

Docker

Virtualisation légère

# DOCKER

## Module 1: Introduction

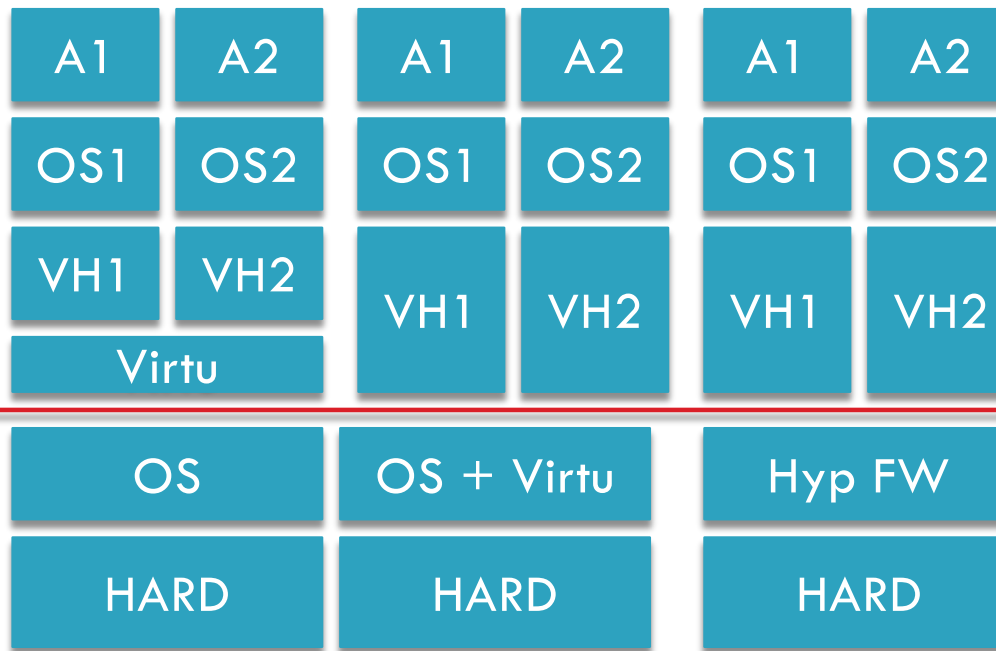
# Problématique: Consolidation des serveurs

## Partionnement



Partitionnement  
Physique

## Virtualisation de Hardware

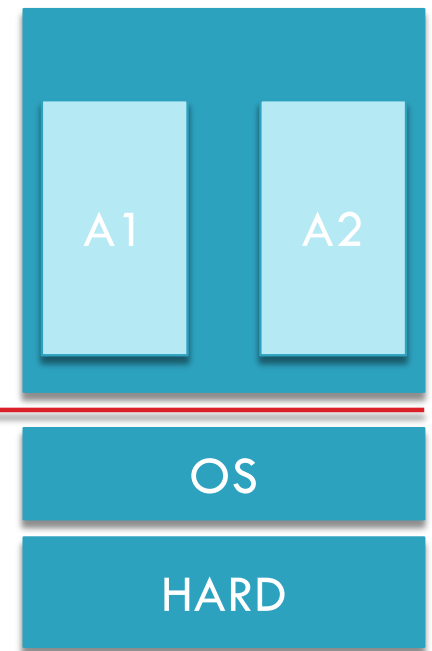


Host Based

Hyperviseur  
Bare Metal

Hyperviseur  
FirmWare

## Virtualisation d' OS

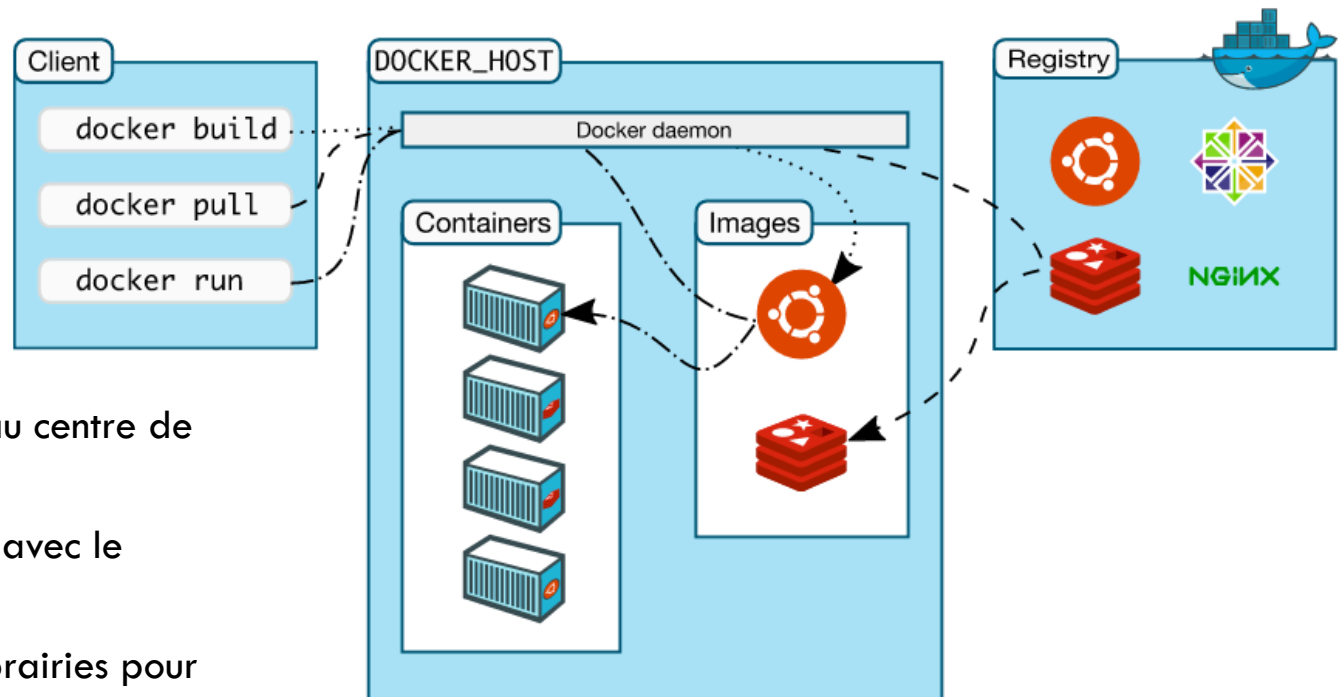


Conteneur

# Architecture de Docker

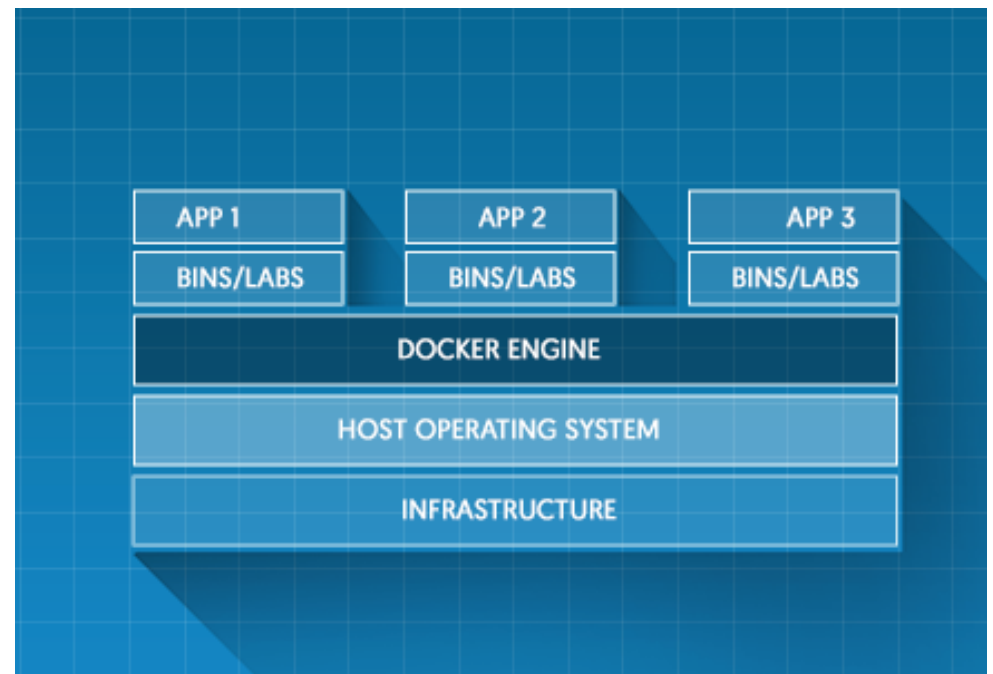
Docker est basé sur une architecture client-serveur :

- ❑ Docker Engine : le Docker serveur au centre de l'architecture.
- ❑ Client : interface CLI pour interagir avec le serveur.
- ❑ Image : image de binaires et de bibliothèques pour exécuter des applications
- ❑ Container : une instance d'une image en exécution
- ❑ Registry : bibliothèque d'images



# Docker Engine

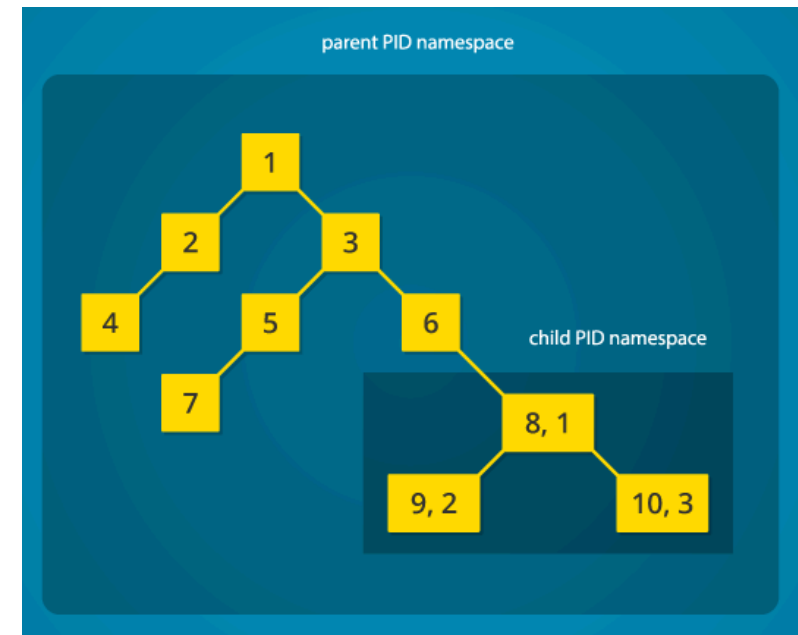
- Docker Engine est le daemon serveur au centre de l'architecture qui permet d'exécuter les conteneurs.
- Docker Engine utilise les namespaces Linux Kernel et les groupes de contrôle.
- L'isolation des containers est garantie par les namespaces.



# Namespaces

## ▣ Namespaces

- Isolation d'une hiérarchie de processus
- Dans un namespace, seuls certains processus sont visibles



# Cgroups



## ▣ Cgroups

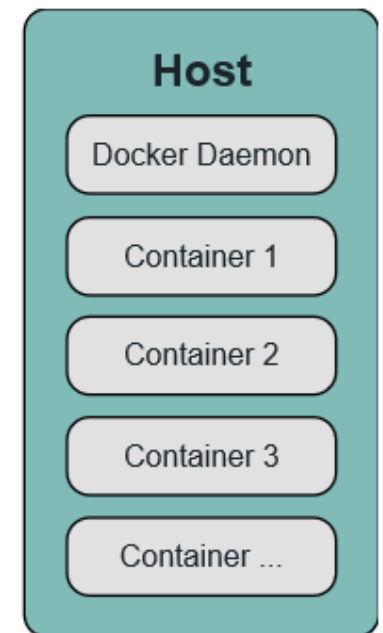
- Contrôler l'accès aux ressources CPU et RAM à un ensemble de processus.
- Capping des ressources RAM et CPU des containers.

# Docker Client

- ❑ Le client Docker se connecte au Docker Engine.
- ❑ Le client peut-être installé sur la même machine que le daemon ou sur une machine différente.
- ❑ Deux types de client Docker :
  - CLI
  - GUI

## Docker Client

`docker pull`  
`docker run`  
`docker ...`





# Images



- ▣ Modèles en lecture seule utilisés pour créer des conteneurs.
- ▣ Construites par vous ou d'autres utilisateurs Docker
- ▣ Stockées dans Docker Hub, Docker de confiance du Registre ou votre propre registre
- ▣ Basé sur une ou plusieurs images.

# Conteneur



- ▣ Espace d'exécution d'application isolé.
- ▣ Contient tout ce qui est nécessaire pour exécuter votre application.
  - Binaires de l'application
  - Bibliothèques requises par l'application

# Registres



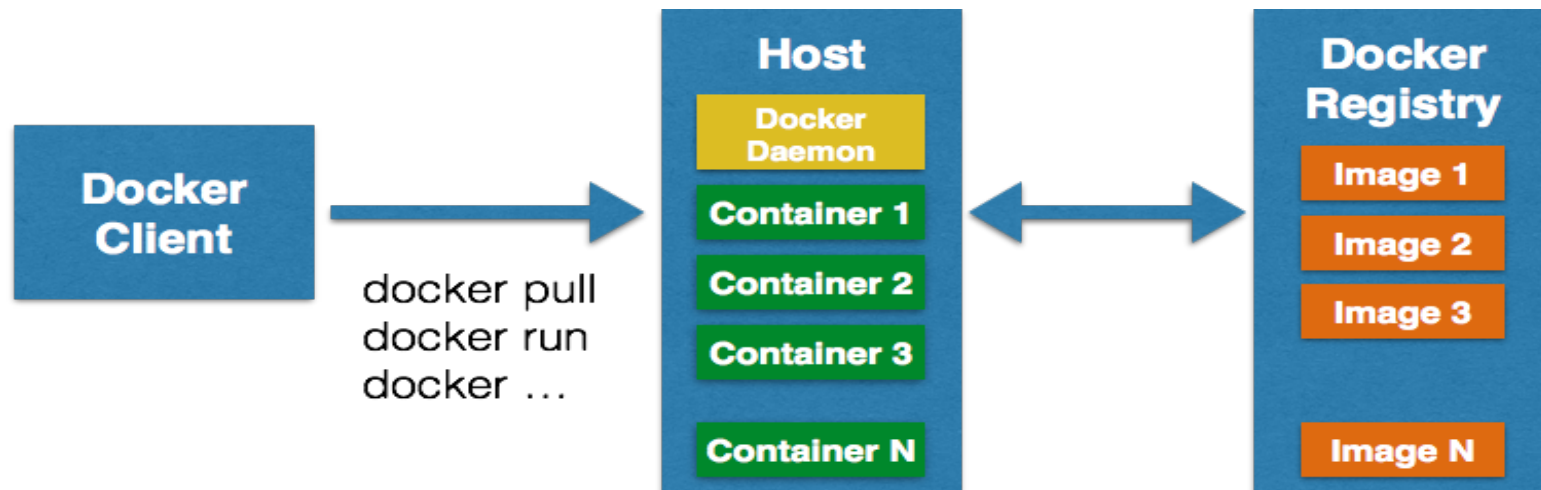
- ▣ Stockages des images.

- Registre Local
- Registre Distant

- ▣ Accès aux images

- Registre Privé.
- Registre Public

# Client/Daemon/Conteneur/Images/Registres



# Docker Orchestration

- Trois outils pour orchestrer des applications distribuées avec Docker
  - Docker machine
    - Outil qui provisionne les Docker hôtes et installe le Docker Engine
  - Docker Swarm
    - Cluster de Docker hôtes et ordonnancement des conteneurs
  - Docker Compose
    - Outil pour créer et gérer des applications multi-conteneurs

# Docker Toolbox

- ❑ Docker Engine ne fonctionne pas en mode natif sur Windows et Mac OSX
- ❑ Docker Toolbox configure ceci :
  - Oracle VirtualBox pour l'exécution d'une VM Linux
  - Docker machine
  - Docker Engine
  - Docker Compose
  - Un Shell pré-configuré pour le client CLI
  - Le client GUI Kitematic

# Docker Enterprise Edition



- Docker EE est une offre commerciale et de support de Docker pour gérer votre propre infrastructure CaaS
- Intégration de développement avec Docker Toolbox
- Images sur Docker Registre de confiance
- Containers déployé à l'aide de UCP Universal Control Plane
- Support de l'API Docker
- Support de Docker Engine (CS Engine)

# Docker Trusted Registry

- ▣ Registre Docker de confiance installé sur votre infrastructure
- ▣ Il comprend
  - Image de registre pour stocker des images
  - les pilotes de stockage
  - Web GUI pour l'administration
  - Mises à jour faciles et transparentes
  - Journalisation



# Points forts de Docker



- ▣ Cycle de développement rapide
- ▣ Portabilité et Evolutivité
- ▣ Plusieurs application sur une machine
- ▣ Facilité de charger et de démarrer un Container, puis d'y lancer un processus.
- ▣ Automatisation du chargement et du lancement d'un Container.
- ▣ Mise à jour et diffusion facile et rapide.

