

Docker / Introduction

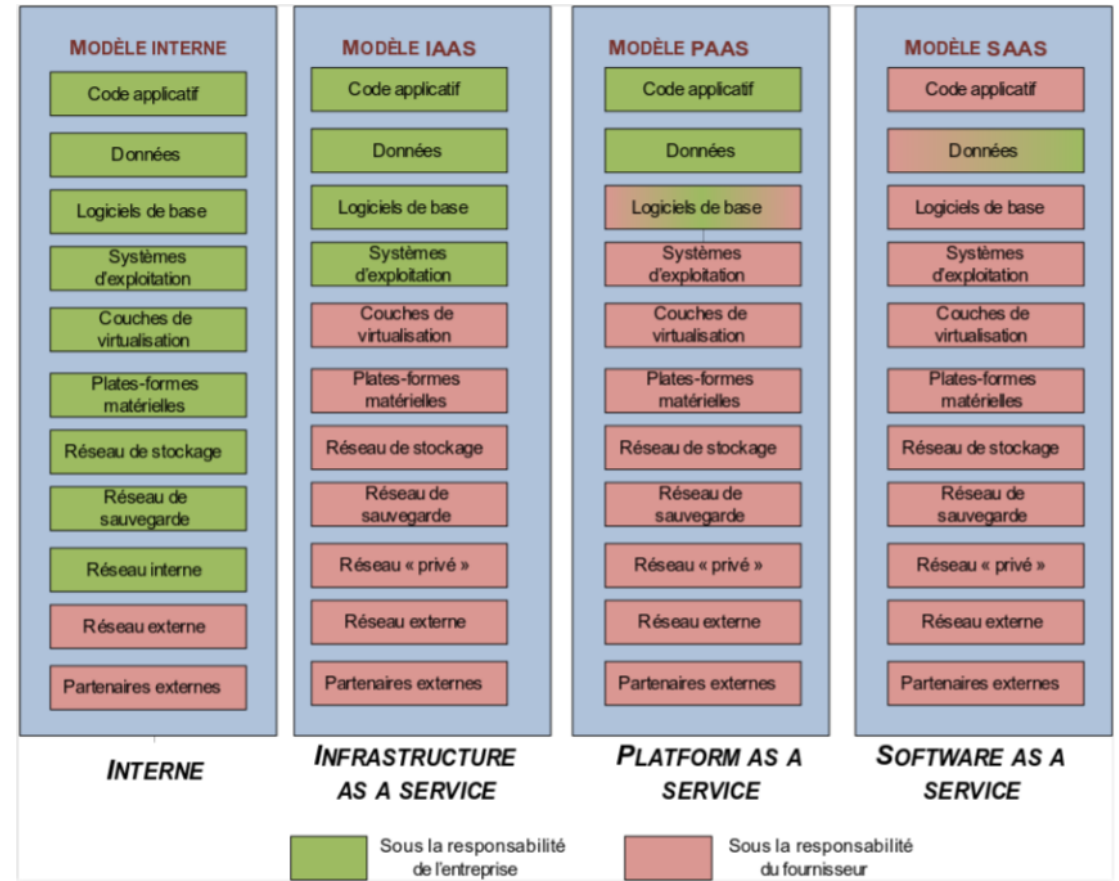
Ecole des Mines de Nantes – 12 & 13/12/2016

Saison 1 - Episode 2/2

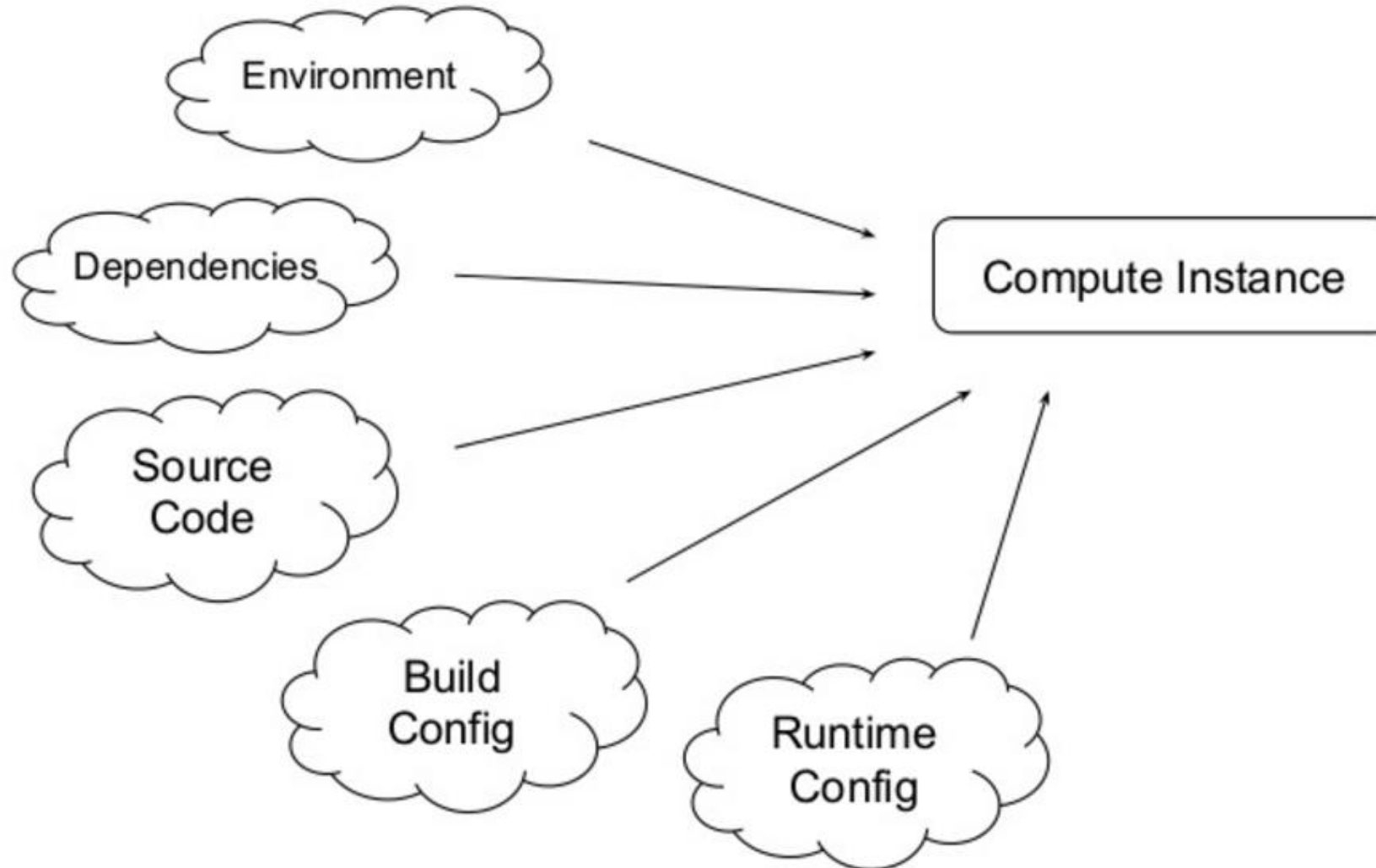
Contexte

L'infrastructure comme une commodité – le cloud

- Ressources de stockage / calcul distantes... externalisées
- Abstraction du matériel
- Accès normalisé via des APIs
- Service et facturation à la demande
- Flexibilité, élasticité



Problème



The matrix from Hell

django web frontend	?	?	?	?	?	?
node.js async API	?	?	?	?	?	?
background workers	?	?	?	?	?	?
SQL database	?	?	?	?	?	?
distributed DB, big data	?	?	?	?	?	?
message queue	?	?	?	?	?	?
	my laptop	your laptop	QA	staging	prod on cloud VM	prod on bare metal

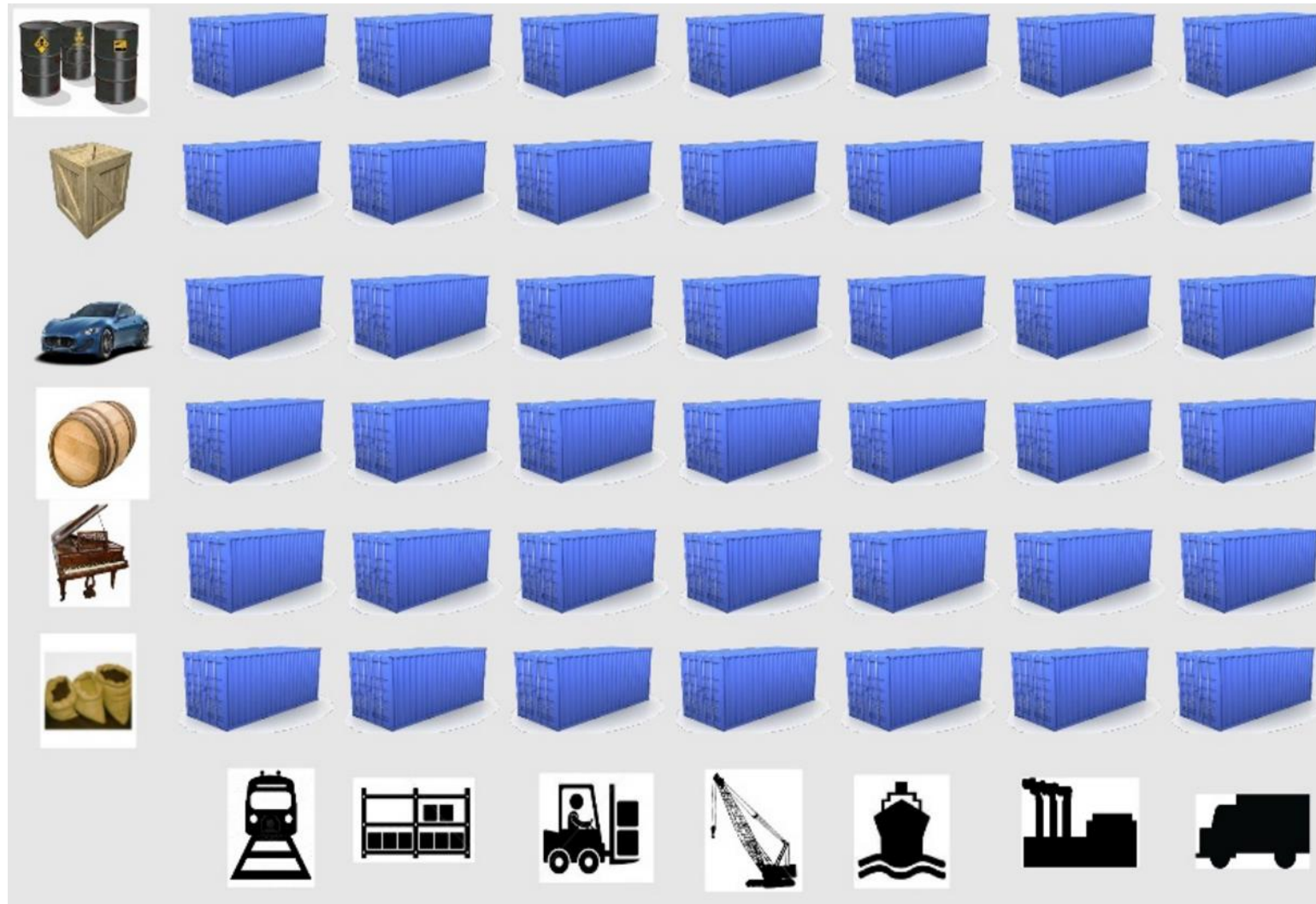
Another Matrix from Hell



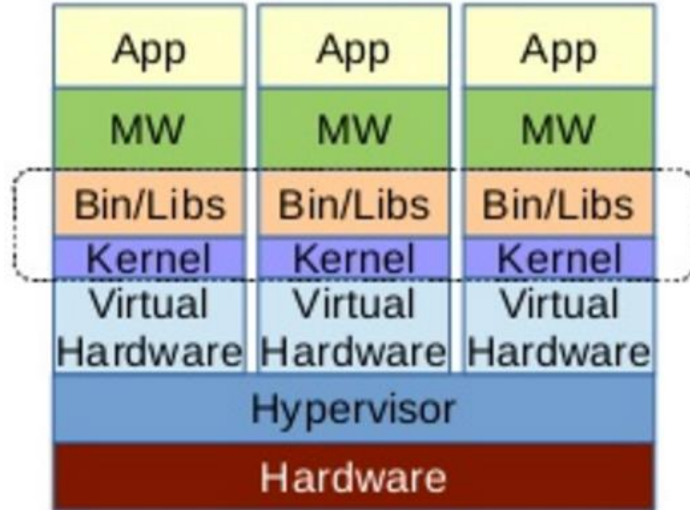
Solution



Problème réglé !

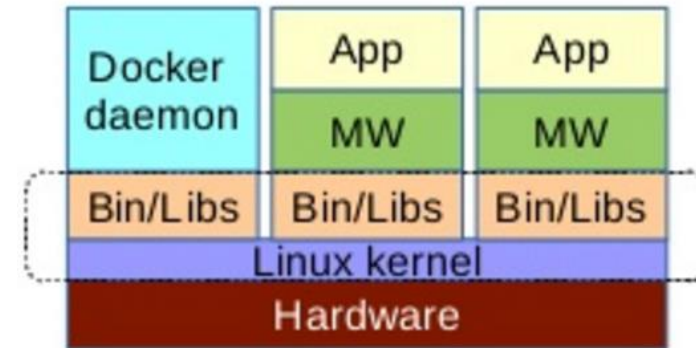


Conteneurs vs Machines Virtuelles



Machines virtuelles

- Abstraction du système physique (infrastructure)
- Incluent les applications, leurs dépendances et la totalité de l'OS (plusieurs dizaines de GB)
- Le hardware doit être optimisé : AMD-V, VT-d, VT-x...
- Sinon, émulation... -50% perf CPU



Conteneurs

- Incluent l'application et ses dépendances
- Partagent l'OS avec les autres conteneurs (kernel, devices, ...)
- S'exécutent comme des process isolés dans le « user space »

Conteneurs vs Machines Virtuelles

	Conteneur	Hypervisor
Redondance	Only Application Code and Data Edited Files	Computation and Memory redundancy
Sécurité	Less Secure, DOS attacks can affect others Containers Container with root privileges could affect other Containers	Full Isolation, Hypervisor is responsible of limiting used resource by VM Cross Guest Access is prohibited
Temps d'instanciation	Almost instantaneous	Even Preexisting VMs need OS Load Time
Performance	Almost Native	Less than native due to middle ware
Consolidation	Limited by actual Application Usage	Limited By OS reserved
Mémoire	Allocation Memory	Allocated Files Disk Allocated Files

Conteneurs / Définition

Les conteneurs fournissent un environnement :

- isolé sur un système hôte,
- semblable à un chroot sous Linux ou une jail sous BSD,
- mais en proposant plus de fonctionnalités en matière d'isolation et de configuration.

Ces fonctionnalités sont dépendantes du système hôte et notamment du kernel



Conteneurisation et DevOps

Dave le développeur

- Mon code
- Mes librairies
- Mon gestionnaire de paquets
- Mes applications

Oscar l'Admin Sys

- Logging
- Accès distant
- Config réseau
- Monitoring



What is Docker ?

