



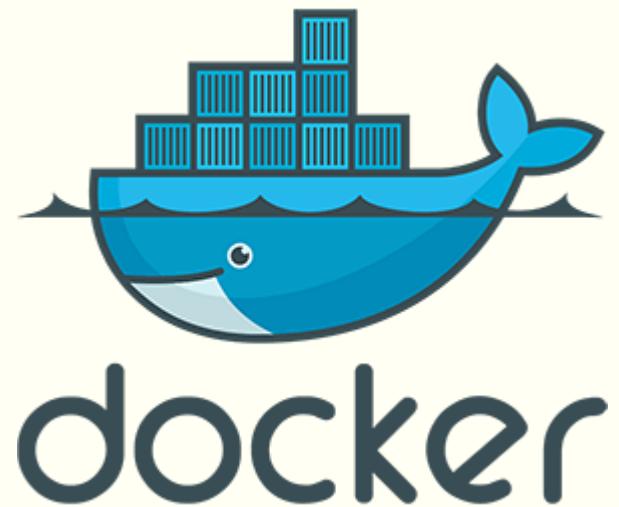
FORMATION DOCKER

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 6: Network
- Module 7: Docker Compose
- Module 8: Docker Swarm
- Module 9: Sécurité



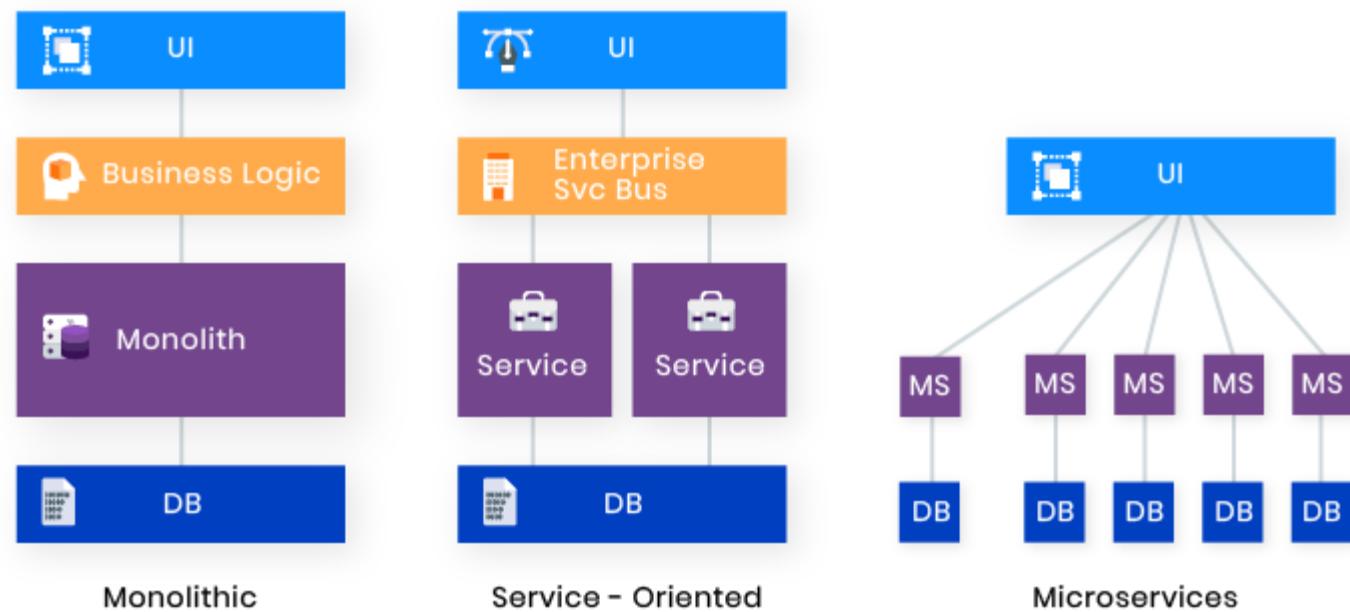
MODULE 1

Introduction à Docker

Introduction à Docker

- Microservice
- 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

Application Monolithique vs Architecture Microservices



Application Monolithique

- « monolithique » signifie formé d'un seul bloc
- conçu pour être autonome
- ses composants sont interconnectés et interdépendants
- chaque composant et ceux qui lui sont associés doivent être présents pour permettre l'exécution ou la compilation du code
- mettre à jour un composant → réécrire toute l'application

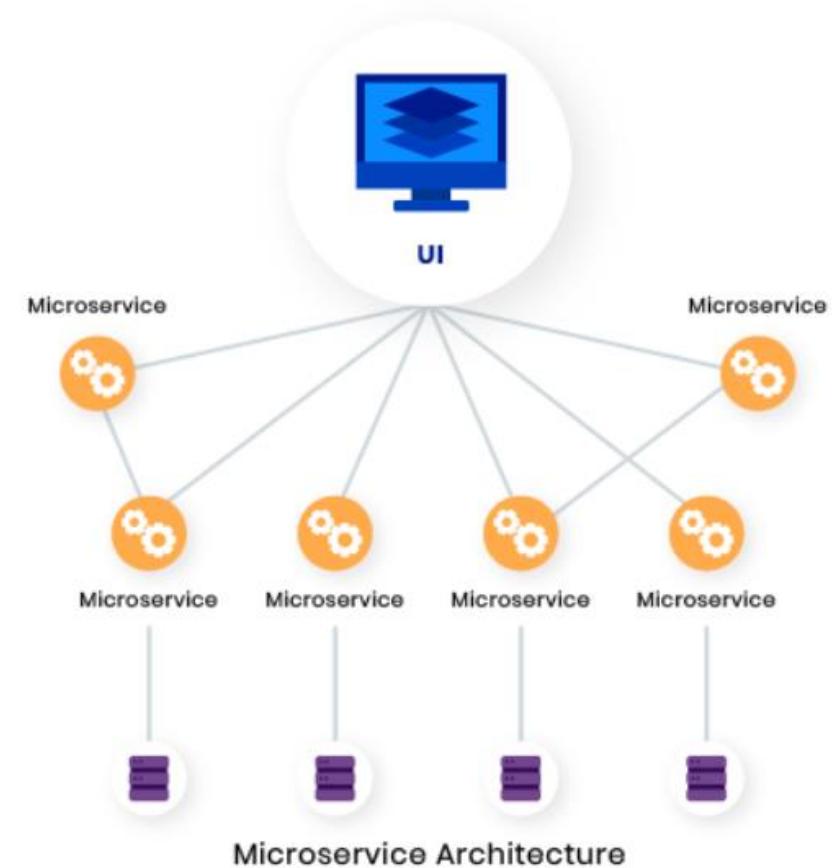


Application Monolithique: Problèmes

- **Problème n°1 :** Comment assurer la haute disponibilité des applications ?
- **Problème n°2 :** Comment répondre rapidement aux nouvelles fonctionnalités métiers et faire évoluer les applications d'une façon très fréquente.
- **Problème n°3 :** Comment être indépendant des technologies utilisées tout en sachant que ces technologies évoluent très vite.

Architecture Microservices

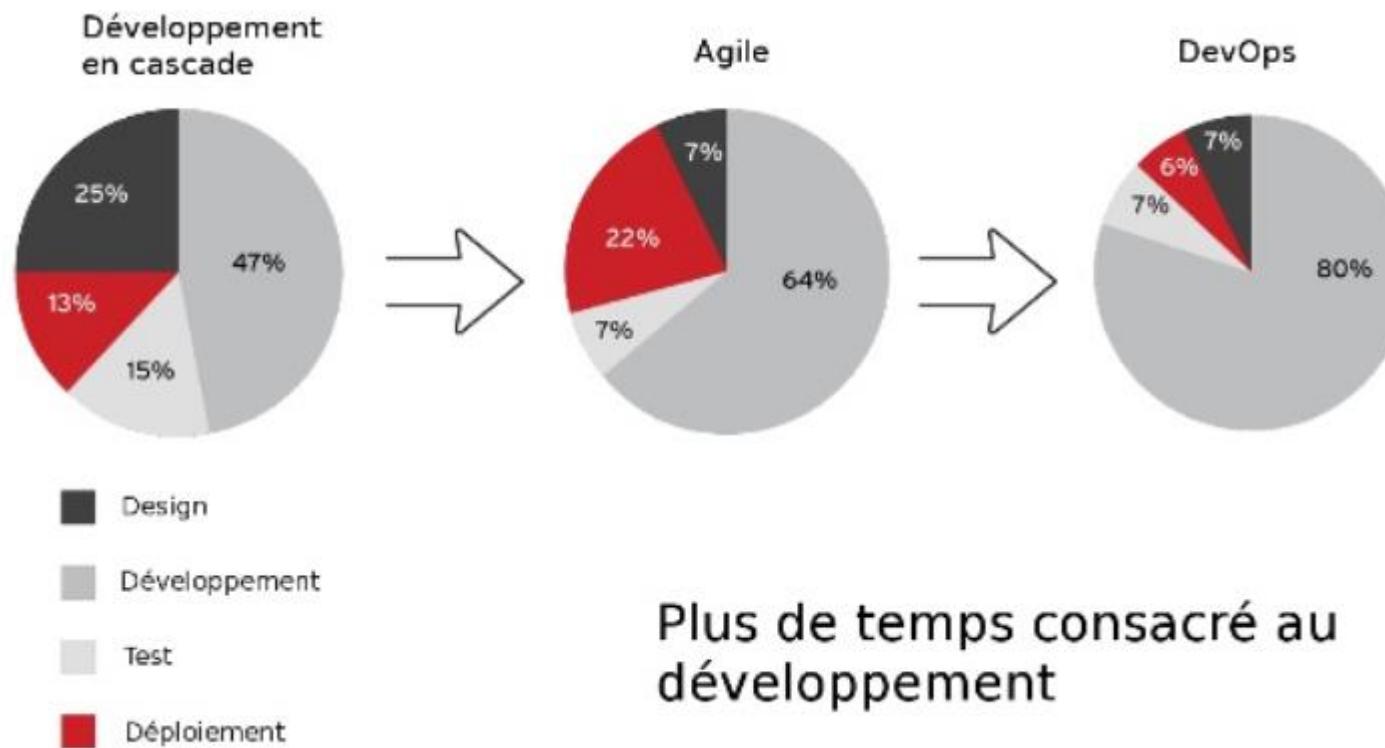
- Composés d'un ou plusieurs services
- Chacun de ces Microservices se concentre sur l'achèvement d'une fonctionnalité
- Développé indépendamment
- Avoir une base de données indépendante
- testé indépendamment
- déployé indépendamment
- (Technologie différente des autres)



Changement de vitesse

Datacenter	Virtualisation	Docker
		
Déploiement dans le mois	⇒ Déploiement dans la minute	⇒ Déploiement dans la seconde
Pendant des années	Pendant des mois	Pendant quelques heures/minutes
Développement en cascade	Agile	DevOps

Plus rapide et de meilleure qualité



De la virtualisation à Docker

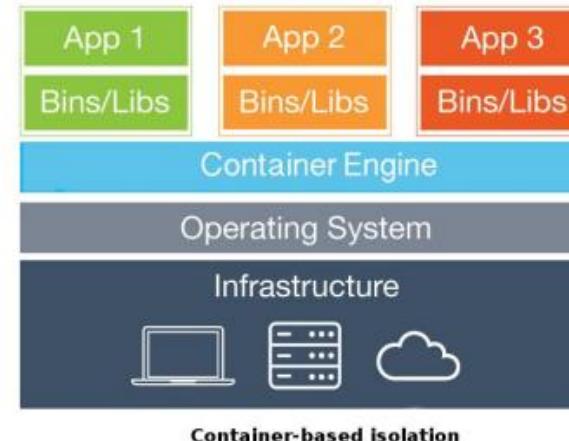
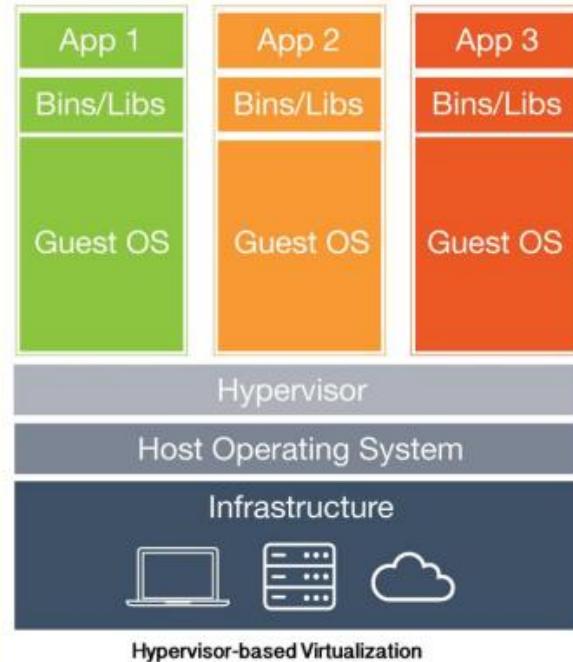
Build, ship and run any app, anywhere

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).

