



Formation Docker

DMR - 28/29/30 mai 2018 - @lucjuggery

Tour de table

L'entreprise dans laquelle vous travaillez

Rôle actuel

Votre background

Ce que vous attendez de cette formation

Mon profil professionnel



Luc Juggery

Dans l'IT depuis +15 ans

Background de développeur

Dans l'écosystème des startups

Organisateur du Meetup Docker Nice

Docker Captain / DCA

<https://medium.com/lucjuggery>

@lucjuggery / luc.juggery@gmail.com

Au programme

- Quick Wins
- Des concepts utiles
- Les containers Linux
- La plateforme Docker
- Les containers avec Docker
- Les images Docker
- Registry
- Stockage
- Docker Machine
- Docker Compose
- Docker Swarm
- Networking
- Sécurité
- Monitoring
- Docker EE
- L'écosystème

Mise en pratique

- Exercices
 - Tout au long du cours
 - <https://github.com/lucj/docker-exercices>
- Projet IoT
 - Construction d'un backend pour un projet IoT
 - <https://github.com/lucj/IoT-demo-project>
 - Exemple d'implémentation : <https://github.com/lucj/iot-api>

Quick Wins

De nombreux logiciels

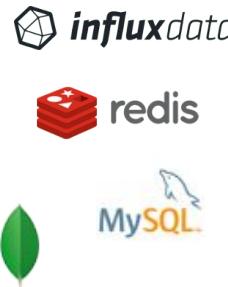
- Disponibles dans le Docker Hub <https://hub.docker.com>
- Packagés dans des **images**
- Utilisables immédiatement



CentOS



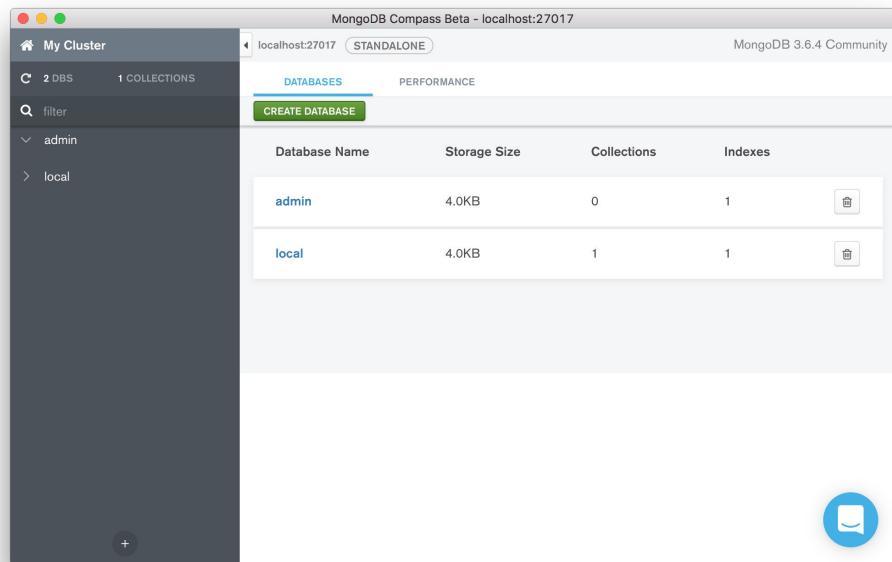
debian



De nombreux logiciels

Exemple: besoin d'une base Mongo 3.6 ?

```
# Lancement du process mongod  
$ docker run -p 27017:27017 -d mongo:3.6
```



The screenshot shows the MongoDB Compass Beta interface connected to a cluster at localhost:27017. The left sidebar displays 'My Cluster' with 2 DBS and 1 COLLECTIONS. The main area shows two databases: 'admin' and 'local'. The 'admin' database has 4.0KB storage size, 0 collections, and 1 index. The 'local' database also has 4.0KB storage size, 1 collection, and 1 index. A green 'CREATE DATABASE' button is visible above the table.

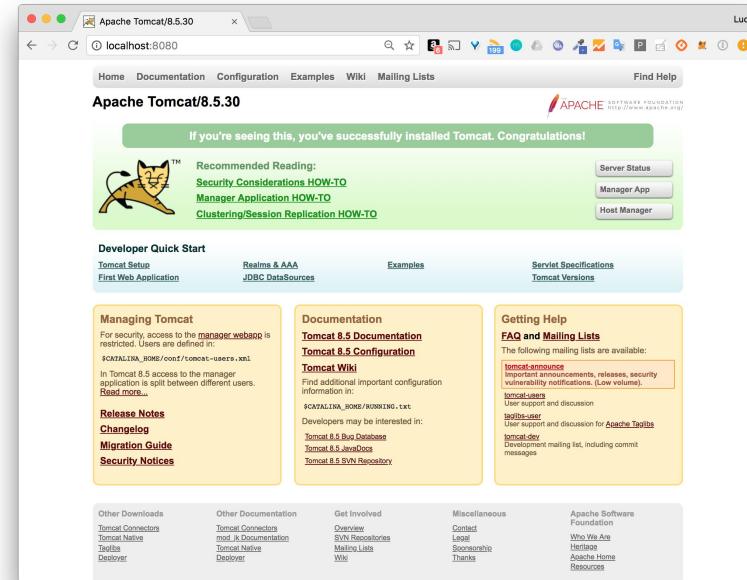
Database Name	Storage Size	Collections	Indexes
admin	4.0KB	0	1
local	4.0KB	1	1

Connexion via MongoDB Compass

De nombreux logiciels

Exemple : besoin de tester Tomcat 8 ?

```
# Lancement de Tomcat 8.5  
$ docker run -p 8080:8080 -d tomcat:8.5
```



Accès à l'interface depuis la machine locale

Des versions différentes en simultanée

- Utilisation de Mongo 3.6

```
# Lancement du process mongod  
$ docker run -p 27017:27017 -d mongo:3.6
```

The screenshot shows the MongoDB Compass interface for a cluster named "My Cluster". The left sidebar shows 2 DBS and 1 COLLECTIONS. The main area displays two databases: "admin" and "local". Both databases have a storage size of 16.0KB, 0 collections, and 1 index. A green "CREATE DATABASE" button is visible at the bottom.

Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	1
local	16.0KB	1	1

- Utilisation de Mongo 3.4

```
# Lancement du process mongod  
$ docker run -p 27018:27017 -d mongo:3.4
```

The screenshot shows the MongoDB Compass interface for a cluster named "My Cluster". The left sidebar shows 2 DBS and 1 COLLECTIONS. The main area displays two databases: "admin" and "local". Both databases have a storage size of 4.0KB, 0 collections, and 2 indexes. A green "CREATE DATABASE" button is visible at the bottom.

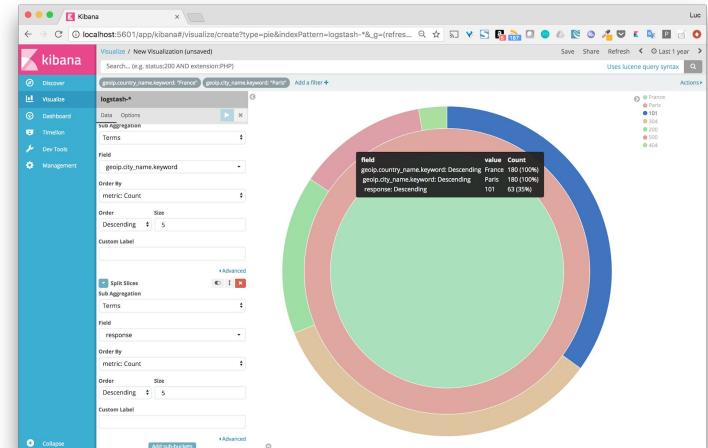
Database Name	Storage Size	Collections	Indexes
admin	4.0KB	0	2
local	4.0KB	1	1

Des stacks complètes

Exemple : Elastic (Logstash, Elasticsearch, Kibana) <https://www.elastic.co/fr/products>

```
version: '3.3'
services:
  logstash:
    image: logstash:5.5.2
    volumes:
      - ./logstash.conf:/config/logstash.conf
    command: ["logstash", "-f", "/config/logstash.conf"]
  elasticsearch:
    image: elasticsearch:5.5.2
    environment:
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
  kibana:
    image: kibana:5.5.2
    ports:
      - 5601:5601
```

```
$ docker-compose up
```

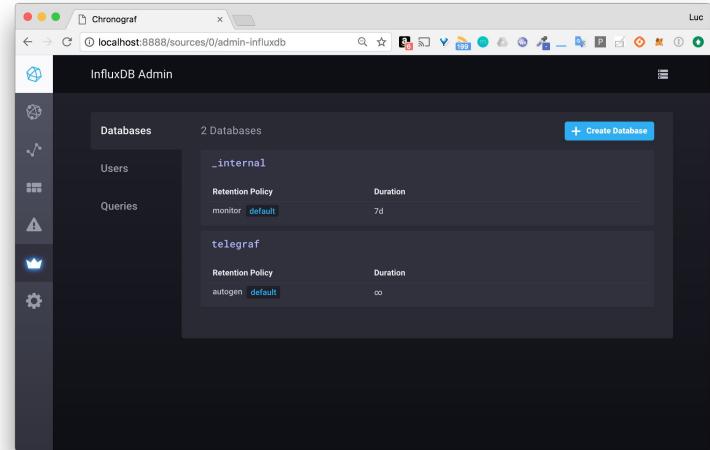


Des stacks complètes

Exemple : TICK <https://www.influxdata.com/time-series-platform/>

```
version: "3.6"
services:
  telegraf:
    image: telegraf
    configs:
      - source: TELEGRAF_CONF
        target: /etc/telegraf/telegraf.conf
    ports:
      - 8186:8186
  influxdb:
    image: influxdb
  chronograf:
    image: chronograf
    ports:
      - 8888:8888
  kapacitor:
    image: kapacitor
    ports:
      - 9092:9092
  configs:
    TELEGRAF_CONF:
      file: ./telegraf.conf
```

```
$ docker stack deploy -c tick.yaml tick
```

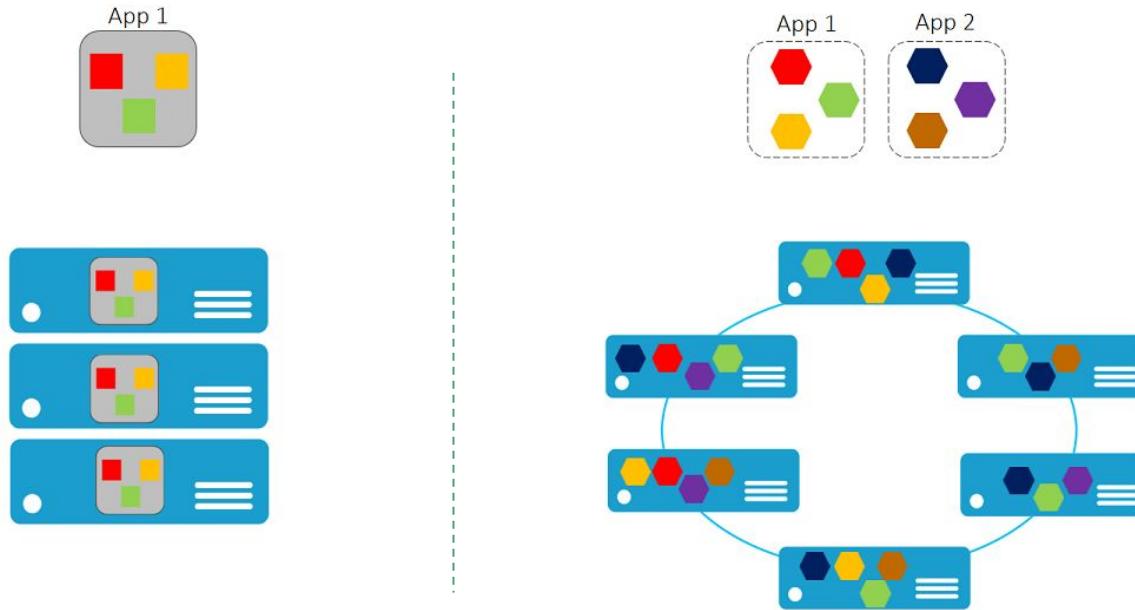


Gestion de vos propres images

- Packaging d'une application et de l'ensemble de ses dépendances
- Facilite la distribution et le déploiement