Packager votre application dans une unité standardisée pour le déploiement

Un conteneur docker enrobe un morceau de logiciel dans un filesystem complet, qui contient :

- Le code, le runtime, les outils systems, les librairies

Cela garantit que le logiciel s'exécutera toujours de la même façon, indépendamment de son environnement



- **Lightweight** : les conteneurs partagent le même OS, démarrent instantanément, système de fichier en couches (mutualisation)
- **Open** : repose sur des mécanismes standards Linux et Windows – compatible avec toutes les infrastructures
- **Secure by default** : les conteneurs sont isolés entre eux

En quelques mots

- Open Source
- Né du mouvement DevOps
- Répond à plusieurs problèmes du cloud computing :
 - Packaging
 - Déploiement d'applications

Problème classique de gestion des dépendances – exemple :

- Une application a besoin d'un conteneur de servlet
- Tomcat a besoin d'une JVM
- Comment utiliser la bonne JVM ?
- Comment gérer les mises à jour...

Habituellement utilisé pour :

- automatiser le packaging et le déploiement d'applications
- la création de PaaS environnement léger et / ou privé
- l'intégration continue, déploiement continu, test automatisé : DevOps
- la gestion de déploiement d'applications "scalable"

Docker pour les développeurs

- Facilite la configuration et la maintenance des environnements de développement
- **Productivité** : se concentrer sur le développement de nouvelles fonctionnalités et livrer le software
- **Liberté technique** : les conteneurs sont isolés, les applications sont libres de leurs librairies et frameworks
- **Consistance des environnements de développement** : les applications sont packagées avec leurs dépendances – le conteneur s'exécute de manière ISO sur tous les environnements

Accélérer le cycle de delivery

- Livrer 7x plus souvent
- **Scaler rapidement** : le démarrage de conteneurs est de l'ordre de quelques secondes – un cluster sait s'adapter à la demande de manière fluide
- **Gérer rapidement les problèmes** : docker permet d'isoler un conteneur qui pose problème, procéder à un rollback et déployer le correctif (possible grâce à l'isolation entre les conteneurs)

Histoire du projet

Auteur : Solomon Hykes

Dotcloud.com / Side Project

1^{ère} version Open Source en mars 2013

- 2014 : levée de fond de 40M\$
- 2015 : levée de fond de 95M\$

Technologies :

- Langage : Go
- LXC (containers Linux)
- cgroups
- Kernel Linux



Domaine d'application :

- Gérer un Cloud de type PaaS
- Gestion de la scalabilité (ex : Cassandra, MongoDB, Riak, ...)
- Faciliter les opérations de déploiement et de maintenance
- Fluidifier la chaîne de développement / intégration / production

Docker et son écosystème



Quelques utilisateurs

illumina®



GROUPON

grubHub®
happy eating

PayPal™

Nerdalize

ORBITZ

BUSINESS
INSIDER



lyft

ebay™

Yandex

Baidu 百度

shopify

Spotify®

yelp.®

Installer Docker

Sur Windows

<https://download.docker.com/win/stable/InstallDocker.msi>

Sur Linux

<https://docs.docker.com/engine/installation/linux/>

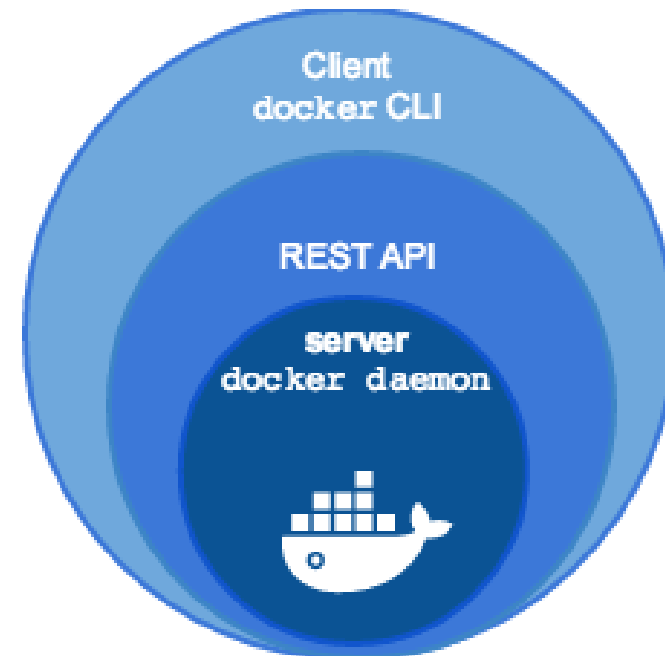
Commandes de base

Docker Engine

- > docker
- > docker **info**
- > docker **version**
- > docker **ps**
- > docker **ps -a**

<https://docs.docker.com/engine/reference/commandline/>

- > docker --help



Démarrer mon premier conteneur

Rechercher

> docker search hello-world

Exécuter

> docker run hello-world

> docker run ubuntu date

Contrôler

> docker images

> docker ps

Prendre la main

> docker run -it ubuntu bash

Exécuter un web server :

> docker run -d -p 8001:80 --name mywebserver nginx

<http://localhost:8001/>

> docker ps

> docker stop mywebserver

> docker start mywebserver

> docker ps mywebserver

> docker rm -f mywebserver

> docker ps -a

Démarrage en arrière plan

```
> docker run --name ubash -it  
ubuntu /bin/bash
```

```
[CTRL]+P+Q
```

```
> docker ps
```

```
> docker attach ubash
```

```
> docker run --name ap -dit  
ubuntu /bin/bash
```

```
> docker ps
```

```
> docker attach ap
```

```
[CTRL]+P+Q
```

Docker Engine

Docker Engine provides the core Docker technology that enables images and containers

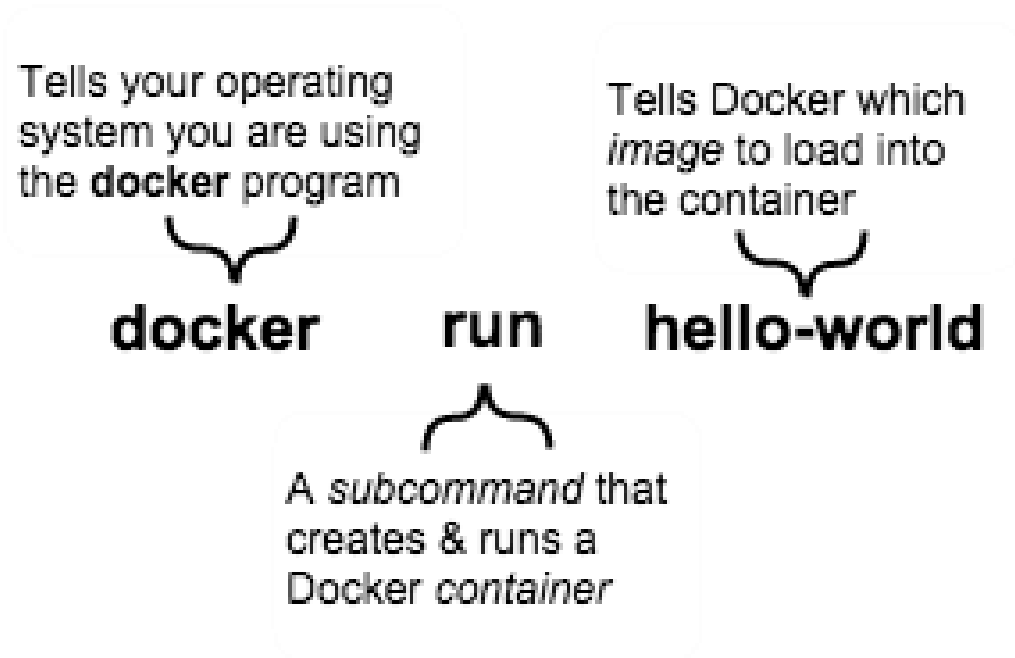
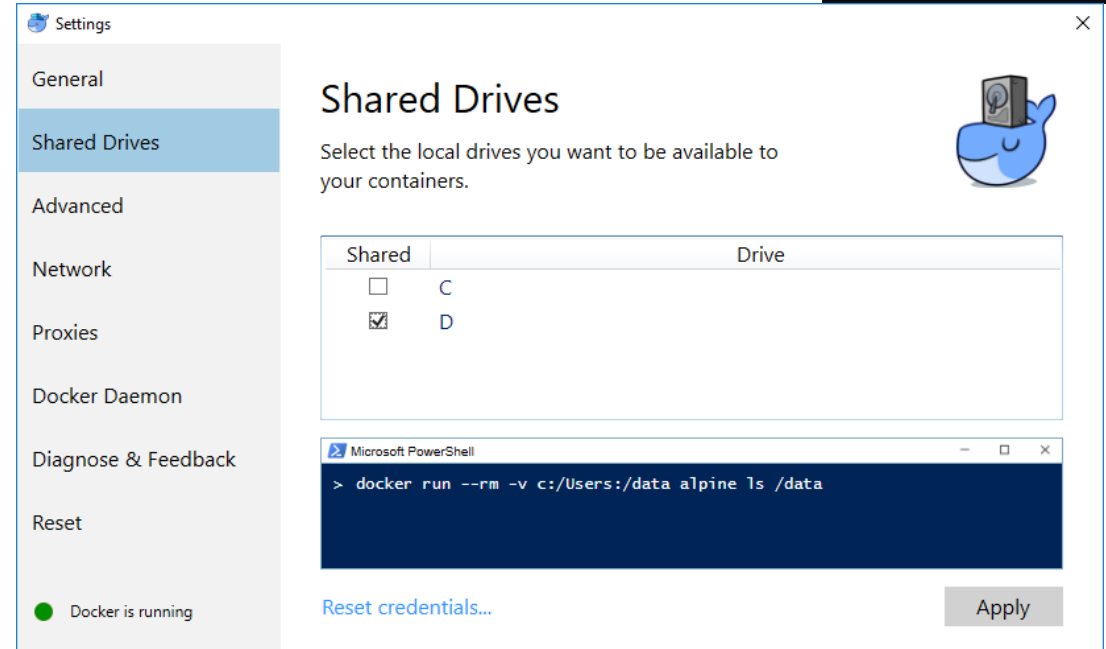
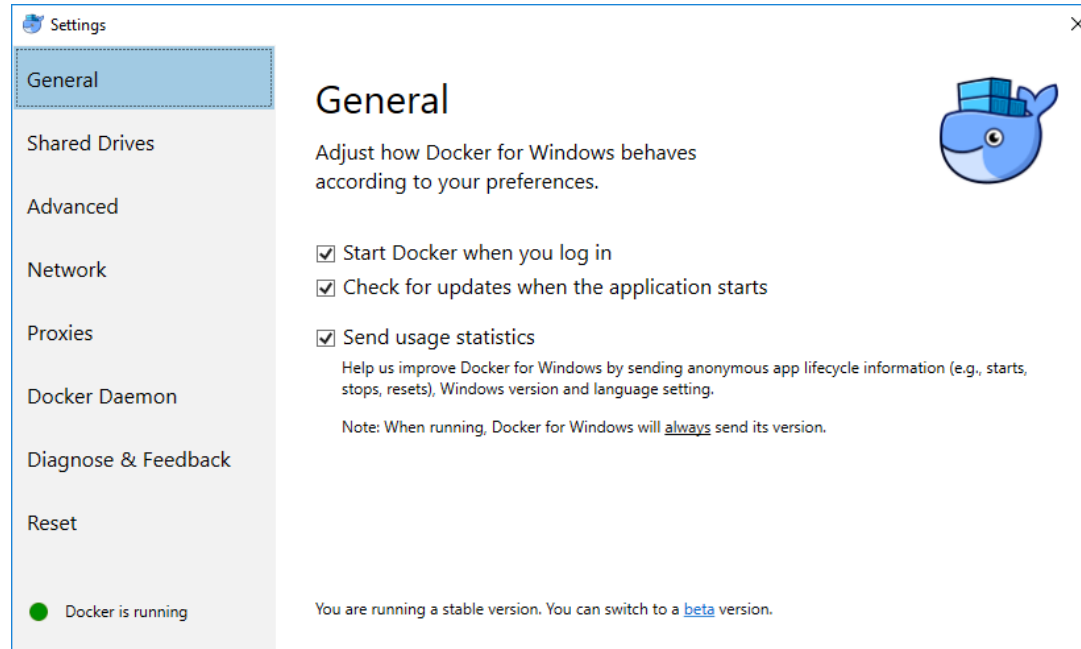
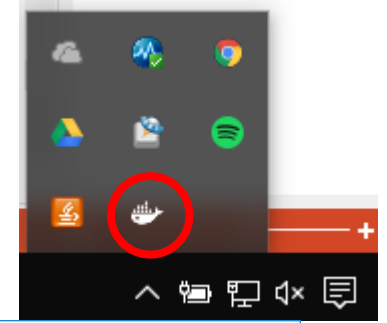


Image : a filesystem and parameters to use at runtime

Container : a running instance of an image

1. checked to see if you had the hello-world software image
2. downloaded the image from the Docker Hub (more about the hub later)
3. loaded the image into the container and “ran” it

Configuration de Docker sous Windows

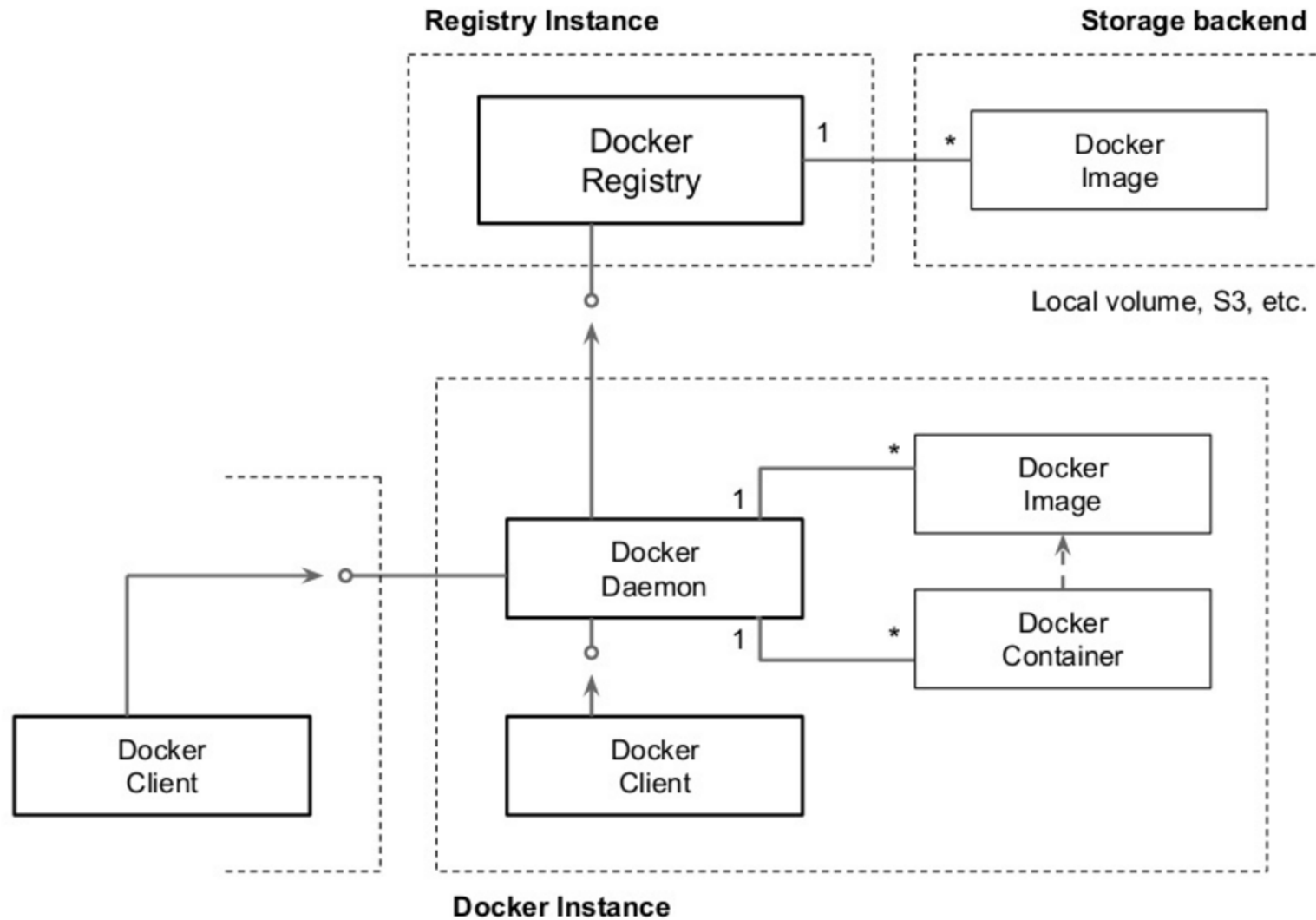


Volume binding :