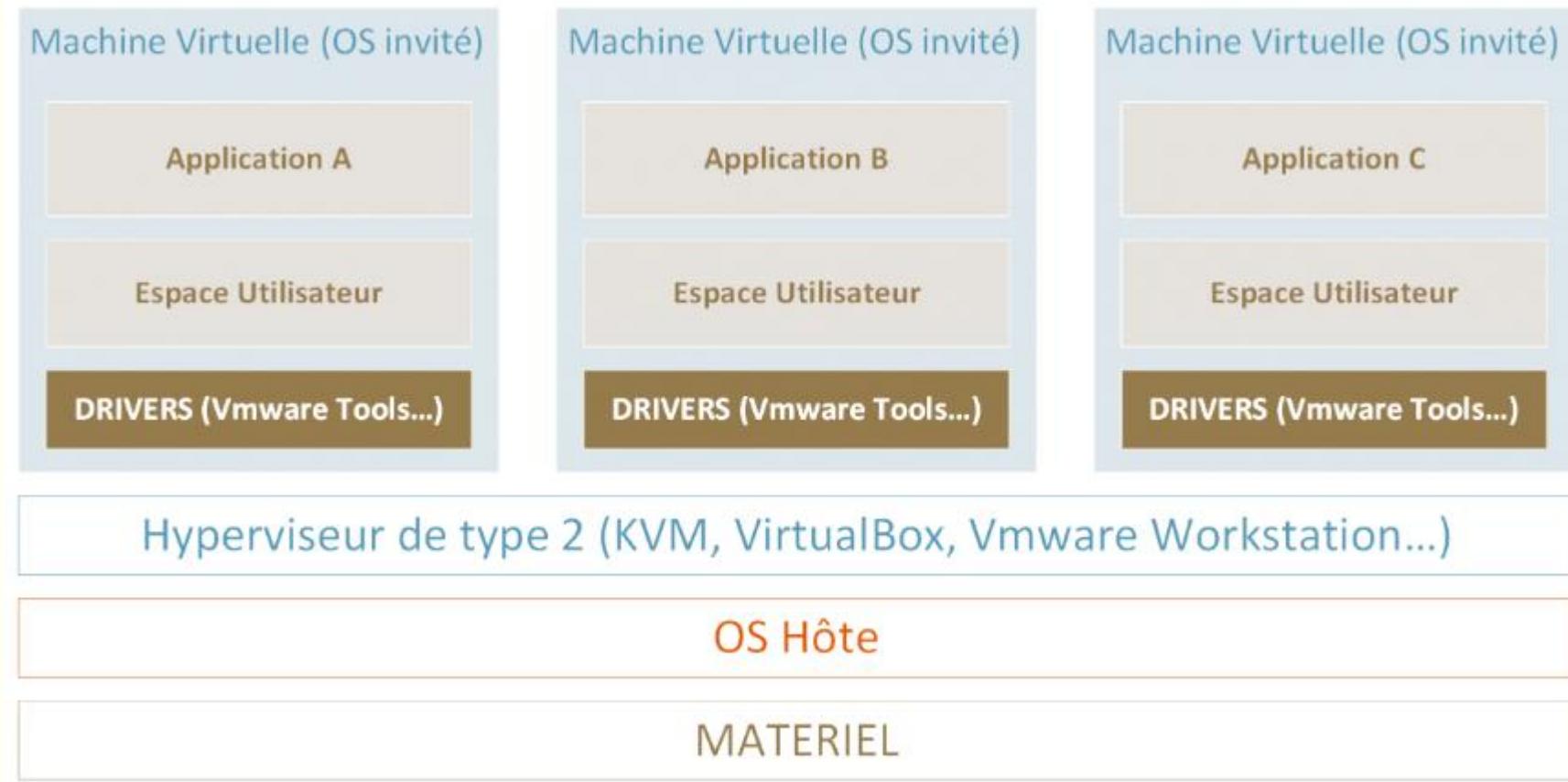
Les différents types de virtualisation

Deux techniques de virtualisation

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



La virtualisation par hyperviseur



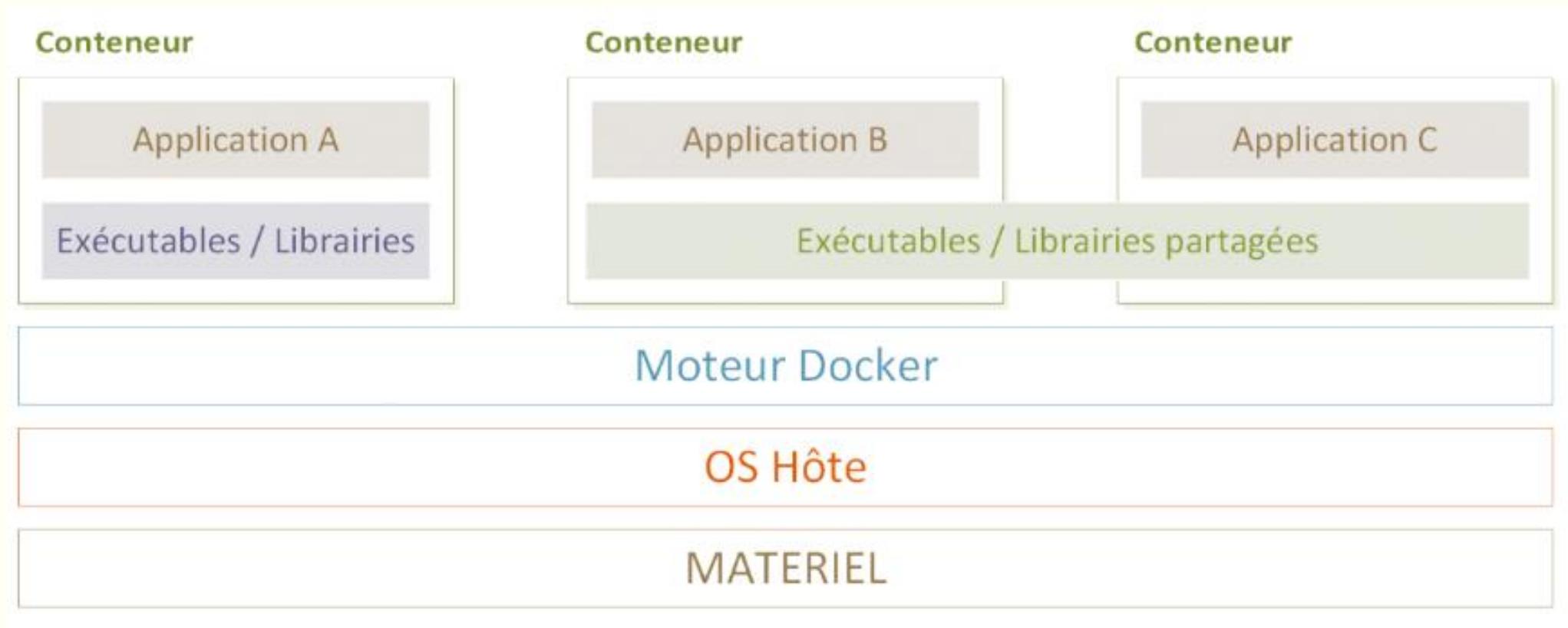
La virtualisation par hyperviseur

- 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 conteneurisation : Namespaces, Control-Groups

- La technologie des conteneurs utilise deux fonctions de base du noyau Linux : les **groupes de contrôle (Cgroupes)** et les **espaces de noms de noyau**.
- 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)** : 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

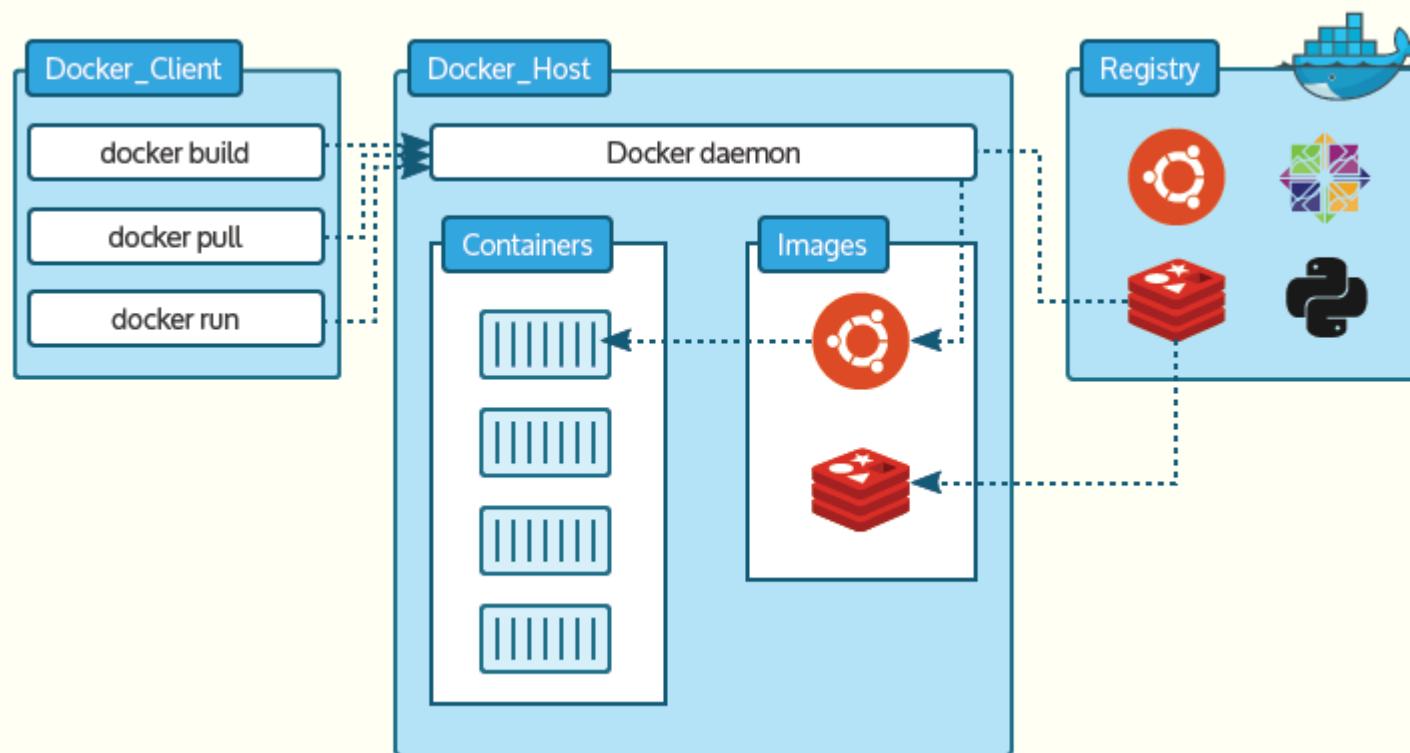
- .
- **Ressources réseau (NET)** : les espaces de noms de réseau permettent d'attribuer à chaque conteneur des ressources réseau distinctes, telles que des adresses IP ou des tables de routage.
- **Point de montage des fichiers système (MNT)** : grâce aux espaces de noms de montage, un processus isolé 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.

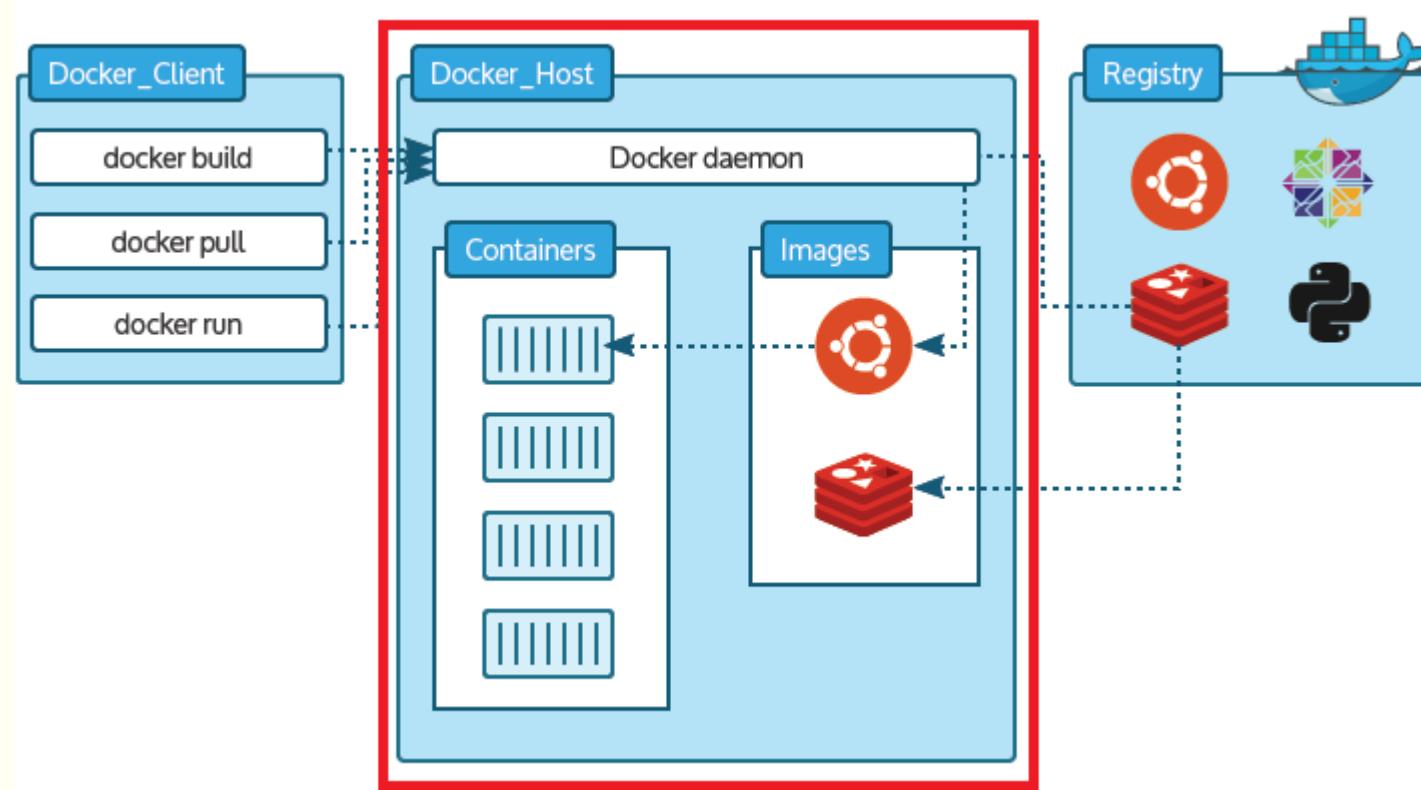
L'architecture de Docker

- 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.



