

$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
alpine              latest   88e169ea8f46  2 weeks ago   3.98 MB
```

# Image Cli:push

---

- Upload une image dans un registry
- Une fois uploadée, l'image pourra être utilisée par les utilisateurs autorisés
- Login nécessaire pour uploader une image sur le Docker Hub

# Image Cli:inspect

---

---

```
$ docker image inspect alpine
[
  {
    "Id": "sha256:88e169ea8f46ff0d0df784b1b254a15ecfaf045aee1856dca1ec242fdd231ddd",
    "RepoTags": ["alpine:latest"],
    ...
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 3983615,
    "VirtualSize": 3983615,
    "GraphDriver": {
      "Name": "aufs",
      "Data": null
    },
    "RootFS": {
      "Type": "layers",
      "Layers": [
        "sha256:60ab55d3379d47c1ba6b6225d59d10e1f52096ee9d5c816e42c635ccc57a5a2b"
      ]
    }
  }
]

$ docker image inspect -f '{{ .Architecture }}' alpine
Amd64

$ docker inspect --format '{{ .ContainerConfig.Cmd }}' mongo:3.2
[/bin/sh -c #(nop)  CMD ["mongod"]]
```

# Image Cli:history

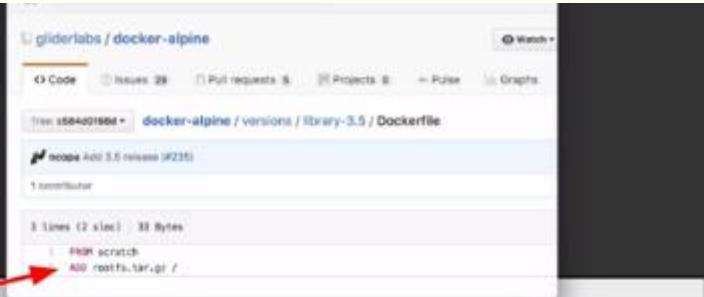
- Montre comment l'image a été créée
- Une entrée d'historique par layer

```
$ docker history alpine
IMAGE      CREATED      CREATED BY
baa5d3471ea  4 days ago  /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872      SIZE      COMMENT
                                                               4.803 MB

Alpine:latest Dockerfile

$ docker image history ubuntu
IMAGE      CREATED      CREATED BY
f753707788c5  9 days ago  /bin/sh -c #(nop)  CMD ["/bin/bash"]      SIZE      COMMENT
<missing>    9 days ago  /bin/sh -c mkdir -p /run/systemd && echo 'doc      0 B
<missing>    9 days ago  /bin/sh -c sed -i 's/^#\s*\(\deb.*universe\)\$/      7 B
<missing>    9 days ago  /bin/sh -c rm -rf /var/lib/apt/lists/*      1.895 kB
<missing>    9 days ago  /bin/sh -c set -xe  && echo '#!/bin/sh' > /u      0 B
<missing>    9 days ago  /bin/sh -c #(nop) ADD file:b1cd0e54ba28cb1d6d      745 B
<missing>    9 days ago  /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872      127.2 MB

Ubuntu:latest Dockerfile
```



A red arrow points from the 'ADD file:7afbc23fda8b0b3872' command in the Alpine Dockerfile output to the 'ADD rootfs.tar.gz /' line in the GitHub Dockerfile snippet.

# Image Cli:ls

---

```
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
node       6.9.4  c5667be18e4e 2 days ago 655 MB
alpine     latest 88e169ea8f46 3 weeks ago 3.98 MB
<none>    <none> d02c4d04476c 11 hours ago 653.2 MB
```

Images créées ou téléchargées

```
$ docker image ls -a
REPOSITORY TAG IMAGE ID CREATED SIZE
<none>    <none> dcb9d60b5a87 About a minute ago 653.2 MB
<none>    <none> 9a77011f0d01 About a minute ago 653.2 MB
node       6.9.4  c5667be18e4e 2 days ago 655 MB
alpine     latest 88e169ea8f46 3 weeks ago 3.98 MB
<none>    <none> d02c4d04476c 11 hours ago 653.2 MB ]
```

Liste les images temporaires créées lors du build

```
$ docker image ls node
REPOSITORY TAG IMAGE ID CREATED SIZE
node       6.9.4  06b984afb149 9 days ago 655 MB
```

Filtre des images par nom

