

# Docker

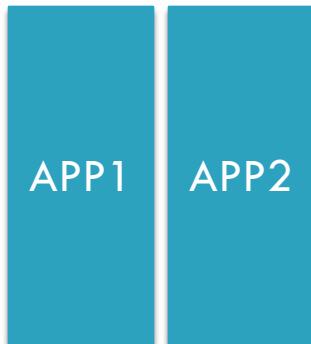
Virtualisation légère

# DOCKER

Module 1: Introduction

# Problématique: Consolidation des serveurs

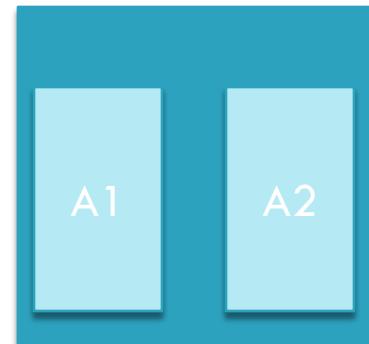
Partitionnement



Virtualisation de Hardware



Virtualisation d' OS



Partitionnement  
Physique

Host Based

Hyperviseur  
Bare Metal

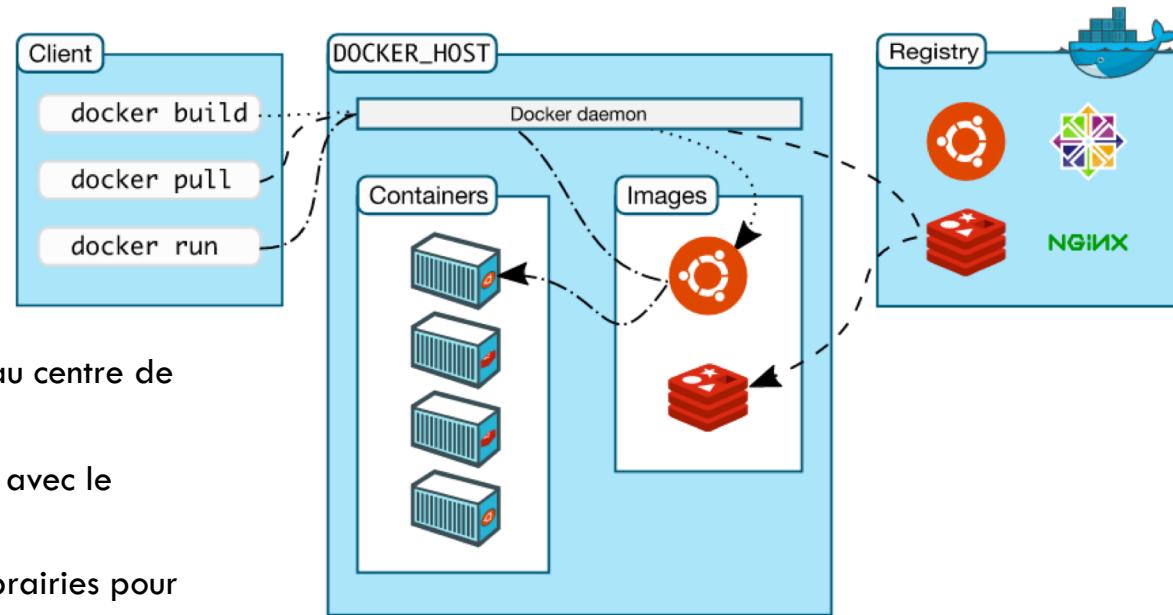
Hyperviseur  
FirmWare

Conteneur

# Architecture de Docker

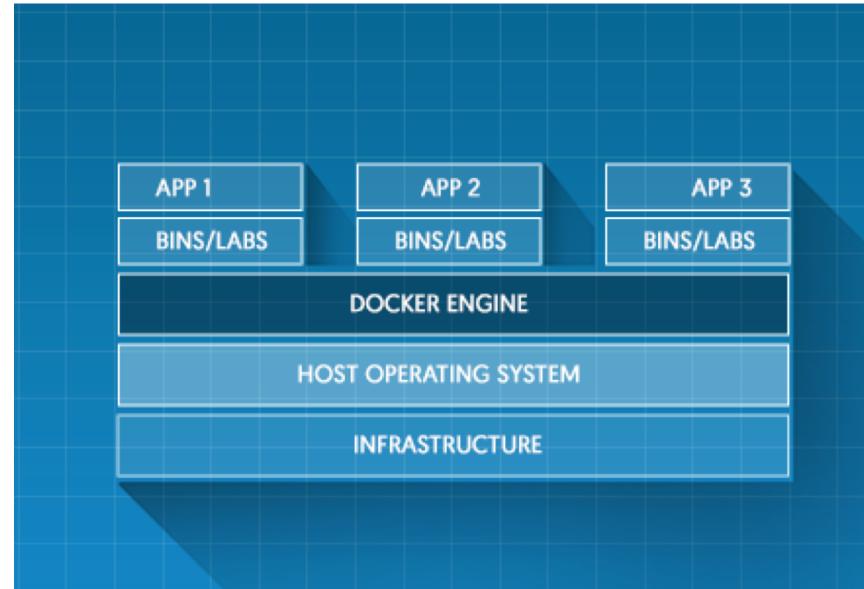
Docker est basé sur une architecture client-serveur :