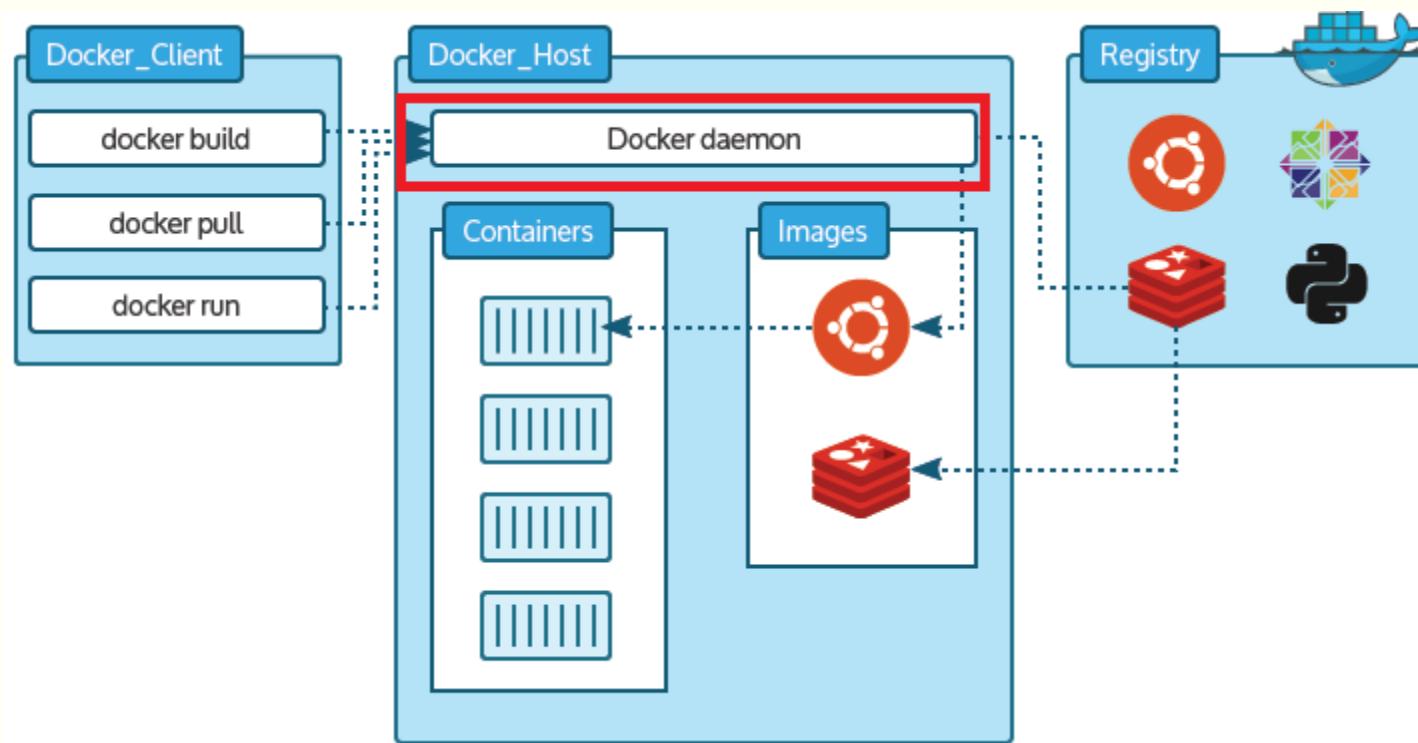
L'architecture de Docker

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



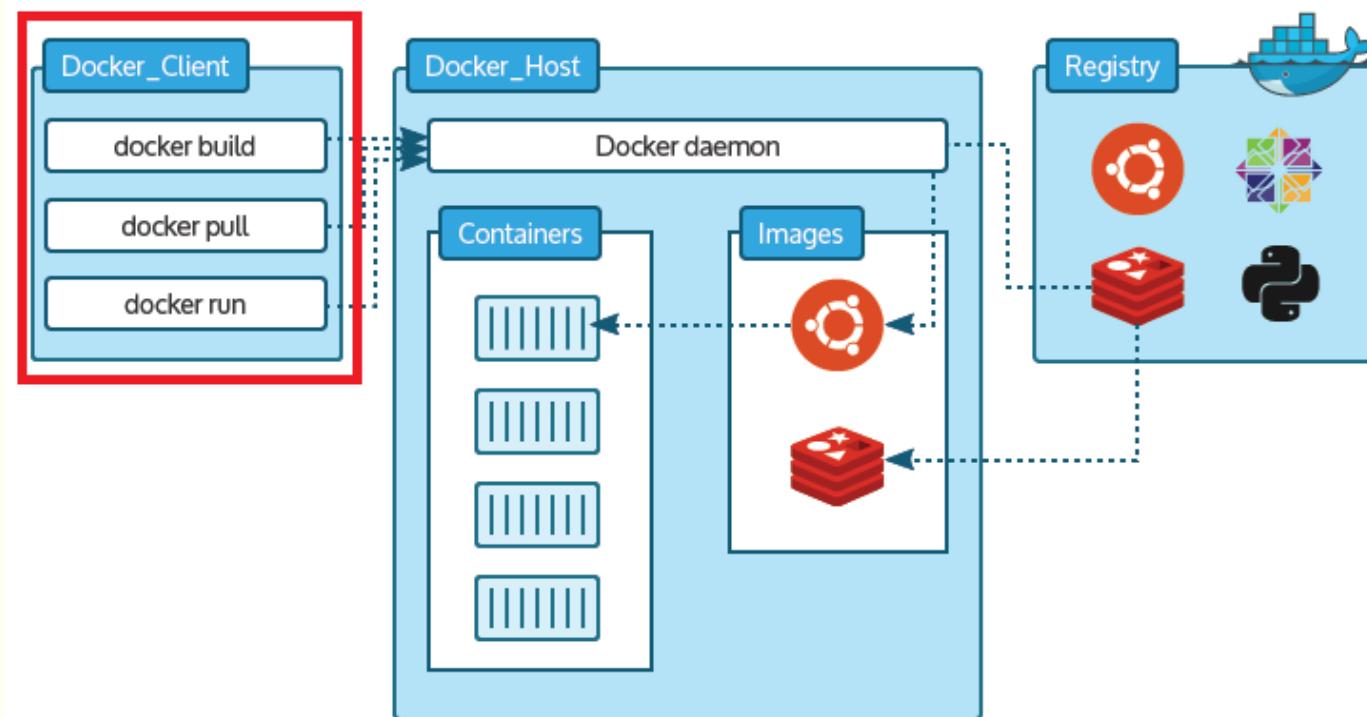
L'architecture de Docker

- 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**.



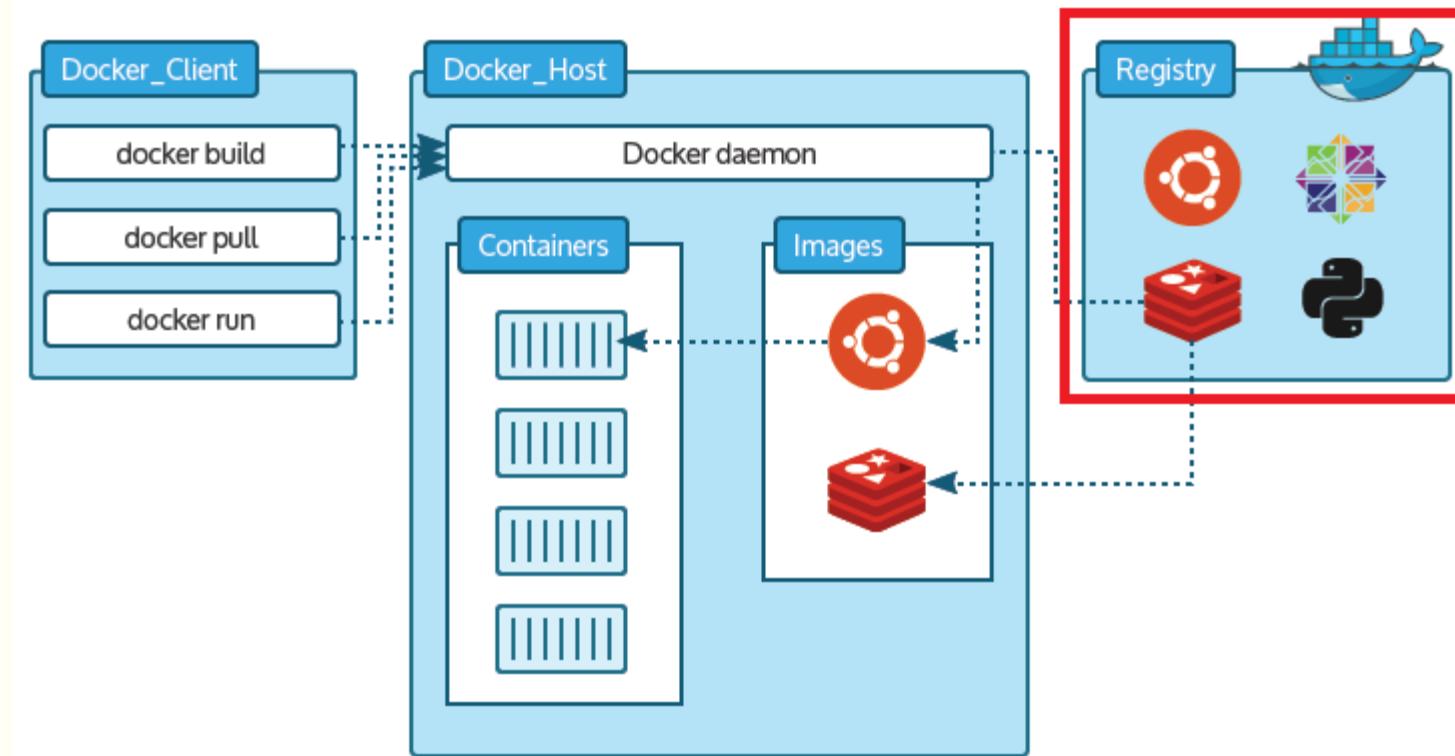
L'architecture de Docker

- 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



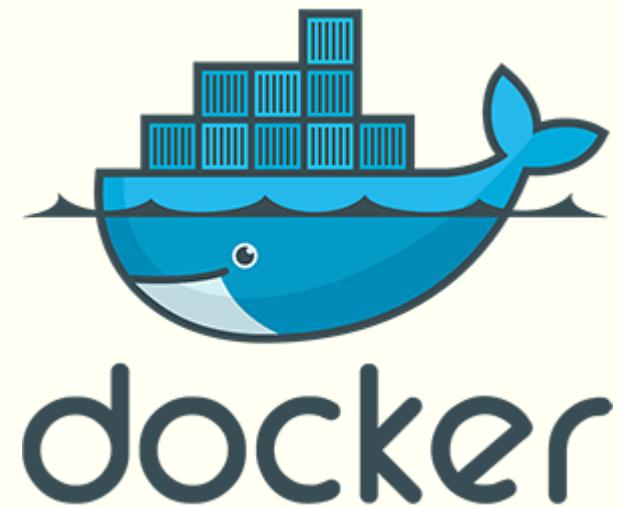
L'architecture de Docker

- 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



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)

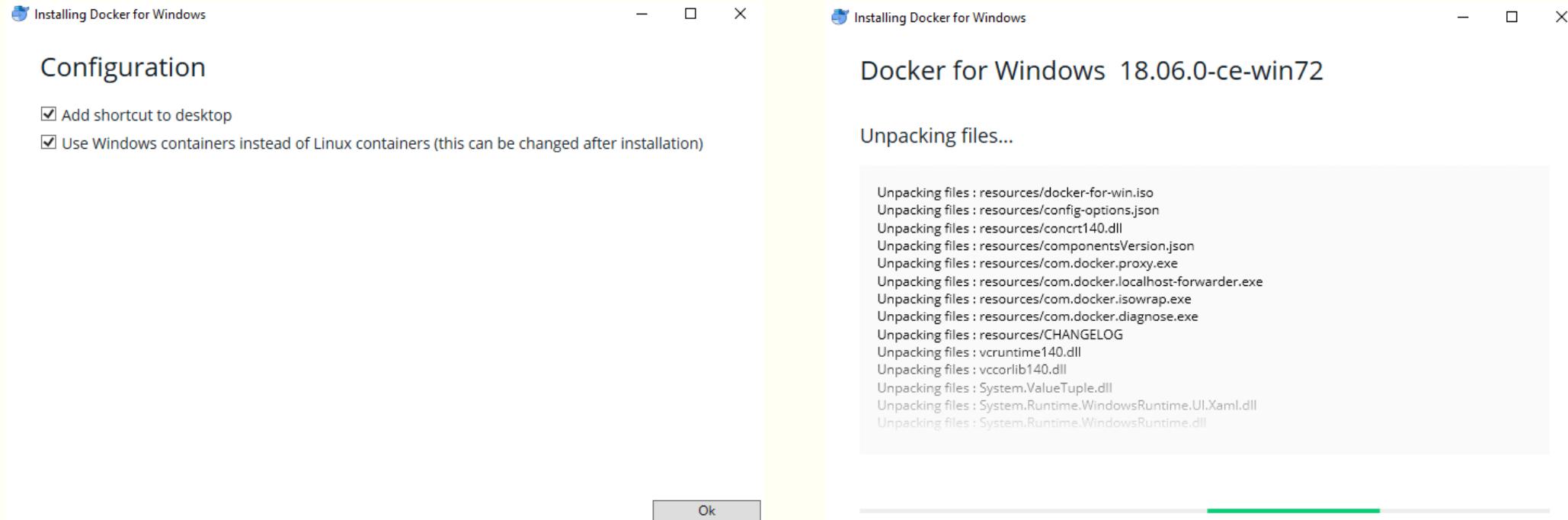
- Vérifiez que la virtualisation est activée sur votre PC :
- Windows 10 : Ouvrir le **Gestionnaire des tâches** > **Onglet Performance** > **Processeur**

Utilisation	Vitesse		Vitesse de base :	3,40 GHz
10%	1,88 GHz		Sockets :	1
Processus	Threads	Handles	Cœurs :	4
193	2681	84895	Processeurs logiques :	4
			Virtualisation :	Activé
Durée de fonctionnement			Cache de niveau 1 :	256 Ko
0:08:31:06			Cache de niveau 2 :	1,0 Mo
			Cache de niveau 3 :	6,0 Mo

- Sinon, l'activer depuis le BIOS.

Installation de Docker (Windows)

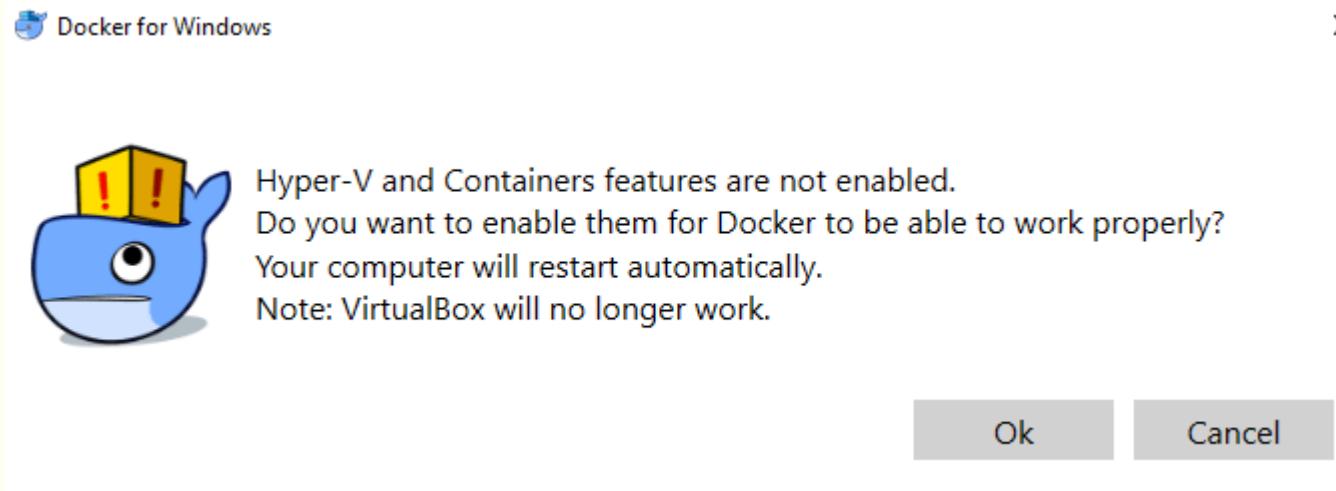
Lancez l'installation de Docker.