# DOCKER

## Module 2: Installation

# Pré-requis

- ▣ Linux 64 bits.
- ▣ Kernel minimum 3.10.

```
$ uname -r  
4.2.0-27-generic
```

- ▣ Installation par un script Docker.
- ▣ Installation par packages.
- ▣ Mac OSX
- ▣ Windows (boot2docker)

# Ubuntu/Debian



- ▣ Mise à jour des apt sources
- ▣ Ajout d'une nouvelle clé GPG
- ▣ Ajout de l'URL source de distribution de Docker
- ▣ Installation du package `docker-engine`
- ▣ Démarrage du service docker

# Exemple d'installation Ubuntu

```
$ sudo apt-get update
```

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

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) stable"
```

```
$ sudo apt-get install docker-ce
```

# RHEL/Centos



- ▣ Mise à jour des packages
- ▣ Ajout de l'URL source de distribution de Docker
- ▣ Installation du package `docker-ce`
- ▣ Démarrage du service docker

# Exemple d'installation CentOS

```
$ sudo yum update
```

```
$ sudo yum-config-manager --add-repo \  
https://download.docker.com/linux/centos/docker-ce.repo  
Loaded plugins: fastestmirror  
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo  
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo  
repo saved to /etc/yum.repos.d/docker-ce.repo
```

```
$ sudo yum list docker-ce.x86_64
```

```
Available Packages  
docker-ce.x86_64                                17.03.1.ce-1.el7.centos                docker-ce-stable
```

```
$ sudo yum install docker-ce-17.03.1.ce-1.el7.centos.x86_64
```

```
Installed:  
  docker-ce.x86_64 0:17.03.1.ce-1.el7.centos
```

```
$ sudo systemctl start docker
```

```
$ sudo systemctl enable docker
```

# Le groupe docker

- ▣ Pour exécuter les commandes `docker` sans nécessiter `sudo`, ajoutez votre compte d'utilisateur au groupe `docker`

```
$ sudo usermod -aG docker <user>
```

- ▣ Le groupe `docker` est créé automatiquement



# Vérification de l'installation

## ▣ Exécuter un premier conteneur

```
$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
.../...
```

## ▣ Exécuter la commande docker --version

```
$ docker --version
Docker version 17.03.1-ce, build c6d412e
```

# Vérification de l'installation

## **\$ docker version**

### Client:

Version: 17.03.1-ce  
API version: 1.27  
Go version: go1.7.5  
Git commit: c6d412e  
Built: Mon Mar 27 17:10:36 2017  
OS/Arch: linux/amd64

### Server:

Version: 17.03.1-ce  
API version: 1.27 (minimum version 1.12)  
Go version: go1.7.5  
Git commit: c6d412e  
Built: Mon Mar 27 17:10:36 2017  
OS/Arch: linux/amd64  
Experimental: false

# Installation Mac OS X

## ▣ Docker Toolbox

- Docker Machine, Docker Compose, Docker CLI & GUI (Kitematic) sont natifs
- Docker Engine s'exécute dans une VM virtualbox

## ▣ Docker for Mac

- Docker Engine s'exécute dans une VM avec hyperviseur natif OS X (xhyve)
- A partir de Yosemite (OS X 10.10)

# Installation Windows

## ▣ Docker Toolbox

- Docker Machine, Docker Compose, Docker CLI & GUI (Kitematic) sont natifs
- Docker Engine s'exécute dans une VM virtualbox

## ▣ Docker for Windows

- Docker Engine s'exécute dans une VM Hyper-V
- A partir de Windows 10


# DOCKER

Module 3: Images Docker


# Docker Hub

## ■ Docker Hub

- Site officiel de Docker Inc
- Images dans plusieurs registres
- Images officielles
- Images non officielles






[Explore](#) [Help](#)



Search

[Sign up](#) [Log In](#)

Explore Official Repositories

	nginx official	4.2K STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>
	busybox official	808 STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>
	redis official	2.8K STARS	10M+ PULLS	<a href="#">&gt; DETAILS</a>

# Dépôts officiels

- Les dépôts officiels sont certifiés et organisés en dépôts Docker qui sont déposés sur le Docker Hub
- Les dépôts proviennent de fournisseurs tels que NGINX, Ubuntu, Red Hat, Redis, etc ...
- Les images sont supportées par leurs éditeurs, optimisées et à jour
- Les images officielles du référentiel sont :
  - Images de systèmes d'exploitation Linux (Ubuntu, CentOS etc ...)
  - Images d'outils de développement, de langages de programmation, d'applications, de bases de données.

# Recherche d'images

## ▣ Syntaxe

```
docker search [OPTIONS] image
```

## ▣ Options

```
--filter <stars=#> <is-automated=true|false> <is-official=true|false>  
--limit=LIMIT  
--no-trunc=true|false
```

### \$ docker search nginx

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
nginx	Official build of Nginx.	4183	[OK]	
jwilder/nginx-proxy	Automated Nginx reverse proxy for docker c...	801		[OK]
richarvey/nginx-php-fpm	Container running Nginx + PHP-FPM capable ...	274		[OK]
million12/nginx-php	Nginx + PHP-FPM 5.5, 5.6, 7.0 (NG), CentOS...	76		[OK]
maxexcloo/nginx-php	Framework container with nginx and PHP-FPM...	58		[OK]
webdevops/php-nginx	Nginx with PHP-FPM	53		[OK]
h3nrik/nginx-ldap	NGINX web server with LDAP/AD, SSL and pro...	29		[OK]
bitnami/nginx	Bitnami nginx Docker Image	18		[OK]
evild/alpine-nginx	Minimalistic Docker image with Nginx	8		[OK]
million12/nginx	Nginx: extensible, nicely tuned for better...	8		[OK]
gists/nginx	Nginx on Alpine	8		[OK]
maxexcloo/nginx	Framework container with nginx installed.	7		[OK]



# Images locales

- ▣ Syntaxe

`docker images [OPTIONS]`

- ▣ Options

`--digests`

`--filter "label=string"` ou `"before=image"` ou `"since=image"`

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	ba6bed934df2	18 hours ago	181.4 MB
ubuntu	latest	45bc58500fa3	4 days ago	126.9 MB
hello-world	latest	c54a2cc56cbb	12 weeks ago	1.848 kB

# Images locales

- ❑ Lors de la création d'un conteneur, Docker va tenter d'utiliser en priorité une image locale.
- ❑ Si aucune image locale n'est trouvée, Docker télécharge l'image à partir de Docker Hub, sauf si un autre registre est spécifié

```
$ docker run nginx
```

```
Unable to find image 'nginx:latest' locally
```

```
latest: Pulling from library/nginx
```

```
6a5a5368e0c2: Downloading [=====>
```

```
] 12.06 MB/51.35 MB
```

```
4aceccff346f: Downloading [=====>
```

```
] 15.76 MB/20.09 MB
```

```
c8967f302193: Download complete
```

# Balise d'une image : tag

- ▣ Les images sont spécifiées par `repository:tag`
- ▣ La même image peut avoir plusieurs balises
- ▣ La balise par défaut est `last`

OFFICIAL REPOSITORY  
**centos** ☆  
Last pushed: 18 days ago

[Repo Info](#) [Tags](#)

Tag Name	Compressed Size	Last Updated
centos5.11	87 MB	18 days ago
5.11	87 MB	18 days ago
6.6	72 MB	18 days ago
centos6.6	72 MB	18 days ago

# Charger une Image

- Utiliser la commande `pull` pour télécharger une image de Docker Hub ou de tout registre.

```
$ docker pull ubuntu
```

```
Using default tag: latest
```

```
latest: Pulling from library/ubuntu
```

```
ff1f1f1de862: Downloading [=====>
```

```
] 30.47 MB/49.79 MB
```

```
0c7b035e2a1a: Download complete
```

```
ac8ee255ff41: Download complete
```

```
$ docker pull ubuntu:12.04
```

```
12.04: Pulling from library/ubuntu
```

```
4bae8cb7faf8: Downloading [=====>
```

```
] 7.47 MB/39.08 MB
```

```
9f6907f8c14c: Download complete
```

```
66f8c8a8de76: Download complete
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	ba6bed934df2	22 hours ago	181.4 MB
ubuntu	latest	45bc58500fa3	5 days ago	126.9 MB
ubuntu	12.04	746fb07d2d18	3 weeks ago	103.6 MB

# DOCKER

Module 4: Conteneurs

# Cycle de vie des conteneurs

- Cycle de vie de base d'un conteneur Docker
  - Créer le conteneur à partir d'une image
  - Exécuter le conteneur avec un processus spécifié
  - Le processus se termine et le conteneur s'arrête
  - Détruire le conteneur
  
- Cycle de vie avancé
  - Créer le contenant à partir d'une image
  - Exécuter le conteneur avec un processus spécifié
  - Interagir et effectuer d'autres actions à l'intérieur du conteneur
  - Arrêter le conteneur
  - Redémarrer le conteneur

# Créer et exécuter un conteneur

## ▣ La commande run de docker

- Crée le conteneur en utilisant l'image spécifiée
- Exécute le conteneur

## ▣ Syntaxe

```
docker run [options] image [commande] [args]
```

```
$ docker run ubuntu:12.04 echo "Hello World"  
"Hello World"
```

```
$ docker run ubuntu ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	25976	1452	?	Rs	19:25	0:00	ps aux

# Lister les conteneurs

▣ Utilisez la commande `docker ps` pour lister les conteneurs en cours d'exécution

■ L'option `-a` pour lister tous les conteneurs

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d879calf2e2e	nginx	"nginx -g 'daemon off'"	2 seconds ago	Up 1 seconds	80/tcp, 443/tcp	silly_galileo

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d879calf2e2e	nginx	"nginx -g 'daemon off'"	24 seconds ago	Up 23 seconds	80/tcp, 443/tcp	silly_galileo
ab7378f76d5c	ubuntu	"ps aux"	6 minutes ago	Exited (0) 6 minutes ago		fervent_golick
8101cdf72c16	ubuntu:14.04	"echo "Hello World\xe2"	7 minutes ago	Exited (0) 7 minutes ago		fervent_spence
aa21052c185b	hello-world	"/hello"	9 hours ago	Exited (0) 9 hours ago		boring_noether



# Conteneur avec un terminal

- ▣ L'option `-i` indique à docker de se connecter au `stdin` du conteneur (interactif)
- ▣ L'option `-t` permet d'obtenir un pseudo-terminal

```
$ docker run -it ubuntu bash
root@e367d73cfbfe:/#
root@e367d73cfbfe:/# exit
$
```

- ▣ `CRTL+P+Q` permet de sortir du terminal sans arrêter le conteneur

# Identifiant du conteneur

- ▣ Les conteneurs peuvent être spécifiés en utilisant leur ID ou le nom
- ▣ Court-ID et le nom peuvent être obtenus en utilisant la commande pour lister les conteneurs `docker ps`
- ▣ L'ID long est obtenu avec l'option `--no-trunc`

```
$ docker ps --no-trunc
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8a7960e90d1c5b55f91f4084101ec77ba59284d12e1cbe6c07dd7bb704440237	ubuntu	"bash"	7 minutes ago	Up 7
minutes	elated_hugle			
d879ca1f2e2e065e1680d1543a962d5c6a4c4d9b112ab0578903be2e44827a5d	nginx	"nginx -g 'daemon off;'"	17 minutes ago	Up 17
minutes	80/tcp, 443/tcp	silly_galileo		

# Identifiant du conteneur

- ▣ Pour lister uniquement les courts ID des conteneurs
  - `docker ps -q`
- ▣ Pour lister le dernier conteneur qui a été lancé
  - `docker ps -l`

```
$ docker ps -q
```

```
8a7960e90d1c  
d879calf2e2e
```

```
$ docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8a7960e90d1c	ubuntu	"bash"	9 minutes ago	Up 9 minutes		elated_hugle

```
$ docker run -d -it ubuntu:14.04 ps
```

```
4870e430ae8b15ffb35764d164999658a4d37853c3b7b439cc0e118a0cf435a
```

```
$ docker ps --filter='status=exited'
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4870e430ae8b	ubuntu:14.04	"ps"	4 seconds ago	Exited (0) 4 seconds ago		

```
fervent_stonebraker
```

# Filtrage

## ▣ Filtrage sur le status du conteneur

■ restarting, running, exited, paused

```
$ docker ps -a --filter "exited=1"
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
f10c84fb706f	ubuntu	"bash"	5 seconds ago	Exited (1) 2 seconds ago
distracted_fermi				

# Exécution en mode détaché

- ▣ Exécution en arrière-plan ou comme un daemon
  - Utiliser l'option `-d`
  - Pour observer la sortie standard utiliser `docker logs conteneur`

```
$ docker run -d ubuntu:14.04 ping 127.0.0.1
2b6449372f8ed91e8ea0cb35f028c335ad5c99a4ec04de11cc97d4f1d6a4cb11

$ docker logs 2b6449372f8ed91e8ea0cb35f028c335ad5c99a4ec04de11cc97d4f1d6a4cb11
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.053 ms
```

# Attacher à un conteneur

- Attacher un client à un conteneur passe le conteneur du mode d'exécution arrière-plan en avant-plan
- La sortie standard du conteneur passe sur sur le terminal

```
$ docker run -d -it ubuntu:14.04 ping 127.0.0.1
d6f0525fefbdaf06de652fc524b4d35708f9a36045ff20228680e72bb735b828

$ docker ps -q
d6f0525fefbd

$ docker attach d6f0525fefbd
64 bytes from 127.0.0.1: icmp_seq=33 ttl=64 time=0.054 ms
64 bytes from 127.0.0.1: icmp_seq=34 ttl=64 time=0.097 ms
64 bytes from 127.0.0.1: icmp_seq=35 ttl=64 time=0.054 ms
```

# Détacher un conteneur

- CTRL + P + Q passe le conteneur en mode arrière-plan, sous condition :
  - L'entrée standard du conteneur est connecté
  - Le conteneur a été démarré avec un terminal
- CTRL + C mettra fin au processus, et donc l'arrêt du conteneur

```
$ docker run -it ubuntu:14.04 ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.044 ms
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3eabel85bef5	ubuntu:14.04	"ping 127.0.0.1"	19 seconds ago	Up 18 seconds		sad_shannon

# Exécuter une commande

- ▣ La commande `docker exec` nous permet d'exécuter des processus supplémentaires à l'intérieur d'un conteneur
- ▣ Généralement utilisée pour avoir accès en ligne de commande
- ▣ La sortie du terminal ne terminera pas le conteneur

```
$ docker run -d -it ubuntu:14.04 ping 127.0.0.1
e9b3879cd6db0f74663157aff36e6435c7499be0531ec804ba913de9db18133e
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e9b3879cd6db	ubuntu:14.04	"ping 127.0.0.1"	5 seconds ago	Up 4 seconds		sad_carson

```
$ docker exec -it e9b3879cd6db bash
```

```
root@e9b3879cd6db:/# ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	01:35	?	00:00:00	ping 127.0.0.1
root	6	0	0	01:35	?	00:00:00	bash
root	19	6	0	01:35	?	00:00:00	ps -ef



# Arrêt d'un conteneur

## ❑ `docker stop`

- envoie un signal SIGTERM au processus principal (pid 1)
- Le processus reçoit ensuite un signal SIGKILL après une période de grâce
- La période de grâce peut être spécifiée avec l'option `-t`

## ❑ `docker kill`

- envoie immédiatement un signal SIGKILL au processus principal

```
$ docker run -d -it ubuntu:14.04 ping 127.0.0.1
1dd32afcef10847500fc2c65f01f83723874d53abd6d216a6903fac249bf5867
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
1dd32afcef10        ubuntu:14.04        "ping 127.0.0.1"   7 seconds ago      Up 7 seconds                tiny_mayer
$ docker stop 1dd32afcef10
1dd32afcef10
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
1dd32afcef10        ubuntu:14.04        "ping 127.0.0.1"   31 seconds ago     Exited (137) 3 seconds ago                tiny_mayer
```

# Redémarrage d'un conteneur

- ❑ `docker start` permet de redémarrer un conteneur qui a été arrêté
- ❑ Le conteneur redémarre en utilisant la même commande et options spécifiées auparavant
- ❑ Le conteneur peut être attaché avec l'option `-a`

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
99d2e3c0744b	ubuntu:14.04	"ping 127.0.0.1"	9 minutes ago	Exited (137) 9 minutes ago

```
hopeful_saha  
$ docker start 99d2e3c0744b  
99d2e3c0744b  
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
99d2e3c0744b	ubuntu:14.04	"ping 127.0.0.1"	10 minutes ago	Up 2 seconds	

```
hopeful_saha
```

# Inspecter un conteneur

- ▣ `docker inspect` affiche tous les détails du conteneur
- ▣ Le résultat est sous forme d'un tableau JSON

```
$ docker inspect 99d2e3c0744b
[
  {
    "Id": "99d2e3c0744bb97e471121bb1649057f57c6bf5f834c20352e6cc616bdccee78",
    "Created": "2016-09-25T01:51:29.80119371Z",
    "Path": "ping",
    "Args": [
      "127.0.0.1"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 5226,
      "ExitCode": 0,
      "Error": ""
    }
  }
]
```

# Filterer docker inspect

- ▣ `--format='{{.champ1.champ2}}'`
- ▣ `--format='{{json .champ2}}'`

```
$ docker inspect --format='{{.State.Status}}' 99d2e3c0744b
Running
```

```
$ docker inspect --format='{{.NetworkSettings.IPAddress}}' 99d2e3c0744b
172.17.0.2
```

```
$ docker inspect --format='{{.State}}' 99d2e3c0744b
{running true false false false false 5226 0 2016-09-25T02:01:37.725256781Z 2016-09-25T01:51:44.11869103Z
<nil>}
```

```
$ docker inspect --format='{{json .State}}' 99d2e3c0744b
{"Status":"running","Running":true,"Paused":false,"Restarting":false,"OOMKilled":false,"Dead":false,"Pid":5226
,"ExitCode":0,"Error":"","StartedAt":"2016-09-25T02:01:37.725256781Z","FinishedAt":"2016-09-
25T01:51:44.11869103Z"}
```

# Supprimer un conteneur

- ▣ On ne peut supprimer que les conteneurs arrêtés
  - Utilisez la commande `docker rm`
  - Indiquez l'ID ou le nom du conteneur
  - Utilisez l'option `-f` (force) pour supprimer un conteneur en cours d'exécution,