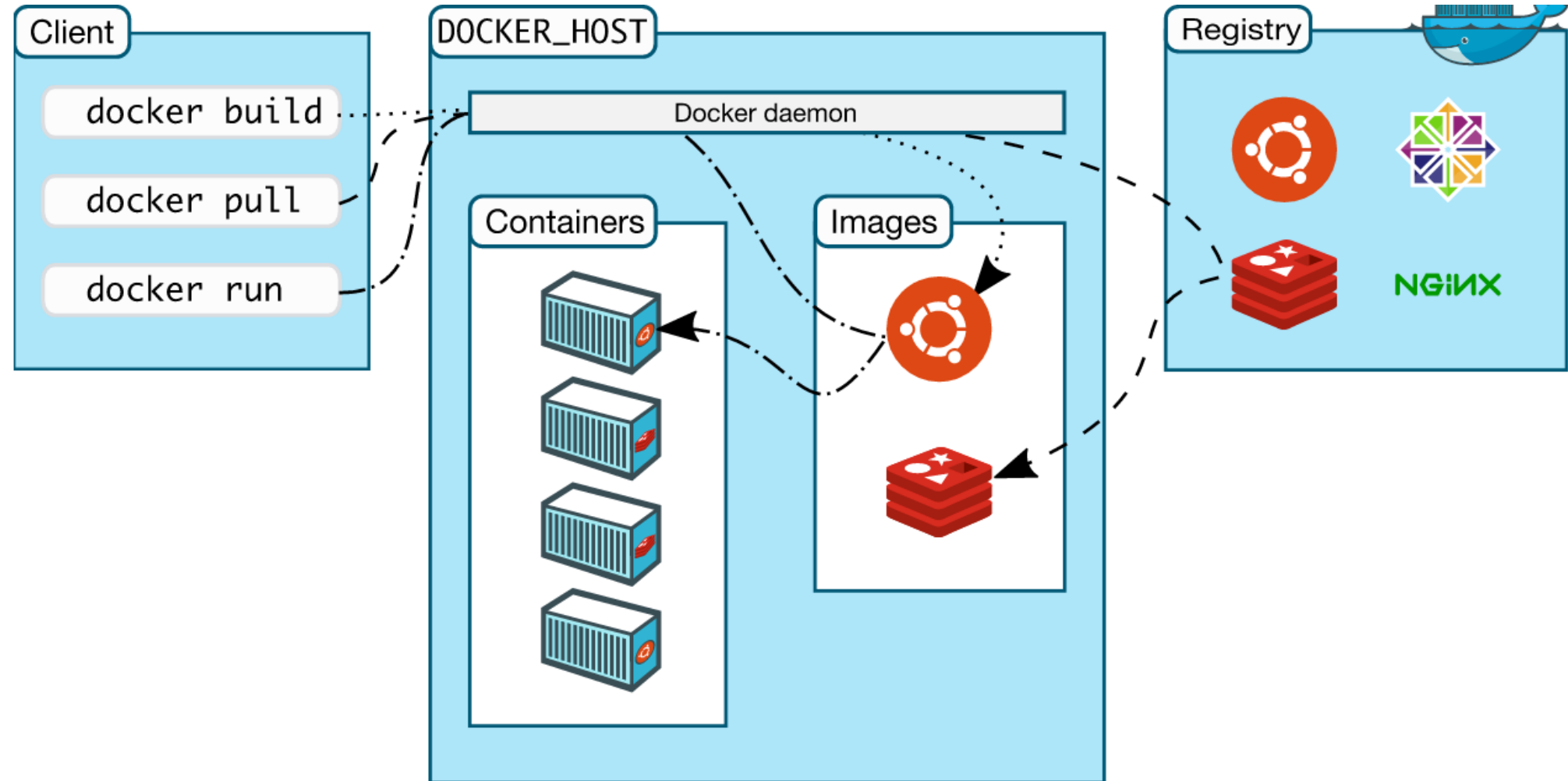
```
> docker run -itv d:/hadrien:/data ubuntu  
bash
```

```
> ls /data
```

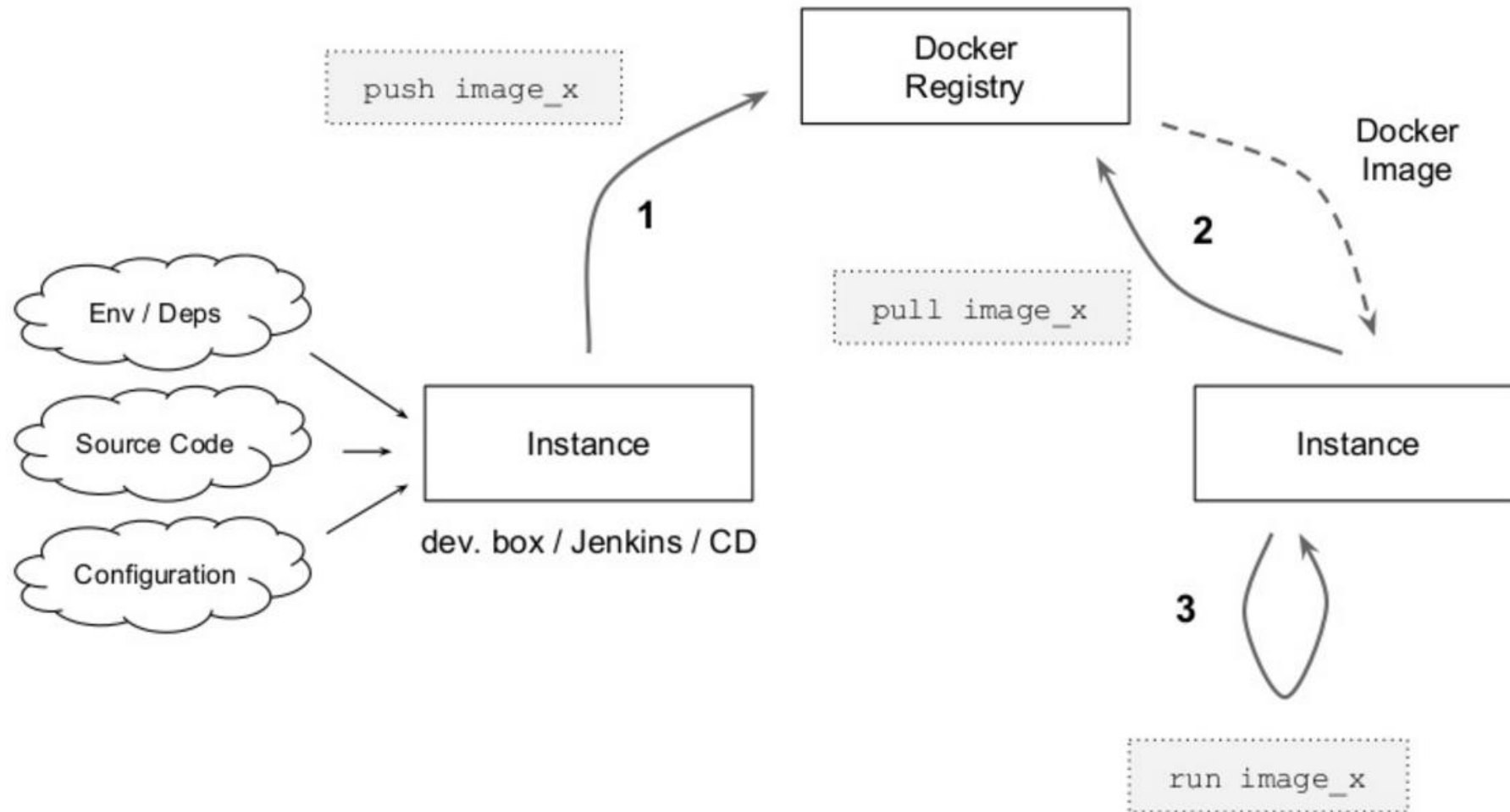

Terminologie



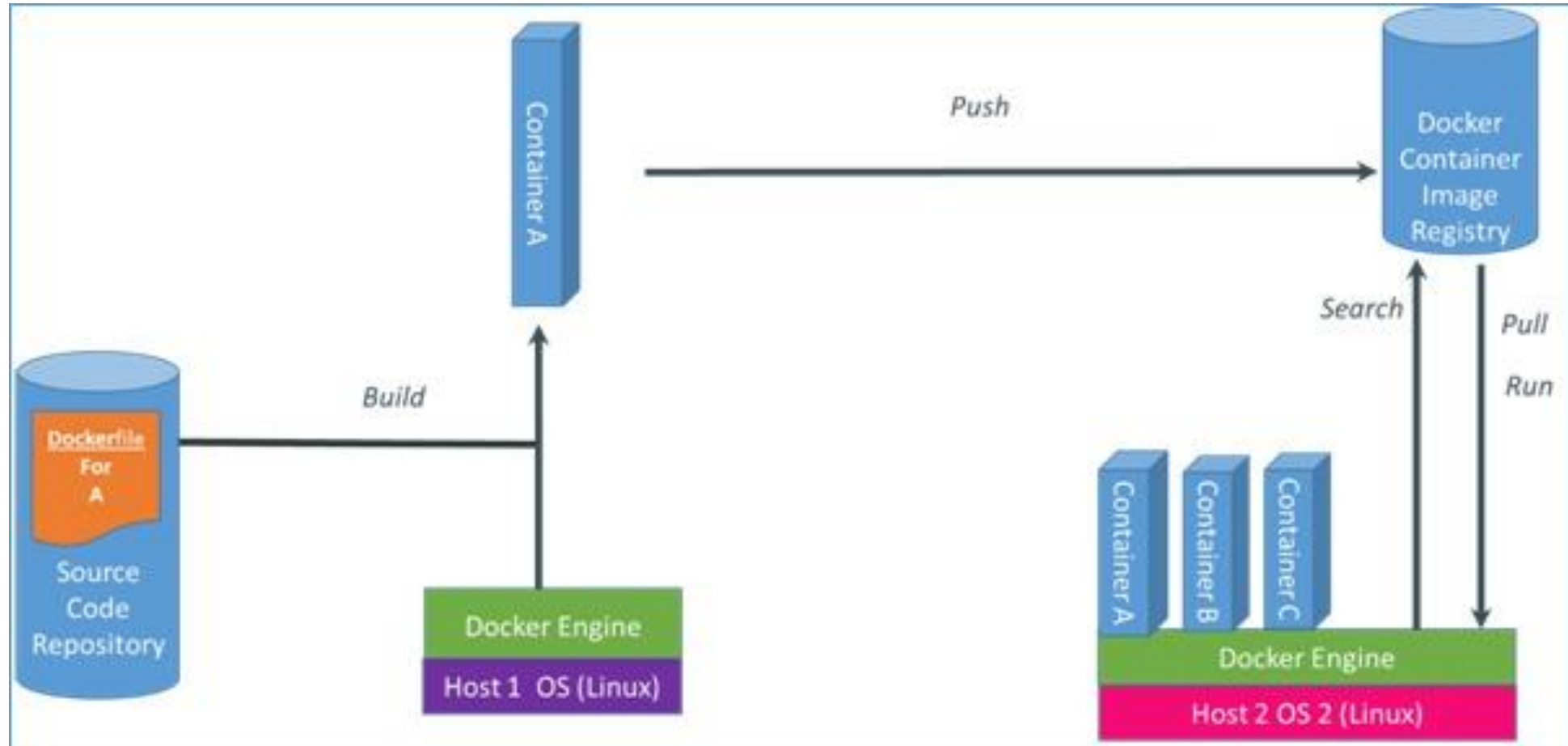
Principe de fonctionnement



Flux de travail – build / ship / run



Cycle de vie des conteneurs



Principes de la conteneurisation

Isolation

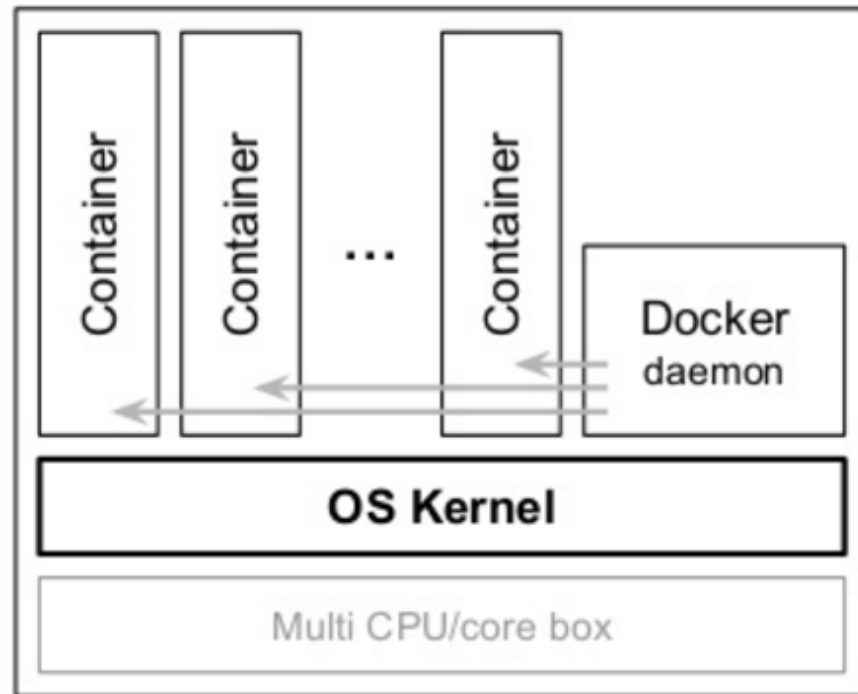
- Identifie un groupe de processus
- Limiter leurs droits
- Isole les processus entre eux