```
$ docker image ls --filter dangling=true
REPOSITORY TAG IMAGE ID CREATED SIZE
<none>    <none> d02c4d04476c 11 hours ago 653.2 MB
```

Les images "dangling" ne sont plus référencées et peuvent être supprimées avec la commande prune

```
$ docker image prune
Total reclaimed space: 653.2 MB
```

udemy

# Image Cli: save / load

---

---

```
$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
alpine          latest        88e169ea8f46   3 weeks ago   3.98 MB

$ docker save -o alpine.tar alpine

$ ls
alpine.tar

$ docker image rm alpine
Untagged: alpine:latest
Untagged: alpine@sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Deleted: sha256:88e169ea8f46ff0d0df784b1b254a15ecfaf045aee1856dca1ec242fdd231ddd
Deleted: sha256:60ab55d3379d47c1ba6b6225d59d10e1f52096ee9d5c816e42c635ccc57a5a2b

$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE

$ docker load < alpine.tar
60ab55d3379d: Loading layer [=====] 4.226 MB/4.226 MB
Loaded image: alpine:latest

$ docker image ls
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
alpine          latest        88e169ea8f46   3 weeks ago   3.98 MB
```

# Image Cli: rm

---

- Supprime une image avec l'ensemble de ses layers
- Plusieurs images peuvent être supprimées en même temps

```
$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
ubuntu              latest   f49eec89601e  16 hours ago  129 MB
mhart/alpine-node  6.9.4    d448eac1cfdb  2 weeks ago   49 MB
alpine              latest   88e169ea8f46  3 weeks ago   3.98 MB

$ docker image rm ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:71cd81252a3563a03ad8daee81047b62ab5d892ebbf71cf53415f29c130950
Deleted: sha256:f49eec89601e8484026a8ed97be00f14db75339925fad17b440976cffcbfb88a
Deleted: sha256:3e2d12b23faf176cf40429fb25be6572212807f27455b8e3e114c397324446f
Deleted: sha256:88a37465e211da3c72acbd999b158ee31c6c7239131f6308f000dcf52d622a7b
Deleted: sha256:99be185bee70b39c64096b8d39b96153d28d3caa7764961a9285ad4d189cd536
Deleted: sha256:fc7e2c65ec42780443f87ae7d9621cd6fcdb371e2127dd461b449d9e50b7ab7b
Deleted: sha256:4f03495a4d7de505ccb8b8e4cf0a8ac201491e1f67ae54b65584e0012aaab9c

$ docker image ls -q
d448eac1cfdb
88e169ea8f46

$ docker image rm $(docker image ls -q)
```

# Disposition de titre et de contenu avec liste

---

---

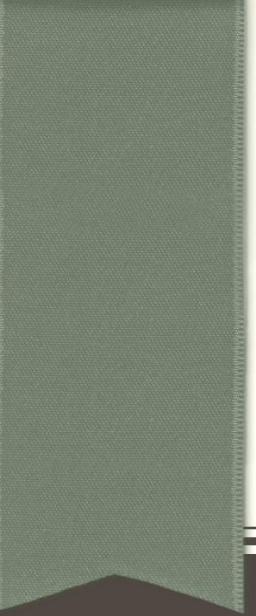
TP

Manipulations des images  
avec la CLI

# Disposition de titre et de contenu avec liste

---

**Question ?**



# MODULE 5

**Volume**

# Présentation

---

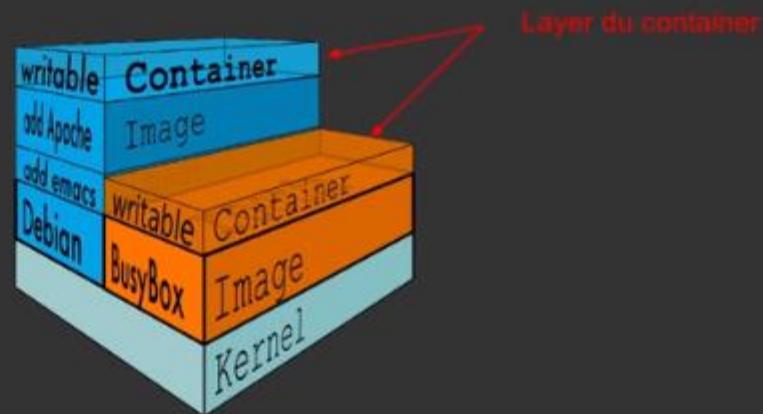
---

- Container et persistance de données
- Générer des volumes
- Aborder les volumes d'hôte
- Partager les volumes
- Définir les volumes dans un Dockerfile

# Container et persistance de données

---

- Un container ne permet pas la persistance des données
- Changements dans la layer du container
  - read-write layer au dessus de l'image
  - créée et supprimée avec le container
- Pour être persistées, les données doivent être gérées en dehors de l'union filesystem



# Container et persistance de données

---