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



# Docker Engine

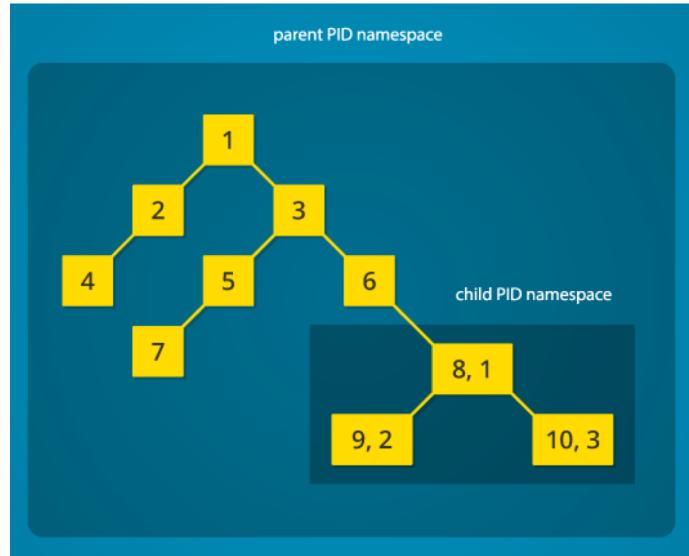
- Docker Engine est le daemon serveur au centre de l'architecture qui permet d'exécuter les conteneurs.
- Docker Engine utilise les namespaces Linux Kernel et les groupes de contrôle.
- L'isolation des containers est garantie par les namespaces.



# Namespaces

## ▣ Namespaces

- Isolation d'une hierarchie 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

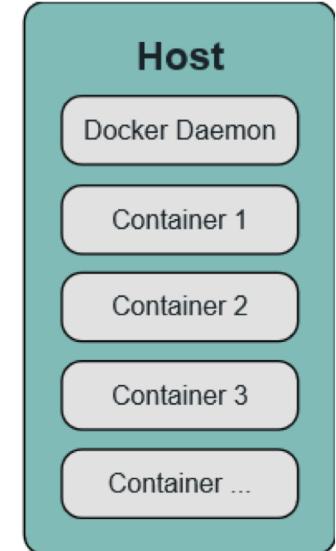
- Le client Docker se connecte au Docker Engine.
- Le client peut-être installé sur la même machine que le daemon ou sur une machine différente.
- Deux types de client Docker :
  - CLI
  - GUI

## 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

# Conteneur

- ❑ Espace d'exécution d'application isolé.
- ❑ Contient tout ce qui est nécessaire pour exécuter votre application.
- ❑ Basé sur une ou plusieurs images.

# Registres

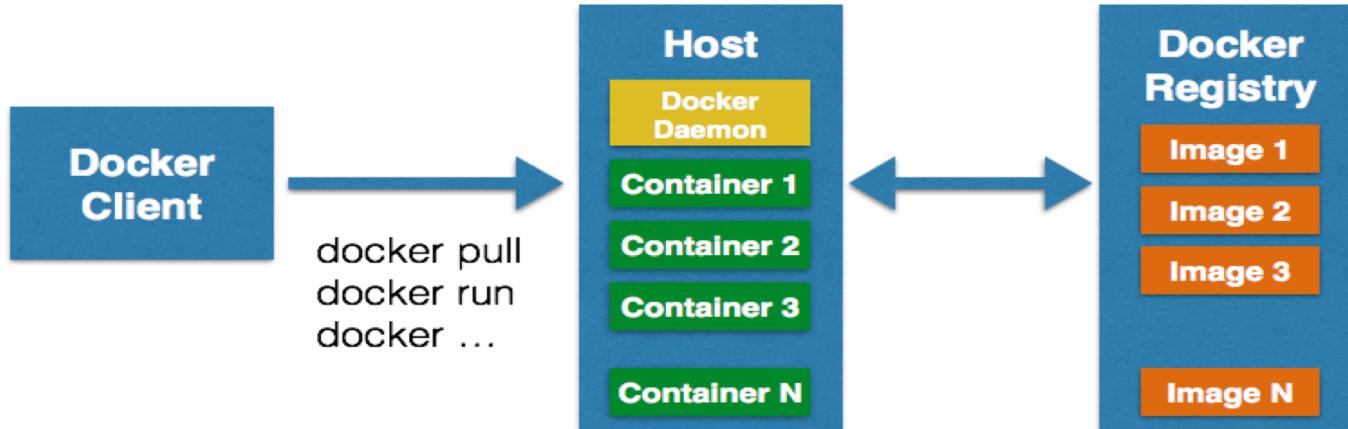
- Stockages des images.