Contrôle des ressources

- Limiter l'utilisation des ressources
- Ressource = Process id, mémoire, CPU, espace disque, réseau, devices

Conteneurs Linux / LXC

Système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation



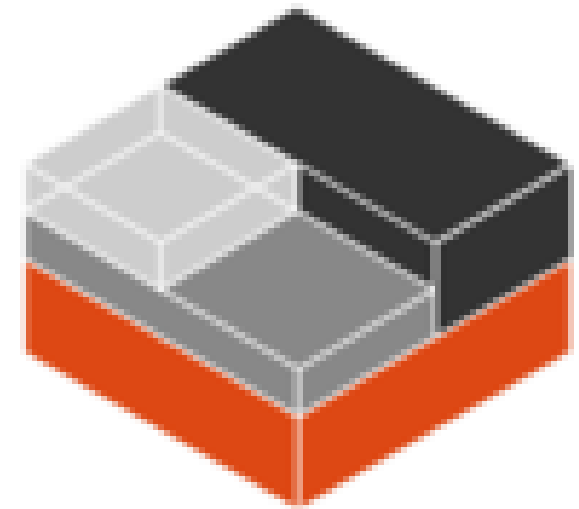
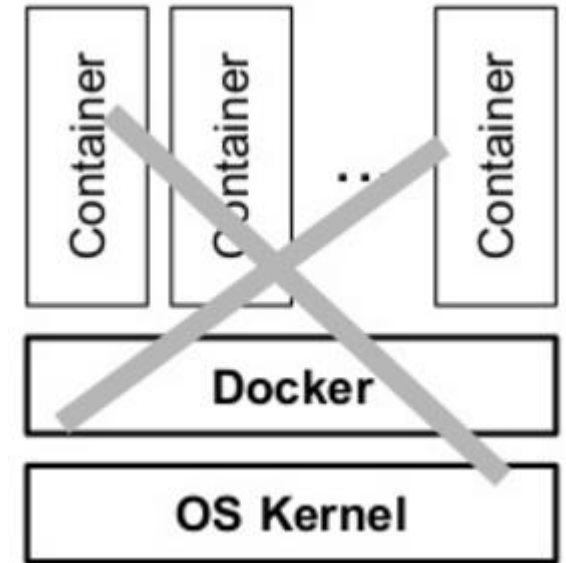
`cgroups` (2007)

Resource isolation

- CPU
- memory
- disk I/O

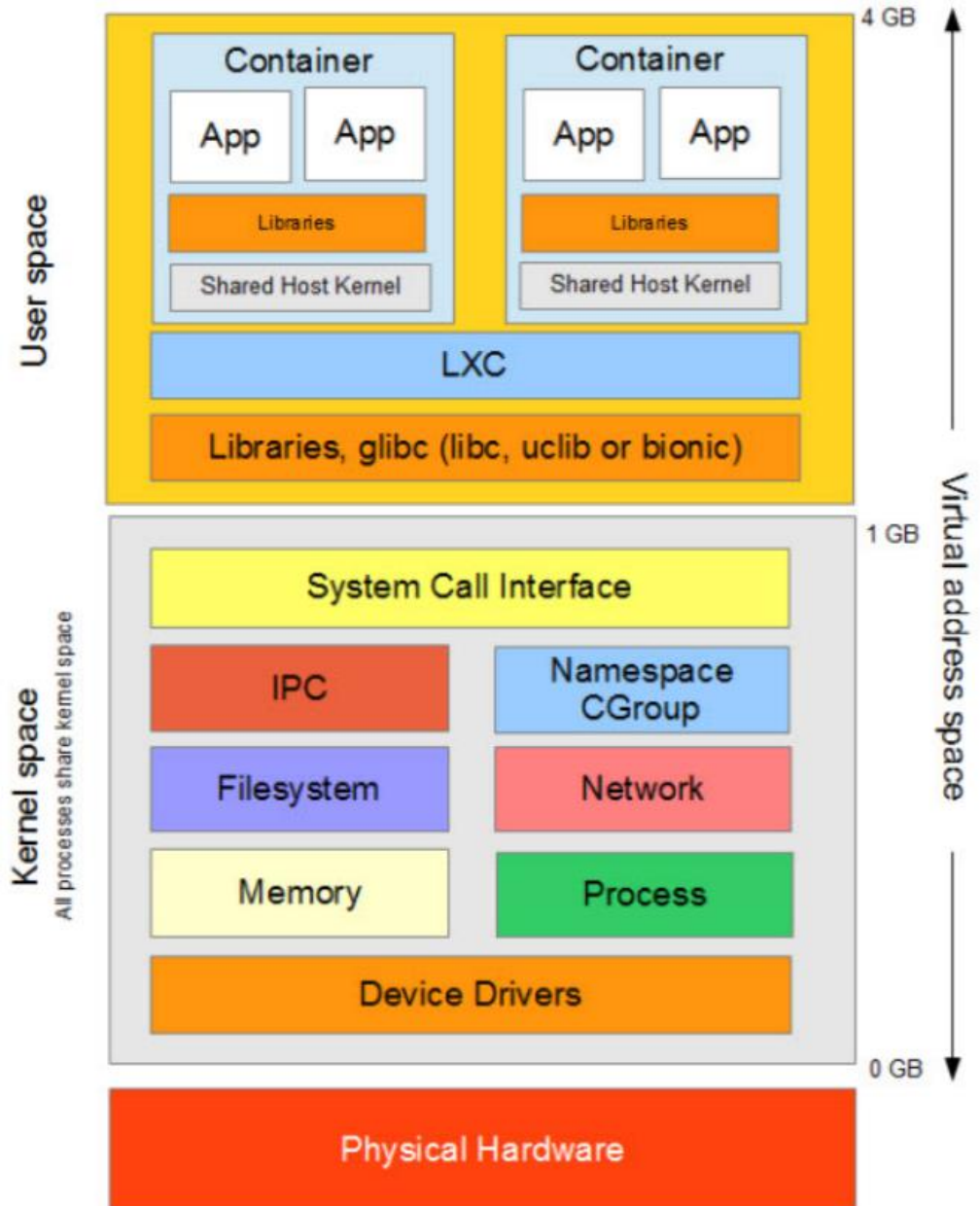
Namespace isolation

- process tree
- network
- user ids
- mounted file systems



LXC Architecture

Les conteneurs sont des groupes de processus isolés grâce à des mécanismes du Noyau Linux



Docker : Sous le capot

Docker s'appuie sur des mécanismes natifs au noyau Linux

Isolation et limitation des droits : Kernel Namespaces

- pid : limiter la visibilité qu'un processus a des autres processus du système
- network : offre une vue indépendante du réseau du système
- ipc : permet d'isoler les communications inter-processus
- Mount : offre une visibilité de ses seuls montages à un conteneur
- uts : permet d'avoir sa propre identité réseau

Contrôler les ressources : Control Groups

cgroup, intégrés en 2007 au kernel

- Regroupement de processus
- Limiter l'accès aux ressources (exemple : limiter la disponibilité mémoire)
- Prioriser certains groupes de contrôle : %CPU, %disk I/O
- Mesurer la consommation
- Contrôle : freeze, stop, restart

Format de conteneur

- Actuellement : libcontainer
- Puis... BSD Jails, Solaris Zones

Notion de « conteneur léger »

Virtualisation par conteneur

- Démarre rapidement : le kernel est déjà démarré
- Le kernel et la mémoire sont partagés
- Exécution native des processus : pas d'overhead (pas d'émulation)

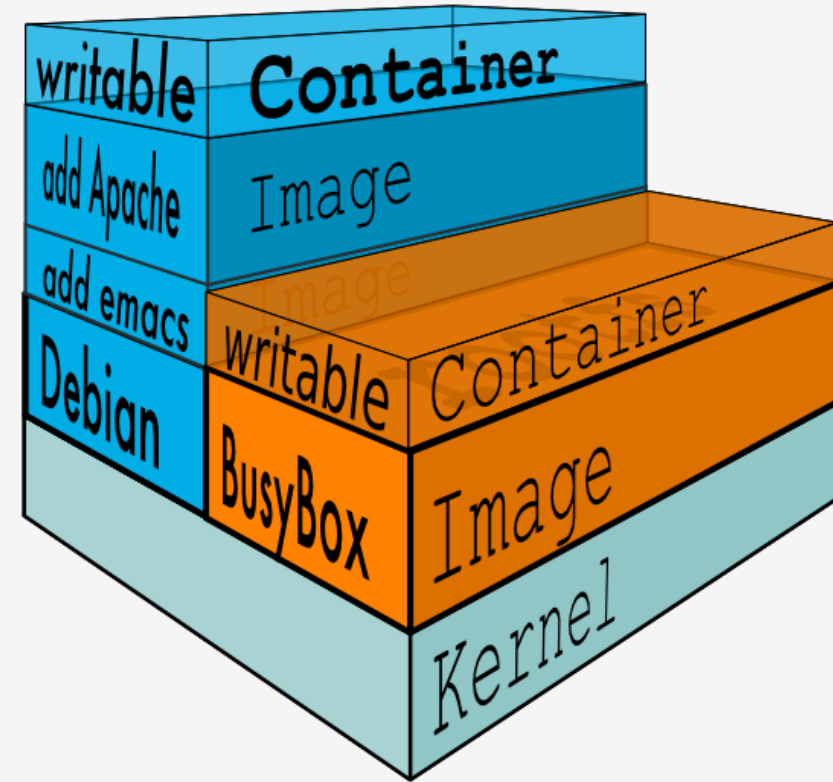
Mécanismes de composition du file system

- Union FS : Plusieurs filesystems sont montés – l'application ne voit qu'un seul filesystem
- Mécanisme de copy on write
- Nombreux autres drivers...

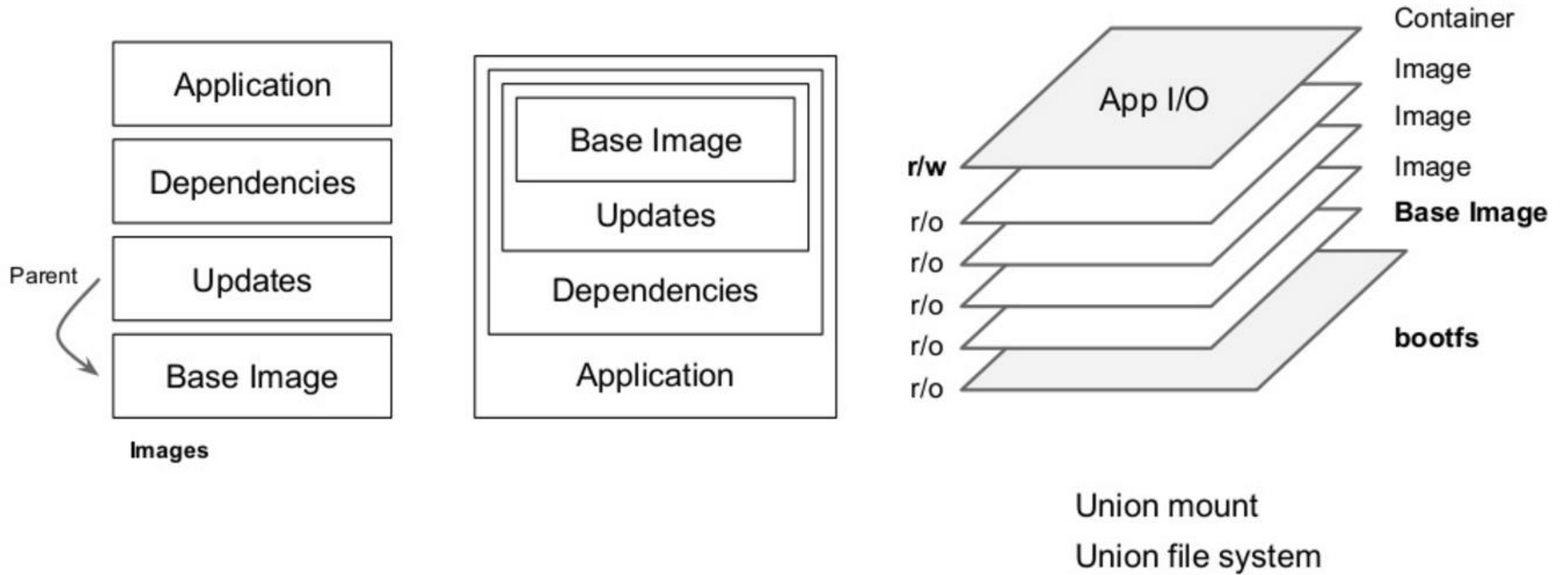
Sous le Capot

Union file system (UnionFs)

- Construction d'un filesystem par couches
- Au dessus d'un filesystem bootable (bootfs)
- Images en lecture seule / mécanisme de Copy on Write
- Variantes d'UFS : AUFS, btrfs, vfs, DeviceMapper, ...



Constitution d'une Image



Constitution d'une image

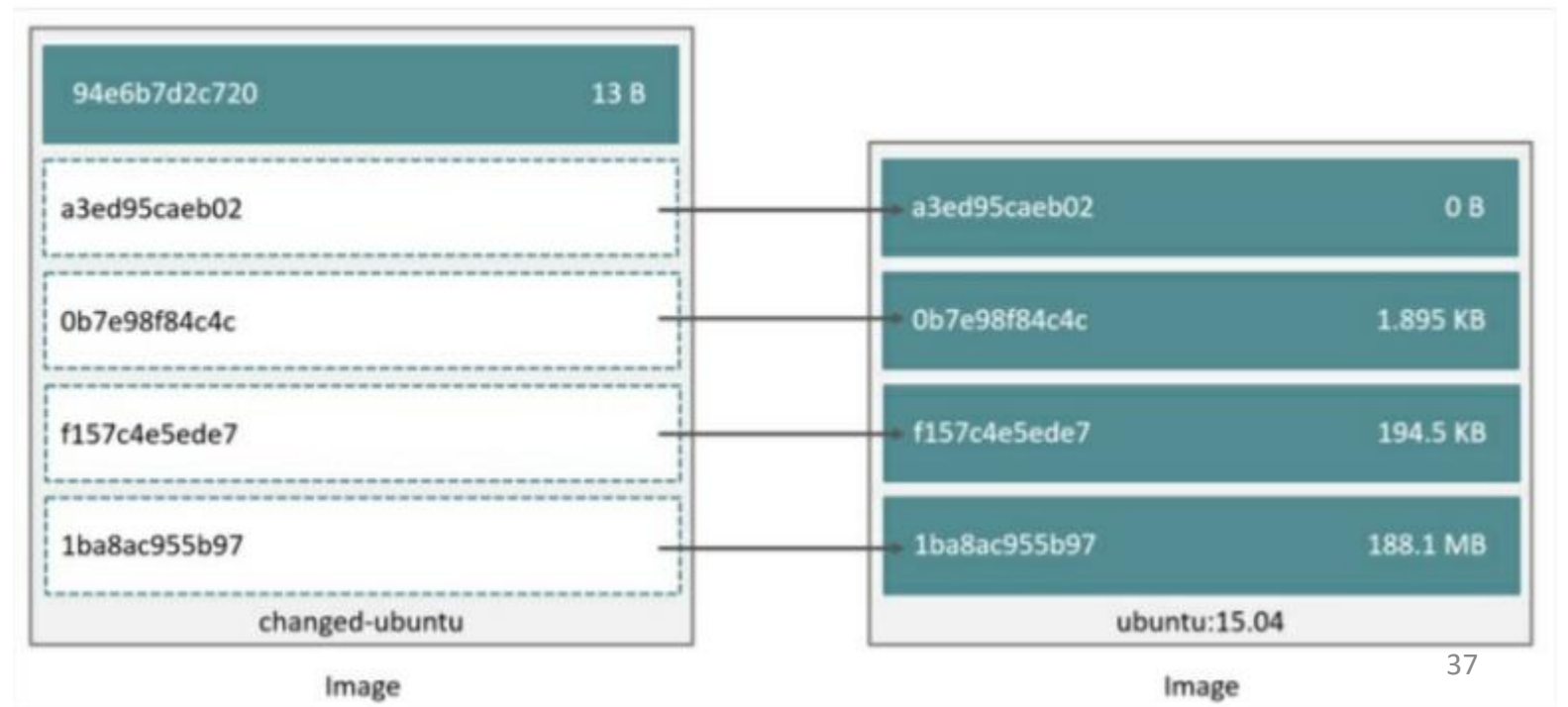
- Collection de fichiers et de méta-données
- Ces fichiers forment le système de fichiers racines d'un conteneur
- Composée de couches, empilées les unes sur les autres
- Chaque couche peut correspondre à un ajout, une modification, ou la suppression de fichiers
- Des images peuvent partager des couches pour optimiser l'espace disque et leur transfert
- Une image est un système de fichiers en lecture seule.