```
$ docker rm 99d2e3c0744b
Error response from daemon: You cannot remove a running container
99d2e3c0744bb97e471121bb1649057f57c6bf5f834c20352e6cc616bdccee78. Stop the container before attempting removal
or use -f

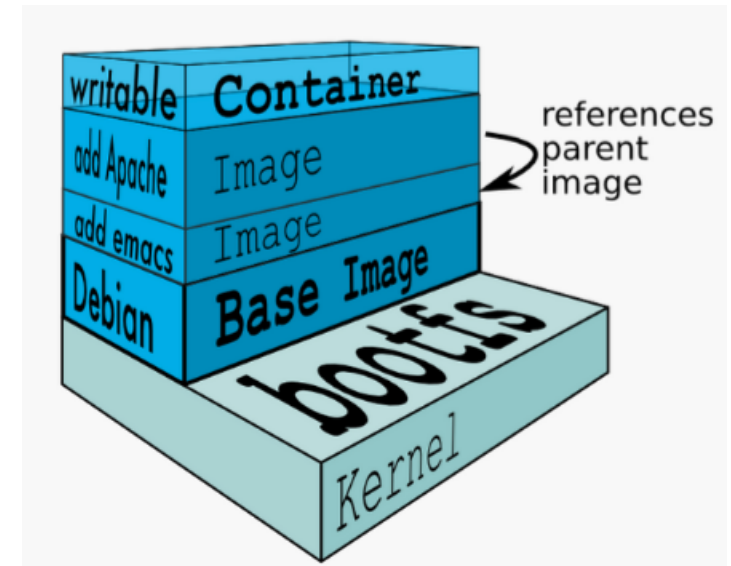
$ docker rm -f 99d2e3c0744b
99d2e3c0744b
```

# DOCKER

## Module 5: Images

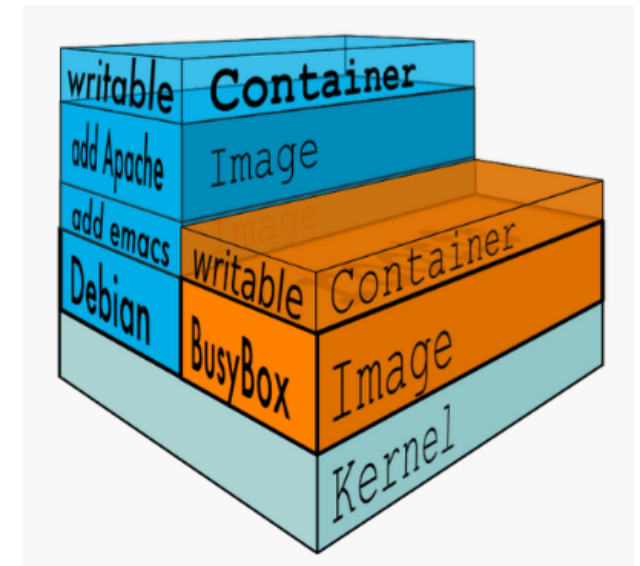
# Docker Layers : couches

- Une image est une collection de fichiers et de méta-données
- Les images sont composées de plusieurs couches
- Une couche est également une image
- Chaque image contient le logiciel que vous souhaitez exécuter
- Chaque image contient une couche de base
- Docker utilise un mécanisme Copy On Write
- Les couches sont en lecture seule



# Docker Layers : couches

- ❑ Docker crée une couche supérieure en écriture pour les conteneurs
- ❑ Les images parentes sont en lecture seule
- ❑ Toutes les modifications sont apportées à la couche en écriture
- ❑ Lors de la modification d'un fichier à partir d'une couche en lecture, le mécanisme COW copie le fichier sur la couche en écriture, ce qui permet de modifier le fichier





# Création d'images

## ■ Trois méthodes

- Valider les modifications d'un conteneur en tant que nouvelle image
  - Nous permet de construire des images de manière interactive
  - Obtenir l'accès terminal dans un conteneur et installer les programmes nécessaires et votre application
  - Ensuite, enregistrez le conteneur comme une nouvelle image en utilisant la commande `docker commit`
- Construire à partir d'un Dockerfile
- Importer une archive dans Docker comme une couche de base autonome

# Création d'images

## ▣ Créer le conteneur, installer l'application

```
$ docker run -it centos bash
[root@8cdeef2d83c8 /]# wget
bash: wget: command not found
[root@8cdeef2d83c8 /]# yum install -y wget
Loaded plugins: fastestmirror, ovl
base                                     | 3.6 kB  00:00:00
extras                                 | 3.4 kB  00:00:00
updates                               | 3.4 kB  00:00:00
(1/4): base/7/x86_64/group_gz          | 155 kB  00:00:00
.../...
Installed:
  wget.x86_64 0:1.14-10.el7_0.1
Complete!
[root@8cdeef2d83c8 /]# exit
```

# Lister les modifications d'un conteneur

- Utilisez la commande `docker diff` pour comparer un conteneur avec son image parent
- L'image parent (l'originale) est comparée avec la nouvelle couche
- Liste les fichiers et les répertoires qui ont changés

```
$ docker diff 8cdeef2d83c8
C /etc
A /etc/wgetrc
C /root
C /usr/bin
A /usr/bin/wget
```

# Lister les modifications d'un conteneur

## ▣ Le nom du dépôt doit être basé sur utilisateur/application

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8cdeef2d83c8	centos	"bash"	17 minutes ago	Exited (0) 15 minutes ago

```
trusting_shirley
```

```
$ docker commit 8cdeef2d83c8 masociete/monapplication:1.0
```

```
sha256:3652dae18983e4cf670bb534a5b4d92539042dbddbcd8ce63d7c5093cdf3dbe2
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
masociete/monapplication	1.0	3652dae18983	3 seconds ago	298.2 MB
nginx	latest	ba6bed934df2	30 hours ago	181.4 MB
ubuntu	14.04	b1719e1db756	5 days ago	188 MB
centos	latest	980e0e4c79ec	2 weeks ago	196.8 MB

# Namespace d'images

- ▣ Les dépôts d'images appartiennent à l'un des trois espaces de nommage
  - Root
    - ubuntu:14.04
    - centos:7
    - Nginx
  - Utilisateur ou organisation
    - masociete/monappli
  - Auto hébergé
    - registry.masociete.com:5000/monappli

# Dockerfile

- ▣ Un Dockerfile est un fichier de configuration qui contient les instructions pour la construction d'une image Docker
- ▣ Fournit un moyen plus efficace de construire des images par rapport à la commande `docker commit`
- ▣ S'adapte dans votre flux de développement et votre processus d'intégration et de déploiement continu

# Processus de création

- ▣ Créez un Dockerfile dans un nouveau dossier
- ▣ Écrivez les instructions pour la construction de l'image
  - Quelle est l'image de base à utiliser
  - Quels sont les programmes à installer
  - Quelle commande à exécuter
- ▣ Exécuter la commande `docker build` pour construire une image à partir du Dockerfile

# Instructions Dockerfile

- FROM spécifie l'image de base utilisée
  - Doit être la première instruction spécifiée dans le Dockerfile
  - Peut être spécifiée plusieurs fois pour créer plusieurs images
  - Chaque FROM marque le début d'une nouvelle image
- RUN spécifie une commande à exécuter
  - Les modifications sont effectuées sur le système de fichiers
  - Utilisée pour installer des bibliothèques, des packages
  - N'enregistre pas l'état des processus
  - Ne démarre pas les daemons automatiquement



# Docker build

## ■ Syntaxe

```
docker build -t [repository:tag] [path]
```

```
$ cat monappli/Dockerfile
```

```
# mon application de test
FROM centos:7.0.1406
RUN yum install -y wget
```

```
$ docker build -t masociete/monappli:1.0 monappli
```

```
Sending build context to Docker daemon 2.048 kB
```

```
Step 1 : FROM centos:7.0.1406
```

```
---> 16e9fdecc1fe
```

```
Step 2 : RUN yum install -y wget
```

```
---> Running in 761e158caf11
```

```
Loaded plugins: fastestmirror
```

```
Determining fastest mirrors
```

```
* base: mirrors.ircam.fr
```

```
---> Package wget.x86_64 0:1.14-10.el7_0.1 will be installed
```

```
Installed:
```

```
wget.x86_64 0:1.14-10.el7_0.1
```

```
---> 211cce1f7cc
```

```
Removing intermediate container 761e158caf11
```

```
Successfully built 211cce1f7cc
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
masociete/monappli	1.0	211cce1f7cc	About a minute ago	307.1 MB
centos	7.0.1406	16e9fdecc1fe	3 weeks ago	210.2 MB

Couche temporaire

Validation de l'image finale

Suppression de la couche temporaire

# Build contexte

- ▣ Le contexte de construction de l'image est le contenu du répertoire que le client envoie au Docker daemon
- ▣ Le répertoire est envoyé sous forme d'une archive
- ▣ Docker daemon va construire en utilisant les fichiers disponibles dans le contexte

```
$ docker build -t masociete/monappli:1.0 monappli  
Sending build context to Docker daemon 2.048 kB
```

# Build cache

- ❑ Docker enregistre un instantané de l'image après chaque étape de construction
- ❑ Avant d'exécuter une étape, Docker vérifie s'il a déjà exécuté cette séquence de construction
  - Si oui, Docker utilisera le résultat (la couche) au lieu d'exécuter à nouveau l'instruction
- ❑ Docker utilise les instructions exactes dans votre Dockerfile à comparer avec le cache
- ❑ En modifiant simplement l'ordre des instructions annulera la cache
- ❑ Pour désactiver le cache utiliser l'option `--no-cache`

```
$ cat monappli2/Dockerfile
```

```
FROM centos:7.0.1406
```

```
RUN yum install -y wget
```

```
RUN yum install -y curl
```

```
$ docker build -t masociete/monappli2:1.0 monappli2
```

```
Sending build context to Docker daemon 2.048 kB
```

```
Step 1 : FROM centos:7.0.1406
```

```
---> 16e9fdecc1fe
```

```
Step 2 : RUN yum install -y wget
```

```
---> Using cache
```

```
---> 211ccef1f7cc
```

```
Step 3 : RUN yum install -y curl
```

```
---> Running in 40de6ceff281
```

```
Loaded plugins: fastestmirror
```

Utilisation du cache

Nouvelle commande dans une nouvelle couche

# Lister les couches

- ▣ `docker history` liste les couches utilisées pour créer une image
- ▣ Affiche la date de création de l'image, sa taille et la commande qui a été exécutée

```
$ docker history masociete/monappli2:1.1
```

IMAGE	CREATED	CREATED BY	SIZE
04bf8be56d5e	47 seconds ago	/bin/sh -c yum install -y curl	11.95 MB
c3138ec3dd96	54 seconds ago	/bin/sh -c yum install -y wget	96.93 MB
16e9fdecc1fe	3 weeks ago	/bin/sh -c #(nop) ADD file:6a409eac27f0c7e043	210.2 MB
<missing>	3 weeks ago	/bin/sh -c #(nop) MAINTAINER The CentOS Proj	0 B

# Instruction CMD

- ❑ L'instruction CMD spécifie la commande par défaut exécutée lorsque le conteneur est créé
- ❑ Ne peut être spécifiée qu'une seule fois dans un Dockerfile
  - Sinon, seule la dernière instruction CMD est exécutée
- ❑ Peut être remplacé au moment de l'exécution
- ❑ Format Shell
  - CMD ping 127.0.0.1 -c 30
- ❑ Format EXEC
  - CMD ["ping","127.0.0.1","-c","30"]

```
$ docker history masociete/monappli2:1.1
```

IMAGE	CREATED	CREATED BY	SIZE
COMMENT			
04bf8be56d5e	47 seconds ago	/bin/sh -c yum install -y curl	11.95 MB
c3138ec3dd96	54 seconds ago	/bin/sh -c yum install -y wget	96.93 MB
16e9fdecc1fe	3 weeks ago	/bin/sh -c #(nop) ADD file:6a409eac27f0c7e043	210.2 MB
<missing>	3 weeks ago	/bin/sh -c #(nop) MAINTAINER The CentOS Proj	0 B

# Instruction ENTRYPOINT

- ▣ Définit la commande qui sera exécuté lorsqu'un conteneur est exécuté
- ▣ Les arguments de la ligne de commande temps d'exécution et de l'instruction CMD sont transmis en tant que paramètres à l'instruction ENTRYPOINT
- ▣ Les deux formats Shell et EXEC sont concernés
- ▣ Les conteneurs s'exécutent essentiellement comme des exécutables
- ▣ Si ENTRYPOINT est utilisé, l'instruction CMD peut être utilisé pour spécifier les paramètres par défaut
- ▣ Les arguments de la ligne de commande remplacent les arguments de l'instruction CMD
- ▣ S'il n'y a pas d'argument sur la ligne de commande, les arguments de CMD seront utilisés pour la commande ENTRYPOINT

# Instruction COPY

- ▣ Syntaxe

`COPY <src> <dest>`

- ▣ Copie les fichiers ou répertoires à partir de la source du hôte vers la destination dans le système de fichier du conteneur
- ▣ Le chemin de la source doit être dans le build context
- ▣ Si la source est un répertoire, tous les fichiers du répertoire sont copiés, mais pas le répertoire lui même.
- ▣ Vous pouvez spécifier plusieurs répertoires sources.

# Instruction ADD

- ▣ Syntaxe

ADD <src> <dest>

- ▣ Copie les fichiers ou répertoires à partir de la source du hôte vers la destination dans le système de fichier du conteneur
- ▣ L'instruction ADD a la capacité d'extraire les fichiers d'une archive (tar)
- ▣ Permet aussi d'utiliser une URL



# Instruction WORKDIR

- ▣ Syntaxe

`WORKDIR /chemin`

- ▣ Par défaut toutes les instructions sont exécutées à partir du dossier racine du conteneur
- ▣ L'instruction WORKDIR permet de spécifier le répertoire de travail pour les commandes RUN, CMD, ENTRYPOINT et COPY
- ▣ L'instruction peut être utilisée plusieurs fois

# Instruction LABEL

- ▣ Syntaxe

`LABEL variable=valeur`

- ▣ Permet d'ajouter des metadatas sur l'image

- ▣ Existe aussi sur les autres objets Docker

- ▣ Exemple

`LABEL maintainer="John Doe jdoe@gmail.com"`

# Instruction ENV

- ▣ Syntaxe

```
ENV  JAVA_HOME /usr/bin/java
```

- ▣ Utilisée pour initialiser des variables d'environnement dans tout conteneur exécuté à partir de l'image

# Bonnes pratiques

- ▣ Chaque ligne dans un Dockerfile crée une nouvelle couche si elle modifie l'état de l'image
- ▣ Vous avez besoin de trouver le juste équilibre entre avoir beaucoup de couches créées pour l'image et la lisibilité du Dockerfile
- ▣ Ne pas installer des paquets inutiles
- ▣ Un ENTRYPOINT par Dockerfile
- ▣ Combiner des commandes similaires en un seul en utilisant "&&" et "\"

# Bonnes pratiques



- ▣ Utilisez au maximum le cache
  - L'ordre des instruction est important
  - Ajouter des fichiers qui sont moins susceptibles de changer en premier et les plus susceptibles de changer à la fin

# Sauvegarde des images

- Produit un référentiel sous forme d'archive `tar` en sortie standard. Contient toutes les couches parentes, et tous les tags et versions.

■ `docker save [OPTIONS] IMAGE [IMAGE...]`

```
$ docker save -o os.tar ubuntu:16.04 centos:7
$ file os.tar
os.tar: POSIX tar archive
$ tar tvf os.tar
drwxr-xr-x  0 0          0          0 10 oct 22:59
13367512b948578a360b2e96ba0f6a25d58bb42cf842329da57918d8744e37dc/
-rw-r--r--  0 0          0          3 10 oct 22:59
13367512b948578a360b2e96ba0f6a25d58bb42cf842329da57918d8744e37dc/VERSION
-rw-r--r--  0 0          0          388 10 oct 22:59
da5ec21813cb48b3e9aa322e11e201e6f0c1ec19becc6823ef901704edd864b3/json
-rw-r--r--  0 0          0      126041088 10 oct 22:59
da5ec21813cb48b3e9aa322e11e201e6f0c1ec19becc6823ef901704edd864b3/layer.tar
-rw-r--r--  0 0          0          704  1 jan  1970 manifest.json
-rw-r--r--  0 0          0          170  1 jan  1970 repositories
```

# Restauration des images

- ▣ charge un référentiel à partir d'une archive.  
Il restaure les images et les tags.

- `docker load [OPTIONS]`
  - `-i, --input string` Read from tar archive file, instead of STDIN
  - `-q, --quiet` Suppress the load output

```
$ docker load -i os.tar
cf516324493c: Loading layer [=====>] 205.2MB/205.2MB
Loaded image: centos:7
0f5ff0cf6alc: Loading layer [=====>] 126MB/126MB
cd181336f142: Loading layer [=====>] 15.87kB/15.87kB
b97229212d30: Loading layer [=====>] 14.85kB/14.85kB
4589f96366e6: Loading layer [=====>] 5.632kB/5.632kB
49907af65b0a: Loading layer [=====>] 3.072kB/3.072kB
Loaded image: ubuntu:16.04
```

# DOCKER

Module 6: Gestion et Distribution des Images



# Distribution

- Pour distribuer votre image il y a deux options
  - Pousser l'image sur le Docker Hub
  - Pousser l'image sur votre propre serveur de registre
  - Utiliser les commandes `docker save` et `load`
- Les images sur Docker Hub peuvent résider dans des dépôts publics ou privés

# Docker Hub

- Les utilisateurs peuvent créer leurs propres dépôts sur Docker Hub
  - Public et privé
- Pousser les images locales sur un dépôt
- Le dépôt réside dans l'espace de noms d'utilisateur ou de l'organisation, par exemple:
  - organisation/monrepo
  - johndoe/monrepo
- Les dépôts publics sont répertoriés et consultables pour un usage public
- Tout le monde peut tirer des images à partir d'un dépôt public

# Docker push

- Syntaxe

`docker push repo/image:tag`

- Le dépôt local doit avoir le même nom et balise que le dépôt sur le Docker Hub
- Seules les couches de l'image qui ont été modifiées sont poussées
- Vous devez vous connecter avec la commande  
`docker login`
- Si vous poussez sur un dépôt local qui n'existe pas sur le Docker Hub, il sera créé automatiquement.

# Images tag

- ▣ Syntaxe

```
docker tag image:tag repo/image:tag
```

- ▣ Utilisé pour renommer un dépôt locale avant de le pousser sur le Docker Hub.
- ▣ Vous serez invité à vous connecter à votre compte Docker Hub
- ▣ Si vous poussez sur un dépôt local qui n'existe pas sur le Docker Hub, il sera créé automatiquement.

# Images tag

## ■ Syntaxe

```
docker tag image:tag repo/image:tag
```

- Utilisé pour renommer un dépôt locale avant de le pousser sur le Docker Hub.
- Vous serez invité à vous connecter à votre compte Docker Hub
- Si vous poussez sur un dépôt local qui n'existe pas sur le Docker Hub, il sera créé automatiquement.