```
# Création d'un container basé sur alpine
$ docker container run -ti --name test alpine sh

# Création d'un fichier dans ce container
/ # touch MYFILE
```

```
# Le process de PID 1 est killé => le container
est stoppé
/ # exit
```

```
# Suppression du container
$ docker rm test
```



```
# MYFILE est présent sur le filesystem de l'hôte
$ find /var/lib/docker -name MYFILE
./aufs/diff/4ef1ca017...4f08f75e2ca36886ed208cdb/MYFILE
./aufs/mnt/4ef1ca017...4f08f75e2ca36886ed208cdb/MYFILE
```

```
# MYFILE est toujours présent
$ find /var/lib/docker -name MYFILE
./aufs/diff/4ef1ca017...4f08f75e2ca36886ed208cdb/MYFILE
```

```
# MYFILE a été supprimé
$ find /var/lib/docker -name MYFILE
8fc5f8a326ad:/var/lib/docker#
```

# Volume

---

- Répertoires / fichiers existant en dehors de l'union filesystem
- Utilisé pour découpler les données du cycle de vie d'un container
- Peut être défini de plusieurs façons
  - instruction VOLUME dans le Dockerfile
  - option -v à la création d'un container
  - via la CLI de l'image

# Volume

---

- Création d'un répertoire local dans `/var/lib/docker/volumes` par défaut
- Copie le contenu du répertoire du container dans le volume
- Création du répertoire dans le container (si non existant)
- Nombreux drivers disponibles
  - Intégration avec des système de stockage externes
  - Ex: Rex-Ray (AWS, GCE, Azure, ..)
- Exemple de cas d'usage: base de données, logs, ...

# Définition dans le Dockerfile

---

## Définition dans le Dockerfile

```
&& mv /etc/mongod.conf /etc/mongod.conf.orig  
  
RUN mkdir -p /data/db /data/configdb \  
    && chown -R mongodb:mongodb /data/db /data/configdb  
VOLUME /data/db /data/configdb  
  
COPY docker-entrypoint.sh /entrypoint.sh  
ENTRYPOINT ["/entrypoint.sh"]  
  
EXPOSE 27017  
CMD ["mongod"]
```

\$ docker container run -d --name mongo mongo:3.2  
893a60693a7196239ffe0ccb158d833cc1dedec268dd874e891c8dccac0f733b

Extract of mongo:3.2 Dockerfile

```
$ docker container inspect -f '{{json .Mounts }}' mongo | python -m json.tool  
[  
  {  
    "Name": "65a7aad62fb00b70b39901cd39cdea5064b7b6a310c5255cb7657dbeec66387b",  
    "Source": "/mnt/sda1/var/lib/docker/volumes/65a7aad62fb00b70b39901cd39cdea5064b7b6a310c5255cb7657dbeec66387b/_data",  
    "Destination": "/data/configdb",  
    "Driver": "local",  
    ....  
  },  
  {  
    "Name": "87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13",  
    "Source": "/mnt/sda1/var/lib/docker/volumes/87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13/_data",  
    "Destination": "/data/db",  
    "Driver": "local",  
    ....  
  }  
]
```

# Définition dans le Dockerfile

---

```
$ ls /var/lib/docker/volumes/87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13/_data
WiredTiger                               WiredTigerLAS.wt                         index-1--8458247895687289006.wt
storage.bson
WiredTiger.lock                           _mdb_catalog.wt                         journal
WiredTiger.turtle                          collection-0--8458247895687289006.wt   mongod.lock
WiredTiger.wt                            diagnostic.data                      sizeStorer.wt