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

Installation de Docker (Windows)

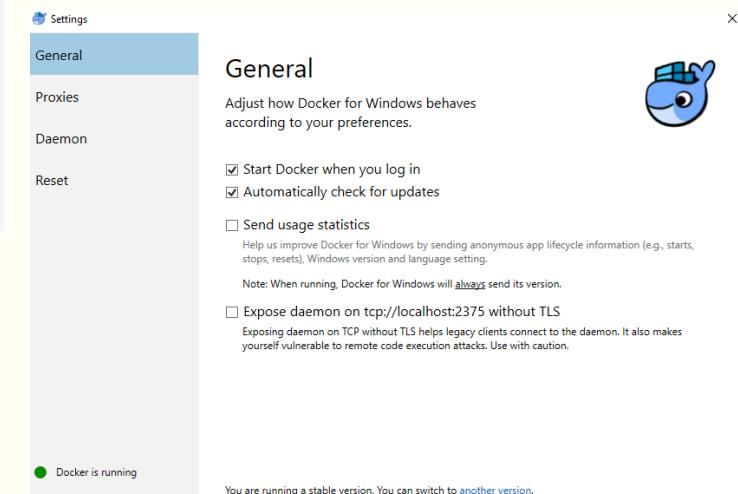
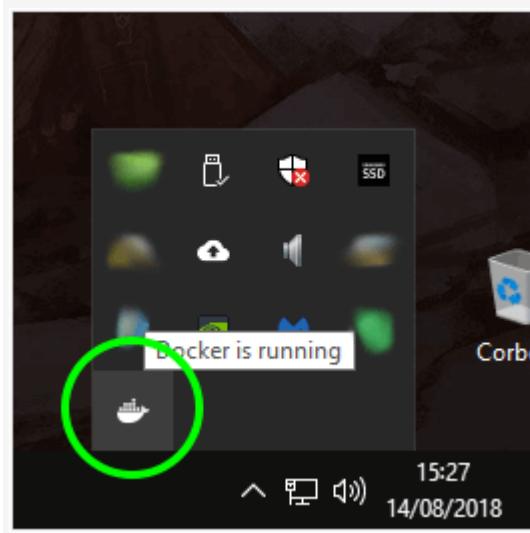
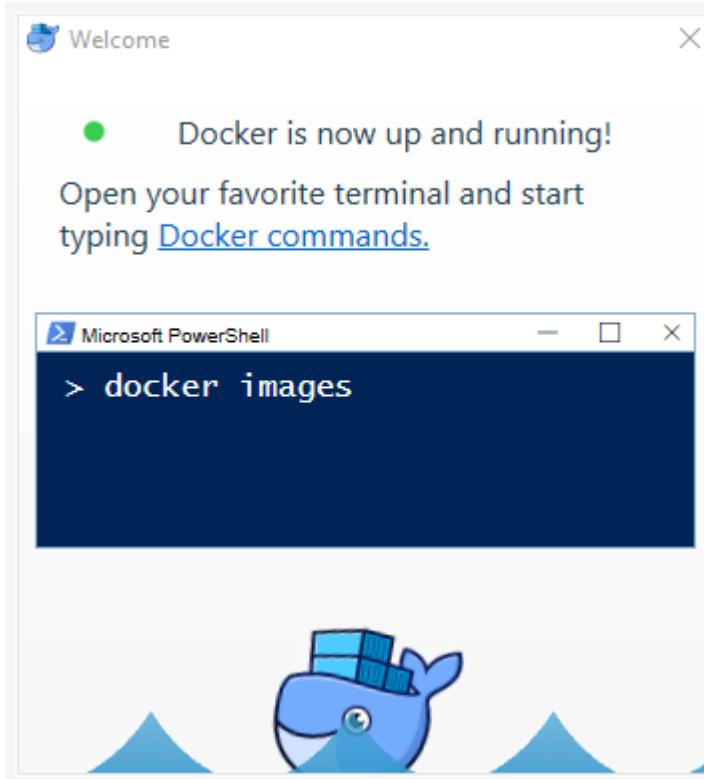
- Après le redémarrage de l'ordinateur, Docker va se lancer automatiquement.
- Ensuite, **un message s'affiche concernant les fonctionnalités Hyper-V**



- **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 et 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. Client:
03.   Version:      18.06.0-ce
04.   API version: 1.38
05.   Go version:   go1.10.3
06.   Git commit:   Offa825
07.   Built:        Wed Jul 18 19:05:28 2018
08.   OS/Arch:      windows/amd64
09.   Experimental: false
10.
11. Server:
12.   Engine:
13.     Version:      18.06.0-ce
14.     API version: 1.38 (minimum version 1.24)
15.     Go version:   go1.10.3
16.     Git commit:   Offa825
17.     Built:        Wed Jul 18 19:23:19 2018
18.     OS/Arch:      windows/amd64
19.     Experimental: false
20. 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)

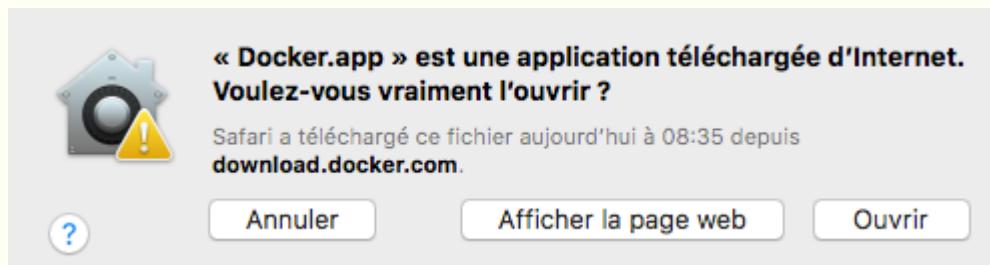


Installation de Docker (Mac)

- La version de mon Mac :

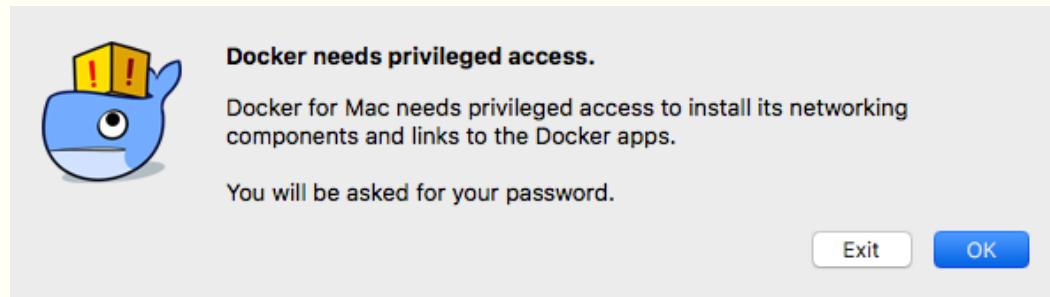


- Vu que l'application n'est pas sur l'Apple Store on a ce message :

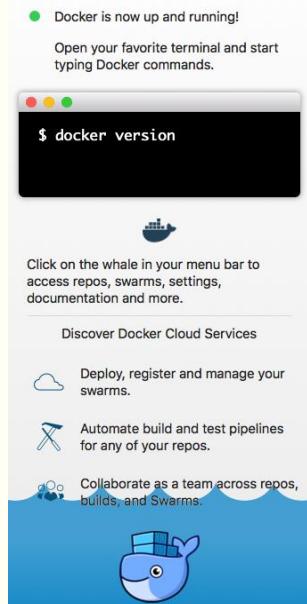


Installation de Docker (Mac)

- 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

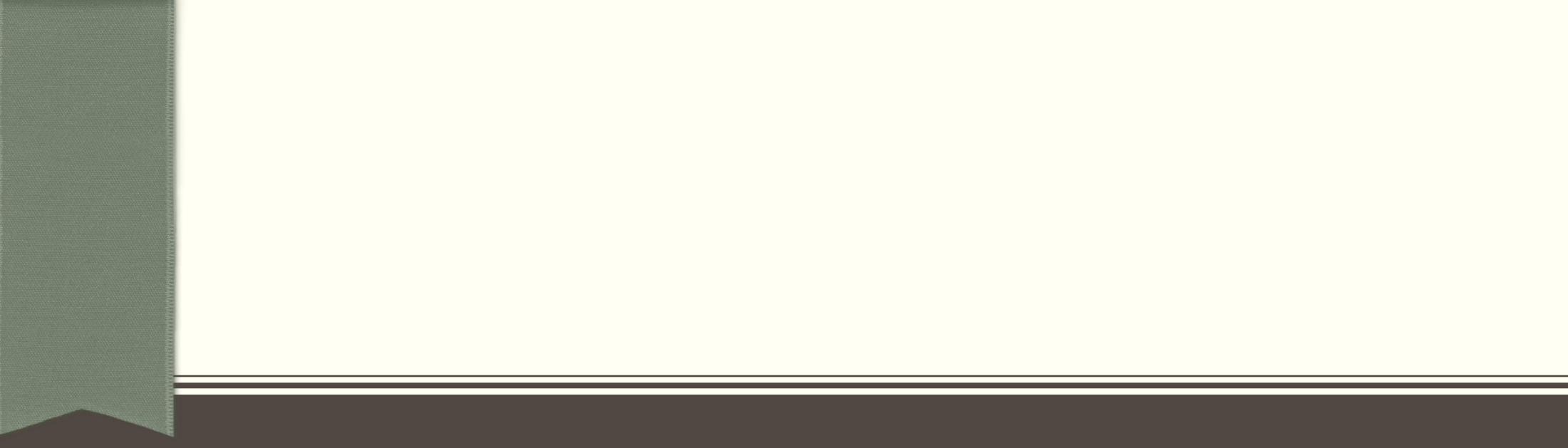
Installation de Vagrant

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: sha256:0add3ace90ecb4adbff777e9aacf18357296e799f81cabcfde470971e499788
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:~$
```

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:/#
```

Création de containers: foreground / background

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

The screenshot shows a terminal window with three entries:

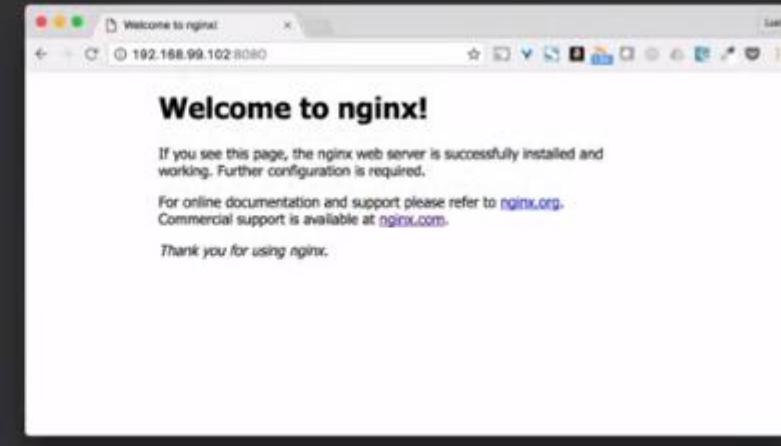
- Création d'un container basé sur l'image nginx en foreground →
\$ docker container run nginx
<process hangs on>
- Création d'un container basé sur l'image nginx en background →
\$ docker container run -d nginx
6f6cac745aa19c01015906480586294f9cc23cd7d1733552a50999a93aef5555
- Création d'un container basé sur l'image alpine en tâche en background →
\$ docker container run -d alpine ping 8.8.8.8
6df4108911d37c357ee6b4928b678f1c996628d336c92617448ffe98a8d0b146

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
```



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
```

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"
}
```

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
/ #
```

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
...
...                ...
```

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

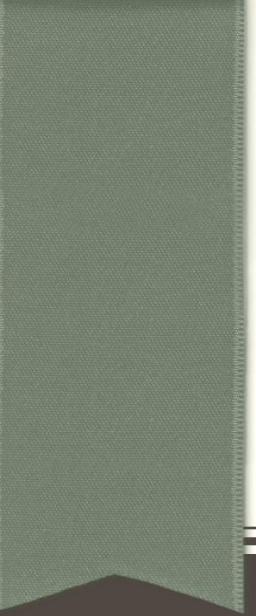
```
$ docker container rm 6df4108911d3
6df4108911d3
```

```
$ 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

Question ?



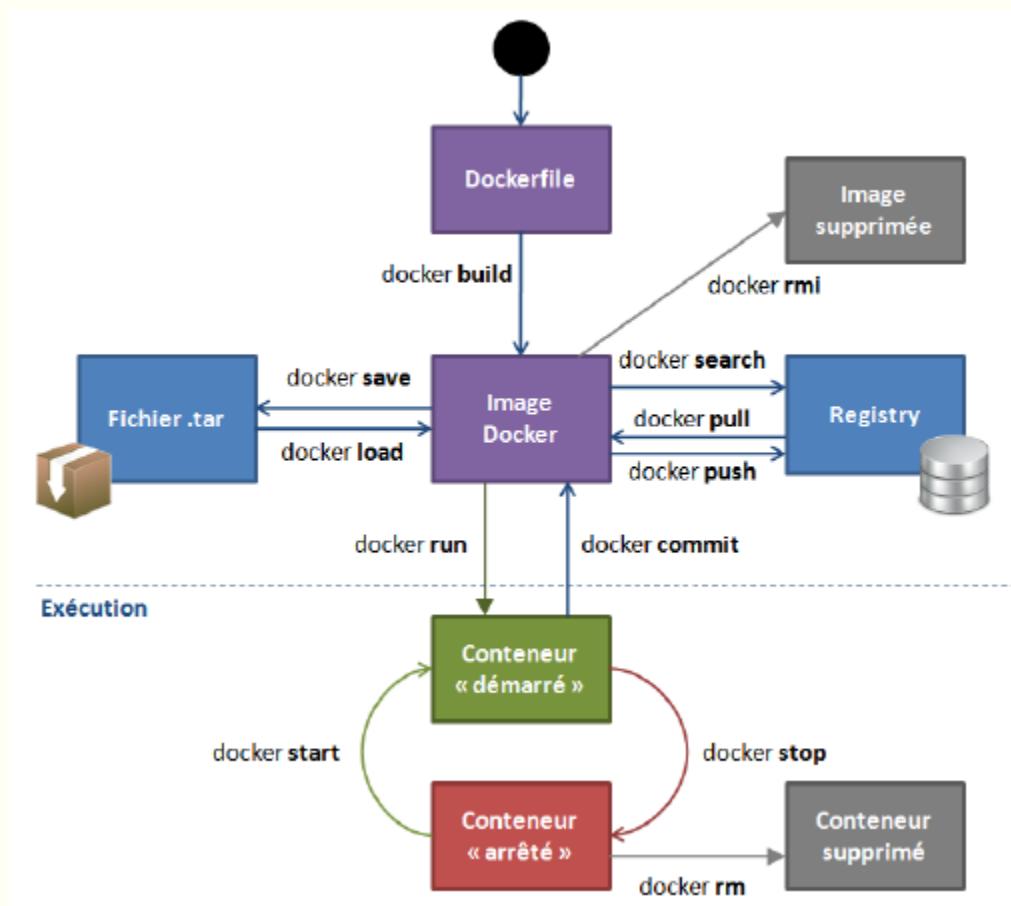
MODULE 4

Création d'images

Présentation

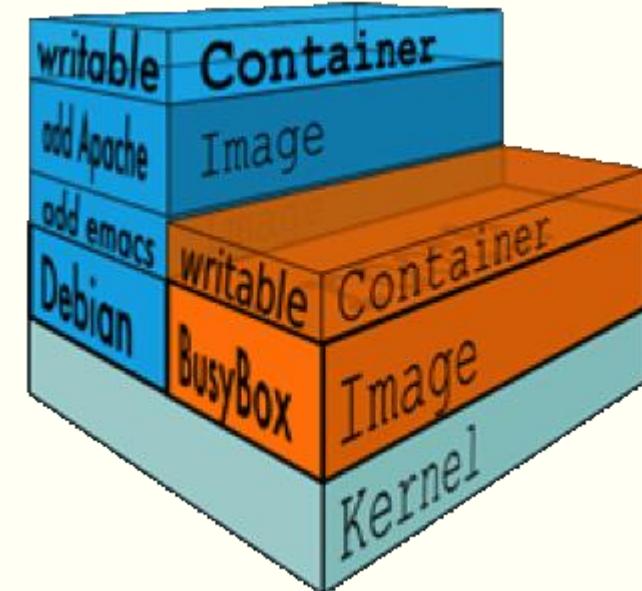
- Anatomie d'une image
- Le Docker Hub
- Pointer sur un registre donné
- **Méthodes de création d'une image**
- Création d'images personnalisées
- Le fichier Dockerfile en détails
- Importance de la prise en compte du cache