```
$ docker tag 2dba9402744e ambre/training:1.0
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
masociete/monappli2	1.0	c903004d057b	2 hours ago	602.1 MB
masociete/monappli1	1.0	2dba9402744e	2 hours ago	585.9 MB
ambre/training	1.0	2dba9402744e	2 hours ago	585.9 MB
centos	7.0.1406	16e9fdecc1fe	3 weeks ago	210.2 MB

# Images push

## ▣ Syntaxe

`docker push repo/image:tag`

## ▣ La même image peut avoir plusieurs tag

## ▣ L'image peut être identifié par son ID, qui est généré en utilisant un hachage du contenu de l'image pour la consistance.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ambre/training	latest	4fc7d44b50e0	About a minute ago	585.9 MB
masociete/monappli1	1.0	4fc7d44b50e0	About a minute ago	585.9 MB
centos	7.0.1406	16e9fdecc1fe	3 weeks ago	210.2 MB

```
$ docker push ambre/training
```

```
The push refers to a repository [docker.io/ambre/training]
```

```
3339caba23b1: Pushing [=====>
```

```
] 1.39 MB/2.277 MB
```

```
a3d098e877cb: Preparing
```

# Suppression des images locales

- ▣ Syntaxe

`docker rmi image_id | repo/image:tag`

- ▣ La même image peut avoir plusieurs tag

- ▣ L'image peut être identifiée par son ID, qui est généré en utilisant un hachage du contenu de l'image pour la consistance.

```
$ docker rmi --no-prune masociete/monappli1:1.0
Untagged: masociete/monappli1:1.0

$ docker rmi 4fc7d44b50e0
Untagged: ambre/training:latest
Deleted: sha256:4fc7d44b50e09760abfeff7dfb7ea2ff98a2738dcf882724f1d0fcc4f1fd3233
Deleted: sha256:09ea582e50d795aa9822f8f22a138e9d5c3dc43786e960c8c75cabbbe2f68412
```

# DOCKER

Module 7: Réseau

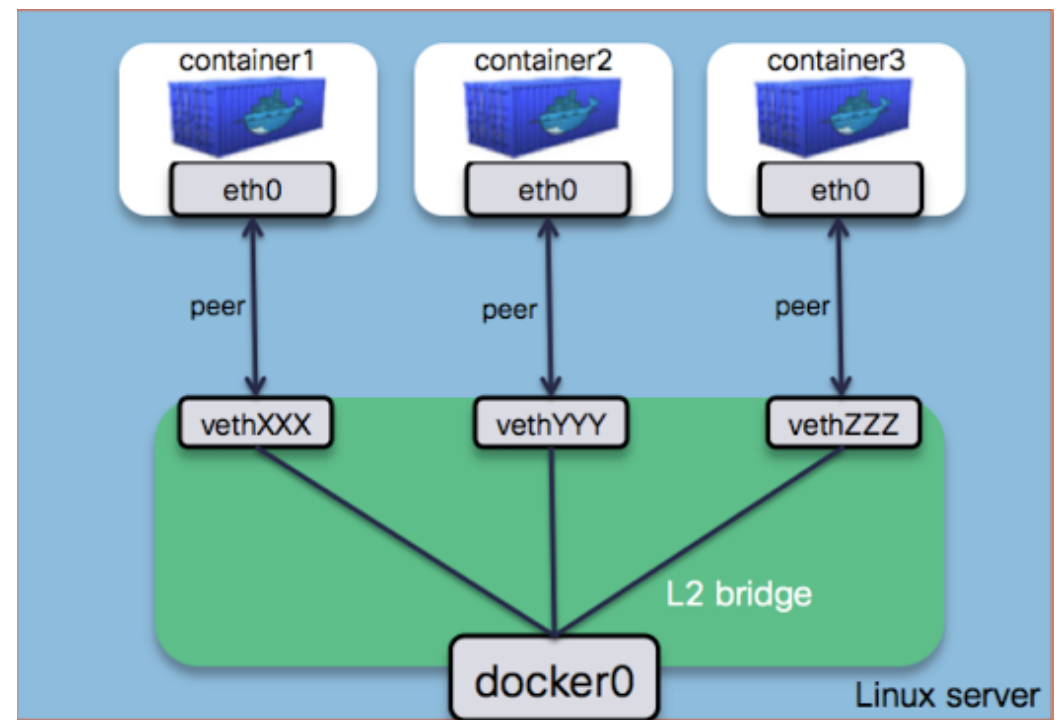


# Modèle Réseau

- ▣ Les conteneurs ne disposent pas d'une adresse IPv4 publique
- ▣ Ils ont une adresse privée
- ▣ Les services en cours d'exécution dans un conteneur doivent être exposés port par port
- ▣ Les ports de conteneurs doivent être mappés sur les ports de l'hôte pour éviter les conflits
- ▣ Lorsque Docker démarre, il crée une interface virtuelle appelée docker0 sur la machine hôte avec une adresse IP privée allouée de manière aléatoire

# Bridge

- L'interface `docker0` est un pont ethernet virtuel
- `docker0` commute les paquets ethernet entre deux interfaces comme un pont ou commutateur physique
  - De l'hôte vers un conteneur
  - D'un conteneur vers un autre conteneur
- Chaque nouveau conteneur obtient une interface attachée au bridge `docker0`



# docker0

```
$ ip a
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:1b:a9:9c:b8 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:1bff:fea9:9cb8/64 scope link
        valid_lft forever preferred_lft forever

$ brctl show docker0
bridge name bridge id                STP enabled interfaces
docker0      8000.02421ba99cb8                    no

$ docker run -d -it ubuntu:14.04
79e1843c0e73901cb4a3ce148d91317b603d4ddfbab8ed1f30804ecfd7965a41

$ docker run -d -it ubuntu:14.04
997f22ef4eb878055caf693367ffd9486f78bbc02b3a38649b20eeae96b80821

$ brctl show docker0
bridge name        bridge id                STP enabled    interfaces
docker0            8000.02421ba99cb8        no             veth24a6c86
                                                           vethd1fa4ad
```

# Réseau par défaut

- ❑ Docker initialise automatiquement 3 réseaux
- ❑ Le réseau bridge est le docker0 bridge
- ❑ Par défaut tous les conteneurs sont connectés au réseau bridge
- ❑ Si on connecte un conteneur au réseau none, le conteneur n'aura pas d'interface réseau
- ❑ Si on connecte un conteneur au réseau host, le conteneur sera sur la même pile réseau que le hôte

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
alf9f5659e41	bridge	bridge	local
da557c8b568c	host	host	local
88c3b697f7e0	none	null	local

```
$ docker run -d -it --net=none ubuntu:14.04
```

```
4e2c0412eabcb02adc2fc08b89082cf5fd52ba7cdda54bfa2640f3aeaedd2707
```

# Inspecter le Réseau

```
$ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "alf9f5659e411cebee106dd64830e77c1c22259be688f352b98406d212b0c42a",
    "Scope": "local",
    "Driver": "bridge",
    "Config": [
      {
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1"
      }
    ]
  },
  "Internal": false,
  "Containers": {
    "Name": "suspicious_hoover",
    "EndpointID": "255292846e0dfb465a53bc490ce63e146da4d1fd0684405803e88e21c97a35f5",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "138b866431b983784600c4e365053e7bd7b8d441bce6cc23a5ce7663fa41750f": {
      "Name": "awesome_shaw",
      "EndpointID": "f05efe1d901ca2d59d68357b7b1cbef49f9dcc1829695e547ddd53d63f3468d",
      "MacAddress": "02:42:ac:11:00:02",
      "IPv4Address": "172.17.0.2/16",
      "IPv6Address": ""
    }
  }
},
]
```

# Créer un Réseau

## ▣ Deux types de réseaux

### ■ Bridge

- Similaire au bridge docker0

### ■ Overlay

- Un bridge à travers plusieurs hôtes

```
$ docker network create --driver bridge mon_bridge
45630a80932f14867f4cbc87f5de3f9ec724c8a3df7210710295c1f616bf69fd

$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
a1f9f5659e41        bridge              bridge              local
da557c8b568c        host                host                local
45630a80932f        mon_bridge          bridge              local
88c3b697f7e0        none                null                local

$ docker run -d -it --net=mon_bridge ubuntu:14.04
72fe6c8fb1412053fb6cf8a7cf93c4211bebbf5f4e3a6d5b0f568c42b49eb737
```

# Serveur DNS intégré

- ▣ Un service intégré qui permet la découverte des containers créés avec un nom valide ou un alias réseau
- ▣ Intégré dans le daemon docker
- ▣ Permet aux conteneurs de communiquer sur le même bridge

```
$ docker run -d -it --net=mon_bridge --name=host1 ubuntu:14.04
07f1d021f3a80d08de768984f28e746b12517592b5ef1e5b264dd8242a004c6f
$ docker run -it --net=mon_bridge --name=host2 ubuntu:14.04
root@00facc8e4f27:/# ping host1
PING host1 (172.18.0.3) 56(84) bytes of data.
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=2 ttl=64 time=0.081 ms
```

# Réseaux multiples

- ▣ Les conteneurs peuvent être connectés à plusieurs réseaux
- ▣ Syntaxe  
`docker network connect <network> <container>`

```
$ docker run -d -it --name=host1 ubuntu:14.04
E83404e00a6df20e3c1caedf47ef3903e3be69dfacea8b20e6c272a7e7725fa77

$ docker run -it --name=host2 --net=mon_bridge ubuntu:14.04
root@9b964ca294fd:/# ping host1
^C

$ docker network connect mon_bridge host1

root@9b964ca294fd:/# ping host1
PING host1 (172.18.0.3) 56(84) bytes of data.
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=2 ttl=64 time=0.068 ms
```



# Mapping des ports

- Les containers en cours d'exécution dans un réseau bridge ne sont accessibles que par l'hôte sur lequel le bridge réside
- Pour qu'un conteneur soit accessible à l'extérieur, nous devons exposer les ports du conteneur et les mapper sur un port de l'hôte.
- Le conteneur est accessible via le port mappé de l'hôte
- Les ports peuvent être mappés manuellement ou automatiquement

# Mapping manuel

- Syntaxe

`docker run -p [host port]:[container port] <image>`

- Utiliser plusieurs fois l'option `-p` pour mapper plusieurs ports

- Visualiser le mapping d'un conteneur

- Syntaxe

`docker port <conteneur>`

```
$ docker run -d -p 8080:80 nginx
54c366ec1c60352ba63d165b0bc2e8a124a739066348ba00c6295f17af56fceb
$ docker ps
CONTAINER
ID          IMAGE          COMMAND          CREATED          STATUS          PORTS
          NAMES
54c366ec1c60 nginx          "nginx -g 'daemon of..." 7 seconds ago    Up 6
seconds     0.0.0.0:8080->80/tcp    pedantic_moser
```

# Mapping automatique

- ▣ Syntaxe

`docker run -P <image>`

- ▣ Mappe automatiquement les ports exposés dans le conteneur sur un numéro de port de l'hôte
- ▣ Les numéros de port hôte utilisés vont de 32 768 à 65 535
- ▣ Utiliser l'instruction EXPOSE pour exposer les ports lorsqu'on utilise un fichier Dockerfile

```
$ docker run -d -P nginx
7b2f2eebd4a132960249856d58f65585d3f08bf6a5cd5373fe82e6c3bbc0637d
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
7b2f2eebd4a1	nginx	"nginx -g 'daemon of..."	4 seconds ago	Up 3	
	seconds	0.0.0.0:32768->80/tcp	pedantic_mirzakhani		

# DOCKER

Module 8: Volumes

# Volumes

- Un volume est répertoire dans un conteneur, qui est conçu pour la persistance des données, indépendamment du cycle de vie du conteneur
- Pas de modification des données du volume lors de la mise à jour d'une image
- Persistant lorsqu'un conteneur est supprimé
- Peut être mappé sur un répertoire du hôte
- Peut être partagé entre conteneurs
- Le système COW n'affecte pas les volumes
- Si on crée une image à partir d'un conteneur, le contenu des volumes ne fait pas partie de l'image
- Si une instruction RUN dans un Dockerfile modifie le contenu d'un volume, ces modifications ne sont pas enregistrées.

# Utilisation des Volumes

- Déconnecter les données qui sont stockées, du conteneur à partir duquel les données ont été créées
- Pour le partage de données entre conteneurs
  - Configuration des données du conteneur qui a un volume monté dans d'autres conteneurs
  - Partager des répertoires entre plusieurs conteneurs
- Contourner le système COW pour obtenir des performances d'I/O disque natives
- Partager un répertoire hôte avec un conteneur
- Partager un fichier unique entre l'hôte et le conteneur

# Création des Volumes

## ▣ Créer un volume

```
docker volume create [--name nom_volume]
```

## ▣ Lister les volumes

```
docker volume ls
```

```
$ docker volume create --name test1
$ docker volume create

$ docker volume ls
DRIVER          VOLUME NAME
local           9771790926a85e35f9e6fe3df8196974480102815b9e766f28fc98f821e1250a
local           test1
```

# Montage de Volumes

```
$ docker run -it -v test1:/www/test1 moncentos bash
```

```
[root@c7d39428c73f /]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/docker-253:0-508564-71ac5904b3390844b362c785e5c5c7dfeedb1c2748ba812f929f3c0350c59875	10G	253M	9.8G	3%	/
tmpfs	497M	0	497M	0%	/dev
tmpfs	497M	0	497M	0%	/sys/fs/cgroup
/dev/sda1	14G	2.0G	12G	14%	/www/test1
shm	64M	0	64M	0%	/dev/shm

## ▣ Monter un volume en lecture seulement

```
docker run -v <name>:<path>:ro
```



# Inspection des Volumes

## ▣ Inspecter un volume

`docker volume inspect <volume>`

```
$ docker volume inspect test1
[
  {
    "Name": "test1",
    "Driver": "local",
    "Mountpoint": "/var/lib/docker/volumes/test1/_data",
    "Labels": {},
    "Scope": "local"
  }
]
```

# Suppression des Volumes

- ❑ Les volumes ne sont pas supprimés lorsque vous supprimez un conteneur
- ❑ Supprimer un volume  
`docker volume rm <volume>`
- ❑ Vous pouvez aussi supprimer tous les volumes associés à un conteneur lorsque vous supprimez celui-ci.  
`docker rm -v <conteneur>`
- ❑ Vous ne pouvez pas supprimer un volume utilisé par un conteneur, même arrêté.

```
$ docker volume rm test2
test2
```

# Volumes d'hôtes

- ❑ Lors de l'exécution d'un conteneur, vous pouvez mapper des dossiers de l'hôte sur un volume
- ❑ Les fichiers du dossier hôte seront présents dans le volume
- ❑ Les modifications apportées à l'hôte sont reflétées dans le volume du conteneur
- ❑ Syntaxe  
`Docker run -v [chemin host]:[chemin conteneur]:[rw|ro]`
- ❑ S'il existe pas le chemin ou le chemin d'accueil du conteneur, il sera créé
- ❑ Si le chemin du conteneur est un dossier avec un contenu existant, les fichiers seront remplacés par ceux du hôte

```
$ docker run -d -v /home/user/www:/www nginx
```

# Volumes partagés

- ▣ Les volumes peuvent être montés dans plusieurs conteneurs
- ▣ Permet aux données d'être partagées entre conteneurs
- ▣ Exemple d'utilisation
  - Un conteneur écrit des données statistiques dans le volume
  - Un autre conteneur exécute une application pour lire les données et générer des graphiques
- ▣ Soyez conscients des conflits potentiels si plusieurs applications sont autorisés à accéder en écriture dans le même volume

```
$ docker run -d -v /home/user/www:/www nginx
```

# Volumes avec Dockerfile

- ❑ L'instruction VOLUME crée un point de montage
- ❑ Possibilité de spécifier des arguments dans un tableau JSON ou chaîne
- ❑ Vous ne pouvez pas mapper des volumes sur les répertoires du hôte
- ❑ Les volumes sont initialisés lorsque le conteneur est exécuté
- ❑ Syntaxe

```
VOLUME /vol1
```