Comment on modifie une image ?

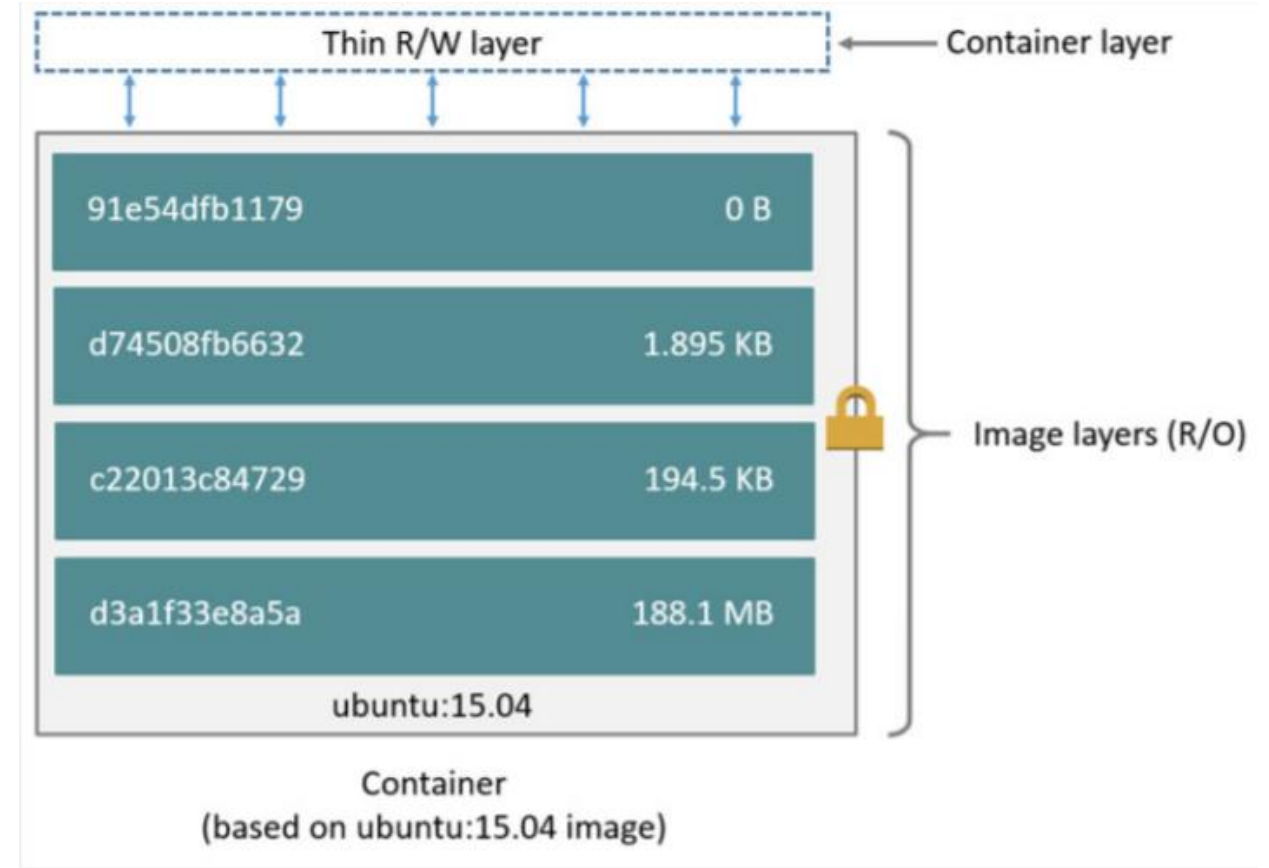
- On ne la modifie pas
- On crée un conteneur depuis une image
- Un conteneur exécute un ou plusieurs processus dans une copie en écriture du système de fichiers de l'image
- On fait des modifications dans un conteneur
- On peut transformer ces modifications en une nouvelle couche
- Une nouvelle image est créée en empilant cette nouvelle couche sur l'ancienne image

Concepts / Layers

- Les images / conteneurs sont composés en couches (Layers)
- Réutilisation des layers entre les conteneurs
- Optimisation de l'espace disque



Concepts / Layers



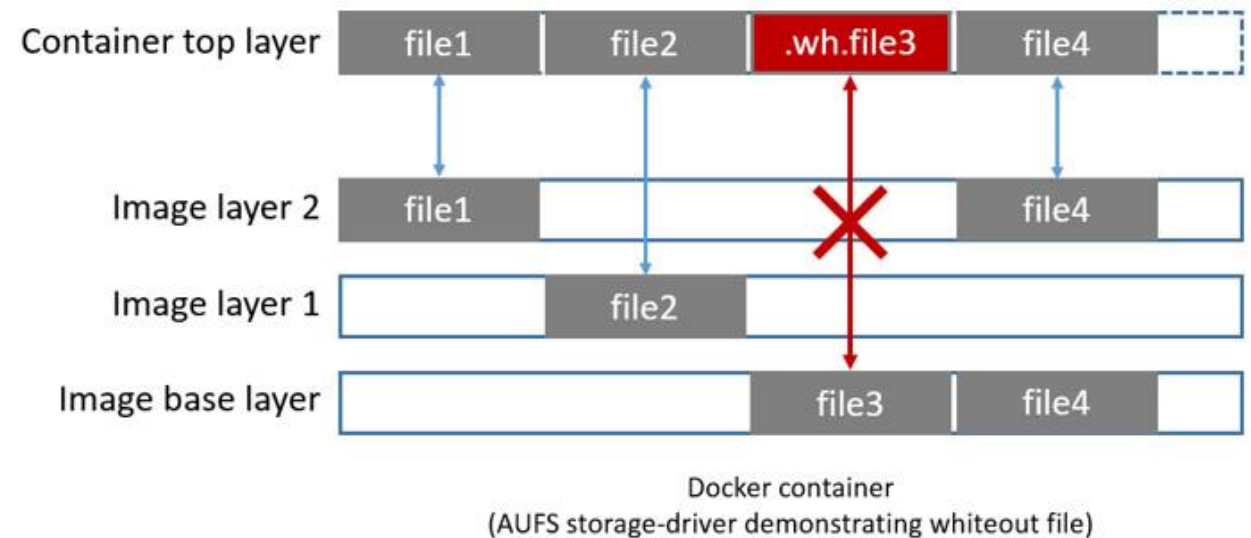
Concepts / Stockage (Union File Systems)

La structure en couche concerne : Exemple avec AUFS

les images, les données des conteneurs, les volumes

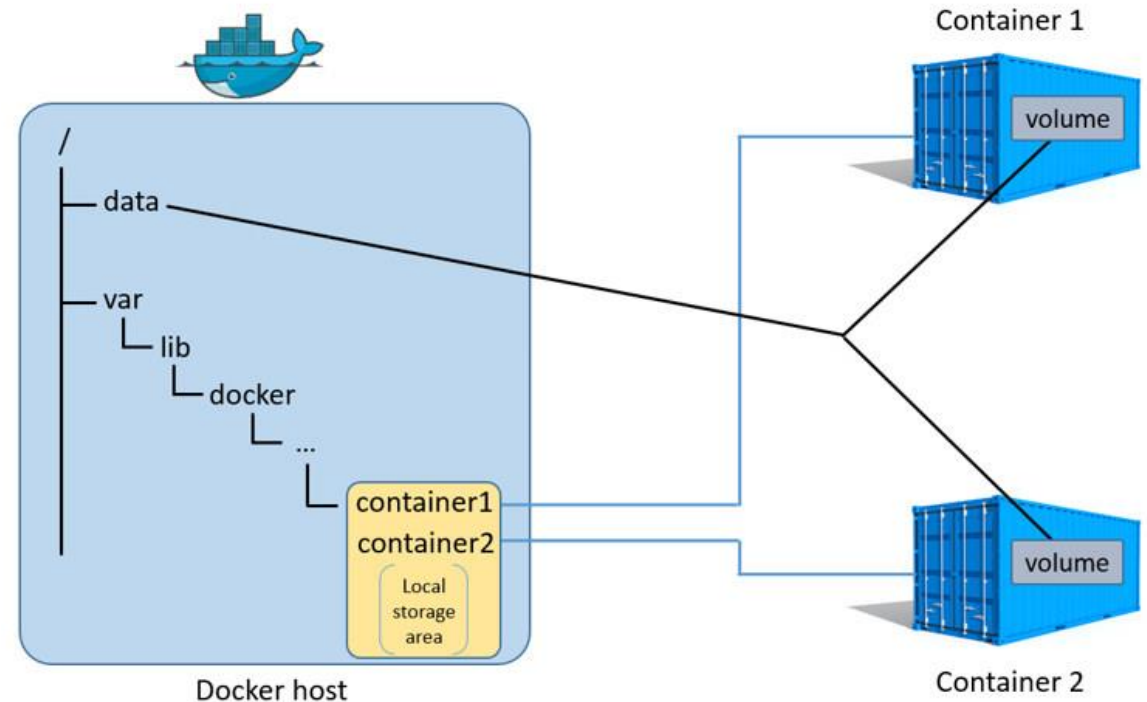
Plusieurs drivers :

- **AUFS**
- DeviceMapper
- OverlayFS
- VFS
- ZFS



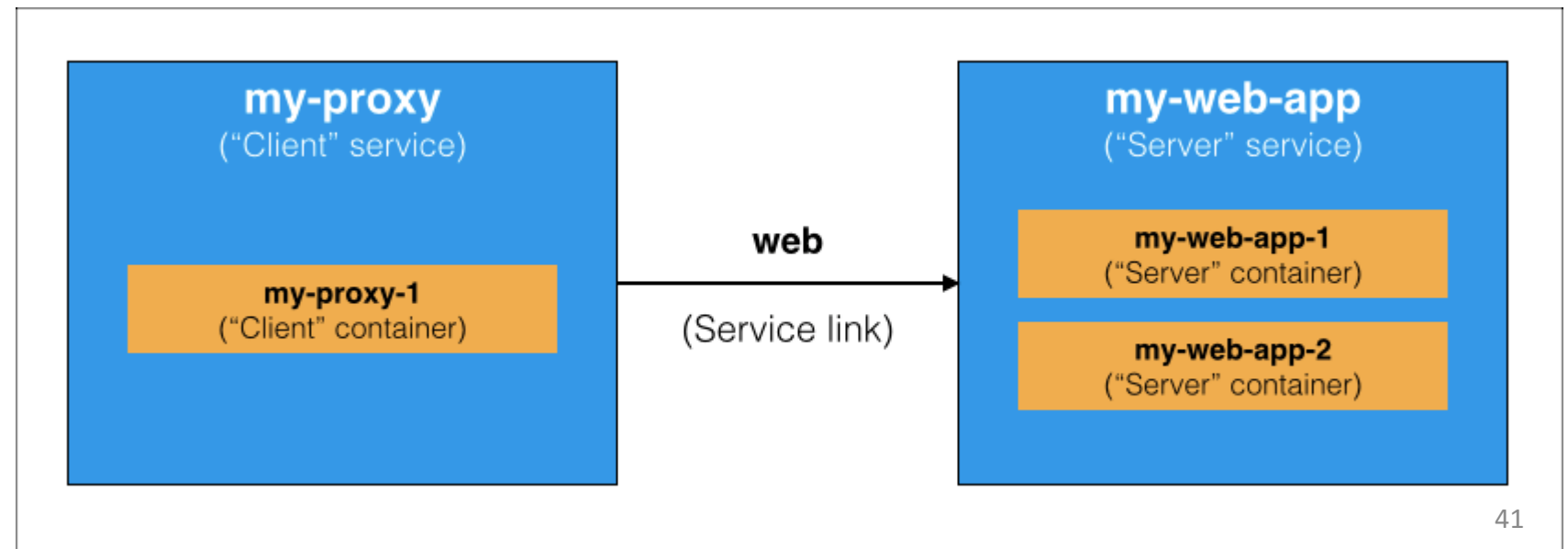
Concepts / Volumes

- Assurer la persistance des données
- Indépendance du conteneur et des layers
- 2 types de volumes :
 - Conteneur : données stockées dans un data container
 - **Hôte : montage d'un dossier de l'hôte docker dans le conteneur**
- Partage d'un volume entre plusieurs conteneurs



Concepts / Links

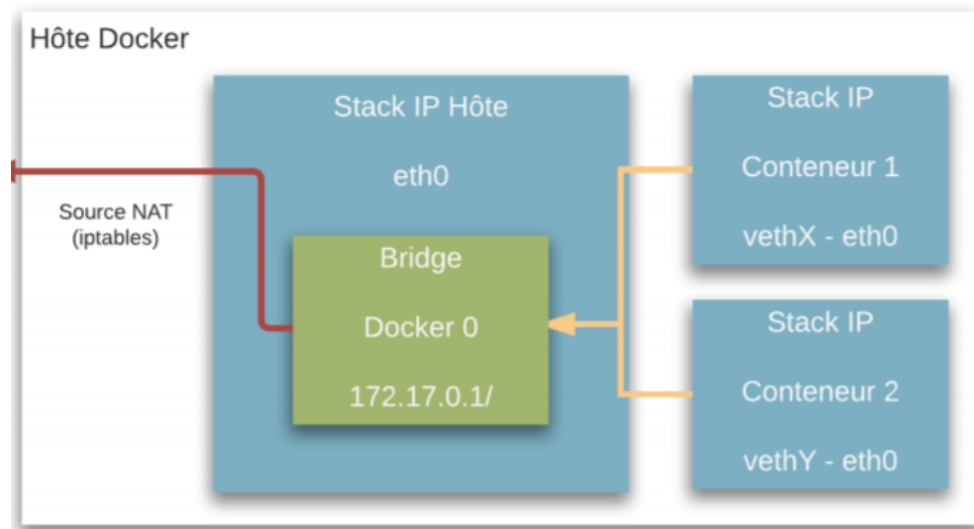
- Les conteneurs d'un même réseau peuvent communiquer via IP
- Les liens permettent de lier deux conteneurs par nom
- Système de DNS rudimentaire (/etc/hosts)
- Complété par les *discovery services*