$ docker container rm -f mongo
mongo

$ ls /var/lib/docker/volumes/87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13/_data
WiredTiger                               WiredTigerLAS.wt                         index-1--8458247895687289006.wt
storage.bson
WiredTiger.lock                           _mdb_catalog.wt                         journal
WiredTiger.turtle                          collection-0--8458247895687289006.wt   mongod.lock
WiredTiger.wt                            diagnostic.data                      sizeStorer.wt
```

Le contenu du volume est présent après la suppression du container

# Définition dans le Dockerfile

## Définition dans le Dockerfile

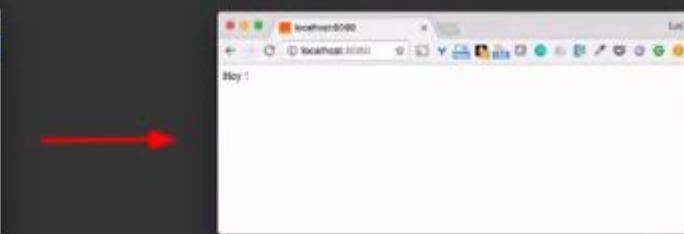
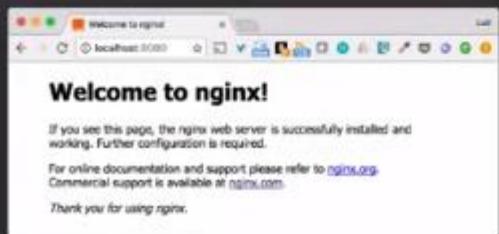
```
$ docker image build -t mynginx .  
  
$ docker run -d --name mynginx -p 8080:80 mynginx  
  
$ docker inspect -f '{{json .Mounts }}' mynginx | python -m json.tool  
[  
  {  
    "Name": "2fd22be72eafea3f8b924b5678c190752f012baef1e0e155aacfb38bb91a524d",  
    "Source": "/var/lib/docker/volumes/2fd22be72eafea3f8b924b5678c190752f012baef1e0e155aacfb38bb91a524d/_data",  
    "Destination": "/usr/share/nginx/html",  
    "Driver": "local",  
    "Type": "volume"  
  }  
]  
  
$ echo '<html><body>Hey !</body></html>' > /var/lib/docker/volumes/2fd22be72eafea...55aacfb38bb91a524d/_data/index.html
```

FROM nginx:1.11.8

VOLUME /usr/share/nginx/html

CMD ["nginx", "-g", "daemon off;"]

Example of custom image based on nginx



# Création à l'exécution

---

- Option `-v` dans la commande *docker container run*
  - `$ docker container run -v /usr/share/nginx/html nginx`
- Même résultat que la définition dans le Dockerfile
- Création d'un volume et copie du contenu du répertoire du container

# Création à l'exécution

---

```
$ docker container run -d --name nginx -p 8080:80 -v /usr/share/nginx/html nginx
$ docker container inspect -f '{{json .Mounts }}' nginx | python -m json.tool
[
    {
        "Destination": "/usr/share/nginx/html",
        "Driver": "local",
        "Name": "b1c6aa7103a4f1440c34f0cb7889df12bf6fe2760149c05c3a046a933871af65",
        "Source": "/var/lib/docker/volumes/b1c6aa7103a4f1440c34f0cb7889df12bf6fe2760149c05c3a046a933871af65/_data",
        "Type": "volume"
    }
]

#/ ls /var/lib/docker/volumes/b1c6aa7103a4f1440c34f0cb7889df12bf6fe2760149c05c3a046a933871af65/_data
50x.html      index.html

$ echo '<html><body>Hey !</body></html>' > /var/lib/docker/volumes/b1c6aa7103a...933871af65/_data/index.html
```

# Bind-Mount

---

---

- Répertoire ou fichier de l'hôte monté dans un container
- A la création d'un container avec l'option -v
- *\$ docker container run -v HOST\_PATH/CONTAINER\_PATH ...*
- Création automatique du folder sur l'hôte ou dans le container
- Cas d'usage
  - en développement: montage du code source dans un container
  - donner accès à la socket unix du daemon Docker
    - *\$ docker container run -v /var/run/docker.sock:/var/run/docker.sock ...*
    - utile pour écouter les évènements du Docker daemon / Swarm

# Bind-Mount

---

- Contenu du répertoire de l'hôte "cache" celui du répertoire du container

```
# Création d'un répertoire et fichier sur le hôte
$ mkdir /tmp/myfolder && touch /tmp/myfolder/file_from_host

# Bind-mount myfolder sur /tmp dans un container basé sur Alpine
$ docker container run -ti -v /tmp/myfolder:/tmp alpine sh