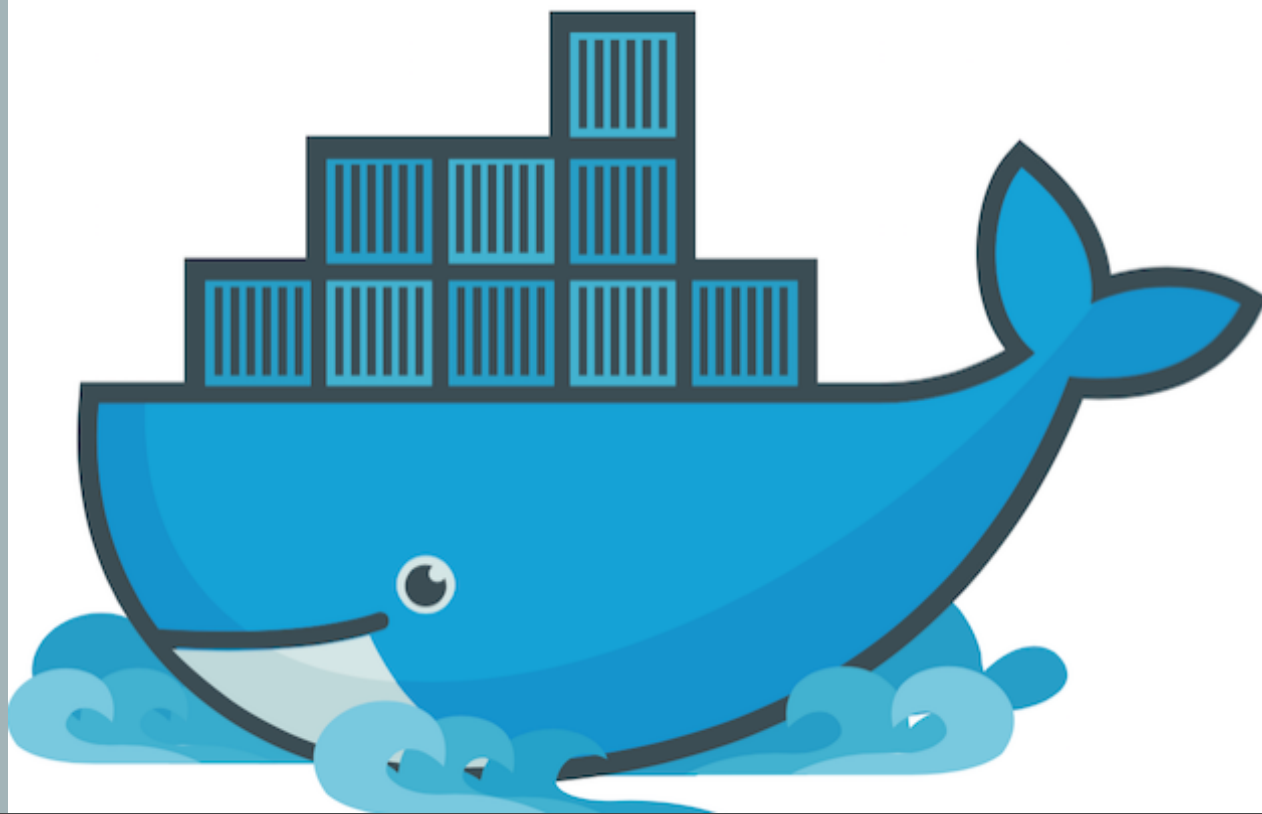
```
VOLUME /www/site1 /www/site2
```

```
VOLUME [ "vol1", "vol2" ]
```

# Conteneur de données

- Un conteneur de données est un conteneur créé dans le but de référencer un ou plusieurs volumes
- Un conteneur de données n'exécute aucune application ou processus
- Utilisé lorsque vous avez des données persistantes qui doivent être partagées avec d'autres conteneurs
- Lors de la création d'un conteneur de données, vous devez lui donner un nom personnalisé pour qu'il soit plus facile à référencer
- Un conteneur de données peut-être utilisé par `--volumes-from`

```
$ docker run --name datas -v /data busybox true
$ docker volume ls
DRIVER          VOLUME NAME
local           aad7200244f1f751767570a13cf7ce02d6930817e46f973a1351dc68a911f705
$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS              PORTS          NAMES
726ab38456b8   busybox    "true"                  45 seconds ago Exited (0) 44 seconds ago           datas
$ docker run -it --volumes-from datas moncentos bash
```



Docker

Virtualisation légère

# MODULE 1: GESTION DU DEMON





# Contrôle et configuration du Daemon

- La configuration du démarrage et arrêt du démon
  - Docker dépend d'un certain nombre de facteurs
    - Le démon s'exécute comme un service ?
    - Quelle distribution Linux
      - `service`
      - `systemctl`
- Exécution en mode interactif en arrière plan

# Exécution en mode service

## Ubuntu et Debian

- Utiliser la commande `service`
  - ▣ `sudo service docker stop`
  - ▣ `sudo service docker start`
  - ▣ `sudo service docker restart`

# Exécution en mode service

## RHEL & CentOS

- Utiliser la commande `systemctl`
  - ▣ `systemctl start docker`
  - ▣ `systemctl stop docker`
  - ▣ `systemctl restart docker`
- Dans les nouvelles version la commande `service` est une redirection vers la commande `systemctl`

# Exécuter le démon de manière interactive

- S'il ne fonctionne pas en tant que service, lancez le démon Docker
  - `sudo docker daemon &` (avant la 1.12)
  - `sudo dockerd &` (depuis la 1.12)
- S'il ne fonctionne pas en tant que service, envoyez le signal `SIGTERM` pour arrêter le démon
  - `sudo kill $(pidof docker)`

# Fichiers de configuration du démon

## Ubuntu et Debian

- `/etc/default/docker`
- La variable `DOCKER_OPTS` est utilisée pour ajouter des options au démon s'il est lancé comme un service
- Relancez le service pour prendre en compte les modifications  
`sudo service docker restart`

# Fichiers de configuration du démon

## RHEL et Centos

- ❑ Le mécanisme `systemd` est utilisé pour exécuter le démon
- ❑ Les options de lancement sont dans le fichier `docker.service`
- ❑ Le fichier `docker.service` se trouve dans  
`/usr/lib/systemd/system` ou `/etc/systemd/service`
- ❑ Exécuter la commande : `find / -name docker.service`

```
[root@docker ~]# find / -name docker.service -print
/etc/systemd/system/multi-user.target.wants/docker.service
/usr/lib/systemd/system/docker.service
```

# Le fichier docker.service

```
[root@docker ~]# cat /etc/systemd/system/multi-user.target.wants/docker.service
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues
still
# exists and systemd currently does not support the cgroup feature set
required
# for containers run by docker
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
ExecStart=/usr/bin/dockerd $OPTIONS \
                $OPTIONS_STORAGE \
                $OPTIONS_NETWORK
ExecReload=/bin/kill -s HUP $MAINPID
```

# Le fichier de configuration du démon

- ❑ Le fichier `docker.service` utilise les fichiers `EnvironmentFile` qui font référence aux fichiers `/etc/sysconfig/docker ...`
- ❑ Ce sont ces derniers fichiers qui seront utilisés pour modifier les options du démon

```
[root@docker ~]# cat /etc/sysconfig/docker
OPTIONS=" \
    -H tcp://0.0.0.0:2375 \
    -H unix:///var/run/docker.sock \
"
```



# Les options du démon

- ▣ Lancer le démon en mode réseau.
  - Le démon écoute sur une socket TCP
- ▣ Activer le mode debug
  - Valider le niveau de journalisation
- ▣ Spécifier un serveur DNS
- ▣ Ajouter des registres non sécurisés
- ▣ Valiser la sécurité du démon avec TLS

# Journalisation du démon

- Démarrez le démon avec l'option `--log-level` et spécifiez le niveau de journalisation
  - ▣ Debug
  - ▣ Info
  - ▣ Warn
  - ▣ Error
  - ▣ Fatal

# Journalisation sur RHEL et CentOS

- Sur les systèmes basés sur `systemd` la journalisation est gérée par le démon `journald`
- La commande `journalctl` permet de visualiser le log
- Utiliser l'option `-u` pour filtrer par service  
`journalctl -u docker.service`

```
[root@docker ~]# journalctl -u docker.service
```

```
-- Logs begin at lun. 2017-10-02 07:57:37 CEST, end at mar. 2017-10-03 09:26:49 CEST. --
oct. 02 11:55:04 poste508.s11.pfd systemd[1]: Starting Docker Application Container Engine...
oct. 02 11:55:04 poste508.s11.pfd dockerd[32645]: time="2017-10-02T11:55:04.797590516+02:00" level=info msg="libcon
oct. 02 11:55:05 poste508.s11.pfd dockerd[32645]: time="2017-10-02T11:55:05.800716404+02:00" level=warning msg="fai
oct. 02 11:55:06 poste508.s11.pfd dockerd[32645]: time="2017-10-02T11:55:06.198230281+02:00" level=info msg="Graph
oct. 02 11:56:23 poste508.s11.pfd dockerd[32645]: time="2017-10-02T11:56:23.479289269+02:00" level=info msg="Attemp
oct. 03 07:46:40 docker systemd[1]: Stopping Docker Application Container Engine...
oct. 03 07:46:40 docker dockerd[32645]: time="2017-10-03T07:46:40.678359275+02:00" level=info msg="Processing signa
oct. 03 07:46:40 docker dockerd[32645]: time="2017-10-03T07:46:40.731091041+02:00" level=info msg="stopping contain
```

# Le démon docker en mode réseau

- ❑ Par défaut, le client Docker et le démon sont sur le même hôte
- ❑ Pour connecter le client à un démon Docker exécuté sur un hôte différent, il faut :
  - ❑ Tout d'abord, le daemon Docker doit écouter sur une socket réseau (TCP)
  - ❑ Pour des raisons de sécurité, le démon peut utiliser une socket cryptée. Il faudra configurer TLS
  - ❑ Modifier le client pour se connecter au démon distant

# Types de socket

- ▣ `unix`
- ▣ `tcp`
- **La socket par défaut est une socket `unix` créée sur `/var/run/docker.sock`**
- **Les permissions root sont requises**

# Erreur de connexion

- ❑ Le type message d'erreur ci-dessous signifie généralement
  - ❑ Le démon Docker ne fonctionne pas
  - ❑ Problème de permission de connexion au démon docker
  - ❑ Votre client Docker essaie de se connecter au démon en utilisant la socket Unix, mais le démon ne l'écoute pas
  - ❑ Vous n'utilisez pas TLS pour vous connecter au démon

```
[root@docker ~]# docker info
```

```
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the  
docker daemon running?
```

# Écoute sur une socket TCP

- ❑ Configurer le démon Docker pour écouter sur une socket TCP, en utilisant l'option `--host` ou `-H` et spécifier l'adresse TCP et le port
- ❑ Pour l'adresse, vous pouvez spécifier une adresse IP pour écouter ou spécifier `0.0.0.0` pour écouter toutes les adresses réseau
- ❑ Le port `2375` pour une communication non chiffrée
- ❑ Le port `2376` pour une communication cryptée

# Écoute sur une socket TCP

- La socket TCP écoute toutes les adresses réseau

```
dockerd -H tcp://0.0.0.0:2375
```

- La socket TCP écoute une adresse spécifique

```
dockerd -H tcp://10.2.1.1:2375
```

```
[root@docker ~]# cat /etc/sysconfig/docker  
OPTIONS="-H tcp://0.0.0.0:2375"
```



# Connexion du client

- Par défaut, le client Docker suppose que le démon écoute sur une socket Unix
- Il faut configurer le client pour se connecter à un démon docker distant
  - ▣ Utiliser l'option `-H` de la commande `docker`
  - ▣ Configurer la variable d'environnement `DOCKER_HOST`

# Ecoute sur plusieurs sockets

- ❑ Le démon Docker peut écouter à la fois la socket Unix et la socket TCP
- ❑ Utiliser l'option `-H` plusieurs fois
  - ❑ Pour la socket unix
    - `unix:///var/run/docker.sock`
  - ❑ Pour la socket réseau
    - `tcp://0.0.0.0:2376`

# Le fichier de configuration

□ `/etc/docker/daemon.json`

```
{  
    "dns": [],  
    "storage-driver": "",  
    "labels": [],  
    "debug": true,  
    "hosts": [],  
    "log-level": "",  
    "tls": true,  
    "default-gateway": ""  
}
```

# MODULE 2: SECURITE TLS



# conteneurs Linux et la sécurité

- ❑ Docker permet de rendre les applications plus sûres car il fournit un ensemble réduit de privilèges
- ❑ Les espaces de noms (namespaces) fournissent une vue isolée du système. Chaque conteneur a son propre :
  - ▣ IPC, stack TCP/IP, file system / etc...
- ❑ Les processus s'exécutant dans un conteneur ne peuvent pas voir les processus d'un autre conteneur
- ❑ Les groupes de contrôle (Cgroups) isolent l'utilisation des ressources par conteneur
- ❑ S'assure qu'un conteneur compromis ne fera pas tomber l'hôte entier en épuisant les ressources

# Considérations

---

- ❑ Le démon Docker doit fonctionner en tant que root
- ❑ Assurez-vous que seuls les utilisateurs de confiance peuvent contrôler le démon Docker
- ❑ Qui fait partie du groupe de docker
- ❑ Si vous liez le démon à une socket TCP, sécurisez-le avec TLS

# TLS Transport Layer Security

- ❑ Evolution de SSL
- ❑ Utilise la cryptographie à clé publique pour crypter les connexions.
- ❑ Les clés sont signées avec des certificats gérés par une partie de confiance.
- ❑ Ces certificats attestent l'identité du serveur
- ❑ Chaque transaction est donc cryptée et authentifiée

# Usage de TLS pour Docker

- ❑ Docker fournit des mécanismes pour authentifier à la fois le serveur et le client.
- ❑ Fournit une authentification forte, une autorisation et un cryptage pour toute connexion API sur le réseau.
- ❑ Les clés client peuvent être distribuées aux clients autorisés
- ❑ **Pré-requis**
  - ❑ OpenSSL 1.0.1 installé
  - ❑ Créez un dossier pour stocker vos clés et assurez-vous que le dossier est protégé en mode 700



# Processus de configuration de TLS

- ❑ Créer l'autorité de certification (CA)
  - Besoin d'une clé privée et d'un certificat CA
- ❑ Configurer la clé privée du serveur
- ❑ Créer une demande de signature de certificat (CSR) pour le serveur
- ❑ Signez la clé du serveur avec le CSR par rapport à notre CA
- ❑ Créer une clé privée client et CSR
- ❑ Signez la clé du client avec le CSR par rapport à notre autorité de certification
- ❑ Exécutez le démon Docker avec TLS activé et spécifiez l'emplacement de la clé privée de l'autorité de certification, du certificat de serveur et de la clé du serveur
  - Et configurez-le pour écouter sur TCP
- ❑ Pointez le client Docker vers l'adresse TCP du démon et spécifiez l'emplacement du certificat client et la clé en tant que clé privée de l'autorité de certification.

# Create the Certificate Authority

- ❑ Nous avons besoin de l'autorité de certification pour signer nos clés de serveur et de client
- ❑ Créez la clé privée de l'autorité de certification. Vous serez invité à entrer une phrase secrète. Assurez-vous de vous en souvenir
- ❑ Créez la clé publique

```
openssl genrsa -aes256 -out ca-key.pem 2048
```

```
openssl req -new -x509 -days 365 \  
            -key ca-key.pem -sha256 -out ca.pem
```

# Configurer la clé du serveur et CSR

- ❑ La demande de signature de certificat (CSR) est nécessaire pour que nous puissions signer notre clé de serveur.
- ❑ Lors de la création du fichier CSR, assurez-vous de spécifier le nom d'hôte de la machine sur laquelle votre démon Docker s'exécute dans l'attribut CN.