Des concepts utiles

Architecture monolithique vs micro-services



<https://docs.microsoft.com>

Architecture micro-services

Pros :

- Découpage de l'application en processus (services) indépendants
- Chacun a sa propre responsabilité métier
- Équipe dédiée pour chaque service
- Plus de liberté de choix dans le langage
- Mise à jour et scaling horizontale
- Containers très adaptés pour les micro-services

Architecture micro-services

Cons :

- Nécessite des interfaces bien définies
- Focus sur les tests d'intégration
- Déplace la complexité dans l'orchestration de l'application globale

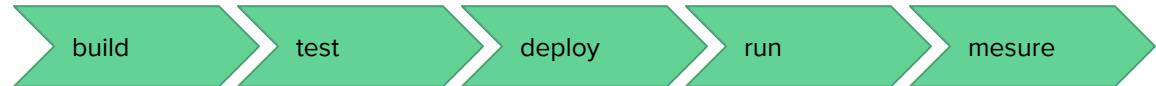
Application Cloud Native

- Application orientée microservices
- Packagée dans des containers
- Orchestration dynamique
- Nombreux projets portés par la CNCF (Cloud Native Computing Foundation)
 - Kubernetes
 - Prometheus
 - Fluentd
 - ...
- cncf.io

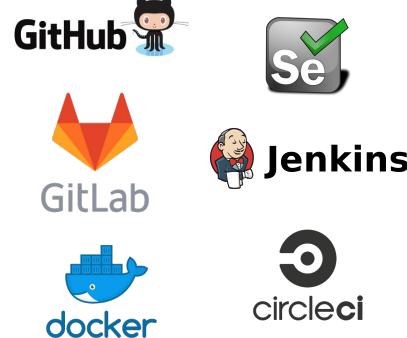
DevOps

- Un objectif : minimiser le temps de livraison d'une fonctionnalité
- Déploiements réguliers
- Mise en avant des tests
- Automatisation des processus
 - provisioning / configuration
 - Infrastructure As Code (IaC)
 - Intégration Continue / Déploiement Continu (CI/CD)
 - monitoring
- Boucle d'amélioration courte

DevOps : quelques outils et produits



Google Cloud Platform



Les containers Linux

Qu'est ce qu'un container ?

- Un processus !
- Isolé des autres processus
- Partage le Kernel de la machine hôte avec les autres containers
- Avec sa propre vision du système sur lequel il tourne (Namespaces)
- Limité dans les ressources qu'il peut utiliser (Control Groups)

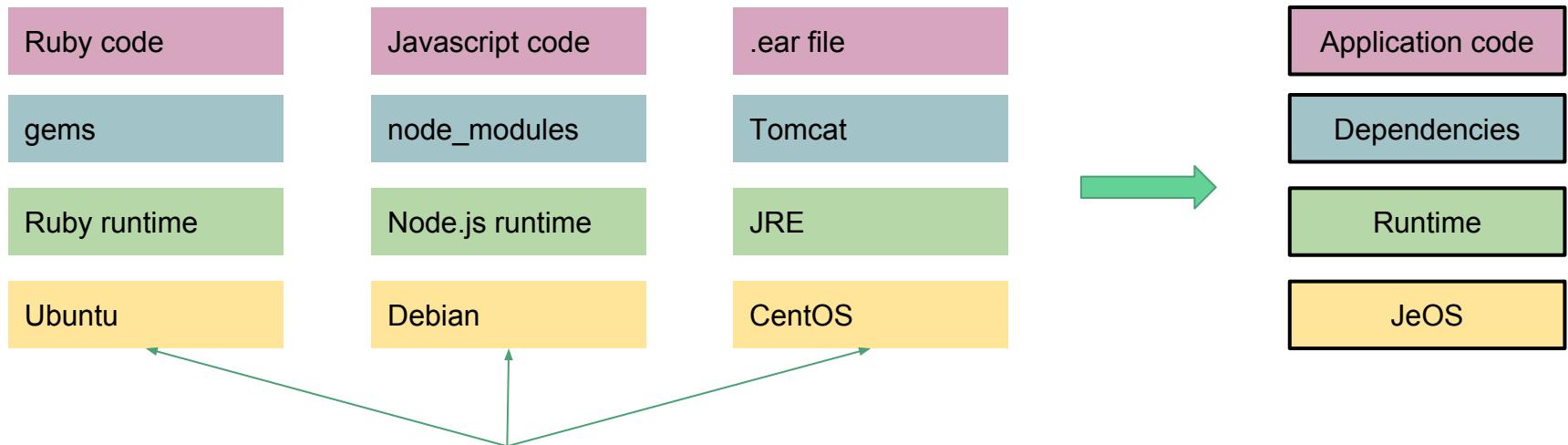
Namespaces

- Technologies Linux pour isoler un processus
- Les namespaces limitent ce qu'un container peut voir
- Différents namespaces
 - pid: isolation de l'espace des processus
 - net: donne une stack réseau privée
 - mount: système de fichiers privé
 - uts: nom du host
 - ipc: isole les communications inter processus
 - user: mapping des UID/GID entre l'hôte et les containers

Namespaces : mount

- Système de fichiers visible par le process
- Contient une application et toutes ses dépendances
- Composition
 - JeOS (Just Enough Operating System) - bibliothèques et binaires système
 - Environnement d'exécution
 - Dépendances applicatives
 - Code applicatif

Namespaces : mount



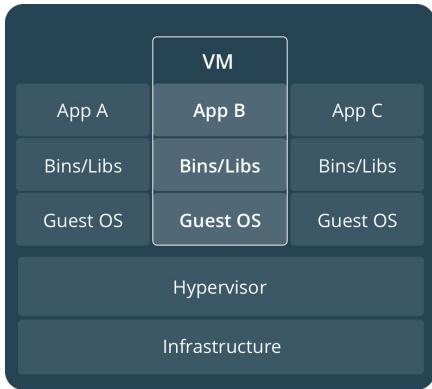
Partie du système de fichiers contenant les binaires et les librairies du système d'exploitation de base

Control Groups (cgroups)

- Technologie Linux
- Limite les ressources qu'un processus peut utiliser
 - RAM
 - CPU
 - I/O
 - Network

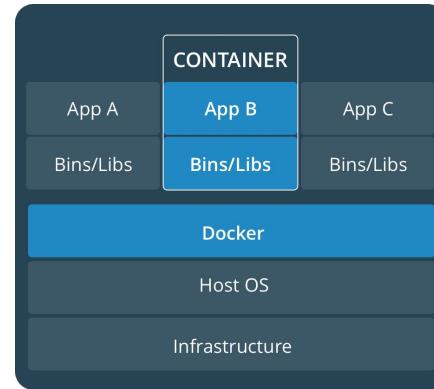
Container vs Machine Virtuelle

Machines virtuelles



- Nécessite un hyperviseur
- Chaque VM a son OS
- Overhead RAM / CPU

Containers



- Processus
- Partage le Kernel de la machine hôte

Ce qu'il faut retenir

- Un container est un processus isolé qui a sa propre vision du système
- Namespaces : ce que le processus peut voir
- Control groups: limitent les ressources que le processus peut utiliser
- Les containers partagent le kernel de la machine hôte

La plateforme Docker

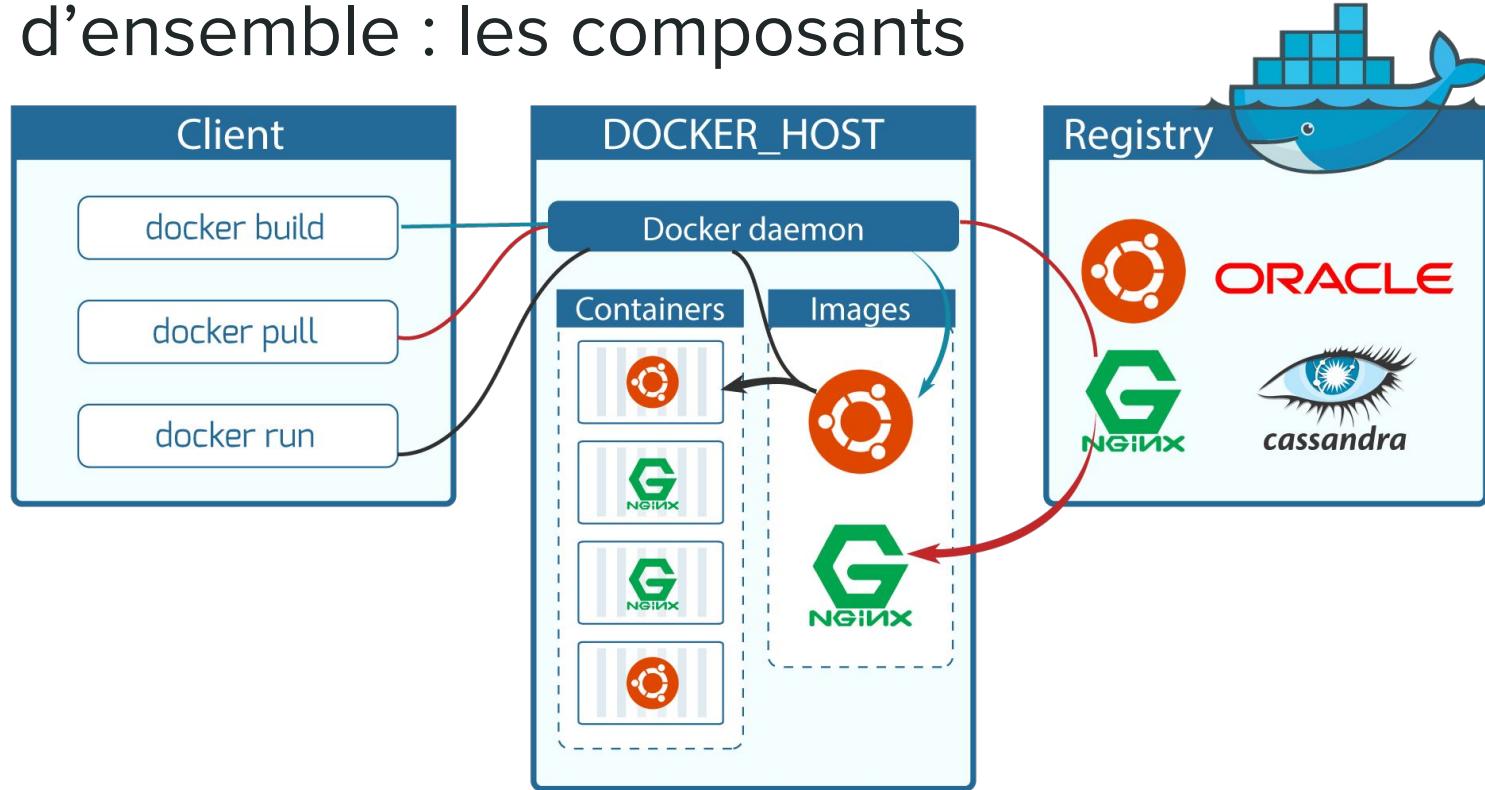
Sommaire

- Vue d'ensemble
- Les différentes éditions
- Modèle client / serveur
- Installation
- Online playground

Vue d'ensemble

- Rend facilement accessible l'utilisation des containers Linux
- Introduit la notion **d'image** pour packager une application et ses dépendances
- Fichier texte **Dockerfile** pour la création d'images
- Facilite la distribution et le déploiement des applications

Vue d'ensemble : les composants



Vue d'ensemble : Docker pour les Devs

- Application / language / stack agnostique
- Expérience utilisateur
- Orchestration intégrée
- RéPLICATION de l'environnement de production