# Le répertoire myfolder "cache" le contenu de /tmp
/ # ls /tmp/
file_from_host

# Création d'un fichier depuis le container
/ # touch /tmp/file/file_from_container

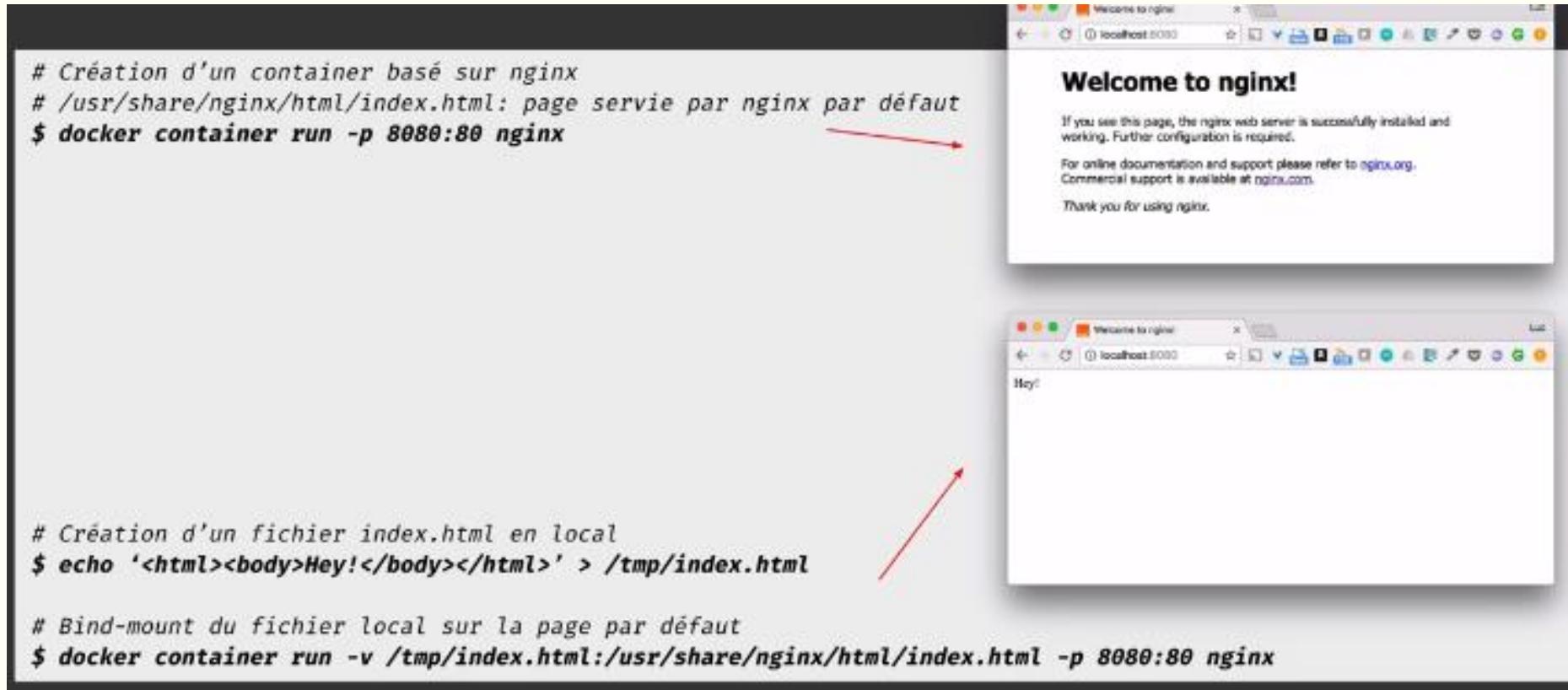
# Le fichier est visible dans le répertoire de l'hôte
$ ls /tmp/myfolder
file_from_host file_from_container
```

66

# Bind-Mount

---

```
# Création d'un container basé sur nginx
# /usr/share/nginx/html/index.html: page servie par nginx par défaut
$ docker container run -p 8080:80 nginx
```



A red arrow points from the first Docker command in the terminal to the top browser window. Another red arrow points from the third Docker command in the terminal to the bottom browser window.