## Créer la clé privée du serveur

```
openssl genrsa -out server-key.pem 2048
```

## Créez le CSR.

```
openssl req -subj "/CN=<host name>" \  
            -new -key server-key.pem -out server.csr
```

## Signez la clé du serveur

- Avant de signer notre clé de serveur, nous allons définir une **extension de certificat** pour spécifier le `subjectAltName`
- `subjectAltName` nous permet de spécifier les **adresses IP de connexion autorisées**

```
echo subjectAltName =  
IP:10.10.10.20,IP:127.0.0.1 > extfile.cnf
```

# Signez la clé du serveur

- Nous signons maintenant la clé du serveur à l'aide de l'autorité de certification que nous avons créée
- Spécifiez le fichier d'extension de certificat (`extfile.cnf`)

```
openssl x509 -req \  
    -days 365 \  
    -in server.csr -CA ca.pem \  
    -CAkey ca-key.pem \  
    -CAcreateserial \  
    -out server-cert.pem \  
    -extfile extfile.cnf
```

# Créer des clés de client

**Nous créons d'abord la clé privée des clients**

```
openssl genrsa -out client-key.pem 2048
```

**Ensuite, nous créons la demande de signature de certificat client**

```
openssl req -subj '/CN=client' \  
            -new \  
            -key client-key.pem \  
            -out client.csr \  
            -
```

# Signer les clés du client

**Nous avons besoin d'un fichier de configuration d'extension avec l'extension `extendedKeyUsage` afin de rendre la clé adaptée à l'authentification du client**

```
echo extendedKeyUsage = clientAuth > extfile.cnf
```

**Maintenant, nous pouvons signer notre clé publique client**

```
openssl x509 -req -days 365 \  
            -in client.csr \  
            -CA ca.pem \  
            -CAkey ca-key.pem \  
            -CAcreateserial \  
            -out client-cert.pem \  
            -extfile extfile.cnf
```

# Activer TLS sur le démon Docker

## Options

Exécuter le daemon avec les options

```
dockerd  
--tlsverify  
--tlscacert=<path to ca cert>  
--tlscert=<path to server  
certificate>  
--tlskey=<path to server key>  
-H=0.0.0.0:2376
```



# Protégez nos certificats et clés

- Pour la clé privée CA, la clé privée du serveur et la clé privée du client, vous voulez rendre les fichiers lisibles seulement pour vous  
`chmod -v 0400 ca-key.pem client-key.pem \`  
`server-key.pem`
- Pour les certificats, vous voudrez supprimer l'accès en écriture  
`chmod -v 0444 ca.pem server-cert.pem client-`  
`cert.pem`
- Il est également recommandé de déplacer le certificat du serveur, la clé privée du serveur et la clé privée CA dans un dossier tel que `/etc/docker`

## Spécification de TLS sur le client

- Maintenant que le démon Docker a TLS activé, lorsque nous utilisons le client, nous devons spécifier pour activer TLS et spécifier notre certificat client et la clé
- `docker`
  - `--tlsverify`
  - `--tlscacert=<path to ca cert>`
  - `--tlscert=<path to client certificate>`
  - `--tlskey=<path to client key>`
  - `-H=<server url>:2376`

# Configuration client

- ❑ Pour configurer le client, nous pouvons placer notre clé et certificat client avec la clé CA dans le dossier caché `.docker`. Ce dossier réside dans notre répertoire personnel.
- ❑ Les fichiers
  - ❑ `ca.pem`, `cert.pem`, `key.pem`
- ❑ Exécuter la commande `docker`, avec l'option TLS ou les variables d'environnement
  - ❑ `--tlsverify`
  - ❑ `-H`  
`docker --tlsverify -H 127.0.0.1:2376 ps -a`

# Variables d'environnement



```
export DOCKER_HOST="tcp://<ipaddress>:2376"  
export DOCKER_TLS_VERIFY=1
```

# MODULE 3:

## RESEAU MULTI-HÔTE



# Réseau Multi-hôte

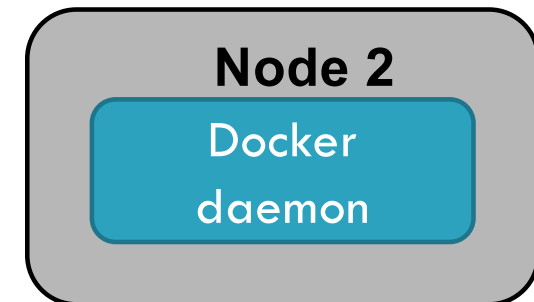
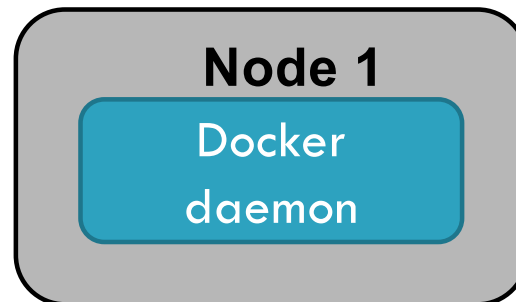
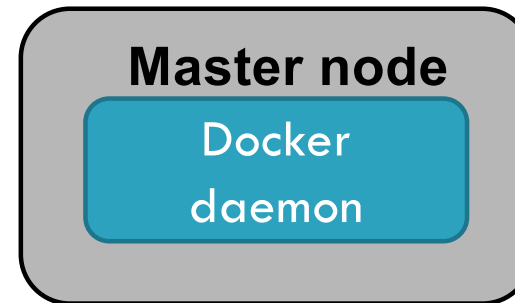
- ❑ Les conteneurs fonctionnant sur des hôtes différents ne peuvent pas communiquer entre eux sans mapper leurs ports TCP aux ports TCP de l'hôte
- ❑ Le réseau multi-hôte permet à ces conteneurs de communiquer sans nécessiter de mapping de port
- ❑ Le Docker Engine prend en charge les réseaux multi-hôtes par le biais du pilote réseau Overlay
- ❑ Pré-requis pour créer un réseau Overlay
  - Accès à un service clé-valeur
  - Un cluster d'hôtes connecté au service clé-valeur
  - Tous les hôtes doivent avoir la version kernel 3.16 ou supérieure

# Service clé-valeur

- Stocke les informations sur l'état du réseau
  - ▣ Service de découverte
  - ▣ Adresses IP
- Solutions prises en charge
  - ▣ Consul
  - ▣ Zookeeper
  - ▣ Etc

# Configuration

- ❑ 1 noeud Master stocke la clé-valeur
- ❑ 2 noeuds





# Configuration de la clé-valeur

## Effectuez ceci sur votre nœud Master

- ❑ Nous utiliserons consul comme service de stockage clé-valeur
- ❑ Exécutez le consul dans un conteneur avec la commande suivante
- ❑ Vérifiez que le consul fonctionne et que le port 8500 est mappé sur l'hôte

```
docker run -p 8400:8400 -p 8500:8500 -p 8600:53/udp -h node1  
progrum/consul -server -bootstrap
```

# Configurer les autres noeuds

---

- Le démon Docker sur les autres noeuds doit être configuré pour:
  - ▣ Écouter sur le port TCP 2375
  - ▣ Utiliser la clé-valeur du Consul sur notre noeud maître

# Configurer démon Docker

- **Modifier la variable OPTIONS dans le fichier :**  
`/etc/sysconfig/docker`

```
OPTIONS="-H tcp://0.0.0.0:2375 \  
        -H unix:///var/run/docker.sock \  
        --cluster-store=consul://<Master Node IP>:8500/network \  
        --cluster-advertise=eth0:2375"
```

# Configurer le réseau Overlay

## **Effectuez ceci sur le Node1 ou le Node2**

- Nous allons créer un réseau Overlay appelé multinet
  - Il sera configuré avec le sous-réseau 10.10.10.0/24
- ```
docker network create -d overlay --subnet  
10.10.10.0/24 multinet
```

# Visualisation

- Une fois que vous avez créé le réseau Overlay, vérifiez qu'il est présent

```
docker network ls
```

- Vérifiez sur l'autre noeud que le réseau est également présent

# Conteneurs sur un réseau multi-hôte

- Pour exécuter un conteneur sur le réseau multi-hôte, il vous suffit de spécifier le nom du réseau sur la commande docker

```
docker run -itd --name c1 --net multinet busybox
```

- A la création d'un réseau Overlay, Docker crée également un autre réseau appelé `docker_gwbridge`.
- Le réseau `docker_gwbridge` fournit un accès externe pour les conteneurs

# MODULE 4:

## SERVEUR DE REGISTRE PRIVE

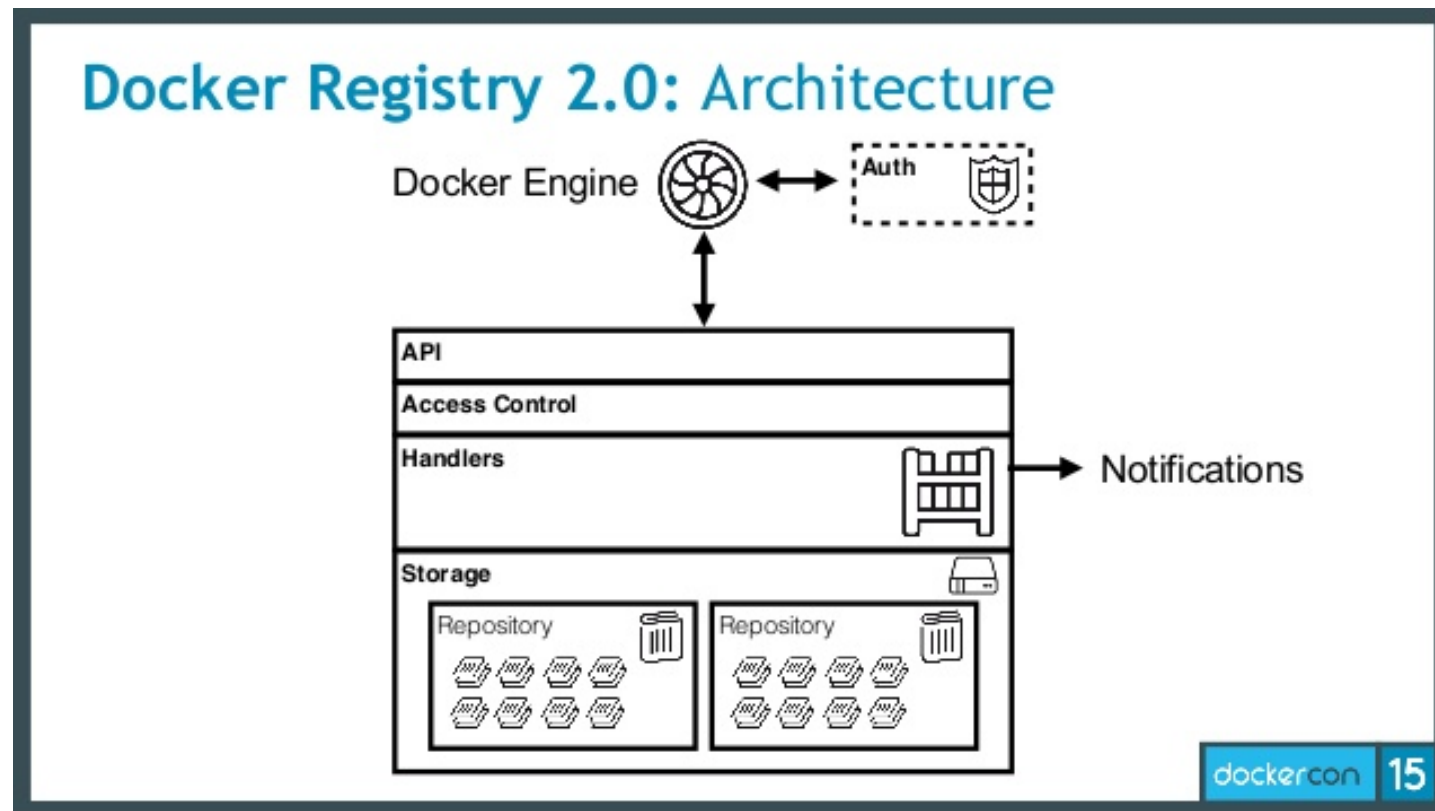


# Serveur de registre

- ❑ Exécutez votre propre serveur de registre pour stocker et distribuer des images au lieu d'utiliser Docker Hub
- ❑ Plusieurs options
  - ❑ Exécuter le serveur de registre en utilisant le conteneur
  - ❑ Docker Trusted Registry
- ❑ Deux versions
  - ❑ Registry v1.0 jusqu'à Docker 1.5
  - ❑ Registry v2.0 depuis Docker 1.6



# Architecture



# Fonctionnalités

- ❑ Stockage configurable
  - ❑ Local disk
  - ❑ Amazon S3
  - ❑ Microsoft Azure
- ❑ Webhooks pour lancer une construction ou envoyer des notifications à certaines personnes lorsque des images ont été poussées
- ❑ Accès sécurisé aux images via TLS

# Public ou private



- Vous pouvez choisir d'exécuter un serveur de registre qui est accessible au public
- Vous pouvez le mettre derrière le pare-feu et le rendre disponible uniquement pour le personnel interne de l'entreprise

# Configuration d'un serveur de registre

- Deux méthodes
  - ▣ Utilisez l'image officielle du registre sur le hub docker
  - ▣ Téléchargez la source de distribution et créez votre propre image de registre personnalisée
- L'image officielle contient une version pré-configurée du registre v2.0
- L'image officielle est destinée à des fins d'évaluation car sa configuration par défaut ne convient pas à l'utilisation de la production
  - ▣ Pas de TLS

```
docker run -d -p 5000:5000 registry:2.0
```

# Push

- D'abord, créer un tag de l'image avec l'hôte IP ou le domaine du serveur de registre, puis exécuter la commande `docker push`

```
[root@docker ~]# docker tag 09ea64205e55 myserver.com:5000/masociete/monimage:1.0
```

```
[root@docker ~]# docker push myserver.com:5000/masociete/monimage:1.0
```

```
The push refers to a repository [myserver.com:5000/masociete/monimage]
```

# Lister les tags dans le registre

- Pour lister les tags d'une image, il faut faire une requête du type :  
`<registry host>:<port>/v2/<repo name>/tags/list`

```
[root@docker ~]# curl http://mycompany.com:5000/v2/masociete/monimage/tags/list  
{ "name": "masociete/monimage", "tags": [ "1.0" ] }
```

# Pull



- Pour extraire une image d'un serveur de registre, il faut
  - ▣ L' URL du serveur et le port
  - ▣ Image repository
  - ▣ Image tag

# Registre non sécurisé

- Par défaut, le démon Docker suppose que tous les registres sont sécurisés et bloque la communication avec des registres non sécurisés
  - ▣ Impossible de faire un `push` ou `pull`
- Pour permettre une communication avec un registre non sécurisé, le Daemon doit être démarré avec l'option `--insecure-registry` et chaque registre doit être ajouté

```
--insecure-registry myregistry:5000
```



# MODULE 5:

# DOCKER MACHINE



# Docker Machine

- ❑ **Docker Machine** est un outil qui provisionne des docker hôtes et installe le démon Docker
- ❑ Crée des docker hôtes supplémentaires sur votre propre ordinateur
- ❑ Crée des docker hôtes sur le cloud (Amazon AWS, DigitalOcean etc...)
- ❑ Docker Machine crée le serveur, installe le démon Docker et configure le client Docker

# Docker Machine



# Installing Machine

- ❑ Téléchargez la version binaire spécifique au système d'exploitation à l'adresse <https://github.com/docker/machine/releases>
- ❑ Placez le binaire dans un dossier de votre système
  - ❑ `/usr/local/bin`

# Version

- ❑ Exécuter `docker-machine -v` pour afficher la version
- ❑ Exécuter `docker-machine -help` pour avoir la liste des commandes

# Docker Machine sur Windows et OSX

---

- ❑ Fait partie du Docker Toolbox
- ❑ Toolbox installe Docker Machine et le client Docker

# docker machine

- La commande `docker-machine` permet de créer et gérer les Docker hosts sur divers environnements :
  - VirtualBox
  - Amazon AWS
  - DigitalOcean
  - Azure
  - Rackspace
  - etc ...
- Chaque environnement possède son propre plugin dans le binaire `docker-machine`

# Création d'un hôte

- ❑ Utilisez la commandes **docker-machine create** et spécifiez le driver de l'environnement
- ❑ Le pilote permet à docker-machine d'interagir avec l'environnement où vous souhaitez créer l'hôte
- ❑

```
docker-machine create --driver <driver>  
<hostname>
```



# VirtualBox



- ❑ L'utilisation de VirtualBox nous permet de fournir rapidement des hôtes Docker supplémentaires sur notre Windows ou Mac
- ❑ `docker-machine create --driver virtualbox testhost`

# VirtualBox

- ❑ `docker-machine` va télécharger la distribution Linux boot2docker, créer et démarrer une machine virtuelle VirtualBox qui exécute Docker
- ❑ `docker-machine` va créer automatiquement la clé SSH pour votre hôte

```
[root@docker ~]# docker-machine create --driver virtualbox host01
Creating CA: /root/.docker/machine/certs/ca.pem
Creating client certificate: /root/.docker/machine/certs/cert.pem
Running pre-create checks...
(host01) Image cache directory does not exist, creating it at
/root/.docker/machine/cache...
(host01) No default Boot2Docker ISO found locally, downloading the latest release...
(host01) Latest release for github.com/boot2docker/boot2docker is v17.09.0-ce
(host01) Downloading /root/.docker/machine/cache/boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/download/v17.09.0-ce/boot2docker.iso...
```

# Provisionnement dans le cloud

- Chaque fournisseur de cloud offre différentes options sur la commande create machine docker et leur propre driver
- Liste des drivers
  - ▣ Amazon Web Services
  - ▣ Google Compute Engine
  - ▣ IBM Softlayer
  - ▣ Microsoft Azure
  - ▣ Microsoft Hyper-V
  - ▣ Openstack
  - ▣ Rackspace
  - ▣ Oracle VirtualBox
  - ▣ VMware Fusion
  - ▣ VMware vCloud Air
  - ▣ VMware vSphere

# DigitalOcean

- ❑ Vous aurez besoin de votre jeton d'accès au compte DigitalOcean
- ❑ Spécifier la droplet (par défaut 512mb)
- ❑ L'image à installer (par défaut ubuntu-14-04-x64)
- ❑ La région

```
docker-machine create
    --driver digitalocean \
    --digitalocean-access-token <your access token> \
    --digitalocean-size 2gb \
    testhost
```

# AWS

- ❑ Pour créer des hôtes dans AWS, vous aurez besoin de votre
  - ❑ Clé d'accès AWS
  - ❑ Clé secrète AWS
  - ❑ L'ID VPC où lancer l'instance
- ❑ L'image par défaut utilisée est Ubuntu 14.04 LTS

```
docker-machine create
    --driver amazec2 \
    --amazec2-access-key <AWS access key> \
    --amazec2-secret-key <AWS secret key> \
    --amazec2-vpc-id <VPC ID> \
    testhost
```

# Liste des machines

- ❑ La commande `docker-machine ls` affiche toutes les machines hôtes qui ont été provisionnées
- ❑ Des hôtes sur différents fournisseurs de cloud

```
[root@docker ~]# docker-machine ls
```

| NAME  | ACTIVE | DRIVER     | STATE   | URL                       | SWARM | DOCKER      | ERRORS |
|-------|--------|------------|---------|---------------------------|-------|-------------|--------|
| aws1  | -      | amazonec2  | Running | tcp://34.215.62.69:2376   |       | v17.09.0-ce |        |
| aws2  | -      | amazonec2  | Running | tcp://35.160.76.216:2376  |       | v17.09.0-ce |        |
| node1 | -      | virtualbox | Running | tcp://192.168.99.100:2376 |       | v17.06.2-ce |        |
| node2 | -      | virtualbox | Running | tcp://192.168.99.101:2376 |       | v17.06.2-ce |        |

# Connexion au hôte

- Il existe 2 méthodes pour se connecter à un hôte que le docker-machine a provisionné
  - ▣ Utiliser la commande `docker-machine ssh`
  - ▣ Définissez les variables d'environnement pour pointer votre client Docker vers le démon sur l'hôte distant