Vue d'ensemble : Docker pour les Ops

- Déploiement plus fréquent
- Scalabilité
- Distribution facilitée
- Sécurité

Vue d'ensemble : Docker pour l'entreprise

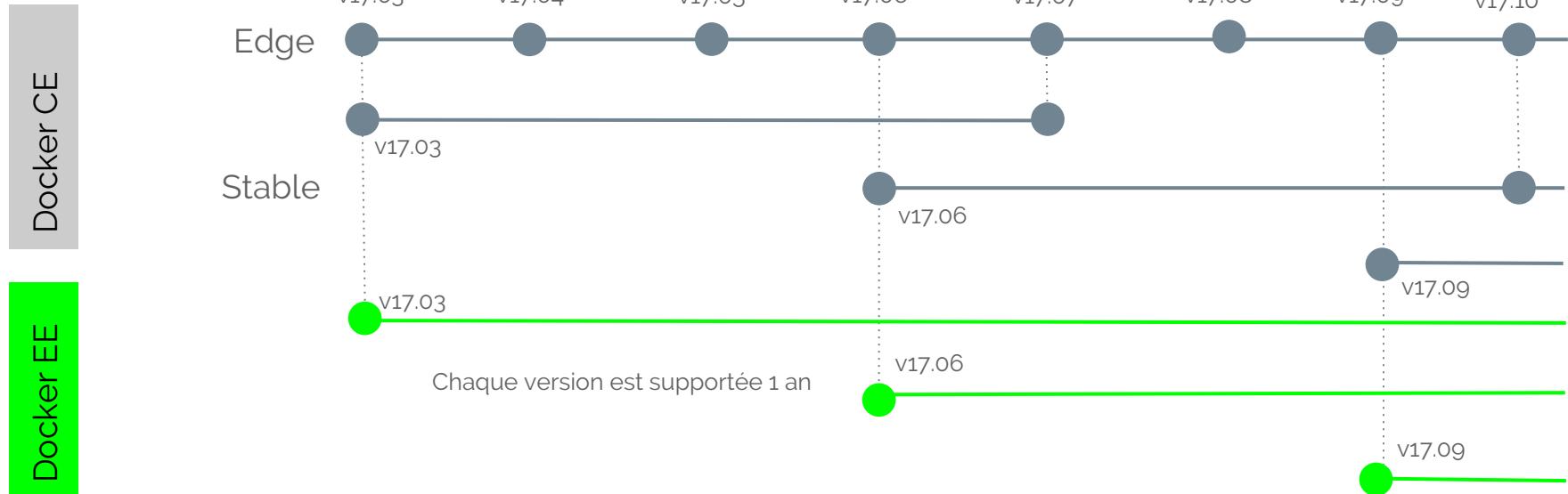
- Une même plateforme pour la gestion de toutes les applications
- Accélère la mise en place de pipelines de déploiement automatiques
- Interface commune qui permet aux Devs et Ops de travailler ensemble
- Intégration dans le workflow de l'entreprise

Les différentes éditions

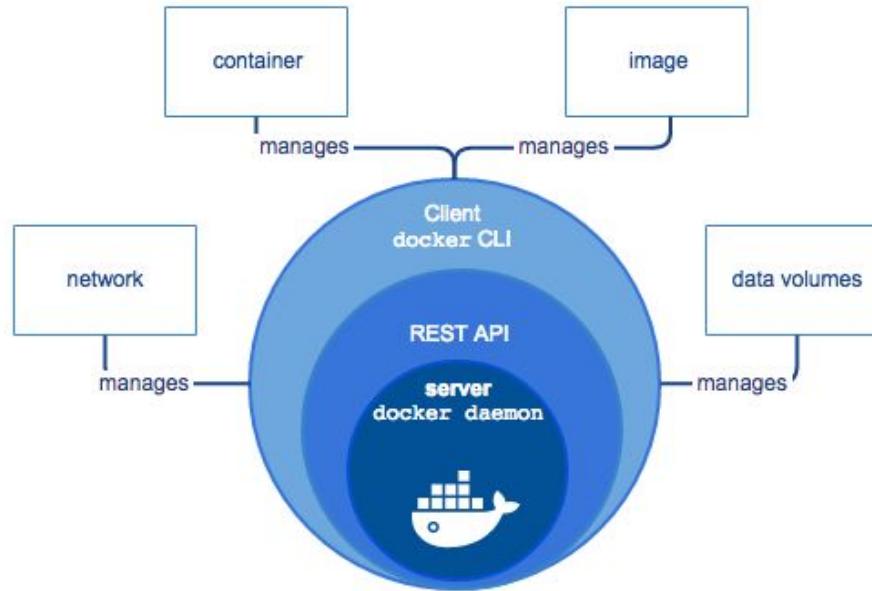
- Docker Community Edition (CE)
 - gratuite
 - une release *stable* tous les trimestres
 - une release *edge* tous les mois
- Docker Enterprise Edition (EE)
 - 3 tiers (basic, standard, avancé)
 - une release tous les trimestres avec 1 an de support
 - plateforme CaaS (Container as a Service)
 - technologie certifiée: Infrastructure, Plugins, Containers, ...

Les différentes éditions : Year.Month

- Depuis mars 2017
- 1.13.1 ⇒ dernière release utilisant le modèle précédent



Modèle client / serveur



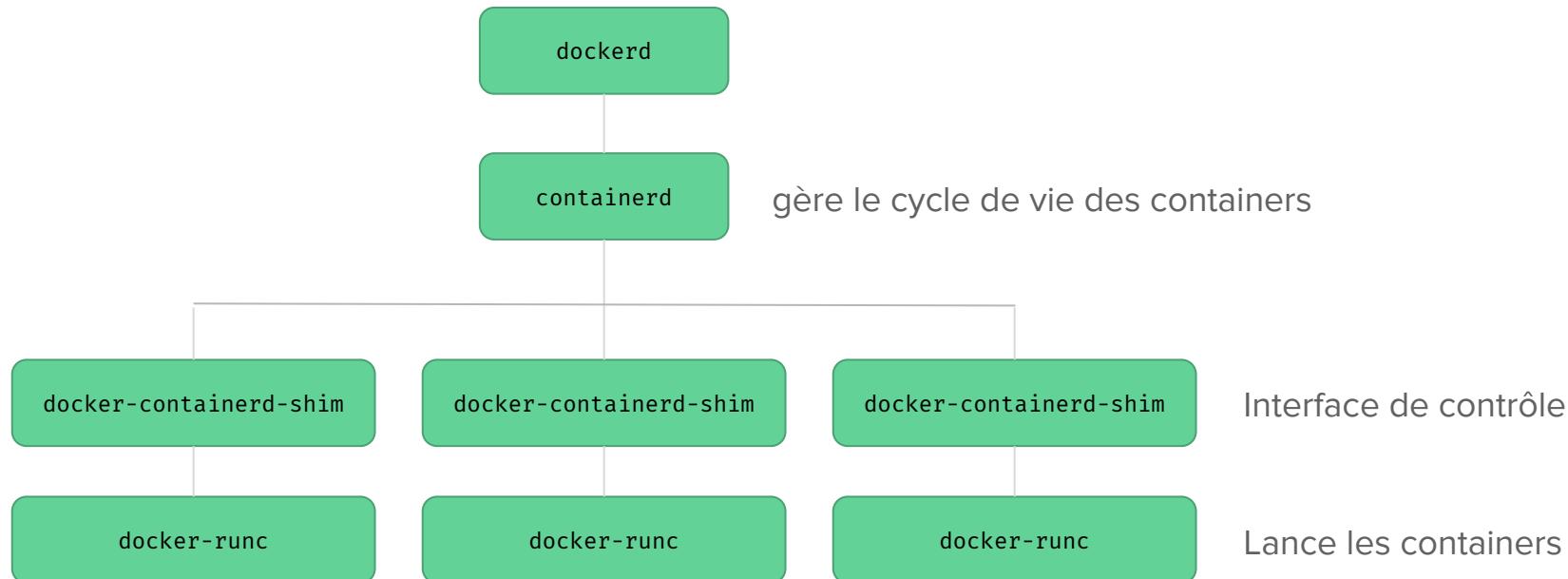
<https://docs.docker.com/>

Modèle client / serveur : côté serveur

- Processus ***dockerd***
- Gestion des images, networks, volumes, cluster
- Expose une API HTTP Rest
- Ecoute sur socket unix et/ou tcp
- Nombreuses options de démarrage

Modèle client / serveur : côté serveur

dockerd délègue la gestion des containers à ***containerd***



Modèle client / serveur : côté serveur

- Configuration du daemon en ligne de commande ou via DOCKER_OPTS
- Fichiers de configuration dépendant du système d'init
 - SystemV / Upstart
 - Ex: Ubuntu 14.04
 - /etc/docker/default
 - Systemd
 - Ex: Fedora, CentOS
 - Fichier dans /etc/systemd/system/docker.service.d/
 - Par défaut
 - /etc/docker/daemon.json

Modèle client / serveur : côté serveur

```
$ systemctl status docker
docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/docker.service.d
             └─10-machine.conf
     Active: active (running) since Wed 2018-02-21 14:17:54 UTC; 4min 34s ago
       Docs: https://docs.docker.com
    Main PID: 5185 (dockerd)
      CGroup: /system.slice/docker.service
              ├─5185 /usr/bin/dockerd -H tcp://0.0.0.0:2376 -H unix:///var/run/docker.sock --storage-driver aufs
--tlsverify --tlscacert /etc/docker/ca.pem
              └─5193 docker-containerd --config /var/run/docker/containerd/containerd.toml

$ cat /etc/systemd/system/docker.service.d/10-machine.conf
[Service]
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2376 -H unix:///var/run/docker.sock --storage-driver aufs --tlsverify
--tlscacert /etc/docker/ca.pem --tlscert /etc/docker/server.pem --tlskey /etc/docker/server-key.pem --label
provider=digitalocean
Environment=
```

Exemple d'options de démarrage sur Ubuntu Xenial (16.04)

Modèle client / serveur : côté serveur

```
$ docker version
```

Client:

```
Version:          18.02.0-ce
API version:     1.36
Go version:      go1.9.3
Git commit:      fc4de44
Built:           Wed Feb  7 21:13:05 2018
OS/Arch:         darwin/amd64
Experimental:    true
Orchestrator:   kubernetes
```



Server:

```
Engine:
Version:          18.02.0-ce
API version:     1.36 (minimum version 1.12)
Go version:      go1.9.3
Git commit:      fc4de44
Built:           Wed Feb  7 21:20:15 2018
OS/Arch:         linux/amd64
Experimental:    true
```

Client tourne sur macOS

Serveur tourne dans une VM boot2docker sur xhyve
Docker-CE version edge (expérimentale)

```
$ docker version
```

Client:

```
Version:          18.02.0-ce
API version:     1.36
Go version:      go1.9.3
Git commit:      fc4de44
Built:           Wed Feb  7 21:16:33 2018
OS/Arch:         linux/amd64
Experimental:    false
Orchestrator:   swarm
```



LINUX

Server:

```
Engine:
Version:          18.02.0-ce
API version:     1.36 (minimum version 1.12)
Go version:      go1.9.3
Git commit:      fc4de44
Built:           Wed Feb  7 21:15:05 2018
OS/Arch:         linux/amd64
Experimental:    false
```

Même architecture côté client et serveur

Docker-CE version stable

Modèle client / serveur : côté client

- Binaire **docker**
- Installé avec le daemon
- Communique avec le daemon à travers une socket unix par défaut
 - /var/run/docker.sock
- Peut communiquer avec des daemons distants via tcp
- Configuration via des variables d'environnement
 - DOCKER_HOST : adresse IP du Docker daemon

Installation

The screenshot shows a web browser window for the Docker Store at <https://store.docker.com>. The main heading is "Get started with Docker". Below it, two sections are shown: "Community Edition (CE)" and "Enterprise Edition (EE)". The CE section features a small icon of a ship in a harbor and a brief description: "A free Docker platform for developers and "do it yourself" ops teams to get started with Docker." A blue "GET DOCKER CE →" button is present. The EE section features a small icon of a city skyline and a brief description: "A subscription with support and certification for IT teams running critical apps in production." A blue "GET DOCKER EE →" button is present. To the right, a large white tablet-like device displays a grid of Docker container icons for various platforms, each with a star rating and pull count. The platforms listed are:

Platform	Rating	Pulls
Docker For Mac	★★★★★	577K
Docker For Windows	★★★★★	483K
Docker For Ubuntu	★★★★★	73K
Docker For AWS	★★★★★	368K
Docker For Azure	★★★★★	175K
Docker For SLES	★★★★★	175K
Docker For Fedora	★★★★★	73K
Docker For CentOS	★★★★★	32K
Docker For Windows Server	★★★★★	327K
Docker For RHEL	★★★★★	37K
Docker For Oracle Linux	★★★★★	226K
Docker For Oracle Linux	★★★★★	226K

Installation

The Docker Store offers several editions of Docker Community Edition:

- Docker Community Edition for Mac**: Docker. The fastest and easiest way to get started with Docker on Mac.
- Docker Community Edition for Windows**: Docker. The fastest and easiest way to get started with Docker on Windows PC.
- Docker Community Edition for AWS**: Docker. A one click template to quickly deploy Docker on Amazon EC2.
- Docker Community Edition for Azure**: Docker. A one click template to quickly deploy Docker on Azure.
- Docker Community Edition for Fedora**: Docker. The best way to run Docker on Fedora.
- Docker Community Edition for CentOS**: Docker. The best way to run Docker on CentOS.
- Docker Community Edition for Ubuntu**: Docker. The best way to run Docker on Ubuntu.
- Docker Community Edition for Debian**: Docker. The best way to run Docker on Debian.