Docker: Vue Globale



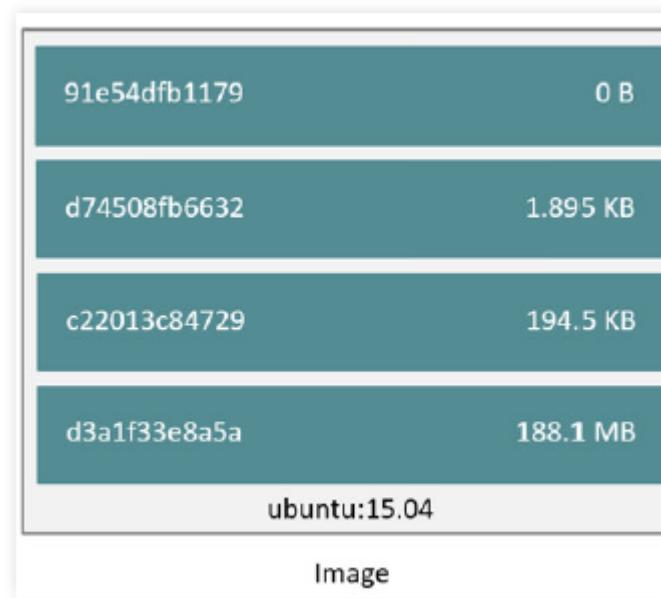
Définition : Image

- 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 méta data
 - Référence une layer parent
 - Est indépendante
 - Est réutilisable par d'autre images
- Une layer read-write créée pour chaque container



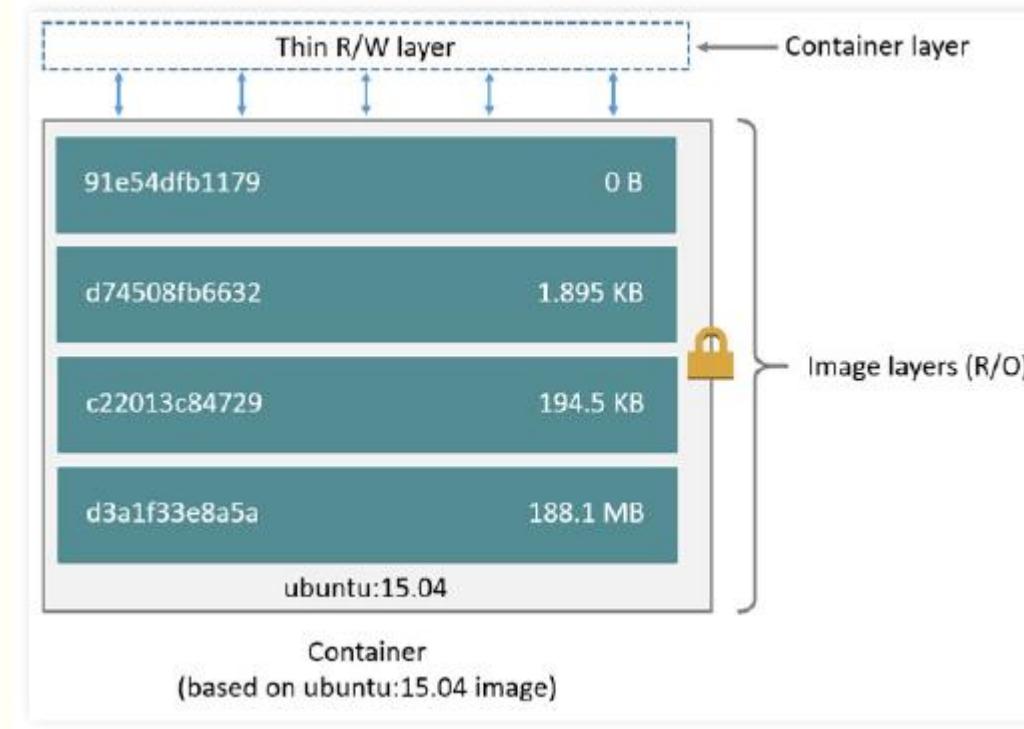
LAYERS

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



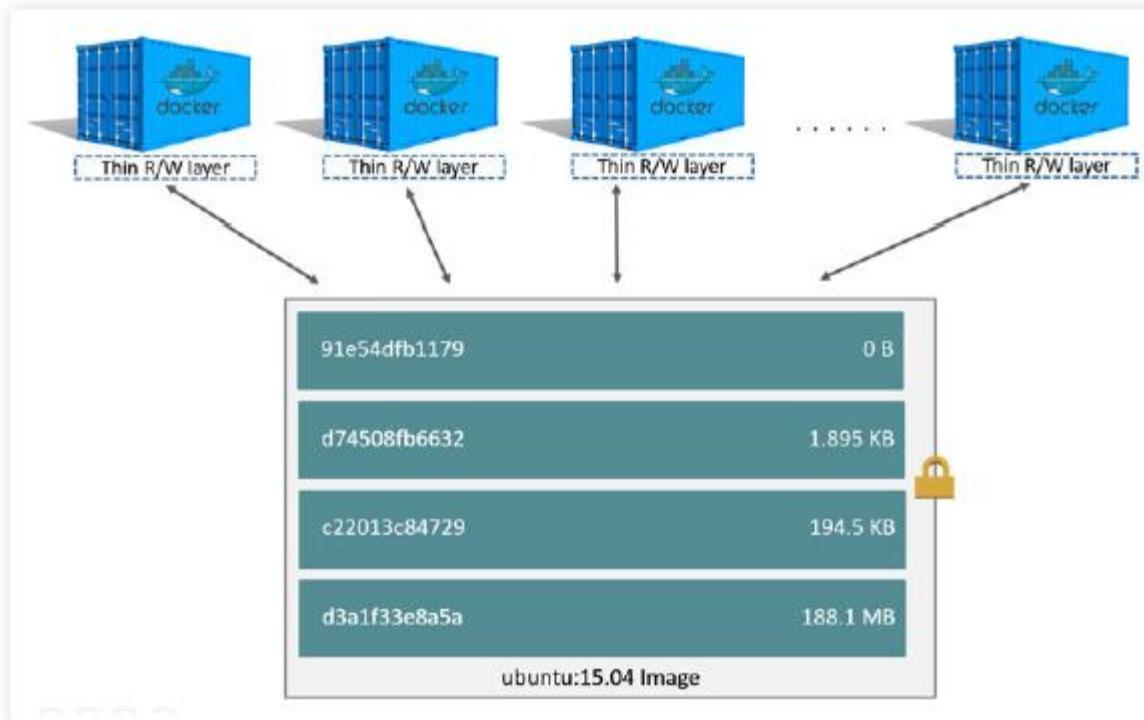
Une image se décompose en layers

LAYERS : UN CONTENEUR



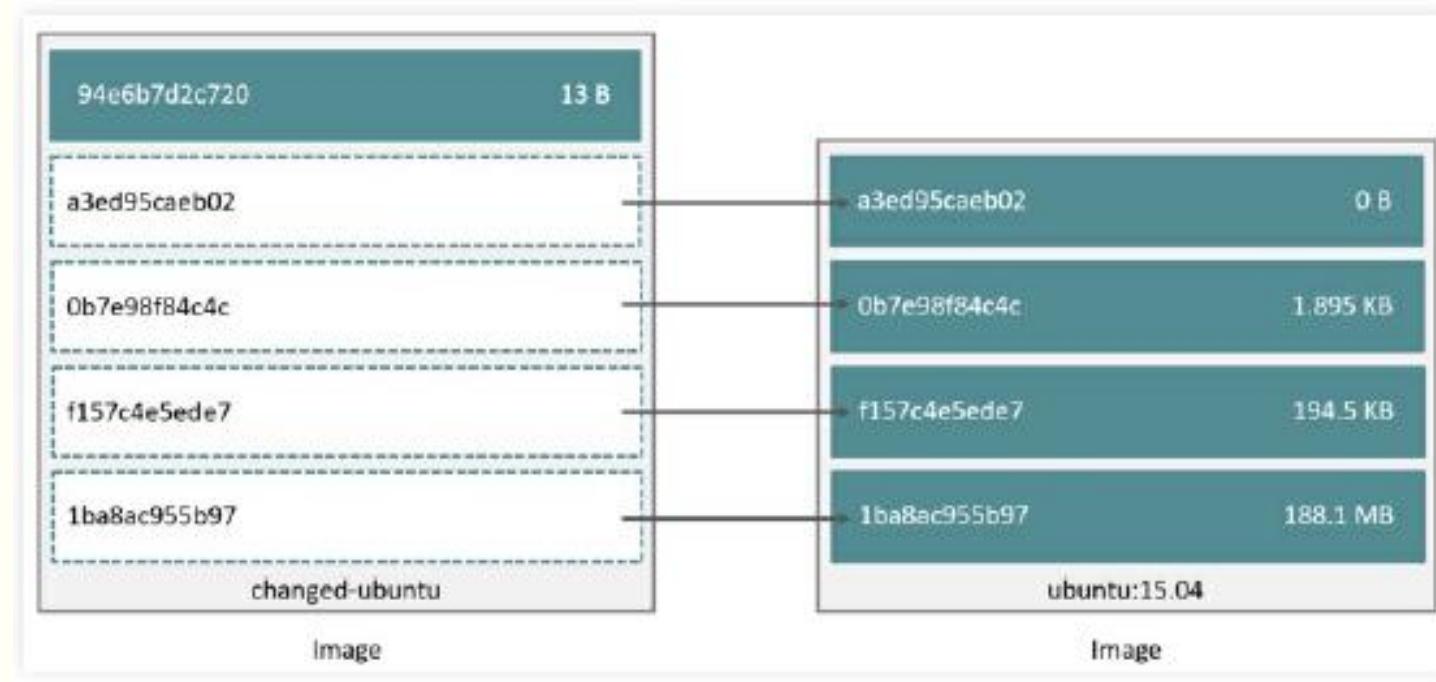
Une conteneur = une image + un layer r/w

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 lié 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" ]
```

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
```



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"]
```

Dockerfile

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
```

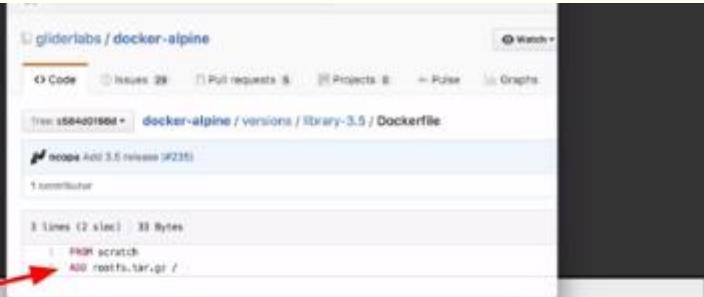


Image Cli:ls

```
$ 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

$ docker image ls -a
REPOSITORY TAG IMAGE ID CREATED SIZE
<none>    <none> dc9d60b5a87 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

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

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

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

Images créées ou téléchargées

Liste les images temporaires créées lors du build

Filtre des images par nom

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

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)
```

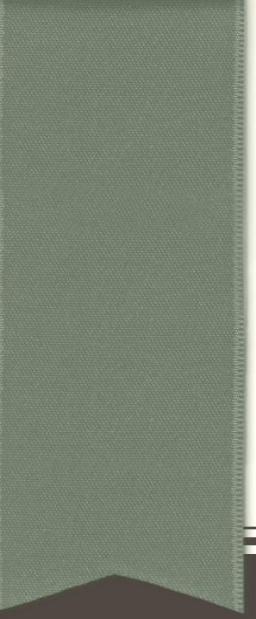
Disposition de titre et de contenu avec liste

TP

Manipulations des images
avec la CLI

Disposition de titre et de contenu avec liste

Question ?