# docker-machine env

- La commande `env` affiche les variables d'environnements qui doivent être configurées pour connecter votre client Docker au démon distant de l'hôte spécifié

□  
`docker-machine env <hostname>`

```
[root@docker ~]# docker-machine env aws1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://34.215.62.69:2376"
export DOCKER_CERT_PATH="/home/user1/.docker/machine/machines/aws1"
export DOCKER_MACHINE_NAME="aws1"
# Run this command to configure your shell:
# eval $(docker-machine env aws1)
```



# Connexion

- ❑ Exécuter la commande `eval $(docker-machine env <hostname>)` pour connecter le client Docker au démon du hôte
  - ❑ Fonctionne en définissant des variables d'environnement sur le client
- ❑ Exécuter la commande `eval $(docker-machine env -u )` pour déconnecter le client Docker du démon du hôte

# Hôte Actif

- ❑ Sur la sortie standard de la commande `docker-machine ls` la colonne “ACTIVE”, indique le hôte actif.
- ❑ L'hôte actif est la machine sur laquelle le client Docker est connecté
- ❑ L'hôte actif est configuré lors de l'exécution de la commande :  
`eval $(docker-machine env <hostname>)`
- ❑ La commande `docker-machine active` indique aussi le hôte actif

# Réglage de l'hôte actif

```
[root@docker ~]# docker-machine ls
```

| NAME  | ACTIVE | DRIVER     | STATE   | URL                       | SWARM | DOCKER      | ERRORS |
|-------|--------|------------|---------|---------------------------|-------|-------------|--------|
| node1 | -      | virtualbox | Running | tcp://192.168.99.100:2376 |       | v17.09.0-ce |        |
| node2 | *      | virtualbox | Running | tcp://192.168.99.101:2376 |       | v17.06.2-ce |        |

```
[root@docker ~]# eval $(docker-machine env node2)
```

```
[root@docker ~]# docker-machine ls
```

| NAME  | ACTIVE | DRIVER     | STATE   | URL                       | SWARM | DOCKER      | ERRORS |
|-------|--------|------------|---------|---------------------------|-------|-------------|--------|
| aws1  | -      | amazonec2  | Stopped |                           |       | Unknown     |        |
| aws2  | -      | amazonec2  | Stopped |                           |       | Unknown     |        |
| node1 | -      | virtualbox | Running | tcp://192.168.99.100:2376 |       | v17.09.0-ce |        |
| node2 | *      | virtualbox | Running | tcp://192.168.99.101:2376 |       | v17.06.2-ce |        |

# Déconnexion du client

- ❑ Pour déconnecter le client Docker du démon distant, il faut désactiver les variables
  - ❑ DOCKER\_TLS\_VERIFY
  - ❑ DOCKER\_CERT\_PATH
  - ❑ DOCKER\_HOST
- ❑ Vous pouvez utiliser la commande unset ou la commande suivante

```
eval $(docker-machine env -u)
```

| Response | Percentage |
|----------|------------|
| Yes      | 75%        |
| No       | 25%        |

- ❑ La commande `docker-machine ssh` nous permet de nous connecter à un hôte provisionné en utilisant SSH
- ❑ Connexion en utilisant la clé SSH créée lors de la création du hôte

```
[root@docker ~]# docker-machine ssh node1
```

##

[illegible]

```
Boot2Docker version 17.09.0-ce, build HEAD : 06d5c35 - Wed Sep 27 23:22:43 UTC 2017
```

Docker version 17.09.0-ce, build afdb6d4

```
docker@node1:~$
```

# Docker machine SSH

- Peut également être utilisé pour exécuter une commande sur la machine spécifiée

```
[root@docker ~]# docker-machine ssh node1  docker version
```

```
Client:
```

```
Version:      17.09.0-ce  
API version:  1.32  
Go version:   go1.8.3  
Git commit:   afdb6d4  
Built:        Tue Sep 26 22:39:28 2017  
OS/Arch:      linux/amd64
```

```
Server:
```

```
Version:      17.09.0-ce  
API version:  1.32 (minimum version 1.12)  
Go version:   go1.8.3  
Git commit:   afdb6d4  
Built:        Tue Sep 26 22:45:38 2017  
OS/Arch:      linux/amd64  
Experimental: false
```

# Démarrer et arrêter un hôte

- ❑ Arrêter une machine hôte
  - ❑ `docker-machine stop <machine name>`
- ❑ Démarrer une machine hôte arrêtée
  - ❑ `docker-machine start <machine name>`
- ❑ Pour redémarrer une machine hôte
  - ❑ `docker-machine restart <machine name>`

```
[root@docker ~]# docker-machine ls
```

| NAME  | ACTIVE | DRIVER     | STATE   | URL                       | SWARM | DOCKER      | ERRORS |
|-------|--------|------------|---------|---------------------------|-------|-------------|--------|
| node1 | -      | virtualbox | Running | tcp://192.168.99.100:2376 |       | v17.09.0-ce |        |
| node2 | *      | virtualbox | Running | tcp://192.168.99.101:2376 |       | v17.06.2-ce |        |

```
[root@docker ~]# docker-machine stop node1
```

```
Stopping "node1"...
```

```
Machine "node1" was stopped.
```

```
[root@docker ~]# docker-machine ls
```

| NAME  | ACTIVE | DRIVER     | STATE   | URL                       | SWARM | DOCKER      | ERRORS |
|-------|--------|------------|---------|---------------------------|-------|-------------|--------|
| node1 | -      | virtualbox | Stopped |                           |       | Unknown     |        |
| node2 | *      | virtualbox | Running | tcp://192.168.99.101:2376 |       | v17.06.2-ce |        |

# Inspecting host details

- `docker-machine inspect` permet d'obtenir des détails sur la machine hôte, les informations du driver, les chemins de certification, l'adresse IP ...

```
[root@docker ~]# docker-machine inspect node2
{
  "ConfigVersion": 3,
  "Driver": {
    "IPAddress": "192.168.99.101",
    "MachineName": "node2",
    "SSHUser": "docker",
    "SSHPort": 57355,
    "SSHKeyPath": "/Users/samir/.docker/machine/machines/node2/id_rsa",
    "StorePath": "/Users/samir/.docker/machine",
    "SwarmMaster": false,
    "SwarmHost": "tcp://0.0.0.0:3376",
    "SwarmDiscovery": "",
    "VBoxManager": {},
    "HostInterfaces": {},
    "CPU": 2,
    "Memory": 2048,
    "DiskSize": 20000,
```



# Obtenir l'adresse IP d'un hôte

- ❑ Vous pouvez voir l'adresse IP en regardant la colonne URL sur la sortie de la commande

```
docker-machine ls
```

- ❑ **Par la commande**

```
docker-machine ip <host name>
```

```
[root@docker ~]# docker-machine ip node2  
192.168.99.101
```

# Suppression d'hôtes

- ❑ La commande `docker-machine rm` supprime l'hôte
- ❑ Cela supprimera l'hôte sur l'environnement (local ou cloud) et supprime le dossier de référence local

`/home/<user>/.docker/machine/machines/<machine name>`

```
[root@docker ~]# docker-machine rm node1
```

```
About to remove node1
```

```
WARNING: This action will delete both local reference and remote instance.
```

```
Are you sure? (y/n): y
```

```
Successfully removed node1
```

# MODULE 6: DOCKER SWARM



# Swarm



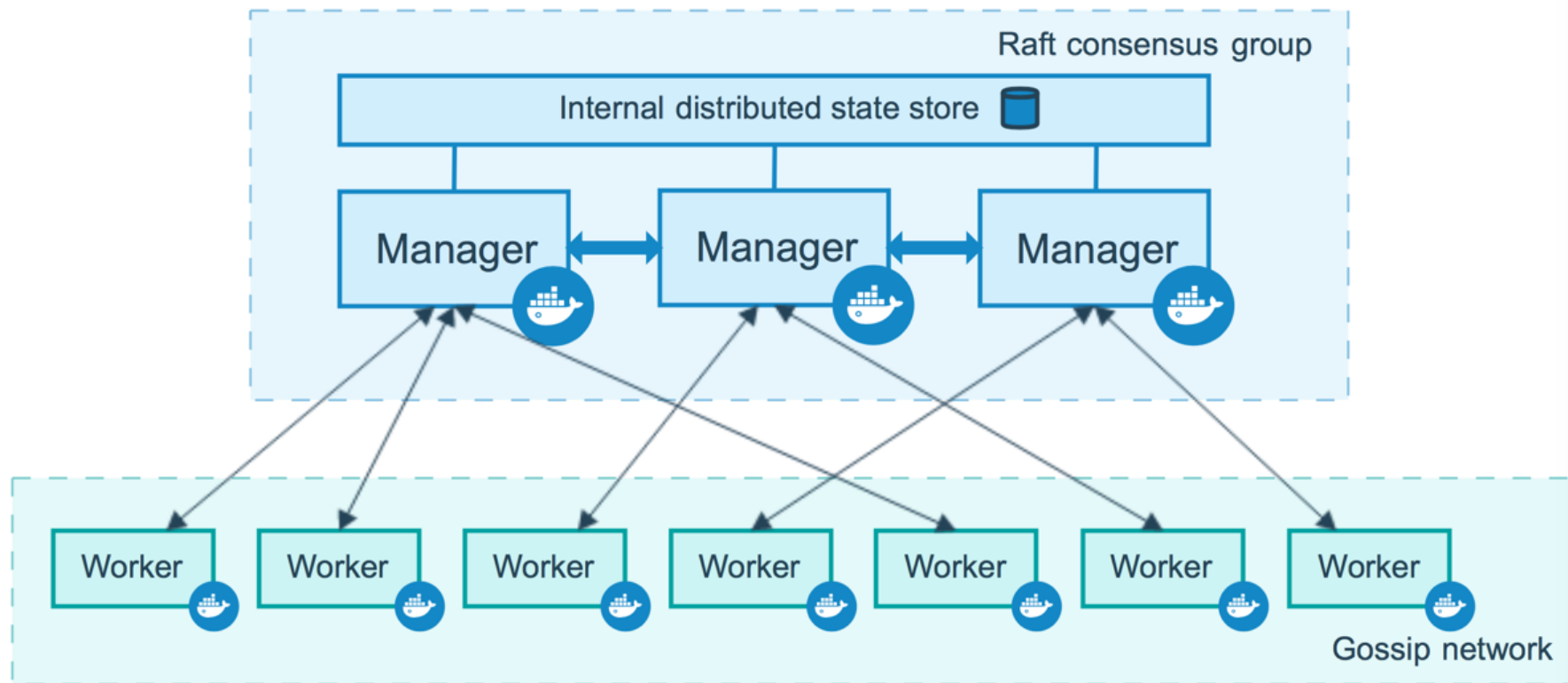
- ❑ **Docker Swarm** est un cluster de hôtes Docker en cluster sur lequel on déploie des conteneurs
- ❑ Permet de distribuer les charges de travail des conteneurs sur plusieurs machines s'exécutant dans un cluster
- ❑ L'API de Docker inclue des commandes pour gérer les noeuds du cluster (ajout, suppression des nœuds), et déployer et organiser des services à travers le cluster

# Modèle



- ❑ Pour déployer votre application dans un cluster, vous soumettez un service à un nœud de Manager. Le nœud Manager distribue les unités de travail appelées tasks aux nœuds Worker.
- ❑ Les nœuds de gestion effectuent également l'orchestration et les fonctions de gestion de cluster requises pour maintenir l'état souhaité du cluster. Les nœuds Manager élisent un seul Maître pour mener des tâches d'orchestration
- ❑ Les nœuds Worker reçoivent et exécutent les tâches envoyées depuis les nœuds Manager. Par défaut, les nœuds Manager exécutent également des services en tant que nœuds Worker, mais vous pouvez les configurer pour exécuter des tâches de gestion

# Architecture



# Agent



- Un agent s'exécute sur chaque nœud de travail et rapporte les tâches qui lui sont affectées. Le nœud Worker notifie au nœud du gestionnaire l'état actuel de ses tâches assignées afin que le gestionnaire puisse maintenir l'état désiré de chaque Worker.

# Service



- Un service est la définition des tâches à exécuter sur le noeud Manager ou les nœuds Worker. C'est la structure centrale du système swarm.
- Lorsque vous créez un service, vous spécifiez l'image du conteneur à utiliser et les commandes à exécuter à l'intérieur des conteneurs en cours d'exécution.



# Modèle



- Dans le modèle de services Replicas, le Manager distribue un nombre de copie spécifié de tâches aux Workers
- Dans le modèle de services Global, swarm exécute une tâche pour le service sur chaque noeud disponible dans le cluster.

# Modèle



- Une tâche comporte un conteneur Docker et les commandes à exécuter à l'intérieur du conteneur. C'est l'unité de planification atomique de swarm.
- Les noeuds Manager attribuent des tâches aux noeuds du travail en fonction du nombre de répliques définies dans l'option scale du service.
- Une fois qu'une tâche est attribuée à un noeud, elle ne peut pas se déplacer vers un autre noeud. Il ne peut fonctionner que sur le noeud assigné ou échouer.

# Création du cluster swarm

- La commande suivante permet de créer un nouveau cluster swarm :

```
docker swarm init --advertise-addr <MANAGER-IP>
```

```
docker@node1:~$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (f9js2mda7uf70bqydetmcgo6p) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-3716t7lma6yeexjlmirjhnma2yh5ydrft5 192.168.99.100:2377
```

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

# Etat du cluster swarm

- La commande suivante permet de visualiser l'état du cluster swarm :

```
docker info
```

```
docker@node1:~$ docker info
```

```
Logging Driver: json-file
```

```
Cgroup Driver: cgroupfs
```

```
Plugins:
```

```
Volume: local
```

```
Network: bridge host macvlan null overlay
```

```
Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
```

```
Swarm: active
```

```
NodeID: f9js2mda7uf70bqydetmcgo6p
```

```
Is Manager: true
```

```
ClusterID: xh1d7tk4ja3qgk5nf5nljvjp1
```

```
Managers: 1
```

```
Nodes: 1
```

```
Orchestration:
```

```
Task History Retention Limit: 5
```

# Liste des noeuds

- La commande suivante permet de lister les neouds du cluster swarm :  
`docker node ls`

```
docker@node1:~$ docker node ls
```

| ID                          | HOSTNAME | STATUS | AVAILABILITY | MANAGER |
|-----------------------------|----------|--------|--------------|---------|
| f9js2mda7uf70bqydetmcgo6p * | node1    | Ready  | Active       | Leader  |

# Ajout des Workers

La commande suivante permet d'ajouter un Worker dans le cluster swarm :

```
docker swarm join -token <token> IP-Manager
```

```
docker@node2:~$ docker swarm join --token SWMTKN-1-3716t7lm29 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@node1:~$ docker node ls
```

| ID                          | HOSTNAME | STATUS | AVAILABILITY | MANAGER |
|-----------------------------|----------|--------|--------------|---------|
| f9js2mda7uf70bqydetmcgo6p * | node1    | Ready  | Active       | Leader  |
| 2bkuf00chye6uy4cebr2728jr   | node2    | Ready  | Active       |         |

# Déployer un service

La commande suivante permet de deployer un service dans le cluster swarm avec le nombre de réplicas désiré :

```
docker service create --replicas <#> Image
```

```
docker@node1:~$ docker service create --replicas 2 --name ping alpine ping 127.0.0.1  
loemi7bv1kne7wo2srkf04kiv
```

Since `--detach=false` was not specified, tasks will be created in the background.