```
# Création d'un fichier index.html en local
$ echo '<html><body>Hey!</body></html>' > /tmp/index.html

# Bind-mount du fichier local sur la page par défaut
$ docker container run -v /tmp/index.html:/usr/share/nginx/html/index.html -p 8080:80 nginx
```

The terminal shows three Docker commands:

- Creation of a Docker container based on nginx.
- Execution of the default page /usr/share/nginx/html/index.html.
- Bind-mounting a local file /tmp/index.html to the default page path /usr/share/nginx/html/index.html.

The top browser window displays the standard "Welcome to nginx!" page. The bottom browser window displays the custom "Hey!" content from the bind-mounted file.

# Volume Cli

---

---

- Introduit dans la version 1.9
- Différents drivers disponibles pour l'orchestration des volumes
  - [https://docs.docker.com/engine/extend/legacy\\_plugins/#/volume-plugins](https://docs.docker.com/engine/extend/legacy_plugins/#/volume-plugins)

```
$ docker volume --help
Usage:docker volume COMMAND

Manage volumes

Options:
  --help  Print usage
Commands:
  create    Create a volume
  inspect   Display detailed information on one or more volumes
  ls        List volumes
  prune    Remove all unused volumes
  rm        Remove one or more volumes
```

Ode

# Volume Cli

---

```
$ docker volume create --name html  
html  
  
$ docker volume ls  
DRIVER      VOLUME NAME  
local       html  
  
$ docker volume inspect html  
[  
  {  
    "Driver": "local",  
    "Labels": {},  
    "Mountpoint": "/var/lib/docker/volumes/html/_data",  
    "Name": "html",  
    "Options": {},  
    "Scope": "local"  
  }  
]  
  
$ ls /var/lib/docker/volumes/html/_data
```

Création du volume *html* en utilisant le driver par défaut

Liste des volumes existants

Inspection du volume *html*

Une fois créé, le volume est un répertoire vide

# Volume Cli

---

# Volume Cli

---

```
$ docker container run -d --name www -v html:/usr/share/nginx/html nginx  
  
$ ls /var/lib/docker/volumes/html/_data  
50x.html index.html
```

Le volume *html* est monté dans le container

Le contenu du répertoire du container est visible dans le volume

```
$ docker container run -ti alpine sh  
/ # mount | grep '/tmp'  
/ #  
/ # exit  
  
$ docker container run -ti -v html:/tmp alpine sh  
/ # mount | grep '/tmp'  
/dev/sda2 on /tmp type ext4 (rw,relatime,data=ordered)  
/ # touch test  
  
$ ls /var/lib/docker/volumes/html/_data  
50x.html index.html test
```

A l'aide de l'option *-v*, le volume est monté sur */tmp* dans le container

Un fichier créé dans */tmp* est créé dans le volume

ude

# Générer des volumes

---

- Un volume est un dossier externe au conteneur, et qui est monté dans l'arborescence du conteneur.
- Ce mécanisme est conçu pour :
  - la pertinence des données, indépendamment du cycle de vie du conteneur,
  - fournir aux conteneurs des fichiers de configuration à l'application qui s'exécute dans le conteneur.
  - la performance, si on veut éviter de passer à travers toutes les couches de l'image pour réaliser des entrées/sorties.
- Pour créer un volume

---

```
[user1@centos1 ~]$ docker volume create --name vol1
vol1
[user1@centos1 ~]$ docker volume ls
DRIVER      VOLUME NAME
local        vol1
```

# Générer des volumes

---

- Pour utiliser le volume, il va falloir le monter au lancement du conteneur.
- Par exemple

```
[user1@centos1 ~]$ docker run -it -v vol1:/www/html centos bash
```

- Vérifier les propriétés du volume. Pour cela, on va utiliser la commande **inspect**.