Filters used in the screenshots:

- Left screenshot (Desktop):** Platforms: Cloud (unchecked), Desktop (checked), Server (unchecked). Operating Systems: Linux (unchecked), Windows (unchecked), MacOS (unchecked).
- Middle screenshot (Cloud):** Platforms: Cloud (checked), Desktop (unchecked), Server (unchecked). Operating Systems: Linux (unchecked), Windows (unchecked), MacOS (unchecked).
- Right screenshot (Server):** Platforms: Cloud (unchecked), Desktop (unchecked), Server (checked). Operating Systems: Linux (unchecked), Windows (unchecked), MacOS (unchecked).

<https://store.docker.com/editions/community/docker-ce-server-ubuntu>

Les différents produits disponibles de la Community Edition

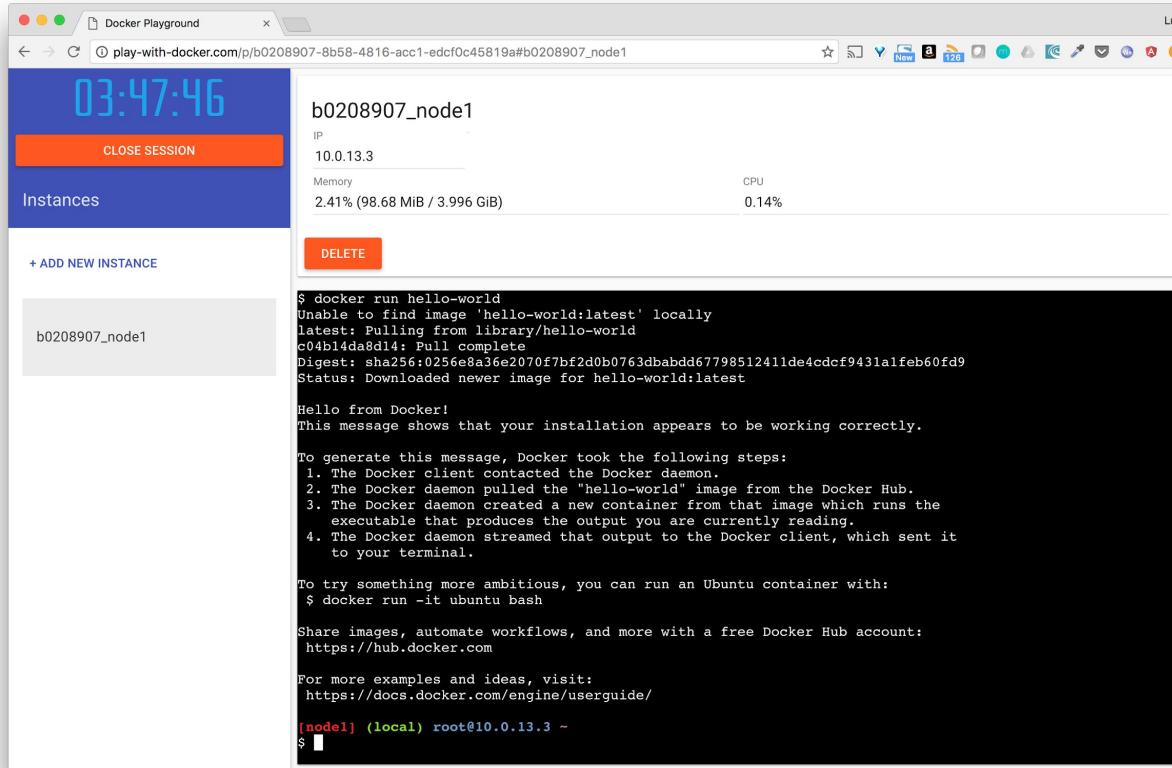
Installation

- Versions CE pour de nombreuses plateformes
 - macOS: Docker for Mac
 - Windows 10: Docker for Windows
 - Windows 7, 8: Docker toolbox
 - Linux: version dédiée (Ubuntu, Debian, CentOS, Fedora)
- Versions EE

Online playground

- Docker depuis un navigateur web !
- Play With Docker (PWD) <https://play-with-docker.com/>
- Créé par Marcos Nils & Jonathan Leibuisky
- Mises à jour très régulières

Online playground



Démo

Premiers pas avec PWD

Les containers avec Docker

Sommaire

- Création d'un container
- Mode interactif
- Foreground vs background
- Publication d'un port
- Limitation des ressources
- Container en mode privilégié
- D'autres options utiles
- Les commandes de base
- Des alias utiles

Création

- Lancement d'un processus dans un container
- Création à partir d'une **image**
 - système de fichiers / package complet d'une application
- *docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]*
- <https://docs.docker.com/engine/reference/commandline/run/>

Création : exemples

Création d'un container basé sur une image **alpine** et lancement de la commande **echo hello** dans le container



```
$ docker container run alpine echo hello
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
3690ec4760f9: Pull complete
Digest:
sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283e
e8200bf9a54f2e5c
Status: Downloaded newer image for alpine:latest
hello
```

L'image **alpine** n'est pas présente en local et est donc téléchargée depuis le repository officiel



La command **echo hello** est exécutée dans le nouveau container



Une fois l'image de base présente en local, la création d'un container est très rapide



```
$ docker container run alpine echo hello
hello
```

Exercice

Mode interactif

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

Mode interactif

Création d'un container, basé sur l'image **ubuntu**, en mode interactif

L'image **ubuntu** n'est pas présente en local,
elle est téléchargée

Un shell est lancé dans le container et
disponible depuis le terminal local



```
$ docker container run -ti ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
aed15891ba52: Pull complete
773ae8583d14: Pull complete
d1d48771f782: Pull complete
cd3d6cd6c0cf: Pull complete
8ff6f8a9120c: Pull complete
Digest:
sha256:35bc48a1ca97c3971611dc4662d08d131869daa692acb281c
7e9e052924e38b1
Status: Downloaded newer image for ubuntu:latest
root@d43ec32a629d:/#
root@d43ec32a629d:/# exit
$
```

Exercice

Foreground vs background

- Container lancé en foreground par défaut
 - prend la main sur le terminal courant
- Option `-d` pour le lancer en background
 - retourne l'identifiant du container (dans sa version longue)

Foreground vs background

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



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

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



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

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



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

Exercice

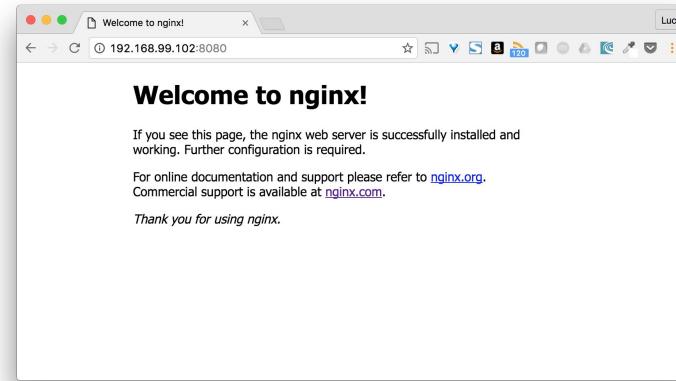
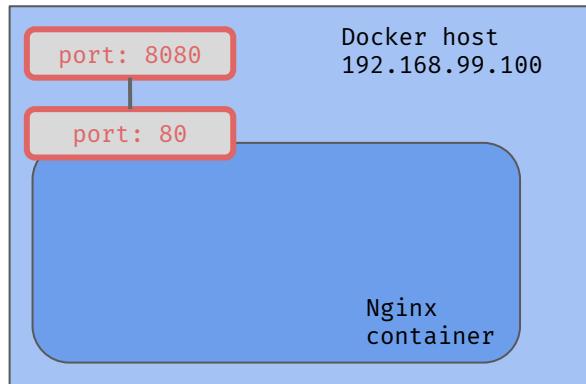
Publication d'un port

- Pour être accessible depuis l'extérieur via un port sur la machine hôte
- Allocation statique d'un port : $-p \text{ HOST_PORT:CONTAINER_PORT}$
- Allocation dynamique de l'ensemble des ports : $-P$
- Conflit si plusieurs containers utilisent le même port de la machine hôte

Publication d'un port

```
// Le port 80 de l'instance Nginx tournant dans le container est publié sur le port 8080 de l'hôte
$ docker container run -d -p 8080:80 nginx
Ed61ef8ff46704e454ec3460b850b0e1eee225252bbeecad7b88b1c2cb419f31
```

```
$ docker container ls
CONTAINER ID    IMAGE      COMMAND                  CREATED          STATUS          PORTS          NAMES
ed61ef8ff467    nginx      "nginx -g 'daemon of..."   About a minute ago   Up 2 seconds   0.0.0.0:8080->80/tcp   angry_chandrasek
```



Exercice

Limitation des ressources

- RAM, CPU, I/O
- <https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources>

```
# Limite de consommation de la RAM
$ docker container run --memory 32m estesp/hogit
→ OOMKilled

# Différentes valeurs de l'option --cpus sur un processeur 4-core

$ docker run -it --rm program/stress --cpu 4                                // Utilisation des 4 cores
⇒ CPU: 98% usr   1% sys   0% nic   0% idle   0% io    0% irq    0% sirq   // => 100% du CPU

$ docker run --cpus 0.5 -it --rm program/stress --cpu 4                      // Utilisation de 50% d'un core
⇒ CPU: 12% usr   0% sys   0% nic   86% idle   0% io    0% irq    0% sirq   // ~12% du CPU

$ docker run --cpus 2 -it --rm program/stress --cpu 4                          // Utilisation de 2 cores
⇒ CPU: 50% usr   2% sys   0% nic   47% idle   0% io    0% irq    0% sirq   // 50% du CPU
```

Container en mode privilégié

- Option *--privileged*
- Donne accès aux devices de la machine hôte
- A n'utiliser qu'en connaissance de cause !