- 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 Datacenter

- Docker Datacenter 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'un 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 par la plupart des distributions, sinon le créer par :

```
$ sudo groupadd docker
```

# 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

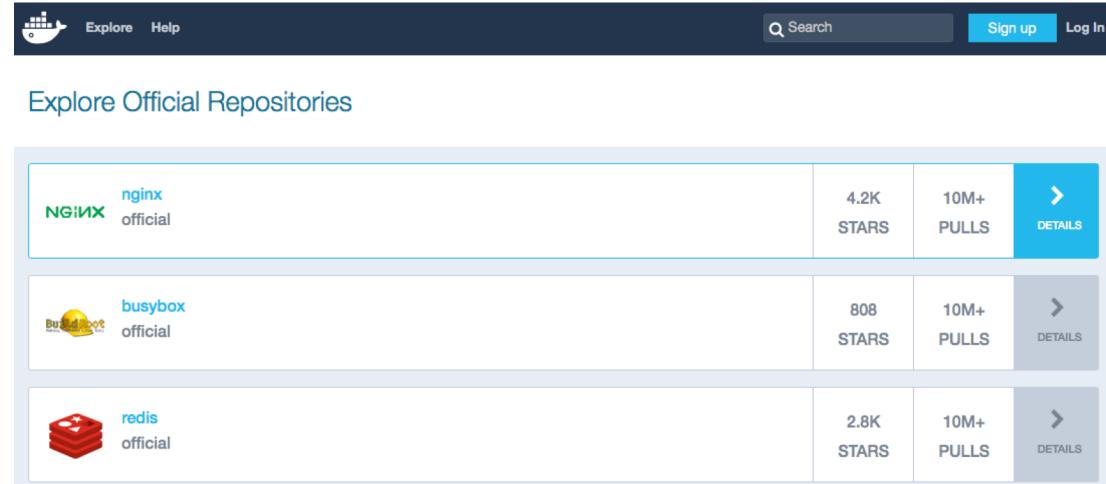
# DOCKER

Module 3: Images Docker

# Docker Hub

## ■ Docker Hub

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



# 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]
maxxcloo/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]
maxxcloo/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 [=====>]
4aceccff346f: Downloading [=====>]
c8967f302193: Download complete
] 12.06 MB/51.35 MB
] 15.76 MB/20.09 MB
```

# 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
```

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
d879ca1f2e2e        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
d879ca1f2e2e        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
8101cdf72c16        ubuntu:14.04      "echo "Hello World\xe2"
                                         "                    7 minutes ago       Exited (0) 7 minutes ago
aa21052c185b        hello-world      "/hello"
                                         "                    9 hours ago        Exited (0) 9 hours ago
```

# 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@e367d73cfbf:/#
root@e367d73cfbf:/# 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		IMAGE	COMMAND	CREATED	STATUS
CONTAINER ID	NAMES				
8a7960e90d1cdb55f91f4084101ec77ba59284d12e1cbe6c07dd7bb704440237	elated_hugle	ubuntu	"bash"	7 minutes ago	Up 7
d879calf2e2e065e1680d1543a962d5c6a4c4d9b112ab0578903be2e44827a5d	silly_galileo	nginx	"nginx -g 'daemon off;'"	17 minutes ago	Up 17

# 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
d879ca1f2e2e

$ 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
4870e430ae8b615ffb35764d164999658a4d37853c3b7b439cc0e118a0cf435a

$ 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
PORTS             NAMES
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
3eabe185bef5	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
```