Concepts / Réseau

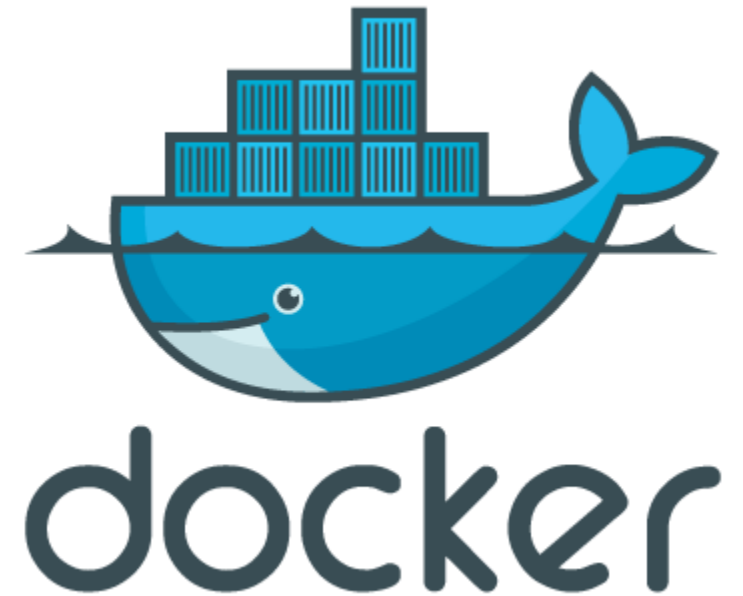
> docker network ls

```
NETWORK ID NAME DRIVER
7fca4eb8c647 bridge bridge
9f904ee27bf5 none null
cf03ee007fb4 host host
```



> ipconfig /all

```
Carte Ethernet vEthernet (DockerNAT) :
    Suffixe DNS propre à la connexion. . . . :
    Description. . . . . :
Hyper-V Virtual Ethernet Adapter
    Adresse physique . . . . . :
00-15-5D-00-00-00
    DHCP activ  . . . . . :
Non
    Configuration automatique activ  e. . . . :
Oui
    Adresse IPv6 de liaison locale. . . . . :
fe80::3952:fb23:9a56:6ca4%2 (pr  f  r  )
    Adresse IPv4. . . . . :
10.0.75.1 (pr  f  r  )
    Masque de sous-r  seau. . . . . :
255.255.255.0
    Passerelle par d  faut. . . . . :
    Serveurs DNS. . . . . :
fec0:0:0:ffff::1%1
```

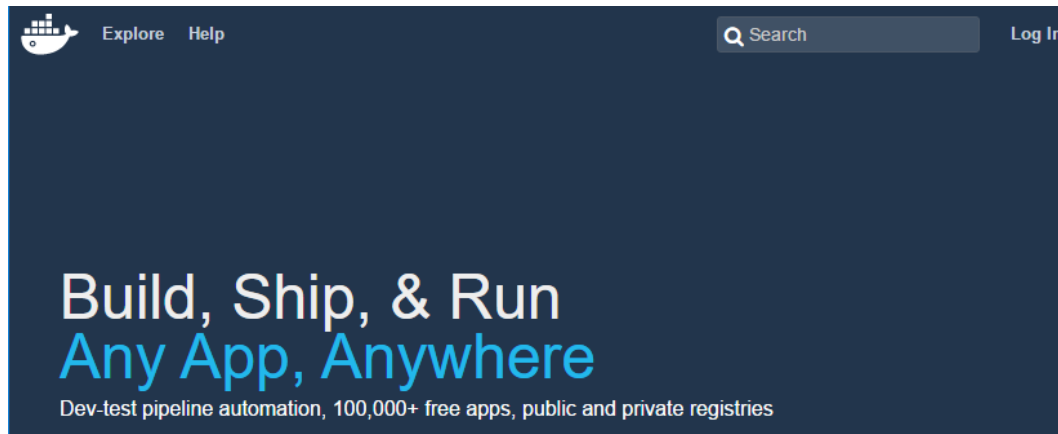


Pourquoi Docker ?

Pour utiliser LXC comme une commodité

Docker Hub

<https://hub.docker.com/>



Exemple :

Rechercher : « docker/whalesay »

<https://hub.docker.com/r/docker/whalesay/>

Exécution :

```
> docker run docker/whalesay  
cowsay boo
```

```
> docker images
```

Créer sa propre image docker

Plusieurs possibilités

- **docker commit** : crée une nouvelle couche (et une nouvelle image) depuis un conteneur
- **docker build** : script de suite de commandes de création automatisée d'image
- **docker import** : charge une archive de fichiers, comme couche de base.

Créer sa propre image Docker

Créer un fichier de configuration

```
> cd d:\docker-emn  
> mkdir mydockerbuild  
> cd mydockerbuild  
> notepad Dockerfile  
FROM docker/whalesay:latest  
RUN apt-get -y update && apt-get  
install -y fortunes  
CMD /usr/games/fortune -a |  
cowsay
```

Construire l'image

```
> docker build -t docker-whale .
```

Contrôler, puis exécuter

```
> docker images  
> docker run docker-whale
```

Uploader l'image sur le Docker Hub

Créer un compte

<https://hub.docker.com/>

Créer un repository

<https://hub.docker.com/add/repository/?namespace=hadrienboye>

Taguer l'image

```
> docker tag ... hadrienboye/docker-whale:latest
```

Pusher l'image

```
> docker login
```

```
> docker push hadrienboye/docker-whale
```

<https://hub.docker.com/r/hadrienboye/docker-whale/tags/>

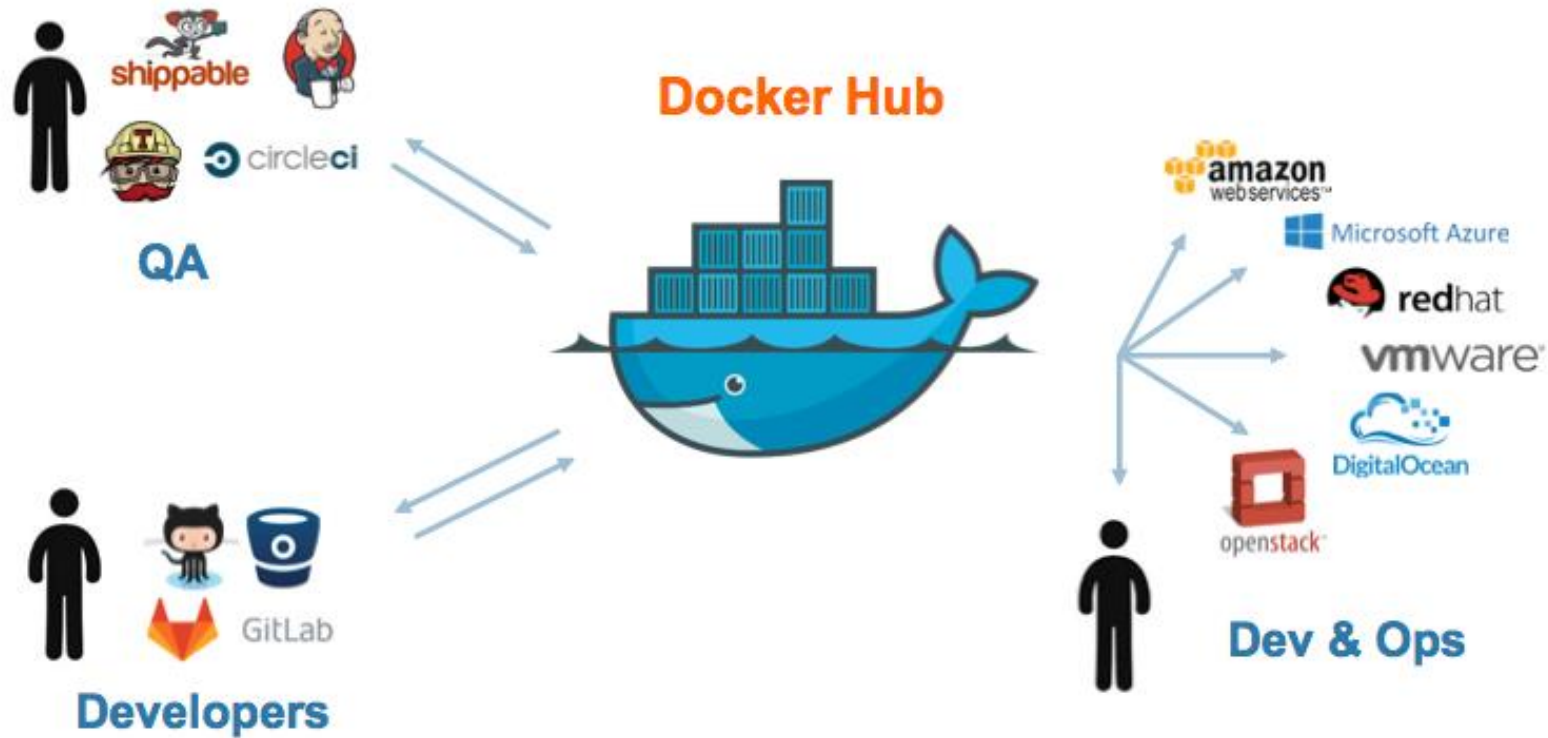
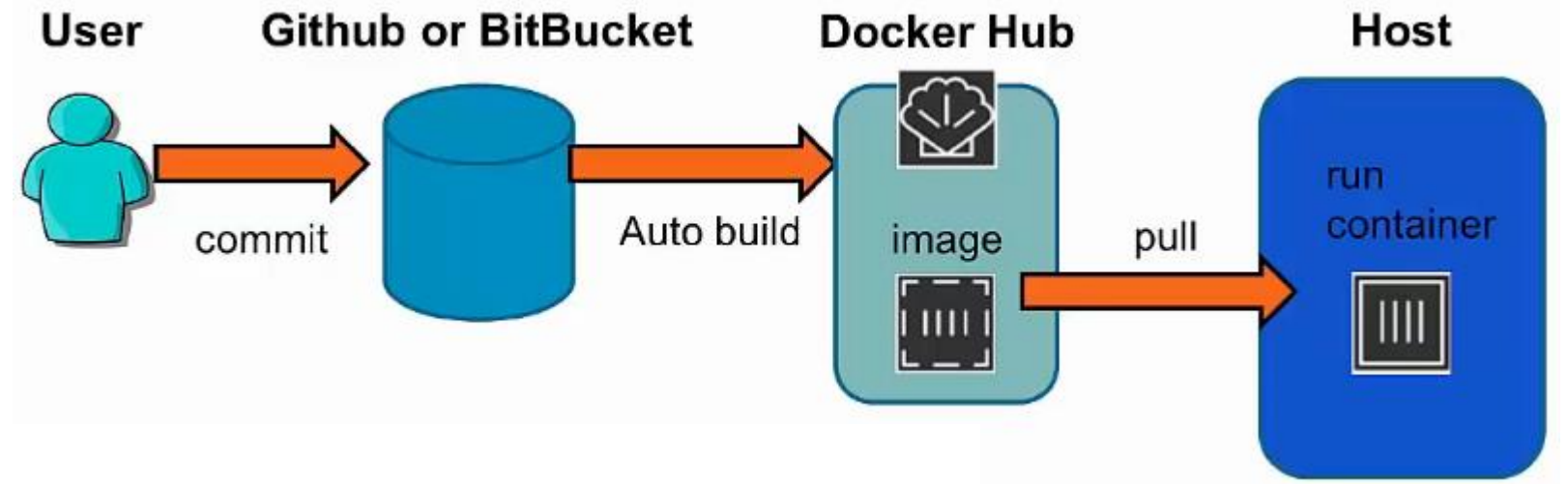
Pull...

```
> docker rmi -f c403668f87fd
```

```
> docker pull hadrienboye/docker-whale
```


```
> docker run hadrienboye/docker-whale
```

Docker Hub




Docker Hub

<https://hub.docker.com/>

 Dashboard Explore Organizations

Q Search

Create

 hadrienboye

hadrienboye

Repositories

Stars


Contributed


Private Repositories: Using 1 of 1 [Get more](#)

Repositories

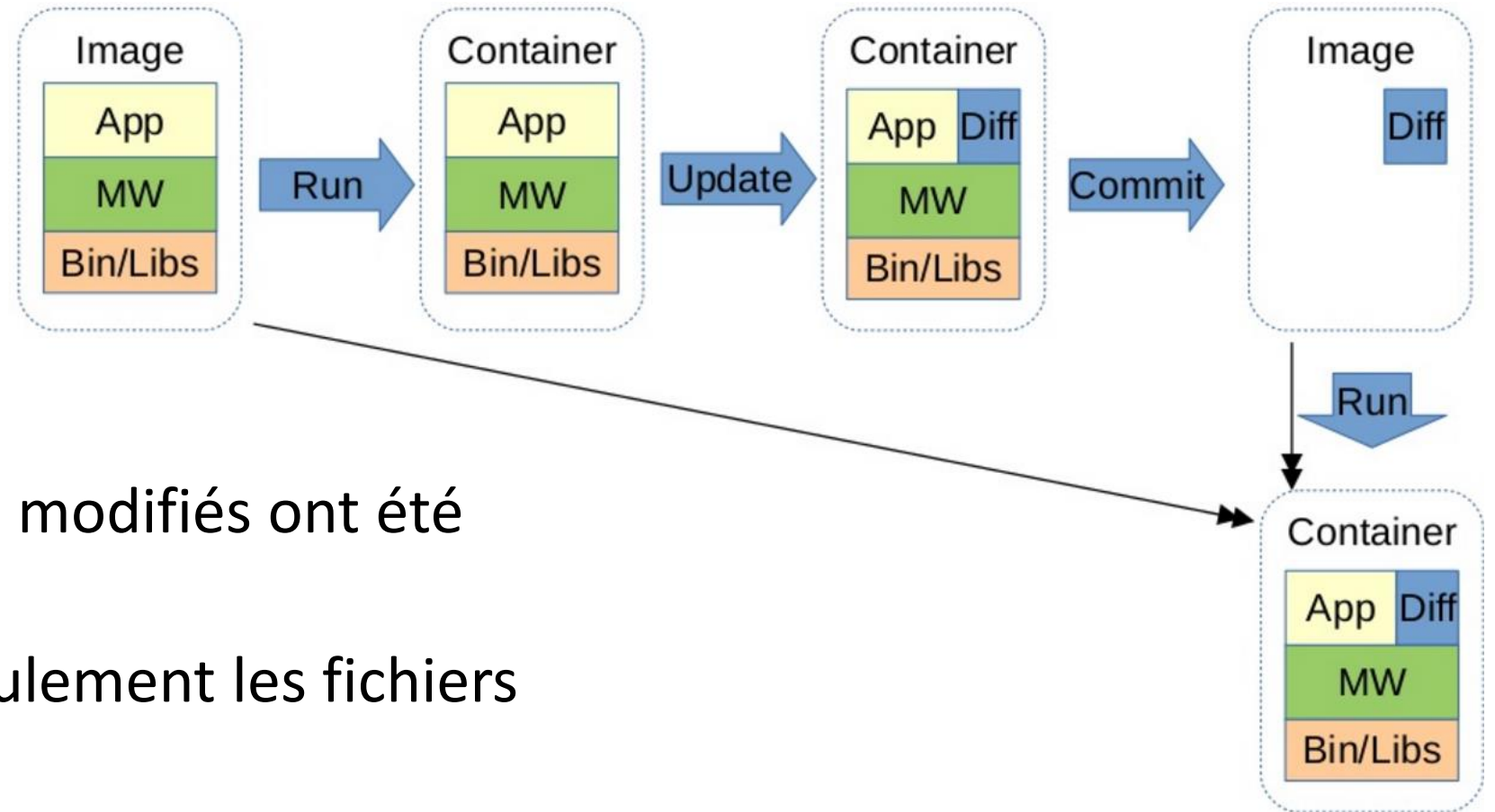
Create Repository +

Type to filter repositories by name

	hadrienboye/docker-whale private	0 STARS	2 PULLS	> DETAILS
---	-------------------------------------	------------	------------	--------------