The screenshot shows a terminal window with the title "2. /tmp (docker)". The command "ls /dev" is run twice: first without privileges and then with the "--privileged" flag. The output lists numerous device files, including console, core, fd, full, mqueue, null, ptmx, pts, random, shm, stderr, stdin, stdout, tty, urandom, zero, bsg, cachefiles, console, core, cpu, cpufreq, cpuidle, cpuidle_latency, cuse, fd, full, fuse, hpet, hwmon, input, kmsg, loop-control, loop0, loop1, loop2, loop3, loop4, loop5, loop6, loop7, loop8, loop9, mapper, mem, memory_bandwidth, mqueue, nbd0, nbd1, nbd10, nbd11, nbd12, net, network_latency, network_throughput, null, nvram, pts, random, shm, stderr, stdin, stdout, tty, urandom, zero, and vcs.

```
~ $ docker container run -ti alpine
/ # ls /dev
console  core    fd      full   mqueue  null    ptmx    pts     random  shm     stderr  stdin   stdout  tty     urandom zero
/ # exit
~ $ docker container run -ti --privileged alpine
/ # ls /dev
bsg      loop1   nbd13   port    stdout   tty21   tty36   tty50   tty8
cachefiles  loop2   nbd14   psaux   tty      tty22   tty37   tty51   tty9
console   loop3   nbd15   ptmx    tty0    tty23   tty38   tty52   tty50
core     loop4   nbd2    pts     tty1    tty24   tty39   tty53   tty51
cpu      loop5   nbd3    random  tty10   tty25   tty4    tty54   tty52
cpufreq  cpuidle cpuidle_latency  loop6   nbd4    tty11   tty26   tty40   tty55
cuse     loop7   nbd5    sda     tty12   tty27   tty41   tty56   uinput
fd       mapper  nbd6    sda1    tty13   tty28   tty42   tty57   urandom
full    mem     nbd7    sg0     tty14   tty29   tty43   tty58   vcs
fuse    memory_bandwidth  nbd8   sg1     tty15   tty3    tty44   tty59   vcs1
hpet    mqueue  nbd9    sg2     tty16   tty30   tty45   tty6    vcsa
hwmon   nbd0    net     shm     tty17   tty31   tty46   tty60   vcsd1
input   nbd1    network_latency  sr0    tty18   tty32   tty47   tty61   vsock
kmsg    nbd10   network_throughput  sr1    tty19   tty33   tty48   tty62   zero
loop-control  nbd11   null    stderr  tty2    tty34   tty49   tty63
loop0    nbd12   nvram   stdin   tty20   tty35   tty5    tty7
/ #
```

D'autres options utiles

- <https://docs.docker.com/engine/reference/run>

```
# Spécification du nom
$ docker container run -d --name debug alpine:3.7 sleep 10000

# Suppression du container quand il est stoppé
$ docker container run --rm --name debug alpine:3.7 sleep 10000

# Redémarrage automatique
$ docker container run --name api --restart=on-failure lucj/api
```

Les commandes de base

run	Création d'un container
ls	Liste des containers
inspect	Détails d'un container
logs	Visualisation des logs
exec	Lancement d'un processus dans un container existant
stop	Arrêt d'un container
rm	Suppression d'un container

Les commandes de base : ls

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

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

# Liste les identifiants des containers actifs et stoppés
$ docker container ls -a -q
5dbae351233f
6df4108911d3
6f6cac745aa1
```

nom attribué au container (via l'option --name ou autogénéré)

Exercice

Les commandes de base : inspect

- Vue détaillée d'un container
- Commande disponible pour chacune des primitives Docker

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

Identifiant ou nom du container

Les commandes de base : inspect

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

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

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

Exercice

Les commandes de base: logs

- Visualisation des logs d'un container
- Option -f pour la mise à jour continue
- Ecriture des logs sur la sortie / erreur standard

Bonne pratique: ne pas écrire les fichiers de logs dans le container

Les commandes de base: logs

```
$ docker container run -d --name ping ubuntu ping 8.8.8.8
```

```
04e44d2b6f4ab2a5380d86c4e12a4415cd7fa2291f87f14a02ce48176ae9a580
```

```
$ docker container logs -f ping
```

```
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=402 ttl=60 time=1.881 ms
64 bytes from 8.8.8.8: seq=403 ttl=60 time=1.939 ms
64 bytes from 8.8.8.8: seq=404 ttl=60 time=1.987 ms
64 bytes from 8.8.8.8: seq=405 ttl=60 time=1.821 ms
64 bytes from 8.8.8.8: seq=406 ttl=60 time=1.857 ms
64 bytes from 8.8.8.8: seq=407 ttl=60 time=2.482 ms
...
...
```

Les commandes de base : exec

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

Les commandes de base : exec

```
# Lancement du process "sleep 10000" dans un container basé sur alpine
$ docker container run -d --name debug alpine:3.6 sleep 10000
```

```
# Exécution d'un shell dans le container debug
```

```
$ docker container exec -ti debug sh
```

Spécification de la commande à lancer

```
/ # ps aux
PID  USER      TIME      COMMAND
 1 root      0:00 sleep 10000
 5 root      0:00 sh
 9 root      0:00 ps aux
/ #
```

Exercice

Les commandes de base : stop

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

Les commandes de base : stop

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

```
6df4108911d3
```

```
$ docker container stop $(docker container ls -q)
```

```
5dbae351233f
```

```
6f6cac745aa1
```

```
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

```
$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

5dbae351233f	nginx	"nginx -g 'daemon ..."	32 minutes ago	Exited (0) 21 seconds ago		goofy_mestorf
--------------	-------	------------------------	----------------	---------------------------	--	---------------

6df4108911d3	alpine	"ping 8.8.8.8"	33 minutes ago	Exited (137) 57 seconds ago		determined_babbage
--------------	--------	----------------	----------------	-----------------------------	--	--------------------

6f6cac745aa1	nginx	"nginx -g 'daemon ..."	33 minutes ago	Exited (0) 21 seconds ago		priceless_turing
--------------	-------	------------------------	----------------	---------------------------	--	------------------

```
...
```

Les commandes de base: rm

- Supprime définitivement un ou plusieurs container
 - \$ docker container rm ID
 - \$ docker container rm \$(docker container ls -aq)
 - \$ docker container rm NAME
- Un container doit être stoppé avant d'être supprimé
- Option -f si le container n'est pas stoppé

Les commandes de base: rm

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

```
6df4108911d3
```

```
$ docker container rm -f $(docker container ls -aq)
```

```
5dbae351233f
```

```
6f6cac745aa1
```

```
47016eee384e
```

```
8653e0b2dd45
```

```
16c68c2264c1
```

```
3c1c4b0b474c
```

```
$ docker container ls -a
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

Exercice

Démo

Les commandes de base

Des alias utiles

```
# Liste les containers actifs
$ alias dls='docker container ls'

# Liste les containers actifs et ceux arrêtés
$ alias dlsa='docker container ls -a'

# Stoppe tous les containers
$ alias dstopall='docker container stop $(docker container ls -aq)'

# Supprime tous les containers
$ alias drmall='docker container rm $(docker container ls -aq)'

# Lance un shell interactif (bash ou sh) dans un container
$ dshell (){
    (docker container exec -ti $1 bash) || (docker container exec -ti $1 sh);
}
```

Exercice

Les images Docker

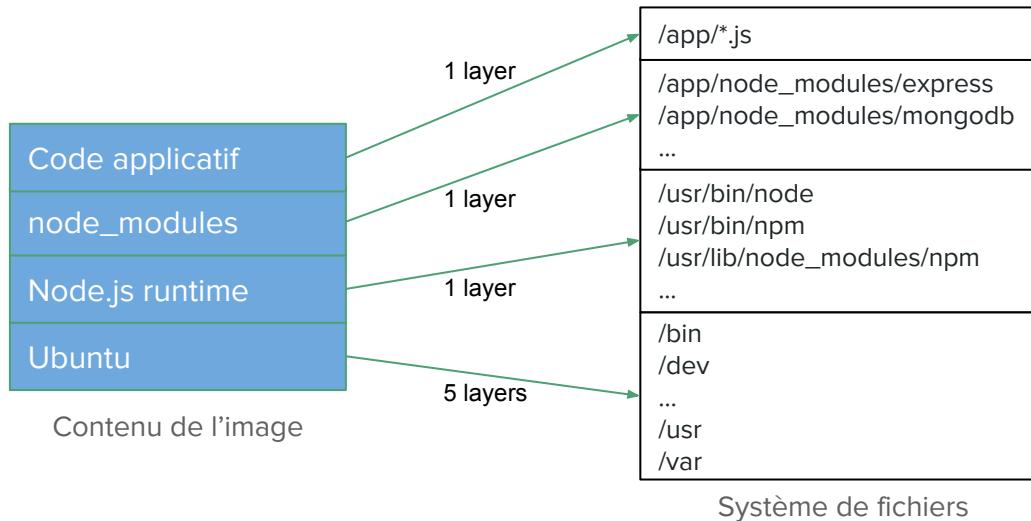
Sommaire

- Définition
- Union File-System
- Copy-On-Write
- Méthodes pour la création d'images
- Dockerfile
- Exemples
- Contexte de build
- Multi-stage build
- Cache
- Les commandes de base

Définition

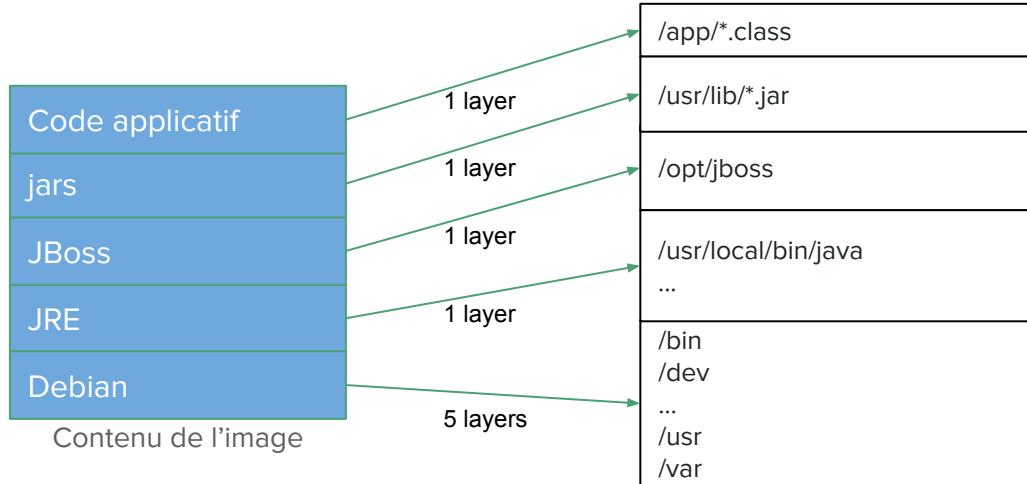
- Un template pour instancier des containers
- Système de fichiers d'un container
- Composée d'une ou de plusieurs layers en lecture
- Chaque layer contient un système de fichiers et des méta data
- Les layers sont partagées entre les images