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
$ 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
PORTS               NAMES
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
NAMES
99d2e3c0744b        ubuntu:14.04       "ping 127.0.0.1"    10 minutes ago   Up 2 seconds
```

# Inspecter un conteneur

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

```
$ docker inspect 99d2e3c0744b
[
  {
    "Id": "99d2e3c0744bb97e471121bb1649057f57c6bf5f834c20352e6cc616bdccce78",
    "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": ""
    }
  }
]
```

# Inspecter un conteneur

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

```
$ docker inspect 99d2e3c0744b
[
  {
    "Id": "99d2e3c0744bb97e471121bb1649057f57c6bf5f834c20352e6cc616bdccce78",
    "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": ""
    }
  }
]
```

# Filtrer docker inspect

- `--format='{{.Champs1.Champs2}}'`
- `--format='{{json .Champs2}}'`

```
$ 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
99d2e3c0744bb97e471121bb1649057f57c6bf5f834c20352e6cc616bcdcee78. Stop the container before attempting removal
or use -f

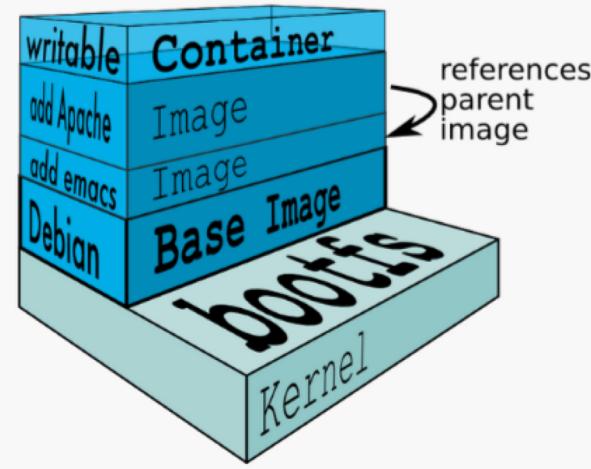
$ docker rm -f 99d2e3c0744b
99d2e3c0744b
```

# DOCKER

Module 5: Images

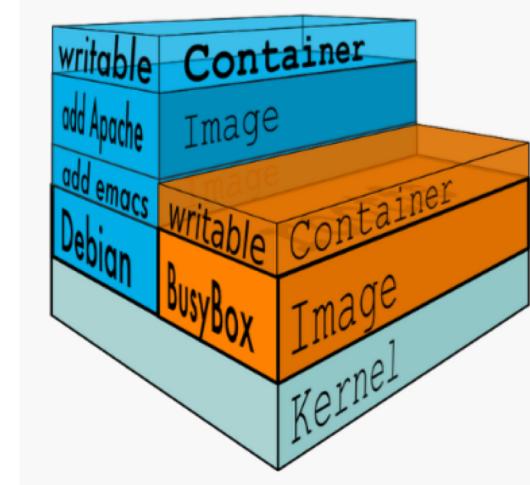
# Docker Layers : couches

- Une image est une collection de fichiers et de métadonnées
- 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

## ■ Crée 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
PORTS              NAMES
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
--> 211cce1f1f7cc
Removing intermediate container 761e158caf11
Successfully built 211cce1f1f7cc
```

← Couche temporaire

← Validation de l'image finale ← Suppression de la couche temporaire

```
$ docker images
REPOSITORY          TAG           IMAGE ID        CREATED         SIZE
masociete/monappli  1.0           211cce1f1f7cc  About a minute ago  307.1 MB
centos              7.0.1406      16e9fdecc1fe   3 weeks ago    210.2 MB
```

# 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 précédemment
  - Si oui, Docker utilisera le résultat (la couche) au lieu d'exécuter à nouveau l'instruction
- Docker utilise les instruction 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

```
$ 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 ← Utilisation du cache
--> 211cceff1f7cc
Step 3 : RUN yum install -y curl
--> Running in 40de6ceff281 ← Nouvelle commande dans une nouvelle couche
Loaded plugins: fastestmirror
```

# 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
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 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
16e9fddecc1fe 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 MAINTAINER

- **Syntaxe**

```
MAINTAINER John Doe <jdoe@organisation.com>
```

- Indique qui est l'auteur du Dockerfile et son email
- En option, mais l'inclure est une bonne pratique
- Habituellement placé juste après l'instruction FROM

# 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 ENTRYPPOINT 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 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
0f5ff0cf6a1c: 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 import et export
- 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 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.