MODULE 5

Volume

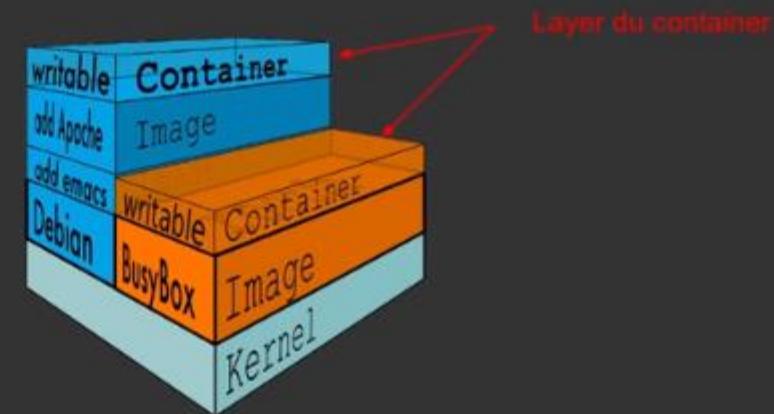
Présentation

- 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

- Instruction VOLUME

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

```
&& 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"]
```

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
```

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
```



```
# 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
```



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

```
$ 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
```

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

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"
  }
]
```

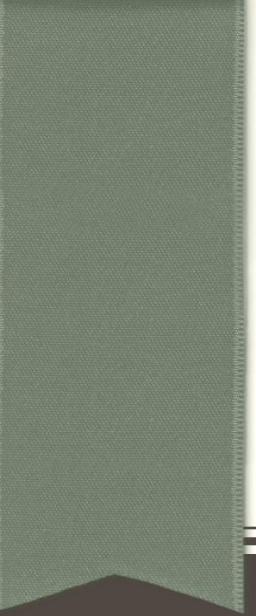
Disposition de titre et de contenu avec liste

Question ?

Disposition de titre et de contenu avec liste

TP

Pratique Volume



MODULE 6

Network

Présentation

- Différents types de réseaux (bridge, overlay)
- Ports mapping entre containers et machine hôte
- Communication entre containers
- Communication multi hôtes

CLI

- Disponible depuis Docker 1.7

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

Manage networks

Options:
  --help    Print usage

Commands:
  connect      Connect a container to a network
  create       Create a network
  disconnect   Disconnect a container from a network
  inspect      Display detailed information on one or more networks
  ls           List networks
  prune        Remove all unused networks
  rm           Remove one or more networks

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

CLI

```
$ docker network create mynet
b38e6d31bc92ff9f57a916ffffe9e9288fccb79dd8a1280dccadc0481cfееad6a

$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
93f9191d1b55    bridge    bridge      local
6bcc6a4f0985    host      host       local
b38e6d31bc92    mynet    bridge      local
87d042e4bd5b    none     null       local

$ docker network inspect -f '{{ json .IPAM.Config }}' mynet | jq
[{
  "Subnet": "172.19.0.0/16",
  "Gateway": "172.19.0.1"
}]

$ docker network rm mynet
mynet
```

User

Les networks de base

- 3 networks créés lors de l'installation de Docker

```
$ docker-machine create --driver virtualbox node1
$ eval $(docker-machine env node1)

$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
fbc4654defd5   bridge    bridge      local
7dcf68373718   host      host       local
98eabcaa9249   none     null       local
```

- Création de l'interface bridgeo sur la machine hôte

```
docker@node01:~$ ip a show docker0
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
      link/ether 02:42:2c:3a:b0:a0 brd ff:ff:ff:ff:ff:ff
        inet 172.17.0.1/16 scope global docker0
              valid_lft forever preferred_lft forever
```

Les networks de base

- Bridge
 - représenté par l'interface dockero créée sur la machine hôte
 - network auquel les containers sont attachés par défaut
 - permet la communication des containers sur un même hôte
- Host
 - stack réseau de la machine hôte
- None
 - pas de connexion sur l'extérieur
 - une seule interface: localhost

Les différents drivers

- Définit le type de network

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
fbc4654defd5	bridge	bridge	local
7dcf68373718	host	host	local
98eabcaa9249	none	null	local

- Plusieurs drivers disponibles par défaut pour la création d'autres networks
 - bridge
 - overlay
 - macvlan

Les différents drivers

- Drivers développés via des plugins (Weave, Kuryr, ...)
- Options spécifiques précisées lors de la création
 - `docker network create --driver DRIVER [OPTIONS] NAME`

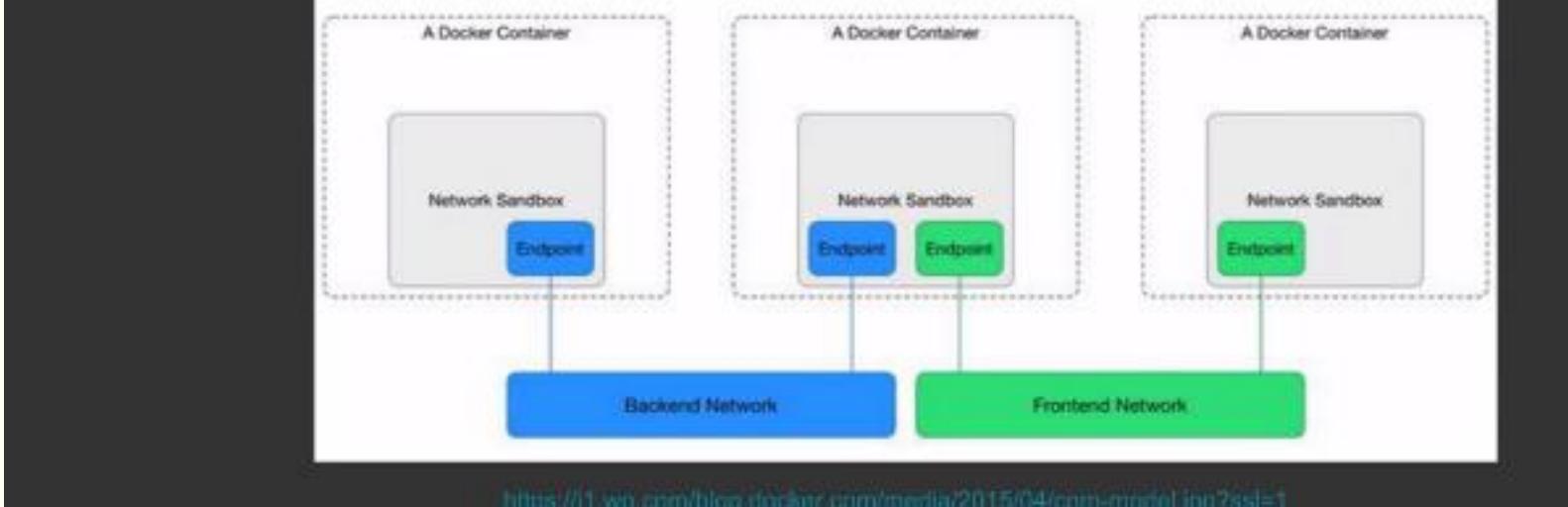
```
# Driver de type bridge utilisé par défaut
docker@node02:~$ docker network create bgnet
e706c80e870c792c5e85659d03dbbdb35def2d1176ddcd182db41b1b701c2ab9

# Création d'un network de type macvlan
docker@node02:~$ docker network create --driver macvlan mvnet
b39891e44e677ea85ab3b70f7ccf528508f1e681e886164aeddb10199e74279c

# Un network Overlay ne peut pas être créé sur un hôte unique
docker@node02:~$ docker network create --driver overlay ovnet
Error response from daemon: datastore for scope "global" is not initialized
```

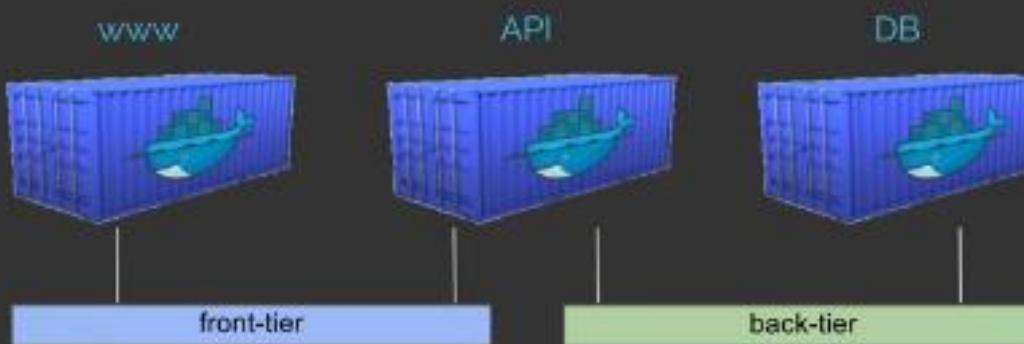
Container Network Model

- Sandbox: stack réseau d'un container
- Endpoint: interface réseau
- Network: un ensemble de Endpoint



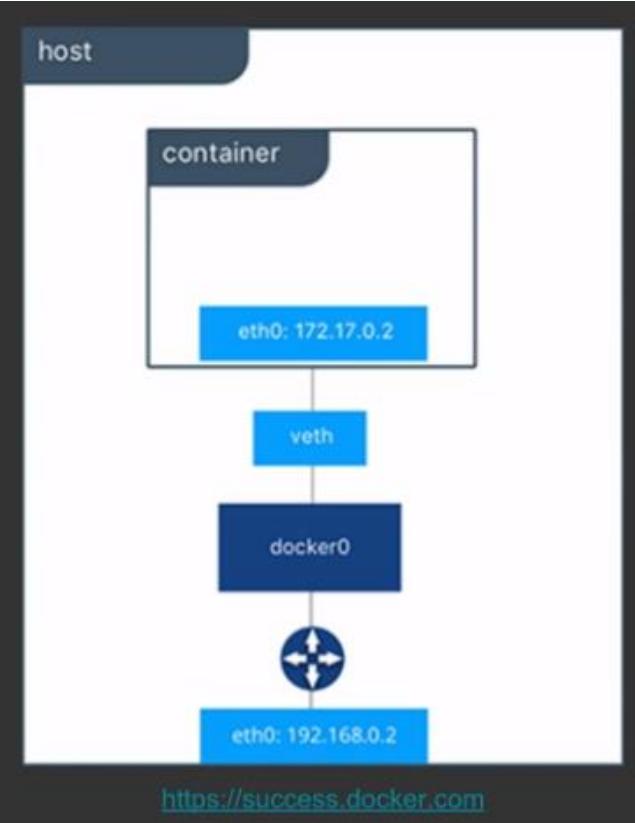
Container Network Model

- Un container peut-être attaché à un network au lancement
 - `docker container run --network NETWORK [OPTIONS] IMAGE`
- Un container peut-être attaché à plusieurs networks
 - front-tier: accessible depuis l'extérieur via le container www
 - back-tier: network interne accessible depuis l'API



Bridge par défaut

- Containers attachés à ce network par défaut
- Représente par l'interface bridge Docker
○ créé à l'installation de Docker
- Création d'un paire d'interfaces virtuelles (veth pair)
○ une interface dans chaque network namespace
 - network namespace du container (ethX)
 - root network namespace (machine hôte)



Bridge par défaut

```
# Creation of an alpine container
$ docker container run -ti alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
...
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
        inet 172.17.0.2/16 scope global eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::42:acff:fe11:2/64 scope link
            valid_lft forever preferred_lft forever

# The other end of the veth pair is attached to the docker0 bridge network
root@node01:~# brctl show
bridge name     bridge id          STP enabled   interfaces
docker0         8000.02428cc9bde4    no           vetha820f30
```

Bridge par défaut

```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
2f00ffa53dd0   bridge    bridge      local
8af0e6c63253   host      host       local
f81df072273d   none     null       local

$ docker run -d --name=nginx nginx
0eee3f18...

$ docker inspect -f '{{.NetworkSettings.IPAddress}}' 0eee
172.17.0.2

$ docker run -ti --name=box busybox sh
/ # ping -c3 nginx
ping: bad address 'nginx'

/ # ping -c3 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.064 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.080 ms
...
```

Les containers attachés au bridge ne peuvent pas communiquer entre eux via leur nom

```
$ docker network inspect 2f00ffa53dd0
...
Containers:
{
    "0eee3f18...": {
        "IPv4Address": "172.17.0.2/16",
        "Name": "nginx",
        ...
    },
    "f4088270...": {
        "IPv4Address": "172.17.0.3/16",
        "Name": "box",
        ...
    }
}
```

Bridge par défaut

```
# Inspect the nginx network settings
$ docker container inspect -f '{{json .NetworkSettings }}' nginx
{
  "Bridge": "",
  "SandboxID": "4969d4507fab66c072ed4c0625f4b300795fcade2ab5220de324f9e3f67c83f",
  "EndpointID": "11f6a79d2d11f5f1772d7a5843ac0d7815e0ad3f8d35446e06c259b54a1d9745",
  "Gateway": "172.17.0.1",
  "IPAddress": "172.17.0.2",
  "MacAddress": "02:42:ac:11:00:02",
  "Networks": {
    "bridge": {
      "NetworkID": "2f00ffa53dd06be3f5b94cc749fc92e13f893ab97579ab5287e144ea10dbfb4c",
      "EndpointID": "11f6a79d2d11f5f1772d7a5843ac0d7815e0ad3f8d35446e06c259b54a1d9745",
      "Gateway": "172.17.0.1",
      "IPAddress": "172.17.0.2",
      "MacAddress": "02:42:ac:11:00:02"
    }
  }
}
```

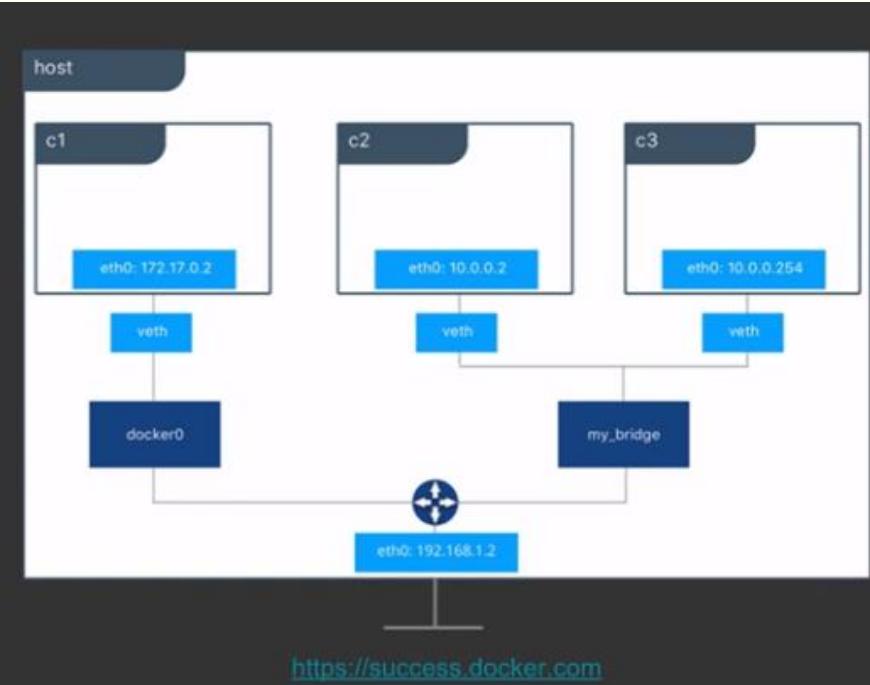
```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
2f00ffa53dd0   bridge    bridge      local
ef94/e210d07   host      host      local
52db6ce3687a   none      null      local
```

Note: part of the output has been removed for readability

ude

"user defined" bridge

- Utilisation du driver "bridge"
- Création d'un bridge Linux
- Utilise des paires d'interfaces virtuelles comme pour le bridge par défaut
- Utilisation du DNS interne pour la résolution des noms des containers



"user defined" bridge

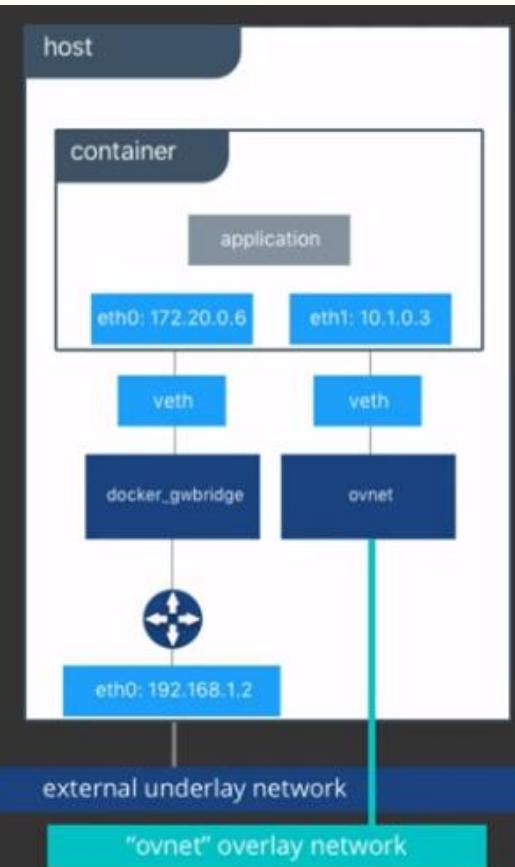
```
$ docker network create --driver=bridge newnet  
49d1e5ee6356...  
  