Définition : exemple pour une application Node.js



Chaque layer contribue au système de fichiers global de l'image

Définition : exemple pour une application Java

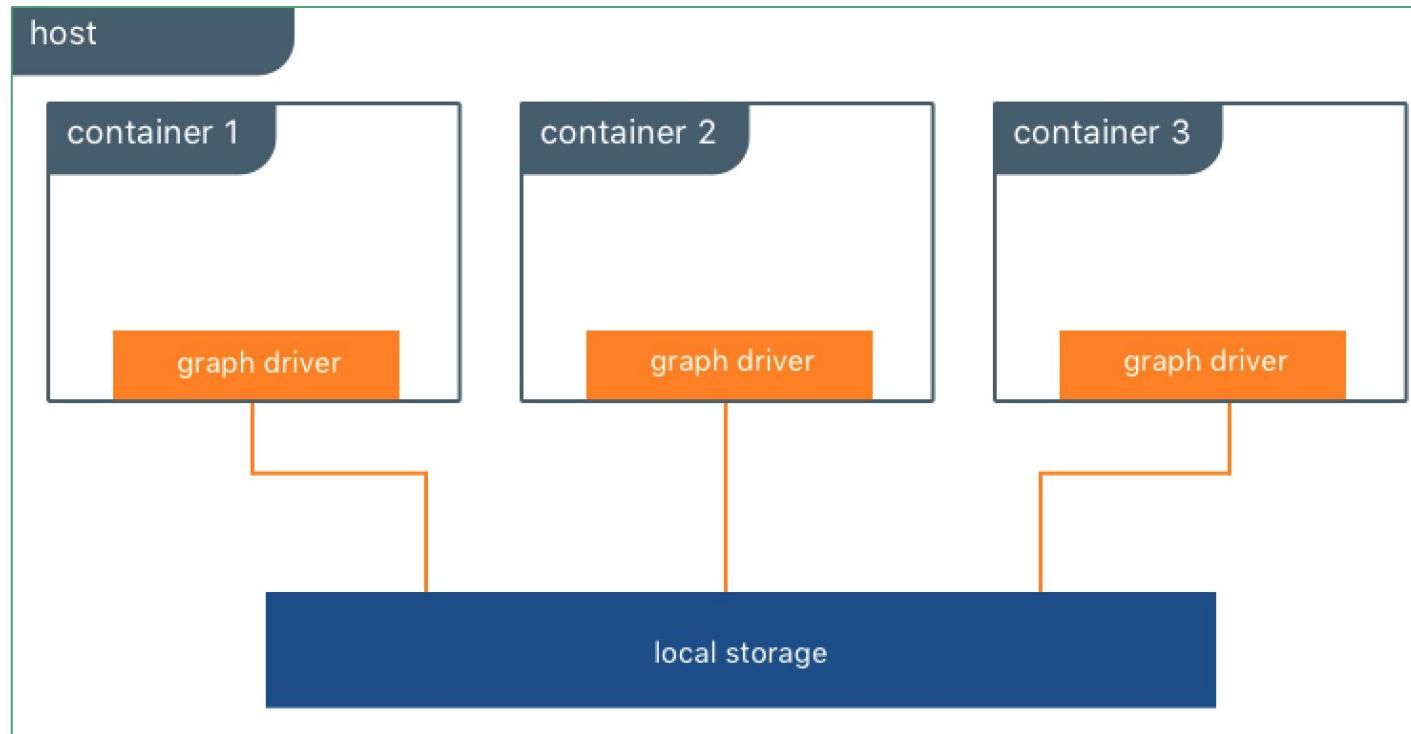


Chaque layer contribue au système de fichiers global de l'image

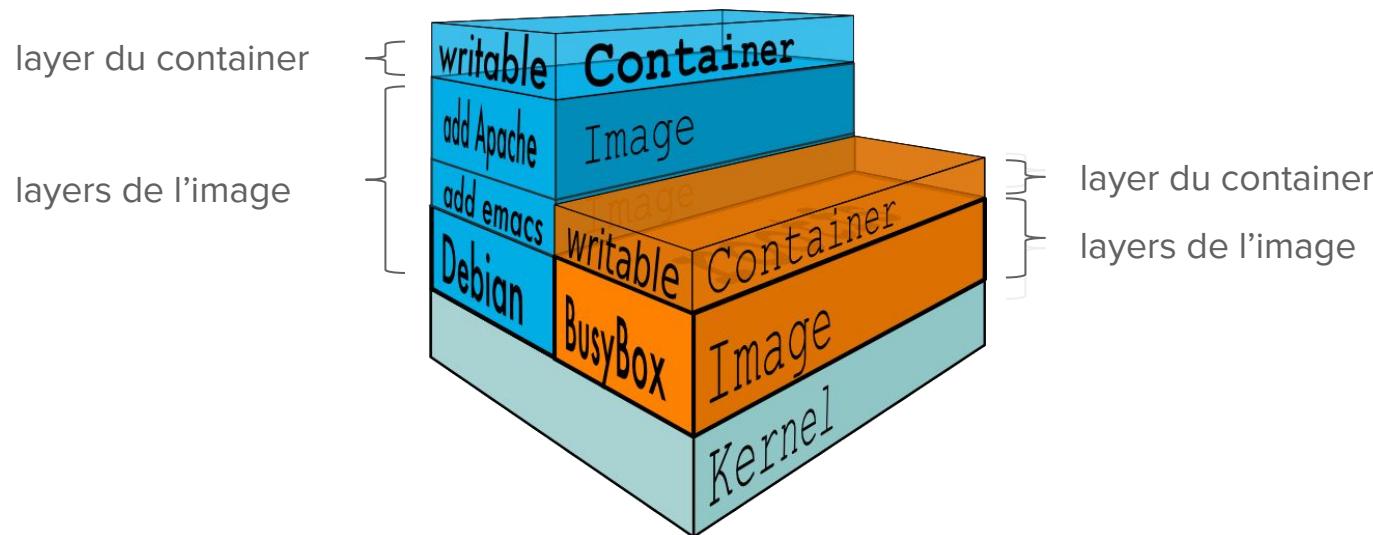
Union filesystem

- Une image est représentée comme un graph de layers
- Utilisation d'un storage driver pour
 - unifier l'ensemble des layer en un seul filesystem
 - ajouter une layer en lecture-écriture liée au container
- Différents drivers disponible en fonction du cas d'usage
 - aufs, devicemapper, btrfs, overlay / overlay2 / zfs
 - <https://docs.docker.com/storage/storagedriver/select-storage-driver>
- Layers stockées dans /var/lib/docker par défaut

Union filesystem



Copy-On-Write



Quand un fichier est modifié dans un container, le fichier original est copié dans la layer du container depuis une layer sous-jacente

Démo

Images et layers

Création d'une image : commit d'un container

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

Création d'une image : commit d'un container

```
$ docker container run -ti --name ping ubuntu:18.04
root@fc663b785b07:/# ping
bash: ping: command not found

root@fc663b785b07:/# apt-get update && apt-get install -y iputils-ping
...
root@fc663b785b07:/# ping -c3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=37 time=20.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=24.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=37 time=20.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 20.289/21.744/24.278/1.806 ms

$
```

CTRL-P + CTRL-Q ← Sort du terminal sans tuer le process (bash) qui tourne dans le container

Création d'une image : commit d'un container

```
$ docker container commit ping myping
```

nom (ou ID) du
container

nom de la
nouvelle image

```
# Lancement d'un container basé sur l'image myping
$ docker container run myping ping -c3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=27.2 ms
...
...
```

Création d'une image : utilisation d'un Dockerfile

- Fichier texte
- Série d'instructions pour construire le système de fichier
- Approche recommandée

Exercice

Dockerfile

- Permet la création d'image personnalisées
- Flow standard
 - utilisation d'une image de base
 - ajout des dépendances
 - ajout et compilation du code applicatif
- Création de l'image
 - *docker image build [OPTIONS] DOCKERFILE_PATH*

```
# Exemple
$ docker image build -t myapp:1.0 .
```

Dockerfile : exemple pour une application Node.js

```
# Image de base
FROM node:8.11.1-alpine

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

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

# Copie du code applicatif
COPY . /app/

# Exposition du port HTTP
EXPOSE 80

# Positionnement du répertoire de travail
WORKDIR /app

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

Dockerfile : les instructions principales

FROM	Image de base
ENV	Définition de variables d'environnement
RUN	Exécution d'une commande, construction du filesystem de l'image
COPY / ADD	Copie de ressources depuis la machine local dans le filesystem de l'image
EXPOSE	Expose un port de l'application
HEALTHCHECK	Vérifie l'état de santé de l'application
VOLUME	Définition d'un volume pour la gestion des données
WORKDIR	Définition du répertoire de travail
USER	Utilisateur auquel appartient le processus du container
ENTRYPOINT	Définie la commande exécutée au lancement du container
CMD	

Démo

Dockerfile : FROM

- Définition de l'image de base
 - Différents choix
 - image d'un OS (Alpine, CentOS, Ubuntu, ...)
 - serveur applicatif
 - environnement d'exécution
 - base de données
 - ...
 - **scratch** : une image particulière
 - pour construire une image de base
 - pour construire une image minimale
- 
- contiennent l'OS de base

Dockerfile : ENV

- Définition de variables d'environnement
- Valeur utilisée dans les instructions suivantes du build
- Substitution possible via \$

```
FROM alpine
ENV path /app
WORKDIR ${path}    # WORKDIR /app
COPY . $path       # COPY . /app
```

Dockerfile : COPY et ADD

- Copie des fichiers et répertoires dans le système de fichiers de l'image
- Engendre la création d'une nouvelle layer
- Option *--chown* pour la mise spécification des droits
- ADD
 - permet de spécifier une URL pour la source
 - unpack un fichier tar.gz
- Privilégier l'utilisation de COPY

Dockerfile : RUN

- Exécute une commande dans une nouvelle layer
- 2 formats
 - Shell : lancé dans un shell avec “/bin/sh -c” par défaut

```
RUN apt-get update -y && apt-get install
```

- Exec : non lancé dans un shell

```
RUN ["/bin/bash", "-c", "echo hello"]
```

Dockerfile : EXPOSE

- Information des ports utilisés par l'application
- Peut être modifié au lancement du container
 - option -p CONTAINER_PORT
 - option -p HOST_PORT:CONTAINER_PORT
 - option -P

```
...  
EXPOSE 27017  
CMD [ "mongod" ]
```

Extrait du Dockerfile de MongoDB

```
...  
EXPOSE 80  
CMD [ "nginx", "-g", "daemon off;" ]
```

Extrait du Dockerfile de Nginx

Dockerfile : VOLUME

- Création d'un point de montage sur la machine hôte
- Découple les données du cycle de vie d'un container
- Gestion des données en dehors de l'union file-system
- Initialisation du volume avec les données présentes dans l'image

```
...
RUN mkdir -p /data/db /data/configdb \
    && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 27017
CMD ["mongod"]
```

Extrait du Dockerfile de MongoDB

Bonne pratique !

Dockerfile : USER

- username ou uid / gid utilisés pour lancer le process du container
- Utilisé par les instructions RUN, CMD, ENTRYPOINT qui suivent
- L'intérêt : ne pas lancer le processus du container en root

Par défaut : root dans le container \Leftrightarrow root sur la machine hôte

Bonne pratique !

Dockerfile : HEALTHCHECK

- Vérification de l'état de santé du container
- Ex : vérification du endpoint /health toutes les 5 secondes

```
FROM node:8.11-alpine
RUN apk update && apk add curl
HEALTHCHECK --interval=5s --timeout=3s --retries=3 CMD curl -f http://localhost:8000/health || exit 1
COPY package.json /app/package.json
WORKDIR /app
RUN npm install
COPY . /app
CMD [ "npm", "start" ]
```

Dockerfile : ENTRYPOINT / CMD

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

```
ENTRYPOINT ["curl"]
CMD ["--help"]
```

Dockerfile : ENTRYPOINT / CMD

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

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



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



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

Dockerfile : ENTRYPOINT / CMD

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

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



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



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

```

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

Dockerfile : ENTRYPOINT / CMD

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry exec_cmd p1_cmd	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry p1_cmd p2_cmd	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

<https://docs.docker.com/engine/reference/builder/#/understand-how-cmd-and-entrypoint-interact>

Exercice

Exemples : application Java (1/2)

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

Main.java

```
FROM openjdk:7

COPY . /usr/src/myapp

WORKDIR /usr/src/myapp

RUN javac Main.java

CMD [ "java", "Main" ]
```

Dockerfile

Exemples : application Java (2/2)

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

Contenu du répertoire courant envoyé au daemon

Une layer est créée pour chaque instruction du Dockerfile



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

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

Exemples : application Python (1/2)

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

barcode.py

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

Dockerfile

Exemples : application Python (2/2)

```
$ docker image build -t barcode .
```

Sending build context to Docker daemon 3.072 kB

Step 1 : FROM python:3

---> 175259937daf

Step 2 : ADD barcode.py /

---> 4bb590da4ba1

Removing intermediate container 8bcc214fb2dd

Step 3 : RUN pip install pystrich

---> Running in ed48c05e8bcd

...