 **Docker Security Scanning**
Protect your repositories from vulnerabilities.
[Try it free](#)

En synthèse



- Seuls les fichiers modifiés ont été sauvegardés
- On push/pull seulement les fichiers « incrémentés »

En synthèse

Build image from source

```
docker build -t IMAGE_NAME .
```

Tag image appropriately

```
docker tag IMAGE_ID REGISTRY_URL:PORT/IMAGE_NAME:TAG
```

Push image to the registry

```
docker push REGISTRY_URL:PORT/IMAGE_NAME:TAG
```

Pull image from the registry

```
docker pull REGISTRY_URL:PORT/IMAGE_NAME:TAG
```

Run container based on the image

```
docker run --restart=always -d -p HOST_PORT:CONTAINER_PORT IMAGE_NAME:TAG
```

Dev box
Jenkins
CD

Prod
Staging
Test

Docker est une application client - serveur

Démon Docker

- Répond aux requêtes sur l'API Docker
- Ecrit en Go
- Gestionnaire open source pour faciliter l'utilisation des conteneurs
- Définit un format standard de conteneur
- Permet de créer des images, selon un format standard et facilement reproductible
- Permet de partager les images via un Registre

Client Docker

- CLI
- Ecrit en Go
- Communique avec le démon Docker via l'API (REST)

Exécuter un conteneur dans un démon

```
> docker run -d ubuntu /bin/sh -c      hello world
"while true; do echo hello world;      hello world
sleep 1; done"                          hello world
> docker ps                            hello world
> docker logs                           hello world
peaceful_chandrasekhar                  hello world
> docker stop                           hello world
peaceful_chandrasekhar                  hello world
                                         hello world
                                         hello world
                                         hello world
                                         hello world
```

Exécuter une application web...

```
> docker run -d -P  
training/webapp python app.py
```

```
> docker ps
```

<http://localhost:32771>

Observer les logs

```
> docker logs -f furious_hoover
```

Lister les process

```
> docker top furious_hoover
```

Ouvrir un shell dans un conteneur

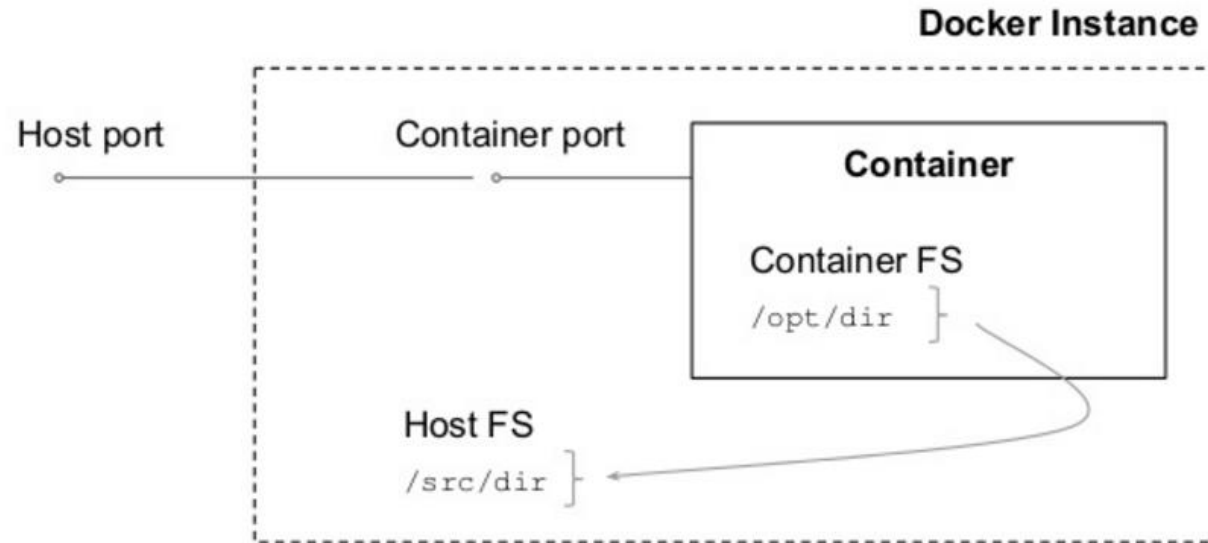
```
> docker exec -it furious_hoover  
bash
```

```
> exit
```

Supprimer le conteneur

```
> docker rm --f furious_hoover
```

Configuration d'un conteneur



Configure port mapping

```
docker run ... -p HOST_PORT:CONTAINER_PORT
```

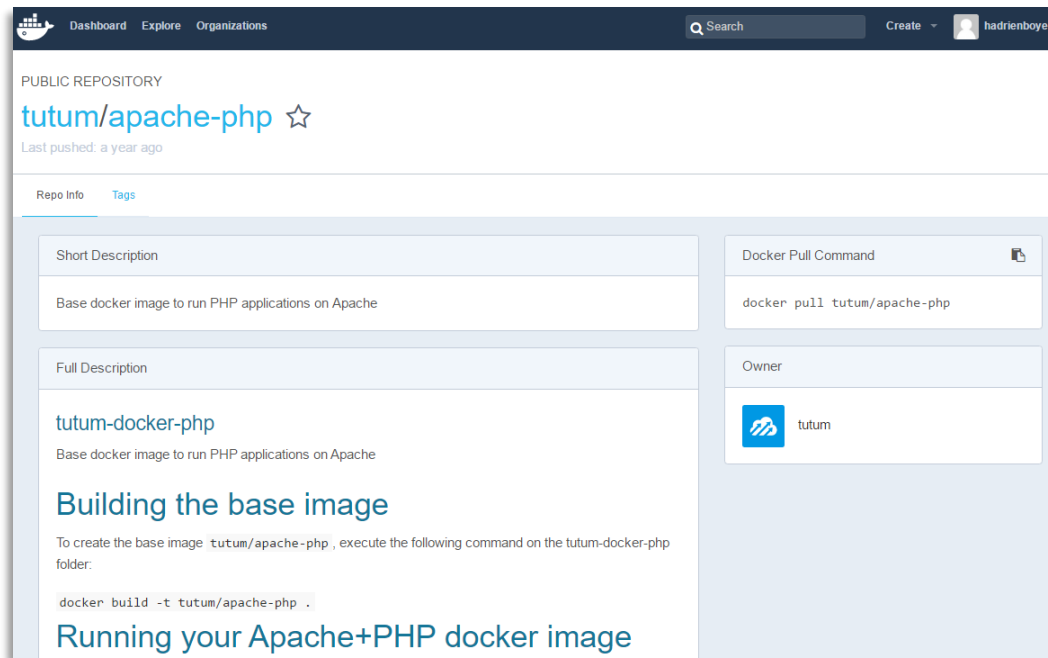
Mount a host directory as a data volume

```
docker run ... -v HOST_DIR:CONTAINER_DIR
```

Monter un environnement de dév Web

Image de départ :

<https://hub.docker.com/r/tutum/apache-php/>



<https://github.com/tutumcloud/apache-php>

“Base docker image to run PHP applications on Apache”

<https://github.com/tutumcloud/apache-php/blob/master/Dockerfile>

tutum-docker-php / Dockerfile

```
FROM ubuntu:trusty
MAINTAINER Fernando Mayo <fernando@tutum.co>
# Install base packages
RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get -yq
install \
    curl \
    apache2 \
    libapache2-mod-php5 \
    php5-mysql \
    php5-mcrypt \
    php5-gd \
    php5-curl \
    php-pear \
    php-apc && \
    rm -rf /var/lib/apt/lists/* && \
    curl -sS https://getcomposer.org/installer
| php -- --install-dir=/usr/local/bin --
filename=composer
```

```
RUN /usr/sbin/php5enmod mcrypt
RUN echo "ServerName localhost" >>
/etc/apache2/apache2.conf && \
    sed -i
"s/variables_order.*/variables_order =
\"EGPCS\"/g" /etc/php5/apache2/php.ini

ENV ALLOW_OVERRIDE **False**

# Add image configuration and scripts
ADD run.sh /run.sh
RUN chmod 755 /*.sh

# Configure /app folder with sample app
RUN mkdir -p /app && rm -fr /var/www/html &&
ln -s /app /var/www/html
ADD sample/ /app

EXPOSE 80
WORKDIR /app
CMD ["/run.sh"]
```

tutum-docker-php / run.sh

```
#!/bin/bash
```

```
chown www-data:www-data /app -R
```

```
if [ "$ALLOW_OVERRIDE" = "***False**" ]; then
```

```
    unset ALLOW_OVERRIDE
```

```
else
```

```
    sed -i "s/AllowOverride None/AllowOverride All/g"  
    /etc/apache2/apache2.conf
```

```
    a2enmod rewrite
```

```
fi
```

```
source /etc/apache2/envvars
```

```
tail -F /var/log/apache2/* &
```

```
exec apache2 -D FOREGROUND
```

On essaye !

```
> docker run --name myapache -rm -p 80:80 tutum/apache-php
```

<http://localhost/>

On regarde ce qui se passe dedans

```
> docker exec -it myapache bash
```

```
> ls
```

```
> pwd
```

Je souhaite mapper le dossier /app avec un dossier local...

```
> docker stop myapache
```

```
> docker run --name myapache -rm -p 80:80 -v d:/docker-emn/phpenv/www:/app tutum/apache-php
```

<http://localhost/phpinfo.php>

Sucharger la configuration de PHP

Permettre l'affichage des erreurs

```
D:\docker-  
emn\phpenv\config\php.ini  
display_errors=1  
error_reporting=E_ALL
```

```
> docker run --name myapache --rm -  
p 80:80 -v d:/docker-  
emn/phpenv/config/php.ini:/etc/php  
5/apache2/conf.d/30-custom.ini -v  
d:/docker-emn/phpenv/www:/app  
tutum/apache-php
```

Générer une erreur

```
D:\docker-  
emn\phpenv\www\phpinfo.php  
  
phpinfo();
```

```
> http://localhost/phpinfo.php
```

Simplifier avec docker-compose

D:\docker-emn\phpenv\docker\docker-compose.yml

```
> cd D:\docker-emn\phpenv\docker  
> docker-compose up
```

web:

image: tutum/apache-php

ports:

- "80:80"

volumes:

- "d:/docker-emn/phpenv/www:/app"

- "d:/docker-emn/phpenv/config/php.ini:/etc/php5/apache2/conf.d/30-custom.ini"

Regardons ce qu'y s'y passe

```
> docker inspect docker_web_1
```

```
[
  {
    "Id":
"830b9a1ef42272219f4bf67a4bea7144b15a7178193f
af6d0d2f0e0ae279fe49",
    "Created": "2016-12-08T16:14:43.7620555Z",
    "Path": "/run.sh",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 15233,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2016-12-
08T16:14:44.4459915Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },

```

```
    "PortBindings": {
      "80/tcp": [
        {
          "HostIp": "",
          "HostPort": "80"
        }
      ]
    },
    "Mounts": [
      {
        "Source": "/d/docker-emn/phpenv/www",
        "Destination": "/app",
        "Mode": "rw",
        "RW": true,
        "Propagation": "rprivate"
      },
      {
        "Source": "/d/docker-
emn/phpenv/config/php.ini",
        "Destination":
"/etc/php5/apache2/conf.d/30-custom.ini",
        "Mode": "rw",
        "RW": true,
        "Propagation": "rprivate"
      }
    ]
  },
]
```

Base de données...

D:\docker-emn\phpenv\docker\docker-compose.yml

web:

..

links:

- db:db

db:

image: mysql

volumes:

- "d:/docker-emn/phpenv/mysql:/var/lib/mysql"

environment:

- MYSQL_ROOT_PASSWORD=root

On teste le lien avec la base de données

> docker exec -it docker_web_1 bash

> ping db

PING db (172.17.0.2) 56(84) bytes of data.