$ docker network ls  
NETWORK ID      NAME      DRIVER      SCOPE  
689cf99b52ef   bridge    bridge      local  
c7695fcd2124   host      host       local  
49d1e5ee6356   newnet   bridge    local  
2328341f0c45   none      null       local  
  
$ docker run -d --name=nginx --network=newnet nginx  
fe46ce155793...  
  
$ docker run -ti --network=newnet busybox sh  
/ # ping -c3 nginx  
PING nginx (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.057 ms  
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.084 ms  
...
```

Les containers attachés à un "user defined" bridge peuvent communiquer entre eux par leur nom via le serveur DNS du daemon Docker

```
$ docker network inspect 49d1e5ee  
...  
Containers:  
{  
    "fe46ce155791...": {  
        "IPv4Address": "172.17.0.2/16",  
        "Name": "nginx",  
        ...  
    },  
    "e43c2872afe4...": {  
        "IPv4Address": "172.17.0.3/16",  
        "Name": "box",  
        ...  
    }  
}
```

Overlay

- Permet la communication entre des containers situés sur des hôtes différents
- Utilise les fonctionnalités VXLAN du Kernel Linux
- Network créé dans un contexte de cluster
- 2 interfaces réseau dans le container
 - connection au docker_gwbridge
 - connection au réseau overlay via un nouveau bridge bridge
- Paires d'interfaces virtuelles (veth pairs)



<https://success.docker.com>

Overlay

```
root@node01:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 42:ef:4b:71:82:85 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 16:32:9d:23:8a:35 brd ff:ff:ff:ff:ff:ff
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:eb:94:c3:7a brd ff:ff:ff:ff:ff:ff

root@node01:~# docker swarm init

root@node01:~# ip link
...
10: docker_gwbridge: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:f3:22:ec:ea brd ff:ff:ff:ff:ff:ff
12: veth23c1b81@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker_gwbridge state UP
mode DEFAULT group default
    link/ether 3a:42:9c:34:9e:01 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

Overlay

```
root@node01:~# docker network create --driver overlay ovnet
Ptk11nhj82v5j4b2m6kas4d2p

root@node01:~# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
efb245433581    bridge    bridge      local
55b9f342458b    docker_gwbridge    bridge      local
5615defdbf54    host      host      local
lk2s876fiaas    ingress    overlay      swarm
993636a4a90a    none      null      local
ptk11nhj82v5    ovnet     overlay      swarm

root@node01:~# docker service create --network ovnet alpine sleep 10000
pynw2rd27xrdz9sv34kubfk1n

root@node01:~# ip link
10: docker_gwbridge: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
12: veth23c1b81@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker_gwbridge state UP mode DEFAULT group default
17: vethd5aa0b6@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker_gwbridge state UP mode DEFAULT group default
```

Overlay

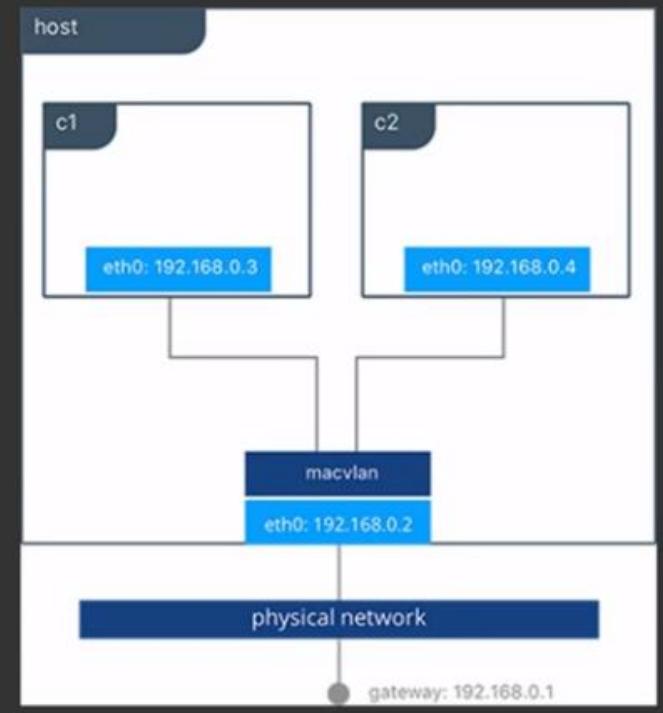
```
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue state UP
    link/ether 02:42:0a:00:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet 10.0.0.2/32 scope global eth0
        valid_lft forever preferred_lft forever
16: eth1@if17: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.3/16 scope global eth1
        valid_lft forever preferred_lft forever
```

Interface permettant la connection au network overlay

Interface permettant la connection au docker_gwbridge

Macvlan

- Pas d'utilisation de bridge
- Léger et dédié à la performance
- Donne à un container un accès direct à une interface de la machine hôte
- IP routable pour chaque container



Macvlan

```
# Creation d'un network macvlan basé sur l'interface parente eth0
$ docker network create -d macvlan --subnet 192.168.0.0/24 --gateway 192.168.0.1 -o parent=eth0 mvnet

# Creation de containers sur le network "mvnet"
$ docker run -it --name c1 --net mvnet --ip 192.168.0.3 busybox sh
$ docker run -it --name c2 --net mvnet --ip 192.168.0.4 busybox sh

# Communication entre les containers
/ # ping 192.168.0.4
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.052 ms
...
/ # ping c2
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.052 ms
...
```

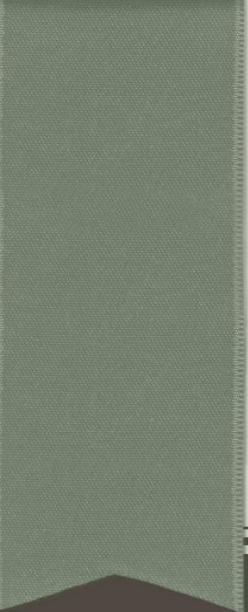
Disposition de titre et de contenu avec liste

TP

Pratique des networks

Disposition de titre et de contenu avec liste

Question ?



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`
 - ...

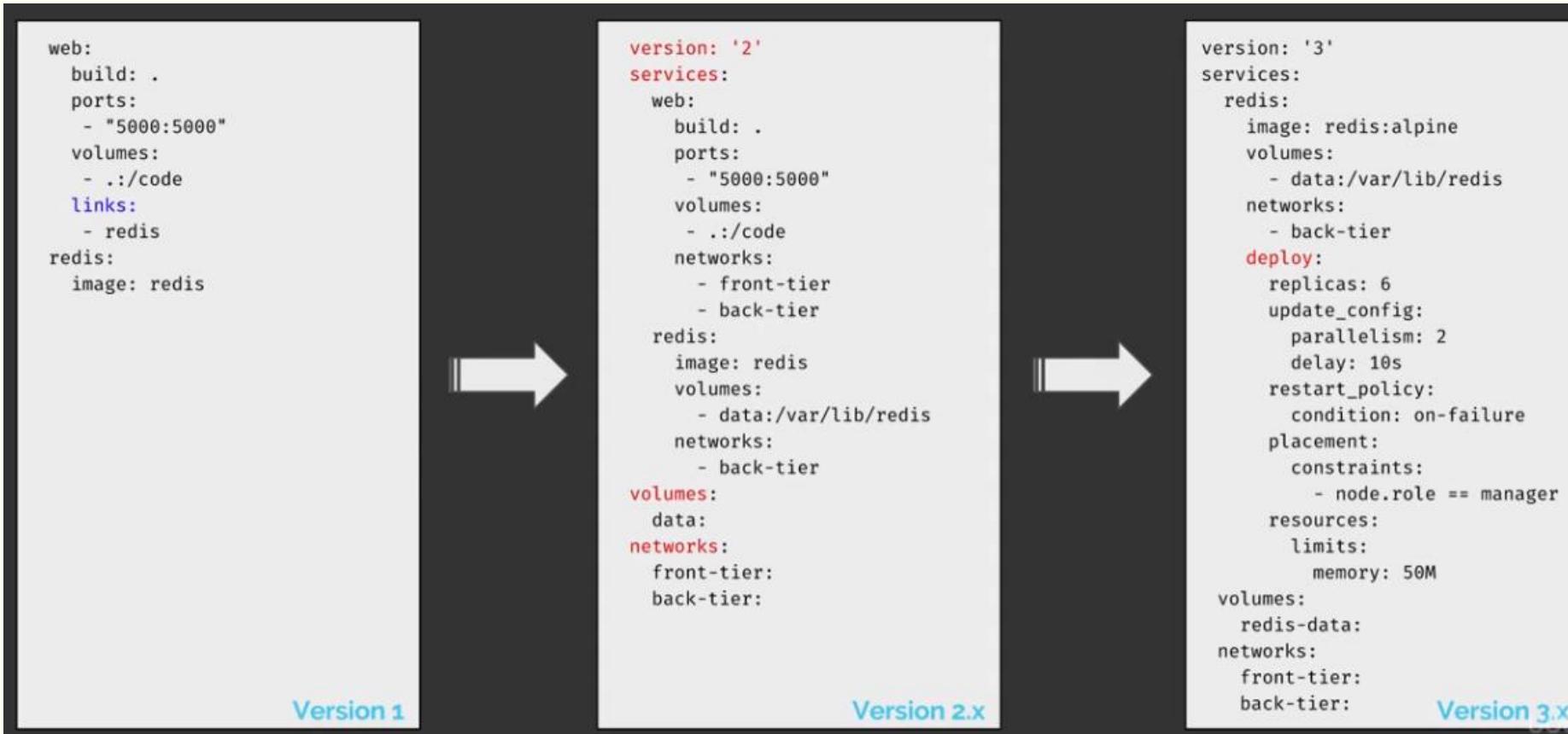
Le fichier docker-compose.yml

- 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

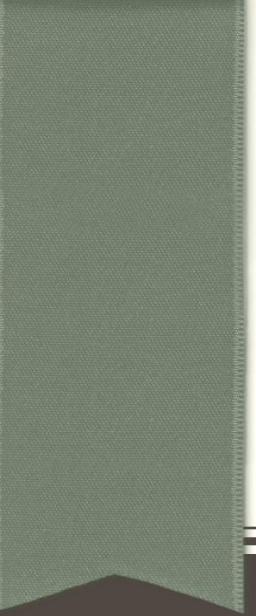
Docker Compose 3 : ELK

TP

Docker Compose 4 : Voting App

Disposition de titre et de contenu avec liste

Question ?



MODULE 8

Docker Swarm

Présentation

- Historique

Historique

- Utilisation d'un key-value store externe (Consul / Etcd / Zookeeper)
- Déploiement complexe
 - options spécifiques du Docker daemon
- Setup manuel des éléments de sécurité
 - certificats
 - clés privées / clés publiques

Historique

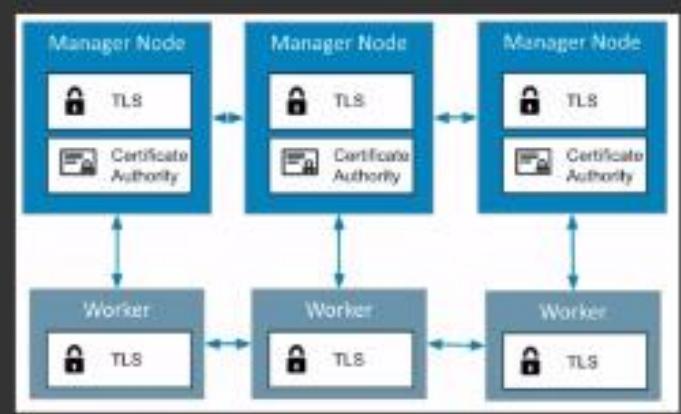
Historique: le cluster Swarm à partir de Docker 1.12

- Disponible depuis Juin 2016
- Orchestration intégrée au Docker daemon
- Swarm mode
- Sécurisé par défaut
- Très facile à mettre en place
- Accessible sur un poste de développement

Swarm mode

Swarm mode

- Orchestrateur
- Intégré au daemon Docker
- Algorithme de consensus basé sur Raft
- Communication entre nodes sécurisée par défaut
- Les primitives
 - Node: machine membre d'un cluster Swarm
 - Service: définition de la façon de lancer les containers d'une application
 - Stack: groupe de services



<https://blog.docker.com>

Swarm mode

- Boucle de reconciliation
- Rolling upgrade
- IP Virtuelle associée à un service
- Load balancing au niveau 4
- Permet le déploiement d'application au format Docker Compose
- Fonctionnalité d'Autolock pour la sécurisation du cluster