---> 25e8a17abb50

Removing intermediate container ed48c05e8bcd

Step 4 : CMD python /barcode.py

---> Running in afc4d6dccc7d

---> 3432b53e2391

Removing intermediate container afc4d6dccc7d

Successfully built 3432b53e2391



```
$ docker run barcode
```

Sending bu

Step 1 : FR

---> 17525

Step 2 : AD

---> 4bb59

oving

Step 3 : R

---> Runni

XX XX

XX XX XX XXXX XX XX XX XXXX

XXXX XX XX XXXX XXXX XXXXXX XX

XX XXXX XXXX XXXX XXXX XX XX

XX XX XXXX XXXX XXXX XXXXXX XXXX

XXXX XX XXXX XXXX XXXX XX XX

XXXX XX XXXXXX XXXX XXXXXX XXXX

XX XX XX XXXXXX XXXX XX XX XX

XX XX XXXX XX XXXX XXXXXX XX

XXXX XX XX XXXX XXXX XXXXXX XX

XX XX XXXXXX XX XX XXXX XX XX

XX XX XXXXXX XXXX XXXXXX XXXX

XXXX XX XX XXXX XXXX XXXXXX XX

XX XX XXXXXX XXXX XXXXXX XXXX

XX XX XXXX XX XX XXXXXXXXXX XX

XX XXXXXXXX XX XXXXXX XX XX XX

XX XX XXXX XX XXXX XX XX XX

XXXX XX XXXX XXXX XXXX XX XX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Exemples : application Node.js (1/2)

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

index.js

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

package.json

```
# Image de base
FROM node:8.11-alpine

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

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

# Copie du code applicatif
COPY . /app/

WORKDIR /app

# Port d'écoute
EXPOSE 80

# Commande à lancer
CMD ["npm", "start"]
```

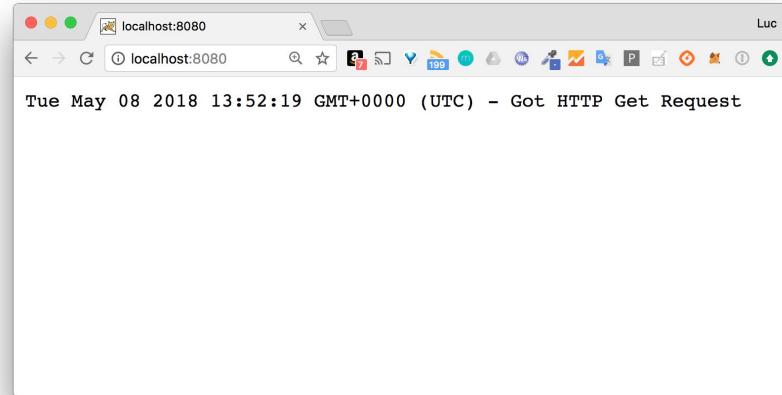
Dockerfile

Exemples : application Node.js (2/2)

```
$ docker image build -t app:1.0 .
Sending build context to Docker daemon 1.698MB
Step 1/7 : FROM node:8.11-alpine
8.11-alpine: Pulling from library/node
605ce1bd3f31: Pull complete
0511902e1bcd: Pull complete
343e34c41f87: Pull complete
Digest:sha256:d0febabb04c15f58a28888618cf5c3f1d475261e257027
41622f375d0a82e050d
Status: Downloaded newer image for node:8.11-alpine
--> e707e7ad7186
Step 2/7 : COPY package.json /app/package.json
--> a88905253359
Step 3/7 : RUN cd /app && npm install
...
Step 4/7 : COPY . /app/
--> 26d9e63ba801
Step 5/7 : WORKDIR /app
Removing intermediate container dbd5c6938baa
--> 4b9c354ba2c0
Step 6/7 : EXPOSE 80
--> Running in 7b15b1b0d37f
Removing intermediate container 7b15b1b0d37f
--> c58f7521de09
Step 7/7 : CMD ["npm", "start"]
--> Running in b7030d04c0b7
Removing intermediate container b7030d04c0b7
--> 63304727df7d
Successfully built 63304727df7d
Successfully tagged app:1.0
```



```
$ docker container run -p 8080:80 app:1.0
```



Multi-stages build

- Build en plusieurs fois
- Plusieurs instructions FROM dans le Dockerfile
- Un cas d'usage courant
 - 1ère étape
 - utilisation d'une image de base avec le tooling pour la compilation
 - compilation des resources
 - 2ème étape
 - utilisation d'une image light (ex: alpine)
 - copie des artefacts générés lors de la première étape

Multi-stages build

```
FROM golang:1.9.0-alpine
WORKDIR /go/src/github.com/lucj/who
COPY http.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o http .
CMD [ "./http" ]
```

Dockerfile “traditionnel”

```
# Création de l'image
$ docker image build -t who:1.0 .
```

```
# Liste des images
$ docker image ls who
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
who	1.0	1f74e07fd4f7	About a minute ago	276MB

276MB pour un simple server web en go !

Multi-stages build

```
FROM golang:1.9.0-alpine as build
WORKDIR /go/src/github.com/lucj/who
COPY http.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o http .

FROM scratch
COPY --from=build /go/src/github.com/lucj/who/http .
CMD [ "./http" ]
```

Dockerfile avec le multi-stages build

```
# Création de l'image
$ docker image build -t who:2.0 .
```

```
# Liste des images
$ docker image ls who
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
who	2.0	21f530664efb	7 minutes ago	6.09MB
who	1.0	1f74e07fd4f7	About a minute ago	276MB

~ 2% de la taille de départ !

Exercice

Cache

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

Cache: exemple sur l'application Node.js

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

```
FROM node:6.10.3
MAINTAINER Luc Juggery <luc.juggery@gmail.com>
ENV LAST_UPDATED 20161209T075500
COPY package.json /app/package.json
RUN cd /app && npm install
COPY . /app/
WORKDIR /app
EXPOSE 80
CMD ["npm", "start"]
```



```
$ docker image build -t app:1.0 .
Sending build context to Docker daemon 1.698MB
Step 1/7 : FROM node:8.11-alpine
--> e707e7ad7186
Step 2/7 : COPY package.json /app/package.json
--> Using cache
--> a88905253359
Step 3/7 : RUN cd /app && npm install
--> Using cache
--> 2f435a0adeca
Step 4/7 : COPY . /app/
--> Using cache
--> 26d9e63ba801
Step 5/7 : WORKDIR /app
--> Using cache
--> 4b9c354ba2c0
Step 6/7 : EXPOSE 80
--> Using cache
--> c58f7521de09
Step 7/7 : CMD ["npm", "start"]
--> Using cache
--> 63304727df7d
Successfully built 63304727df7d
Successfully tagged app:1.0
```

Build context

- Répertoire utilisé par le Docker daemon lors du build
- Contenu package dans un .tar et envoyé au daemon

```
$ docker image build -t app:v2.0 .
Sending build context to Docker daemon 4.096kB
```

- Gestion avec le fichier *.dockerignore*

- rapidité du build
- exclusion des données sensibles

```
.git  
node_modules
```

exemple: *.dockerignore* pour une application Node.js

Les commandes de base

```
$ docker image --help
```

```
Usage:docker image COMMAND
```

```
Manage images
```

```
Options:
```

```
    --help     Print usage
```

```
Commands:
```

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

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

Les commandes de base : build

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

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

Les commandes de base : pull

- Download une image depuis un registry (Docker Hub par défaut)
- Chaque layer de l'image est downloadée
- Image automatiquement downloadée lors de la création d'un container
- Format de nommage: USER/IMAGE:VERSION

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

```
$ docker image pull alpine
```

```
Using default tag: latest
```

```
latest: Pulling from library/alpine
```

```
0a8490d0dfd3: Pull complete
```

```
Digest: sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
```

```
Status: Downloaded newer image for alpine:latest
```

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	88e169ea8f46	2 weeks ago	3.98 MB

Les commandes de base : push

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

Les commandes de base : inspect

```
$ docker image inspect alpine
[
  {
    "Id": "sha256:88e169ea8f46ff0d0df784b1b254a15ecfaf045aee1856dca1ec242fd231ddd",
    "RepoTags": [ "alpine:latest" ],
    ...
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 3983615,
    "VirtualSize": 3983615,
    "GraphDriver": {
      "Name": "aufs",
      "Data": null
    },
    "RootFS": {
      "Type": "layers",
      "Layers": [
        "sha256:60ab55d3379d47c1ba6b6225d59d10e1f52096ee9d5c816e42c635ccc57a5a2b"
      ]
    }
  }
]

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

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

Les commandes de base : history

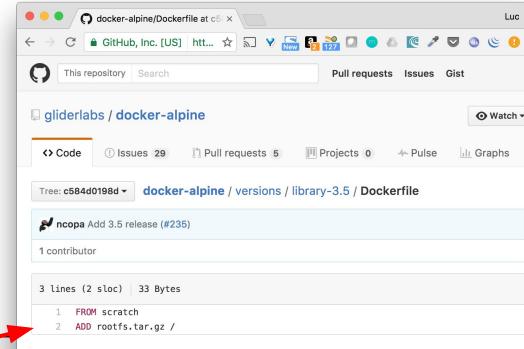
- Montre comment l'image a été créée
- Une entrée d'historique par layer

```
$ docker history alpine
IMAGE      CREATED      CREATED BY
baa5d63471ea  4 days ago  /bin/sh -c #(nop) ADD file:7afbc23fda8b0b3872
SIZE      COMMENT
4.803 MB
```

[Alpine:latest Dockerfile](#)

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

[Ubuntu:latest Dockerfile](#)



Les commandes de base : ls

```
$ docker image ls
```

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

Images créées ou téléchargées

```
$ docker image ls -a
```

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

Liste les images temporaires créées lors du build

```
$ docker image ls node
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
node	6.9.4	06b984afb149	9 days ago	655 MB

Filtre des images par nom

```
$ docker image ls --filter dangling=true
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	d02c4d04476c	11 hours ago	653.2 MB

Les images “dangling” ne sont plus référencées

```
$ docker image prune
```

Total reclaimed space: 653.2 MB

Suppression des images “dangling”

Les commandes de base : save / load

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	88e169ea8f46	3 weeks ago	3.98 MB

```
$ docker save -o alpine.tar alpine
```

```
$ ls
```

```
alpine.tar
```

```
$ docker image rm alpine
```

```
Untagged: alpine:latest
```

```
Untagged: alpine@sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
```

```
Deleted: sha256:88e169ea8f46ff0d0df784b1b254a15ecfaf045aee1856dca1ec242fdd231ddd
```

```
Deleted: sha256:60ab55d3379d47c1ba6b6225d59d10e1f52096ee9d5c816e42c635ccc57a5a2b
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	88e169ea8f46	3 weeks ago	3.98 MB

```
$ docker load < alpine.tar
```

```
60ab55d3379d: Loading layer [=====] 4.226 MB/4.226 MB
```

```
Loaded image: alpine:latest
```

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	88e169ea8f46	3 weeks ago	3.98 MB

Les commandes de base : rm

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

```
$ docker image ls
```

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

```
$ docker image rm ubuntu
```

```
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:71cd81252a3563a03ad8daee81047b62ab5d892ebbfb71cf53415f29c130950
Deleted: sha256:f49eec89601e8484026a8ed97be00f14db75339925fad17b440976cffcbfb88a
Deleted: sha256:3e2d12b23fafef176cf40429fb25be6572212807f27455b8e3e114c397324446f
Deleted: sha256:88a37465e211da3c72acbd999b158ee31c6c7239131f6308f000dcf52d622a7b
Deleted: sha256:99be185bee70b39c64096b8d39b96153d28d3caa7764961a9285ad4d189cd536
Deleted: sha256:fc7e2c65ec42780443f87ae7d9621cd6fcdb371e2127dd461b449d9e50b7ab7b
Deleted: sha256:4f03495a4d7de505ccb8b8e4cf0a8ac201491e1f67ae54b65584e0012aaab9c
```

```
$ docker image ls -q
```

```
d448eac1cfdb
88e169ea8f46
```

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

Démo

Les commandes de base avec PWD

Exercices

Registry



Sommaire

- Utilisation
- Différents providers
- Docker Hub
- Docker Registry
- Docker Trusted Registry

Utilisation

- Librairie d'images
- Facilite la distribution entre environnements
- Download d'une image: `$ docker image pull IMAGE`
- Upload d'une image: `$ docker image push IMAGE`
- Docker Hub (<https://hub.docker.com>) utilisé par défaut