# 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 consistence.

```
$ 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]
3339cabab23b1: 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 consistence.

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

```
Untagged: masociete/monappli1:1.0
```

```
$ docker rmi 4fc7d44b50e0
```

```
Untagged: ambre/training:latest
```

```
Deleted: sha256:4fc7d44b50e09760abfeff7dfb7ea2ff98a2738dcf882724f1d0fcc4f1fd3233
```

```
Deleted: sha256:09ea582e50d795aa9822f8f22a138e9d5c3dc43786e960c8c75cabbe2f68412
```

# DOCKER

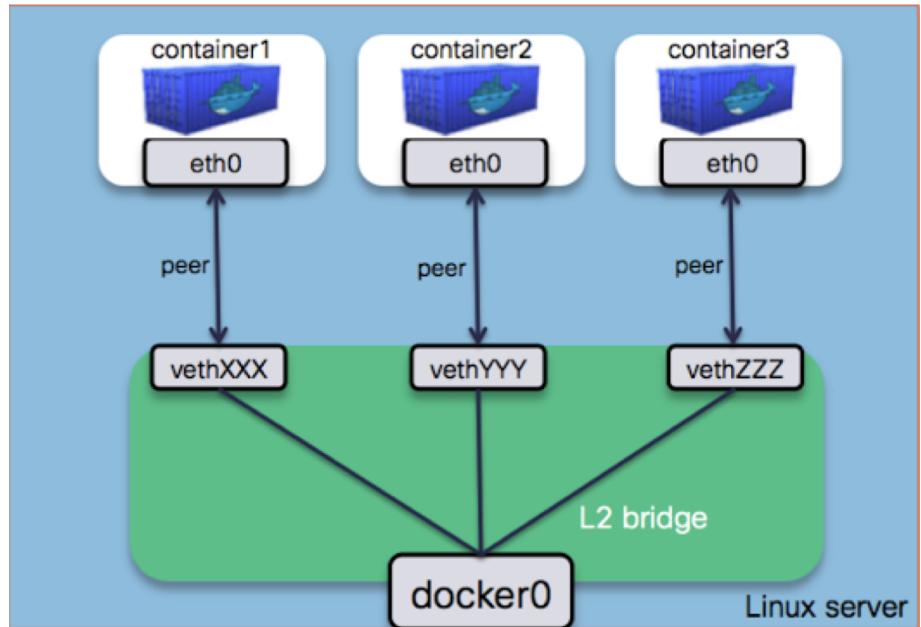
Module 7: Réseau

# 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
a1f9f5659e41    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": "a1f9f5659e411cebee106dd64830e77c1c22259be688f352b98406d212b0c42a",
        "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": "f05efee1d901ca2d59d68357b7b1cbef49f9dcc1829695e547ddd53d63f3468d",
            "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
E83404e00a6df20e3c1caedf47ef3903e3be69dfa8b20e6c272a7e7725fa77

$ 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]:[host container] <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 80:8080 nginx
A88adeed130e306cc5de15cbe808f959b178f5645ad391416dc61b18ab813363
```

```
$ docker rm -f `docker ps -aq`
A88adeed130e
```

```
$ docker run -d -p 80:80 -p 8080:8080 --name ngx nginx
ee3e0361ee27fc7cba5111eea5ab6e7678fc0c6864c34910bd33f027b478fa5b
```

```
$ docker port ngx
8080/tcp -> 0.0.0.0:8080
80/tcp -> 0.0.0.0:80
```

# 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 49.153 à 65.535
- Utiliser l'instruction EXPOSE pour exposer les ports lorsqu'on utilise un fichier Dockerfile

```
$ docker run -d -P --name ngx nginx
02791edeb4c1ed159729b76f9f8a00783dca9808a352506e28ce8b273b7c8245
```

```
$ docker port ngx
443/tcp -> 0.0.0.0:32772
80/tcp -> 0.0.0.0:32773
```

# DOCKER

Module 8: Volumes

# Volumes

- Un volume est répertoire dans un conteneur, qui est conçu pour la persistence 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

## ■ Crée 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-253:0-508564-71ac5904b3390844b362c785e5c5c7dfeeb1c2748ba812f929f3c0350c59875  10G  253M  9.8G  3% /
tmpfs                   497M    0  497M   0% /dev
tmpfs                   497M    0  497M   0% /sys/fs/cgroup
/dev/mapper/centos-root  14G  2.0G  12G  14% /www/test1
shm                      64M    0   64M   0% /dev/shm
```

## Monter un volume en lecture seule

```
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
$ docker run -it --volumes-from datas moncentos bash
```