```
[user1@centos1 ~]$ docker volume inspect vol1
[
  {
    "CreatedAt": "2018-02-07T17:18:58+01:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
    "Name": "vol1",
    "Options": {},
    "Scope": "local"
  }
]
```

# Disposition de titre et de contenu avec liste

---

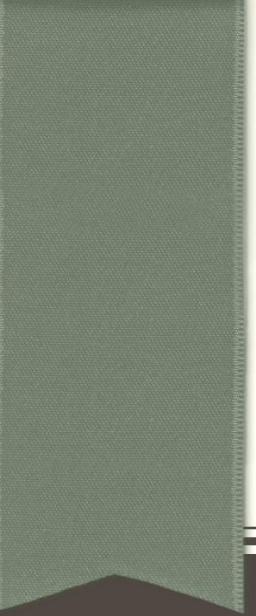
**Question ?**

# Disposition de titre et de contenu avec liste

---

TP

Pratique Volume



# MODULE 7

Docker Compose

# Présentation

---

---

- Présentation et Installation
- Définition d'une application multicontainers
- Déploiement sur le poste de travail
- Déploiement sur un Docker host créé avec Docker Machine

# Présentation

---

---

- Outil permettant de définir et de gérer des applications complexes
- Convient bien à une architecture de micro-services
- Format de fichier
  - docker-compose.yml
  - définition de l'application
- Binaire
  - docker-compose
  - écrit en Python
  - permet de lancer une application sur un hôte Docker

# Présentation

---

---

- Hôte unique
  - application lancée avec le binaire docker-compose
  - `$ docker-compose up`
- Cluster Swarm
  - application lancée par le client Docker
  - notion de Stack comme un ensemble de services
  - `$ docker stack deploy`
- Certaines options utilisables dans un seul des 2 cas
  - `build` uniquement prise en compte par `docker-compose`
  - `deploy` uniquement prise en compte par `docker stack`
  - ...

# Le fichier docker-compose.yml

---

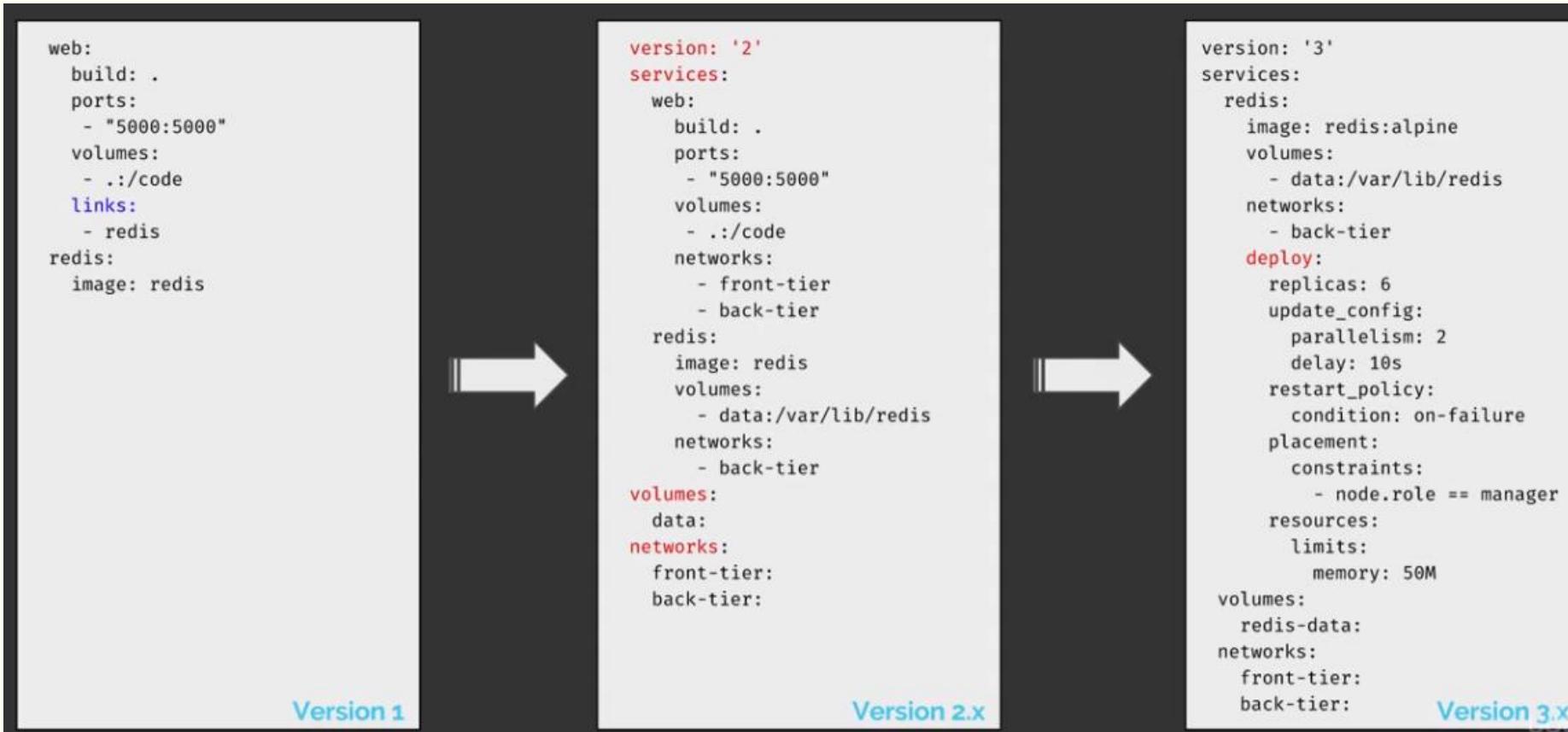
- Définition des composants de l'application
  - Services
    - spécifie une image à utiliser
    - spécifie une commande à lancer
    - instancié en un ou plusieurs containers
  - Volumes
  - Networks
  - Secrets (Swarm)
- De nombreuses options disponibles
  - <https://docs.docker.com/compose/compose-file/>