64 bytes from db (172.17.0.2): icmp_seq=1 ttl=64
time=0.212 ms

> printenv

HOSTNAME=59777584e3c8

DB_1_PORT_3306_TCP_PROTO=tcp

DB_NAME=/docker_web_1/db

DB_PORT=tcp://172.17.0.2:3306

DB_PORT_3306_TCP_PORT=3306

Administrer la BDD

<https://www.adminer.org/#download>

"D:\docker-emn\phpenv\www\adminer-4.2.5-mysql.php"

<http://localhost/adminer-4.2.5-mysql.php>

Authentification

Système	MySQL ▼
Serveur	db
Utilisateur	root
Mot de passe
Base de données	

☐ Authentification permanente

Docker File / Principales instructions

- FROM : baseimage utilisée
- RUN : Commandes effectuées lors du build de l'image
- EXPOSE : Ports exposées lors du run (si-P est précisé)
- ENV : Variables d'environnement du conteneur à l'instanciation
- CMD : Commande unique lancée par le conteneur
- ENTRYPOINT : "Préfixe" de la commande unique lancée par le conteneur

Bonnes pratiques :

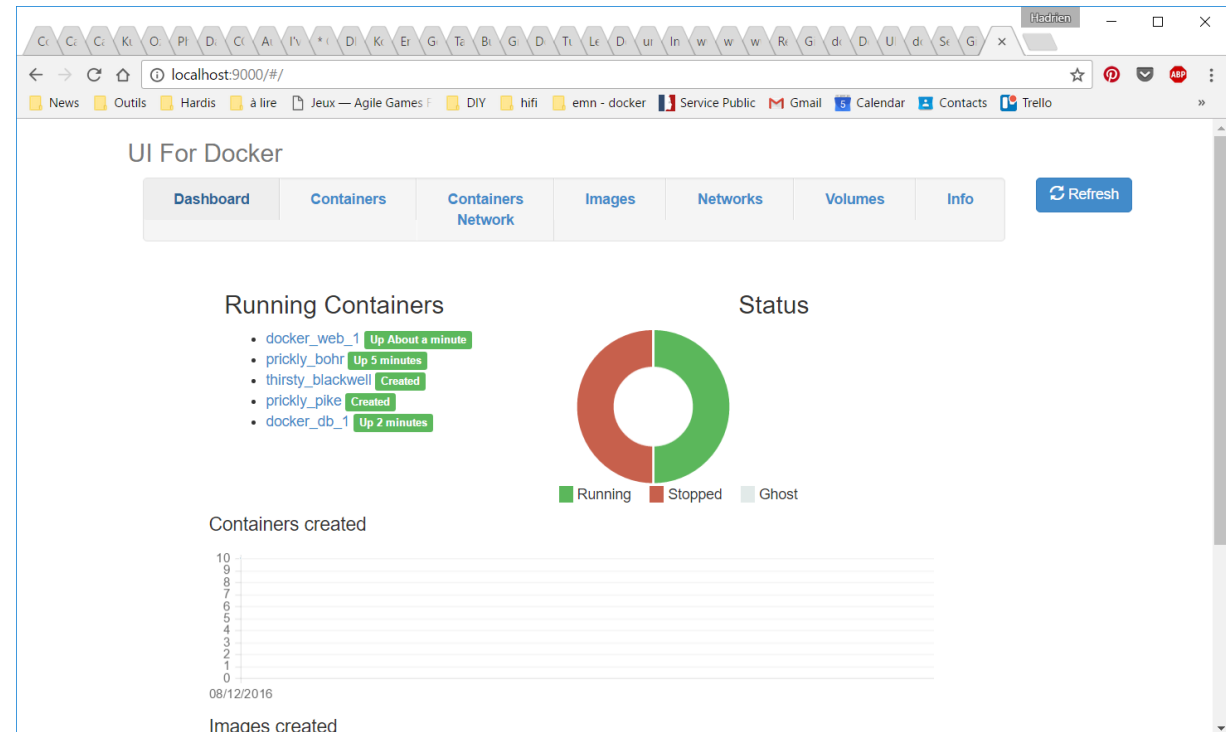
- Bien choisir sa baseimage
- Chaque commande Dockerfile génère un nouveau layer
- Comptez vos layers !

Interfaces graphiques / Docker UI

```
> docker run -d -p 9000:9000 --  
privileged -v  
/var/run/docker.sock:/var/run/do  
cker.sock uifd/ui-for-docker
```

<https://github.com/kevana/ui-for-docker>

<http://localhost:9000/#/>

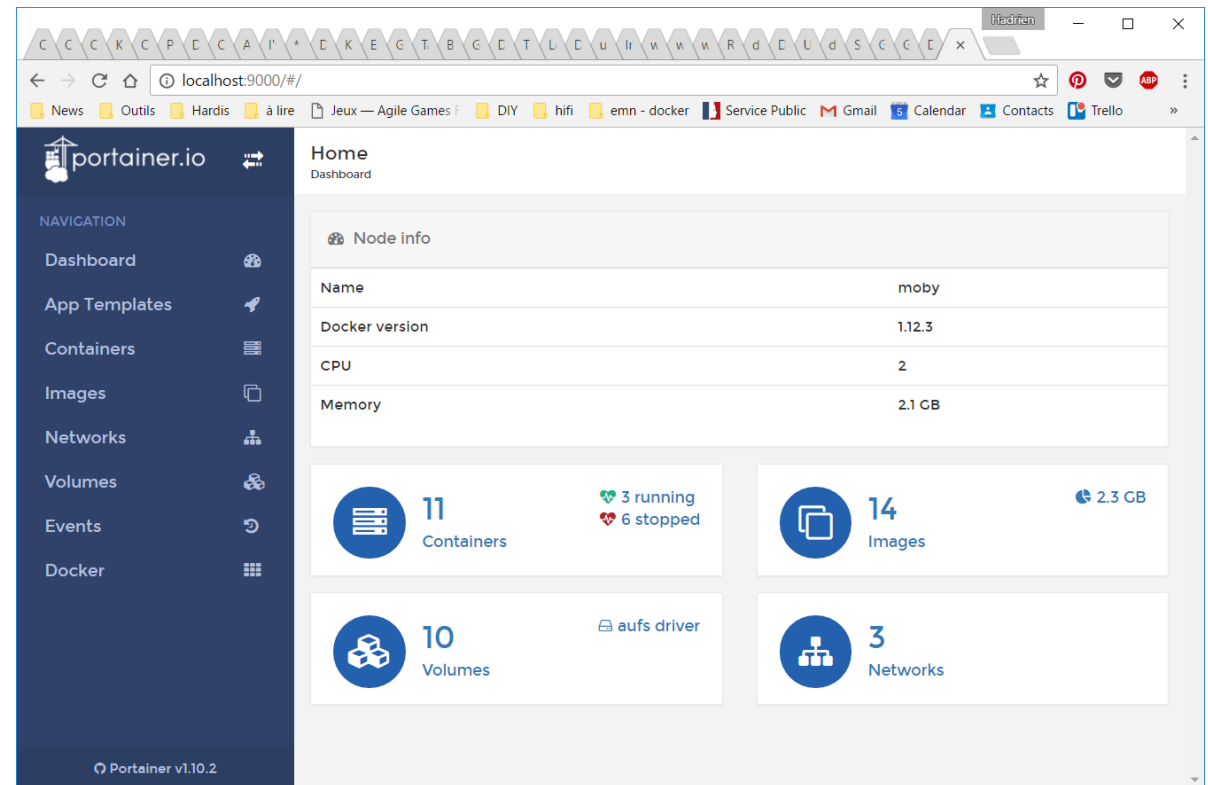


Interfaces Graphiques / Portainer

```
> docker run -d -p 9000:9000 -v  
/var/run/docker.sock:/var/run/do  
cker.sock portainer/portainer
```

<http://portainer.io/>

<http://localhost:9000/#/>



Commandes utiles

Supprimer tous les conteneurs

```
> docker rm $(docker ps -a -q)
```

Sauvegarder un conteneur

```
> docker commit mon-conteneur  
backup/mon-conteneur
```

Exporter une image

```
> docker save -o mon-image.tar  
backup/mon-conteneur
```

Importer une image

```
> docker import mon-image.tar  
backup/mon-conteneur
```

Les outils

- **Docker compose** : permet de définir des applications multi-conteneur en un fichier
- **Docker machine** : permet de créer des hôtes Docker en local, sur le cloud et dans un data center
- **Docker swarm** : permet de créer et gérer des clusters de conteneurs

Gérer un cluster

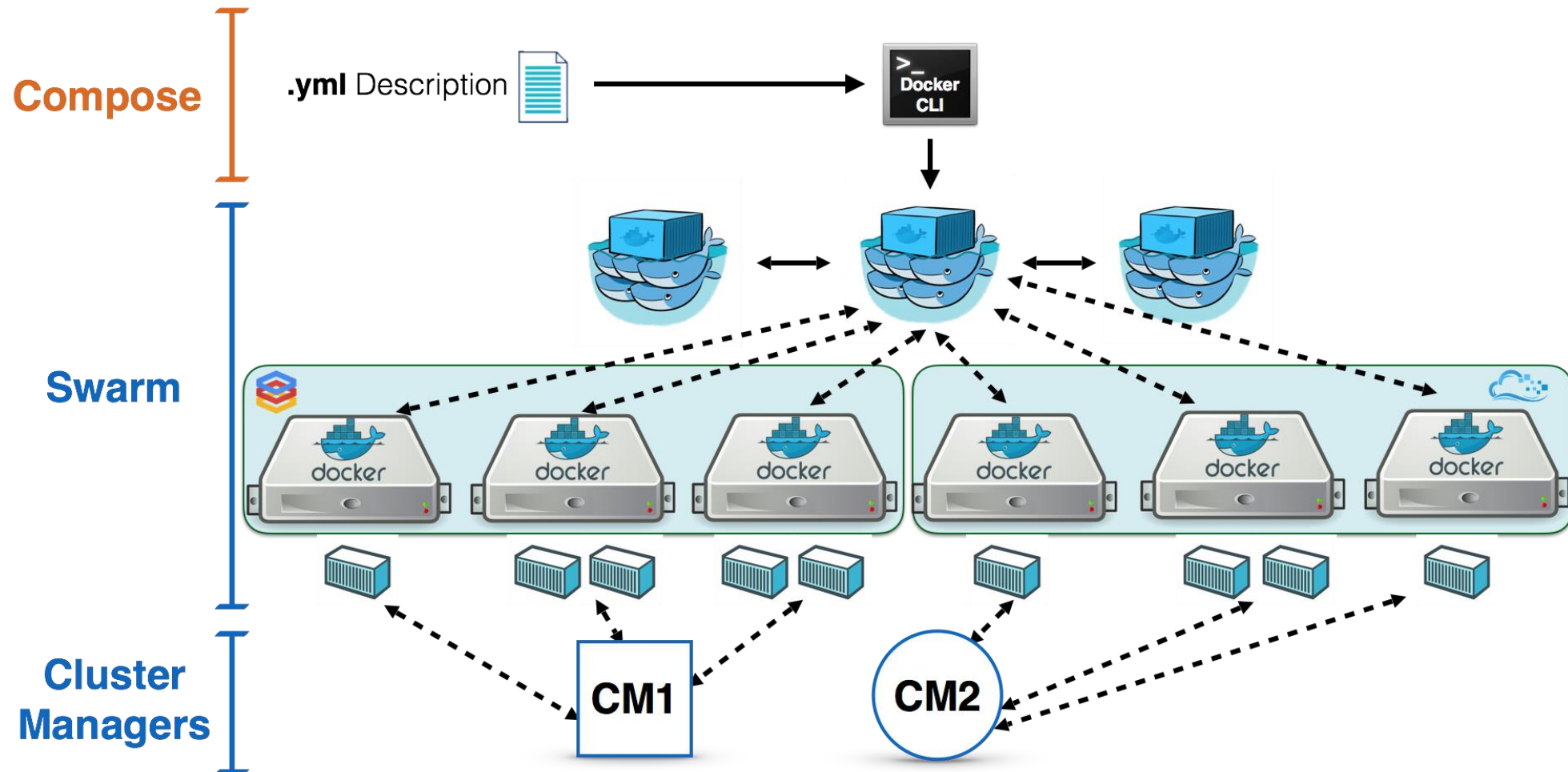
Swarm

- Go
- A pour but d'exposer sous la forme standard de l'api docker un ensemble d'hôtes Docker
- Permet de gérer un cluster pour déployer les conteneurs tout en mimant la façon dont se comporterait une machine hôte
- Compatible avec Compose et Machine

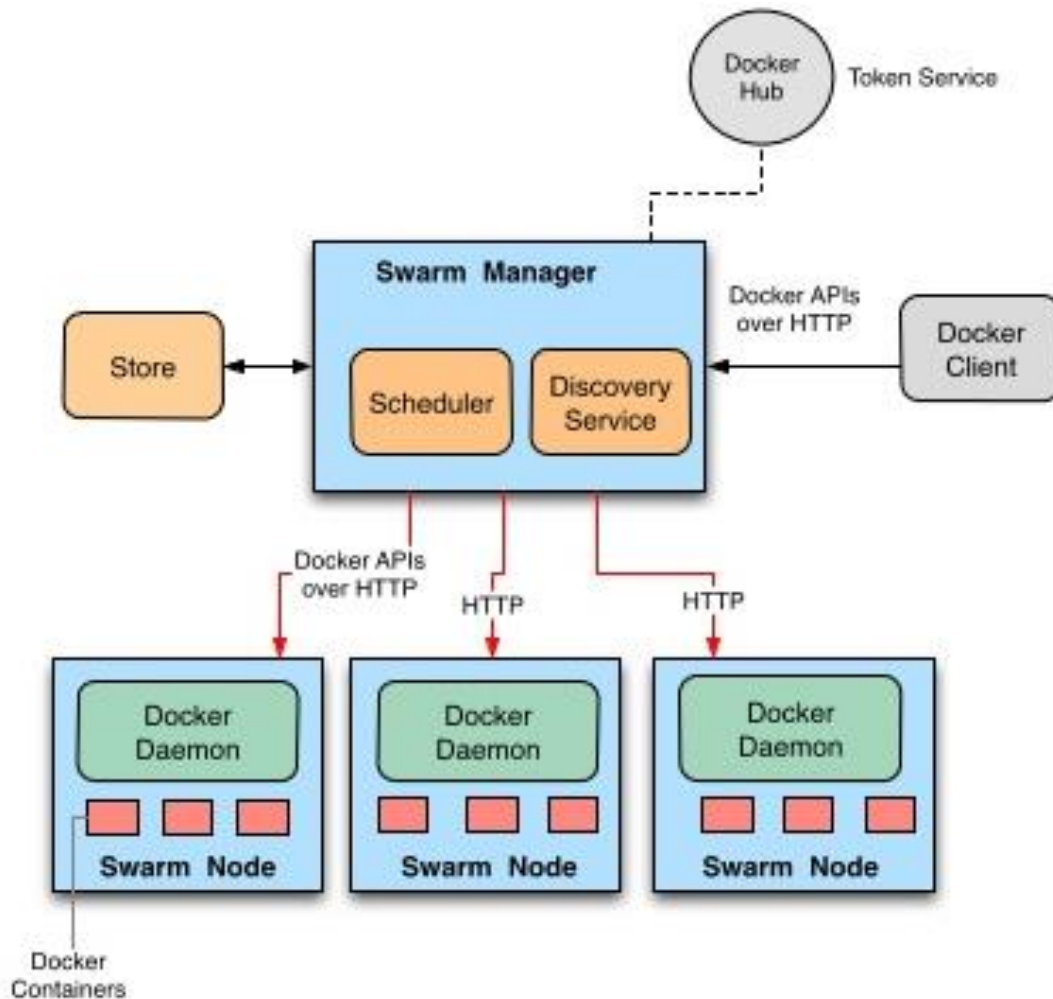
Kubernetes

- Go
- Equivalent très avancé de Swarm
- Supporte différentes technologies de conteneurs
 - Docker,
 - rkt,
 - lxc

Swarm + Machine + Compose



Swarm Architecture



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

To **add** a worker to this swarm, **run** the following command: `docker swarm join \ --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \ 192.168.99.100:2377`

```
docker node ls
```

```
docker swarm join \ --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c \ 192.168.99.100:2377
```

```
docker service create --replicas 1 --name  
helloworld alpine ping docker.com
```

```
docker service ls
```

```
docker service inspect --pretty helloworld
```

```
docker service scale helloworld=5
```

```
docker service ps helloworld
```

```
docker service rm helloworld
```


Ressources

- <https://www.docker.com/what-docker>
- <https://www.docker.com/products/docker>
- <http://www.slideshare.net/endhrk/introduction-to-docker-36472476>
- <http://www.slideshare.net/egorpushkin/docker-demo>
- <file:///C:/Users/HBO/Desktop/veille/docker%20emn%20-%20qualifi%C3%A9/docker/docker%20-%20cours%20ozone.pdf>
- <file:///C:/Users/HBO/Desktop/veille/docker%20emn%20-%20qualifi%C3%A9/docker/introduction%20%C3%A0%20docker%20-%20cours%20uns.pdf>