Swarm mode

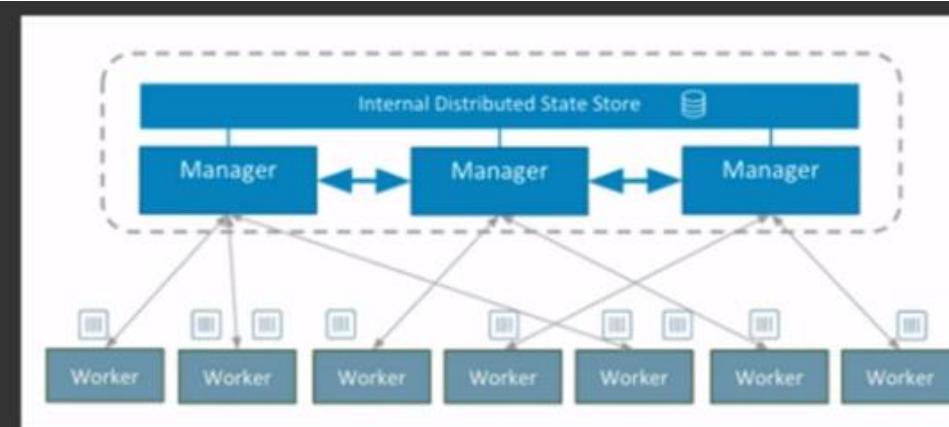
```
$ docker swarm --help

Usage: docker swarm COMMAND
Manage Swarm
Options:
  --help    Print usage
Commands:
  init      Initialize a swarm
  join      Join a swarm as a node and/or manager
  join-token Manage join tokens
  leave     Leave the swarm
  unlock    Unlock swarm
  unlock-key Manage the unlock key
  update    Update the swarm
```

- Création d'un Swarm : *docker swarm init*
- Ajout d'un node: *docker swarm join*
- Management des clés d'encryption

Node

- Machine faisant partie du Swarm
- Manager
 - schedule et orchestre les containers
 - gère l'état du cluster
 - algorithme de consensus RAFT
- Worker
 - exécute les containers
 - protocol Gossip entre les workers
- Un manager est aussi un Worker
- Communication Manager → Worker
 - protocol gRPC (basé sur HTTP/2 et Protocol Buffers)



<https://blog.docker.com>

Node

- Active
 - le scheduler peut assigner des tâches à ce node
- Pause
 - le scheduler ne peut pas assigner de nouvelles tâches à ce node
 - les tâches tournant sur ce node sont inchangées
- Drain
 - le scheduler ne peut pas assigner de nouvelles tâches à ce node
 - les tâches tournant sur ce node sont stoppées et relancées sur d'autres nodes

Node

```
$ docker node --help

Usage:docker node COMMAND

Manage Docker Swarm nodes

Options:
    --help    Print usage

Commands:
    demote      Demote one or more nodes from manager in the swarm
    inspect     Display detailed information on one or more nodes
    ls          List nodes in the swarm
    promote     Promote one or more nodes to manager in the swarm
    rm          Remove one or more nodes from the swarm
    ps          List tasks running on a node
    update     Update a node

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

Gestion d'un Swarm

```
docker@node1:~$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (3gd0odtwxxp6nvomf3xfv05ww) is now a manager.
To add a worker to this swarm, run the following command:
  docker swarm join \
    --token SWMTKN-1-07990ineggmmqgp2zdh7w85y26wd494wwa0f0hjqckim1jq11x-78dzw3pq8drzgq5b1377rsre4 \
    192.168.99.110:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
docker@node1:~$ docker node ls
ID                  HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
3gd0odtwxxp6nvomf3xfv05ww *  node1     Ready   Active        Leader
```

- Le node sur lequel le Swarm est initialisé devient Leader
- Les Instructions pour ajouter des nodes supplémentaires sont fournies

Note: l'option **--advertise-addr** spécifie l'interface réseau utilisée pour que le leader soit accessible. L'option est obligatoire si il y a plusieurs interfaces réseau.

Gestion d'un Swarm

- Machine pouvant communiquer avec le manager du Swarm
- Les commandes Docker doivent être lancées sur un manager

```
docker@node2:~$ docker swarm join --token
SWMTKN-1-07990ineggmmqgp2zdh7w85y26wd494wwa0f0hjqckim1jq11x-78dzw3pq8drzgq5b1377rsre4
192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@node1:~$ docker node ls
ID                      HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
lf2kfkj0442vplsxmgr05lxjq *  node1    Ready   Active        Leader
pes38pa6w8xteze6dbj9lxx54  node2    Ready   Active
```

```
docker@node2:~$ docker node ls
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or
modify cluster state. Please run this command on a manager node or promote the current node to a
manager.
```

Gestion d'un Swarm

Gestion d'un Swarm: ajout d'un manager

- Récupération d'un token depuis le manager
- Un manager non Leader a le statut Reachable

```
docker@node1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:
  docker swarm join \
    --token SWMTKN-1-07990ineggmmqgp2zdh7w85y26wd494wwa0f0hjqckim1jq11x-auytk3hyxqijes0mpanl5yax1 \
    192.168.99.100:2377

docker@node3:~$ docker swarm join --token
SWMTKN-1-07990ineggmmqgp2zdh7w85y26wd494wwa0f0hjqckim1jq11x-auytk3hyxqijes0mpanl5yax1 192.168.99.100:2377
This node joined a swarm as a manager.

docker@node1:~$ docker node ls
ID                      HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
1m9rdh9hlkktiwaz3svweeppp  node3    Ready   Active        Reachable
lf2kfkj0442vplsxmgr05lxjq *  node1    Ready   Active        Leader
pes38pa6w8xteze6dbj9lxx54  node2    Ready   Active
```

Gestion d'un Swarm

Gestion du Swarm: promotion / destitution d'un node

- Commandes lancées depuis un manager

```
# Destitution d'un node manager en worker  
docker@node1:~$ docker node demote node1  
Manager node1 demoted in the swarm.
```

```
# Promotion d'un node worker en manager  
docker@node3:~$ docker node promote node2  
Node node2 promoted to a manager in the swarm.
```

```
# Etat du Swarm  
docker@node2:~$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
1m9rdh9hlkktiwaz3svweeppp	node3	Ready	Active	Leader
lf2kfkj0442vplsxmgr05lxjq *	node1	Ready	Active	
pes38pa6w8xteze6dbj9lxx54	node2	Ready	Active	Reachable

Gestion d'un Swarm

Gestion du Swarm: availability

- Une valeur parmi: Active / Pause / Drain
- Contrôle le déploiement des tâches sur un node

```
# Le node2 ne pourra plus recevoir de nouvelles tâches
docker@node1:~$ docker node update --availability pause node-2

# Les tâches du node2 seront schedulees sur d'autres node du cluster
docker@node1:~$ docker node update --availability drain node-2

# Le node2 repasse en mode actif et pourra recevoir de nouvelles tâches
docker@node1:~$ docker node update --availability active node-2
```

Disposition de titre et de contenu avec liste

TP

Swarm cluster commandes

TP

Set up docker swarm cluster

Service

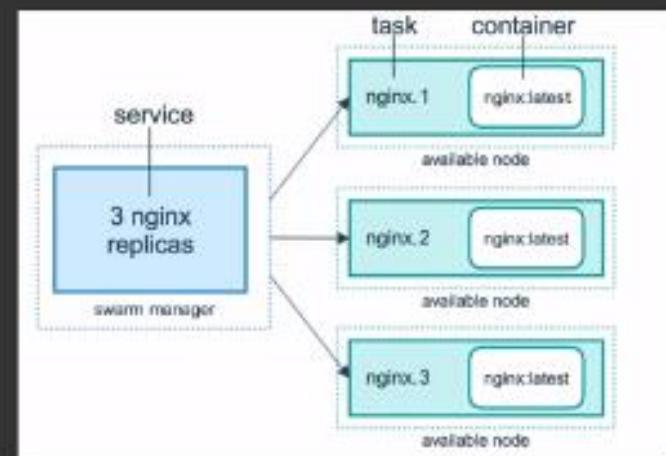
- Définition de la façon de lancer les containers d'une application
- Boucle de réconciliation
- Mode *répliqué* ou *global*
- Adresse IP virtuelle associée à chaque service
- Load balancer de niveau 4 (IPVS)
- Publication d'un port / Routing Mesh

Service

- Configuration
 - image utilisée
 - nombre de réplicas
 - ports exposés
 - stratégie de redémarrage
 - contraintes de déploiement
 - configuration des mises à jour
 - Health check
- https://docs.docker.com/engine/reference/commandline/service_create/

Service

- Définition d'un nombre de réplicas
- Instancié en une ou plusieurs tâches
- Chaque tâche est déployée sur un node
- Une tâche exécute un container



<https://docs.docker.com/engine/swarm>

Service

Service: CLI

```
$ docker service --help

Usage:docker service COMMAND

Manage services

Options:
  --help  Print usage

Commands:
  create      Create a new service
  inspect     Display detailed information on one or more services
  ls          List services
  ps          List the tasks of a service
  rm          Remove one or more services
  scale       Scale one or multiple replicated services
  update     Update a service

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

Service

Service: exemple d'un serveur HTTP

Image nginx avec déclaration de 3 réplicas

```
$ docker service create --name www -p 8080:80 --replicas 3 nginx
sux9upku57t1tvwca15svoich

$ docker service ls
ID           NAME    MODE      REPLICAS  IMAGE
sux9upku57t1  www     replicated  3/3      nginx:latest

$ docker service ps www
ID           NAME    IMAGE      NODE    DESIRED STATE  CURRENT STATE          ERROR  PORTS
ubi81q9y1f28  www.1  nginx:latest  node2   Running       Running 56 seconds ago
yoxlel0bwdaa  www.2  nginx:latest  node1   Running       Running 7 minutes ago
fwfssuhokp6e  www.3  nginx:latest  node3   Running       Running 56 seconds ago

$ docker service scale www=1
www scaled to 1

$ docker service ps www
ID           NAME    IMAGE      NODE    DESIRED STATE  CURRENT STATE          ERROR  PORTS
fwfssuhokp6e  www.3  nginx:latest  node1   Running       Running about a minute ago
```

Service

Service: exemple d'une base de données

Image mongodb avec utilisation d'un volume

```
$ docker service create \
  --mount type=volume,src=data,dst=/data/db \
  --name db \
  --publish 27017:27017 \
  mongo:3.4

$ docker service ls
ID          NAME      MODE      REPLICAS      IMAGE      PORTS
zicokstoycg9  db       replicated  1/1        mongo:3.4    *:27017->27017/tcp
```

```
$ docker volume ls
DRIVER      VOLUME NAME
local      86920eb910b7b346fd670efabbbd2e38851fa5ef1245bb40e395aaf6aff9777d
local      data
```

Volume créé par l'instruction mount

Volume défini dans le Dockerfile de mongo:
VOLUME /data/configdb

Service: rolling upgrade

- Gestion des mises à jour du service

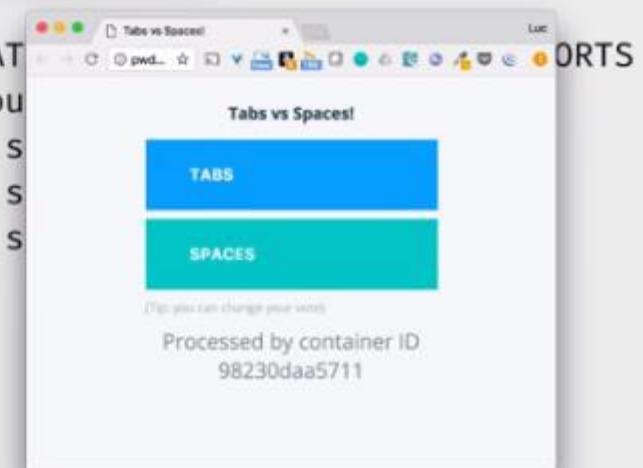
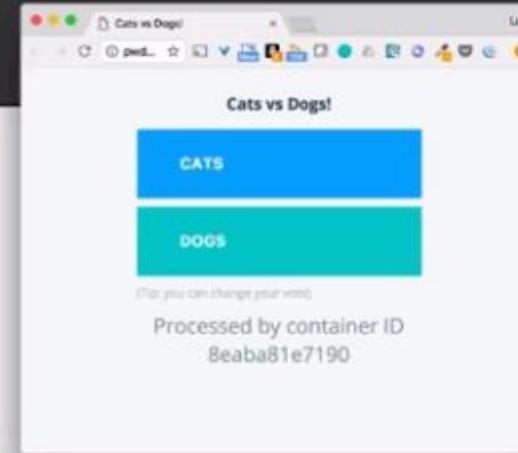
```
$ docker service create  
  --update-parallelism 2 \\\n  --update-delay 10s \\\n  --publish 8080:80 \\\n  --name vote \\\n  instavote/vote
```

```
$ docker service scale vote=4
```

```
$ docker service ps vote
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
detcfbxk72s0	vote.1	instavote/vote:latest	node1	Running	Running about 1 min
wvo6kl3nj21k	vote.2	instavote/vote:latest	node1	Running	Running 59 s
xo3vyvvbleit	vote.3	instavote/vote:latest	node1	Running	Running 59 s
zfk296jpqj9w	vote.4	instavote/vote:latest	node1	Running	Running 59 s

```
$ docker service update --image instavote/vote:indent vote
```



Disposition de titre et de contenu avec liste

TP

Create and Deploy nginx
service over swarm

TP

Déploiement de services

Stack

- Un groupe de services
- Déployée à partir d'un fichier au format Docker Compose

```
$ docker stack --help

Usage:docker stack COMMAND
Manage Docker stacks
Options:
    --help    Print usage

Commands:
    deploy      Deploy a new stack or update an existing stack
    ls          List stacks
    ps          List the tasks in the stack
    rm          Remove the stack
    services    List the services in the stack

Run 'docker stack COMMAND --help' for more information on a comma
```

Stack

Stack: example with the Voting App

```
docker@node1:~$ git clone https://github.com/docker/example-voting-app/ && cd example-voting-app

docker@node1:~$ docker stack deploy -c docker-stack.yml vote
Creating network vote_backend
Creating network vote_default
Creating network vote_frontend
Creating service vote_worker
Creating service vote_visualizer
Creating service vote_redis
Creating service vote_db
Creating service vote_vote
Creating service vote_result

docker@node1:~$ docker stack ls
NAME      SERVICES
vote      6

docker@node1:~$ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
0b7wliblok8l  vote_vote  replicated  2/2      dockersamples/examplevotingapp_vote:before
nwl97sw2xci5  vote_redis  replicated  2/2      redis:alpine
oa3k3lqhpkle  vote_visualizer  replicated  1/1      dockersamples/visualizer:stable
otmy5huzv333  vote_worker  replicated  1/1      dockersamples/examplevotingapp_worker:latest
oub9r3pwunqd  vote_db    replicated  1/1      postgres:9.4
w1aek7v8f059  vote_result  replicated  1/1      dockersamples/examplevotingapp_result:before
```



TP

Stack in PWD

Disposition de titre et de contenu avec liste

Question ?

Reference

- <https://www.scalair.fr/blog/docker>
- <https://www.ionos.fr/digitalguide/serveur/configuration/tutoriel-docker-installation-et-premiers-pas/>
- <https://pixelabs.fr/installation-docker-sous-windows-10/>
- <https://www.wanadev.fr/24-tuto-docker-demarrer-docker-partie-2/>