```
version: '3'
services:
  db:
    image: mongo:3.4
    volumes:
      - data:/data/db
    networks:
      - backnet
    restart: always
  api:
    image: org/api:1.2
    networks:
      - backnet
      - frontnet
    restart: always
  web:
    image: org/web:2.3
    networks:
      - frontnet
    restart: always
volumes:
  data:
networks:
  frontend:
  backend:
```

Application web décrite dans un fichier docker-compose.yml

# Le fichier docker-compose.yml

---



# Le fichier docker-compose.yml

---

## docker-compose.yml: Dev vs Prod

- Même fichier de base pour différents environnement
- Mais des contraintes et options différentes
  - pas de bind-mount du code applicatif
  - binding de ports
  - variables d'environnement
  - règles de redémarrage
  - services supplémentaires (tls, logs, monitoring, ...)
  - contraintes de placement (dans le cas d'un cluster Swarm)
  - ...

# Le binaire docker-compose

---

- Gestion du cycle de vie d'une application au format Compose
- Installation indépendante
  - <https://docs.docker.com/compose/install/>
- *docker-compose [ -f <arg>... ] [options] [COMMAND] [ARGS...]*
  - ex: *docker-compose up --scale api=3*
- Se base sur le fichier docker-compose.yml par défaut
- Plusieurs fichiers peuvent être utilisés

# Le binaire docker-compose

---

---

Commande	Utilisation
up / down	Création / Suppression d'une application (services, volumes, réseaux)
start / stop	démarrage / arrêt d'une application
build	Build des images des services (si instruction build utilisée)
pull	Téléchargement d'une image
logs	Visualisation des logs de l'application
scale	Modification du nombre de container pour un service
ps	Liste les containers de l'application

Liste des commandes les plus utilisées. La liste complète est obtenue avec `docker-compose --help`

# Le binaire docker-compose

---

---

```
# Construction / téléchargement des images
# Création des volumes
# Lancement des containers
$ docker-compose -f docker-compose-simple.yml up -d
...

# Liste des containers de l'application
$ docker-compose -f docker-compose-simple.yml ps
Name          Command           State    Ports
-----
examplevotingapp_db_1   docker-entrypoint.sh postgres   Up      5432/tcp
examplevotingapp_redis_1  docker-entrypoint.sh redis ... Up      0.0.0.0:32768->6379/tcp
examplevotingapp_result_1 nodemon --debug server.js     Up      0.0.0.0:5858->5858/tcp, 0.0.0.0:5001->80/tcp
examplevotingapp_vote_1   python app.py                 Up      0.0.0.0:5000->80/tcp
examplevotingapp_worker_1  /bin/sh -c dotnet src/Work ... Up

# Scale le nombre d'instance du service worker
$ docker-compose -f docker-compose-simple.yml up --scale worker=3
```

# Communication entre services

---

- Utilisation du DNS du daemon Docker pour la résolution des services
- Un service communique avec un autre service via son nom

```
version: '3'
services:
  db:
    image: mongo:3.4
    volumes:
      - data:/data/db
    restart: always
  api:
    image: org/api:1.2
    restart: always
    volumes:
      data:
```

Extrait d'un fichier docker-compose.yml

Le service **api** utilise le service de base de données par son nom

```
// Mongodb connection string
url = 'mongodb://db/todos';
// Connection to database
MongoClient.connect(url, (err, conn) => {
  if(err){
    return callback(err);
  } else {
    return callback(null, conn);
  }
});
```

Extrait d'un code Node.js: connexion à la base de données

# Disposition de titre et de contenu avec liste

---

---

TP

Docker compose 1 : Mysql

TP

Docker Compose 2 : Wordpress

TP

Docker Compose 3 : ELK

TP

Docker Compose 4 : Voting App

# Disposition de titre et de contenu avec liste

---

**Question ?**