In a future release, `--detach=false` will become the default.

```
docker@node1:~$ docker service ls
```

| ID           | NAME | MODE       | REPLICAS | IMAGE         |
|--------------|------|------------|----------|---------------|
| loemi7bv1kne | ping | replicated | 2/2      | alpine:latest |

# Inspector un service

```
docker service inspect --pretty <service>
```

```
docker@node1:~$ docker service inspect --pretty ping
```

```
ID:                loemi7bvlkne7wo2srkf04kiv
Name:              ping
Service Mode:      Replicated
  Replicas:         2
Placement:
UpdateConfig:
  Parallelism:      1
  On failure:        pause
  Monitoring Period: 5s
  Max failure ratio: 0
```



# Distribution des services

```
docker service ps <service>
```

```
docker@node1:~$ docker service ps ping
```

| ID                     | NAME          | IMAGE         | NODE  | DESIRED |
|------------------------|---------------|---------------|-------|---------|
| STATE                  | CURRENT STATE | ERROR         | PORTS |         |
| nd9ntu992wex           | ping.1        | alpine:latest | node1 | Running |
| Running 10 minutes ago |               |               |       |         |
| acixdloq7zrq           | ping.2        | alpine:latest | node2 | Running |
| Running 10 minutes ago |               |               |       |         |

# Modifier le nombre de replicas

```
docker service scale service=#
```

```
docker@node1:~$ docker service scale ping=4
```

```
ping scaled to 4
```

Since `--detach=false` was not specified, tasks will be scaled in the background.  
In a future release, `--detach=false` will become the default.

```
docker@node1:~$ docker service ps ping
```

| ID                      | NAME          | IMAGE         | NODE  | DESIRED |
|-------------------------|---------------|---------------|-------|---------|
| STATE                   | CURRENT STATE | ERROR         | PORTS |         |
| nd9ntu992wex            | ping.1        | alpine:latest | node1 | Running |
| Running 14 minutes ago  |               |               |       |         |
| acixdloq7zrq            | ping.2        | alpine:latest | node2 | Running |
| Running 14 minutes ago  |               |               |       |         |
| js2ubbkubb8a            | ping.3        | alpine:latest | node3 | Running |
| Preparing 3 seconds ago |               |               |       |         |
| 64gkoa3invol            | ping.4        | alpine:latest | node3 | Running |
| Preparing 3 seconds ago |               |               |       |         |

100

# Noeud en état Drain

```
docker node update -availability drain <service>
```

```
docker@node1:~$ docker node update --availability drain node2
Node2
```

```
docker@node1:~$ docker service ps ping
```

| ID                      | NAME          | IMAGE         | NODE  | DESIRED  |
|-------------------------|---------------|---------------|-------|----------|
| STATE                   | CURRENT STATE | ERROR         | PORTS |          |
| nd9ntu992wex            | ping.1        | alpine:latest | node1 | Running  |
| Running 20 minutes ago  |               |               |       |          |
| jbr3gzlmywgc            | ping.2        | alpine:latest | node1 | Running  |
| Running 10 seconds ago  |               |               |       |          |
| acixdloq7zrq            | \_ ping.2     | alpine:latest | node2 | Shutdown |
| Shutdown 10 seconds ago |               |               |       |          |
| js2ubbkubb8a            | ping.3        | alpine:latest | node3 | Running  |
| Running 5 minutes ago   |               |               |       |          |
| 64gkoa3invol            | ping.4        | alpine:latest | node3 | Running  |
| Running 5 minutes ago   |               |               |       |          |

# Noeud en état active

```
docker node update -availability active <node>
```

```
docker@node1:~$ docker node update --availability active node2
```

```
node2
```

```
docker@node1:~$ docker service ps ping
```

| ID                         | NAME          | IMAGE         | NODE     | DESIRED     |
|----------------------------|---------------|---------------|----------|-------------|
| STATE                      | CURRENT STATE | ERROR         | PORTS    |             |
| adj283f3acaq               | ping.1        | alpine:latest | node1    | Running     |
| Running 48 seconds ago     |               |               |          |             |
| sed6bo01e5o5               | \_            |               |          |             |
| ping.1                     | alpine:latest | node2         | Shutdown | Shutdown 50 |
| seconds ago                |               |               |          |             |
| rqcjsjz9qaxn               | ping.2        | alpine:latest | node1    | Running     |
| Running 2 minutes ago      |               |               |          |             |
| 4jt9khyb4exe               | ping.3        | alpine:latest | node3    | Running     |
| Running about a minute ago |               |               |          |             |
| 1522wwvt3ol8               | ping.4        | alpine:latest | node3    | Running     |
| Running about a minute ago |               |               |          |             |

# Noeud en état pause

```
docker node update -availability pause <node>
```

```
docker@node1:~$ docker node update --availability pause node2
```

```
node2
```

```
docker@node1:~$ docker node ls
```

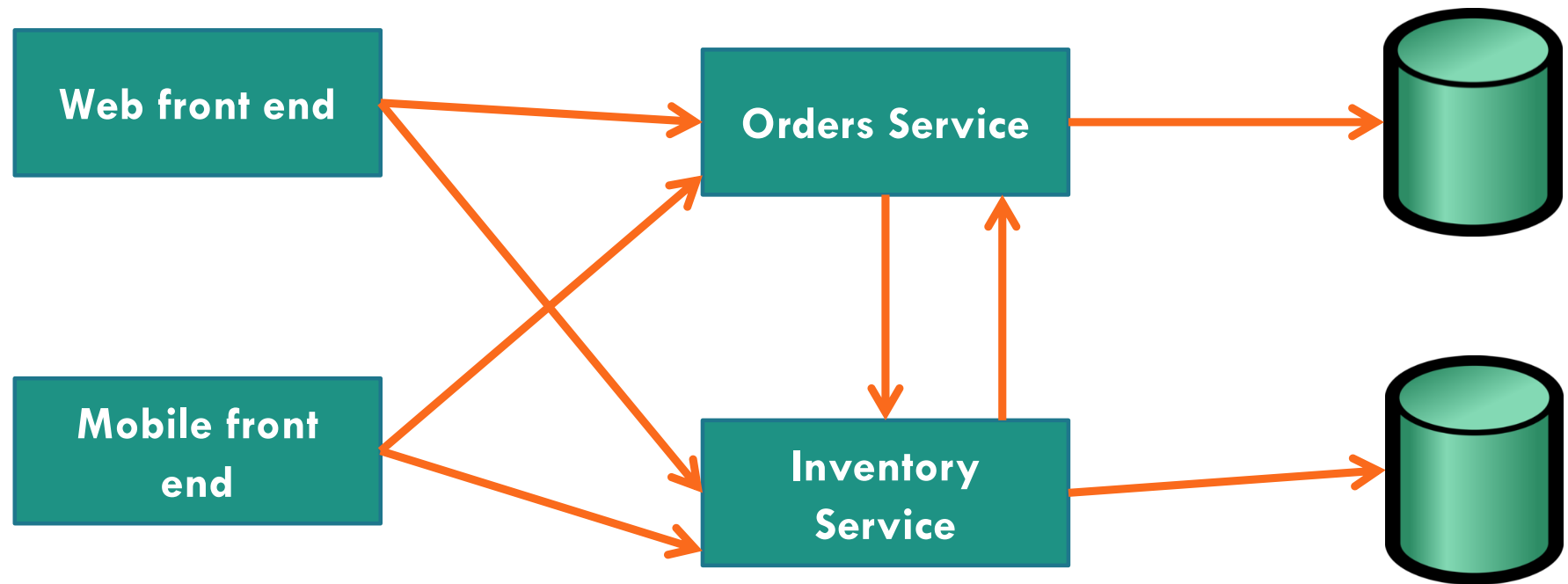
| ID                        | HOSTNAME   | STATUS | AVAILABILITY | MA       |
|---------------------------|------------|--------|--------------|----------|
| up8ru65qmpdu4vpe03qqr0u1  |            |        |              |          |
| * node1                   | Ready      | Active | Leader       | 18.05.0- |
| ce                        |            |        |              |          |
| oz3yp2dmri87hb8gnvpsqvi0h | node2      | Ready  | Pause        |          |
|                           | 18.05.0-ce |        |              |          |
| t9rhmo12lne238hngrq3j95vj | node3      | Ready  | Active       |          |
|                           | 18.05.0-ce |        |              |          |
| d                         |            |        |              |          |

# MODULE 7:

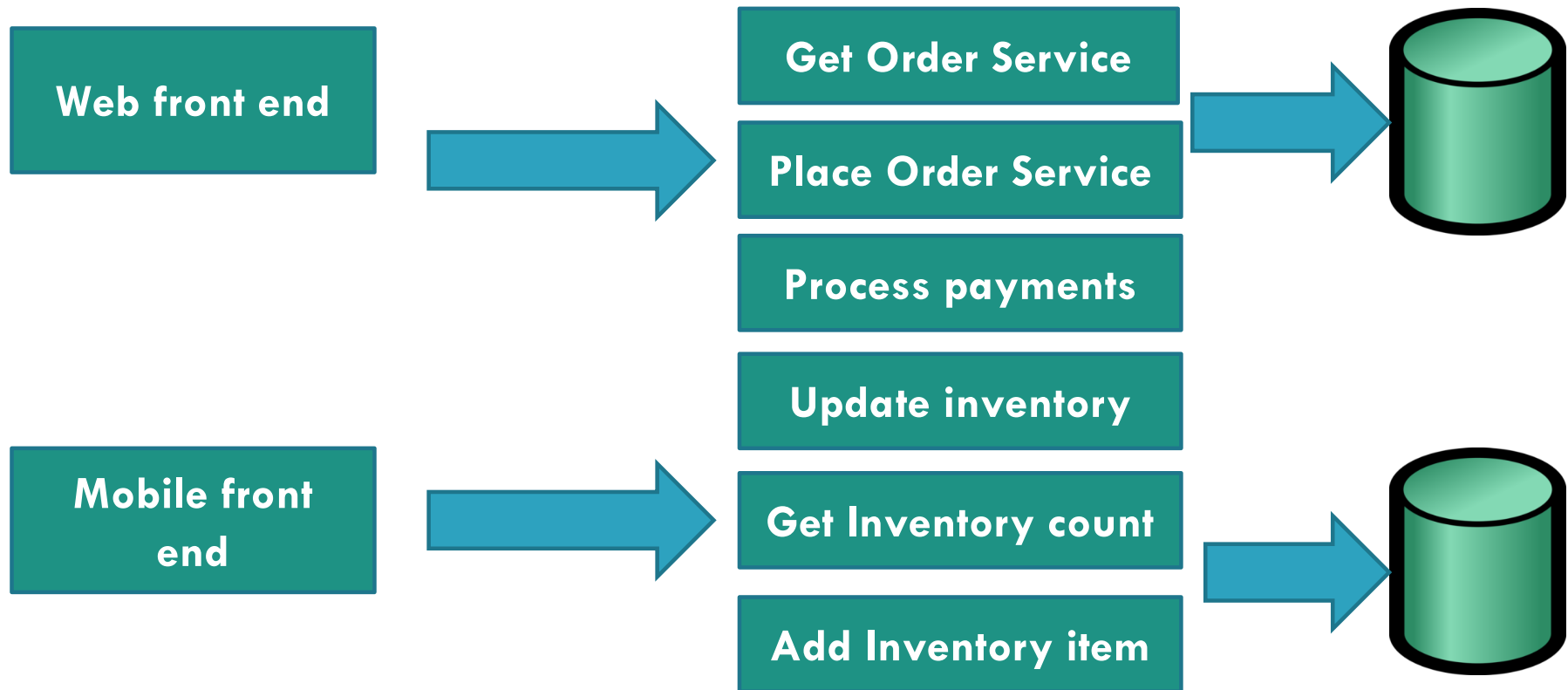
# DOCKER COMPOSE



# Exemple d'application



# Les micro-service de l'application





# Compose

- ❑ Docker Compose est un outil pour créer et gérer des applications multi-conteneurs
- ❑ Les conteneurs sont tous définis dans un seul fichier appelé `docker-compose.yml`
- ❑ Chaque conteneur gère un composant / service particulier de votre application
  - ❑ Web front end
  - ❑ User authentication
  - ❑ Payments
  - ❑ Database
- ❑ Les liens entre conteneurs sont définis
- ❑ Compose lancera tous vos conteneurs en une seule commande

# Installer Compose

- Vérifiez la dernière version sur

<https://github.com/docker/compose/releases>

**MAC et Windows** : Docker Compose est inclus dans Docker Toolbox

# Compose fichier yml

- ❑ Définit les services qui composent votre application
- ❑ Chaque service contient des instructions pour construire et gérer un conteneur

**service**

## Example

```
javaclient:
  build: .
  command: java HelloWorld
  links:
    - redis
redis:
  image: redis
```

# Compose fichier yml

- ❑ Version
- ❑ Référence top-level

**Utilisation**

**Déclaration**

```
version: "3.2"
services:
  web:
    image: nginx:alpine
    volumes:
      - dbdata:/var/lib/postgresql/data
volumes:
  mydata:
  dbdata:
```

# Instruction Build

- ❑ Build définit le chemin d'accès au fichier Docker qui sera utilisé pour créer l'image
- ❑ Le conteneur sera exécuté à l'aide de l'image construite
- ❑ Le chemin d'accès à la construction peut être un chemin relatif. Par rapport à l'emplacement du fichier yml



```
javaclient:
  build: .
orderservice:
  build:
    /src/com/company/service
```

# Instruction Image

- ❑ L'image définit l'image qui sera utilisée pour exécuter le conteneur
- ❑ L'image peut être locale ou distante
- ❑ Peut spécifier une étiquette ou un ID d'image
- ❑ Tous les services doivent comporter une instruction build ou image

Use the latest redis  
Image from Docker  
Hub

```
javaclient:
  image:
johnnytu/myclient:1.0
redis:
  image: redis
```

# Exemple

- Deux services
  - ▣ Java client
  - ▣ Redis
- Le client java est une classe Java simple qui se connecte à notre serveur Redis et obtient la valeur d'une clé
- Code disponible sur <https://github.com/johnny-tu/HelloRedis.git>

# Code Java

```
import redis.clients.jedis.Jedis;

public class HelloRedis
{
    public static void main (String args [])
    {
        Jedis jedis = new Jedis("redisdb");

        while (true) {
            try {
                Thread.sleep(5000);
                System.out.println("Server is running: "+jedis.ping());
                String bookCount = jedis.get("books_count");
                System.out.println("books_count = " + bookCount);
            }
            catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

}



# Dockerfile

```
FROM java:7
COPY /src /HelloRedis/src
COPY /lib /HelloRedis/lib

WORKDIR /HelloRedis
RUN javac -cp lib/jedis-2.1.0-
sources.jar -d . \
    src/HelloRedis.java
```

# Links

- ❑ Crée une entrée pour l'alias à l'intérieur des conteneurs dans le fichier `/etc/hosts`

```
javaclient:
  build: .
  command: java HelloWorld
  links:
    - redis
redis:
  image: redis
```

# Exécuter l'application

- ❑ **docker-compose up**
- ❑ La commande Up
  - ▣ Créez l'image pour chaque service
  - ▣ Créer et démarrer les conteneurs
  - ▣ Les conteneurs source sont démarrés avant les destinataires
- ❑ Les conteneurs peuvent tous fonctionner au premier plan ou en mode détaché

# Exécuter l'application

```
docker@node1:~/HelloRedis-master$ docker-compose up
Pulling redis (redis:latest)...
latest: Pulling from library/redis
065132d9f705: Pull complete
Status: Downloaded newer image for redis:latest
Building javaclient
Step 1/6 : FROM java:7
7: Pulling from library/java
6263baad4f89: Pull complete
Creating helloredismaster_redis_1 ...
Creating helloredismaster_javaclient_1 ...
Attaching to helloredismaster_redis_1, helloredismaster_javaclient_1
redis_1      | 1:C 05 Oct 04:39:55.519 # oO00oO00oO00o Redis is starting oO00oO00oO00o
redis_1      | 1:M 05 Oct 04:39:55.521 * Ready to accept connections

javaclient_1 | Server is running: PONG
javaclient_1 | books_count = null
```

# Visualiser containers

- ❑ `docker-compose ps` permet de lister les services lancés par Compose
- ❑ Cela affichera uniquement les services qui ont été lancés depuis Compose tel que définis dans le fichier `docker-compose.yml`
- ❑ La commande doit être exécutée dans le dossier avec le fichier `yml`

```
docker@node1:~/HelloRedis-master$ docker-compose ps
```

| Name                          | Command                        | State | Ports    |
|-------------------------------|--------------------------------|-------|----------|
| helloredismaster_javaclient_1 | java HelloRedis                | Up    |          |
| helloredismaster_redis_1      | docker-entrypoint.sh redis ... | Up    | 6379/tcp |

# Start et stop

- ❑ **Arrêt d'un service**

`docker-compose stop <service name>`

- ❑ **Arrêt de tous les services**

`docker-compose stop`

- ❑ **Redémarrage d'un service**

`docker-compose start <service name>`

- ❑ **Redémarrage de tous les services**

`docker-compose start`

# Suppression

- ❑ Vous pouvez supprimer manuellement chaque conteneur de service avec la commande `docker rm`
- ❑ Ou `docker-compose rm` pour supprimer tous les conteneurs de service qui ont été arrêtés
- ❑ Spécifier un service à supprimer  
`docker-compose rm <service name>`
- ❑ Utilisez l'option `-v` pour supprimer les volumes associés  
`docker-compose rm -v <service name>`

# logs



`docker-compose logs`

- Si un service n'est pas spécifié, le journal agrégé de tous les conteneurs sera affiché



# Scale

- Dans une architecture de micro-service, nous avons la flexibilité d'étendre un service particulier pour gérer une charge plus grande

```
docker-compose scale <service name>=<instances>
```

```
docker@node1:~/HelloRedis-master$ docker-compose scale javaclient=2
Starting helloredismaster_javaclient_1 ... done
Creating helloredismaster_javaclient_2 ...
Creating helloredismaster_javaclient_2 ... Done
```

```
docker@node1:~/HelloRedis-master$ docker-compose ps
```

| Name                          | Command                        | State | Ports    |
|-------------------------------|--------------------------------|-------|----------|
| helloredismaster_javaclient_1 | java HelloRedis                | Up    |          |
| helloredismaster_javaclient_2 | java HelloRedis                | Up    |          |
| helloredismaster_redis_1      | docker-entrypoint.sh redis ... | Up    | 6379/tcp |

## Compose et Dockerfile

- ❑ La plupart des paramètres dans un fichier `docker-compose.yml` ont une instruction équivalente dans Dockerfile
- ❑ Les options déjà spécifiées dans Dockerfile sont respectées par Docker Compose et n'ont pas besoin d'être spécifiées à nouveau

# Docker Compose build

- ❑ `docker-compose build` command construira les images pour vos services définis sans démarrer les services
- ❑ **Tags des images** `<project name>_<service name>:latest`
- ❑ `--no-cache` **Ne pas utiliser le cache lors de la construction de l'image**

```
docker@node1:~/HelloRedis-master$ docker images
```

| REPOSITORY                  | TAG    | IMAGE ID     | CREATED       | SIZE  |
|-----------------------------|--------|--------------|---------------|-------|
| helloredismaster_javaclient | latest | 0179fc6723b3 | 5 minutes ago | 585MB |

# Variable substitution

- ❑ Les variables d'environnement peuvent être utilisées dans votre fichier de configuration YAML
- ❑ **Syntaxe** `$VARIABLE` ou `${VARIABLE}`

```
webapp:  
  image: jtu/mywebapp:${MYAPP_VERSION}  
  ...  
  ...
```

# Specifiser autre fichier yml

- ❑ Le fichier de configuration par défaut est `docker-compose.yml`
- ❑ Nous pouvons spécifier un autre fichier de configuration en utilisant l'option `-f`  
`docker-compose -f docker-ver3.yml`

# Multiplés fichiers yml

- Lorsque plusieurs fichiers de configuration Compose sont spécifiés à l'aide de l'option -f, la configuration de ces fichiers est combinée.

```
docker-compose -f docker-compose.yml \  
                -f docker-compose.debug.yml
```

- La configuration est construite dans l'ordre des fichiers listés