Utilisation: exemple

```
# Récupération de l'image alpine
$ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine } L'image alpine est composée d'une seule layer
c0cb142e4345: Pull complete
Digest:sha256:ca7b185775966003d38ccbd9bba822fb570766e4bb69292ac23490f36f8a742e
Status: Downloaded newer image for alpine:latest

# Envoi de l'image lucj/randomcity:v2.0
$ docker image push lucj/randomcity:v2.0
f51c5c3da2f3: Pushed
c50a9d715b70: Layer already exists
a920baf03ef8: Layer already exists
8f01a53880b9: Layer already exists } Chaque layer de l'image est uploadée dans le registry
0.1: digest: sha256:48f715baf24b4e8920bb2ee40afb6024fe0a8a378a9b5edea5cb08caaee2d753b size: 1790
```

Différents providers : les registries Docker

- Docker Hub - <https://hub.docker.com/>
 - Registry officiel hosté par Docker
- Docker Registry
 - Solution Open-Source
- Docker Trusted Registry
 - Solution commerciale
 - Disponible avec Docker-EE standard / avancé

Différents providers : autres registries de l'écosystème

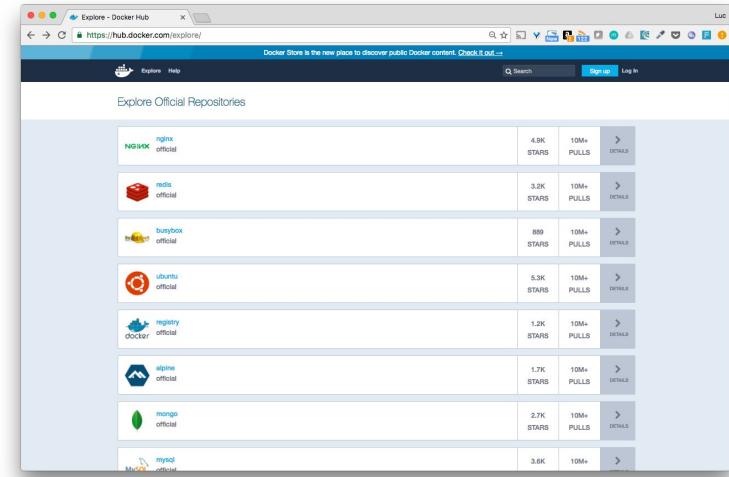
- Amazon EC2 container registry
- Google Container Registry
- Quay.io (CoreOS)
- GitLab container registry
- ...

Docker Hub : overview

- <https://hub.docker.com>
- Nombreuses images officielles - <https://hub.docker.com/explore>
- Repository public ou privé
- Intégration avec Github et BitBucket
- Intégration dans un pipeline d'intégration continu / déploiement continu

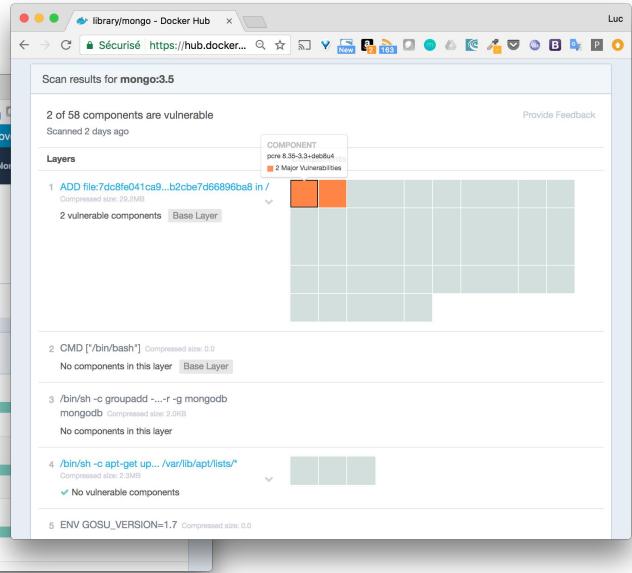
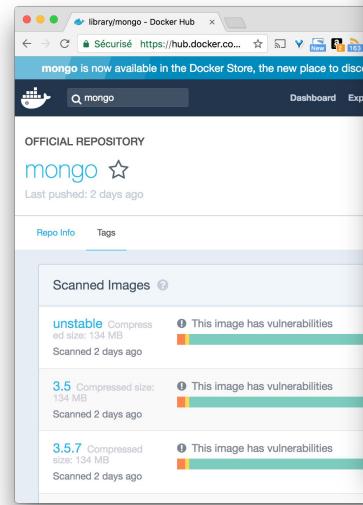
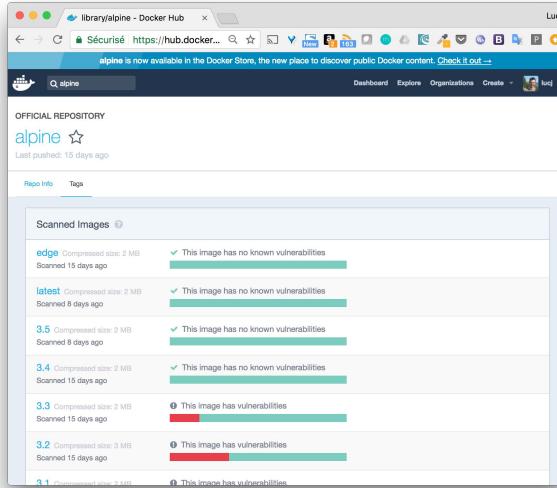
Docker Hub : les images officielles

- Nombreuses images disponibles
 - distributions Linux (ex: ubuntu, centos, alpine)
 - serveurs HTTP (ex: nginx, apache)
 - bases de données (ex: mongo, redis, mysql)
 - runtime (ex: nodejs, java, clojure, php)
 - ...
- Images “fiables”



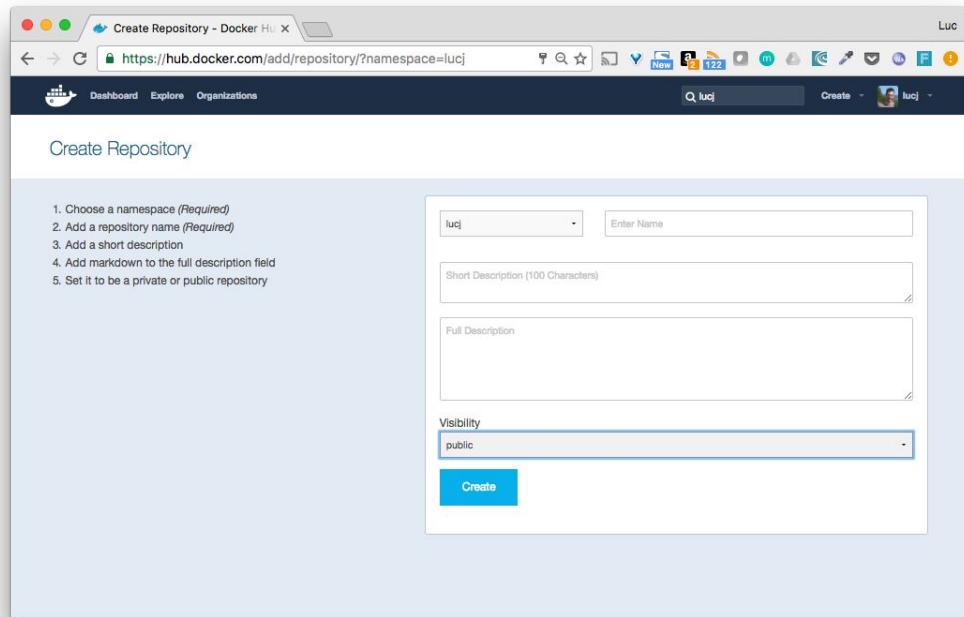
Docker Hub : analyse des vulnérabilités

- Analyse effectuée au niveau binaire
- Basée sur une base de données de CVE
- Scan automatique si nouvelles vulnérabilités



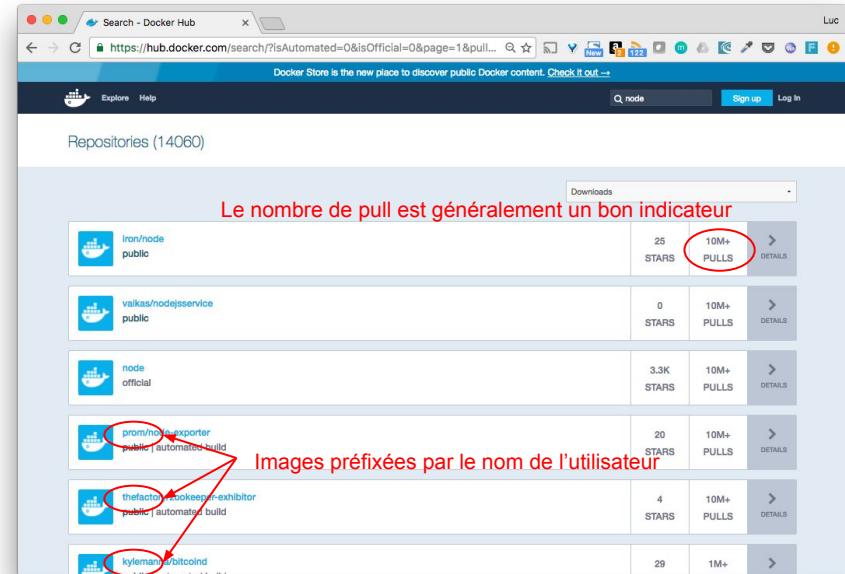
Docker Hub : repository

- Contient l'ensemble des versions d'une image
- Visibilité publique ou privée
- Tag ⇒ *user/repository:version*
- Login nécessaire pour pusher



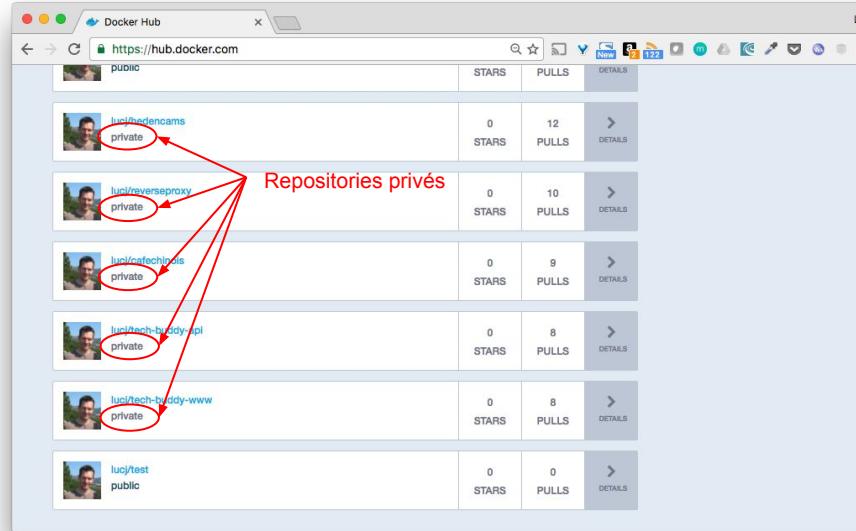
Docker Hub : les images publiques

- Utilisable librement sans login
- A utiliser avec précaution
 - peuvent contenir des failles de sécurité
 - mises à jour non assurées



Docker Hub : les images privées

- Peuvent être utilisées par les utilisateurs autorisés



Docker Hub : les images privées

- Login nécessaire pour le download d'images privées

```
$ docker pull lucj/labs:1.2
```

```
Error response from daemon: repository lucj/labs not found: does not exist or no pull access
```

```
$ docker login
```

```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
```

```
Username: lucj
```

```
Password:
```

```
Login Succeeded
```

```
$ docker pull lucj/labs:1.2
```

```
1.2: Pulling from lucj/labs
```

```
e12c678537ae: Pull complete
```

```
333547110842: Pull complete
```

```
4df1e44d2a7a: Pull complete
```

```
3852ed97f353: Pull complete
```

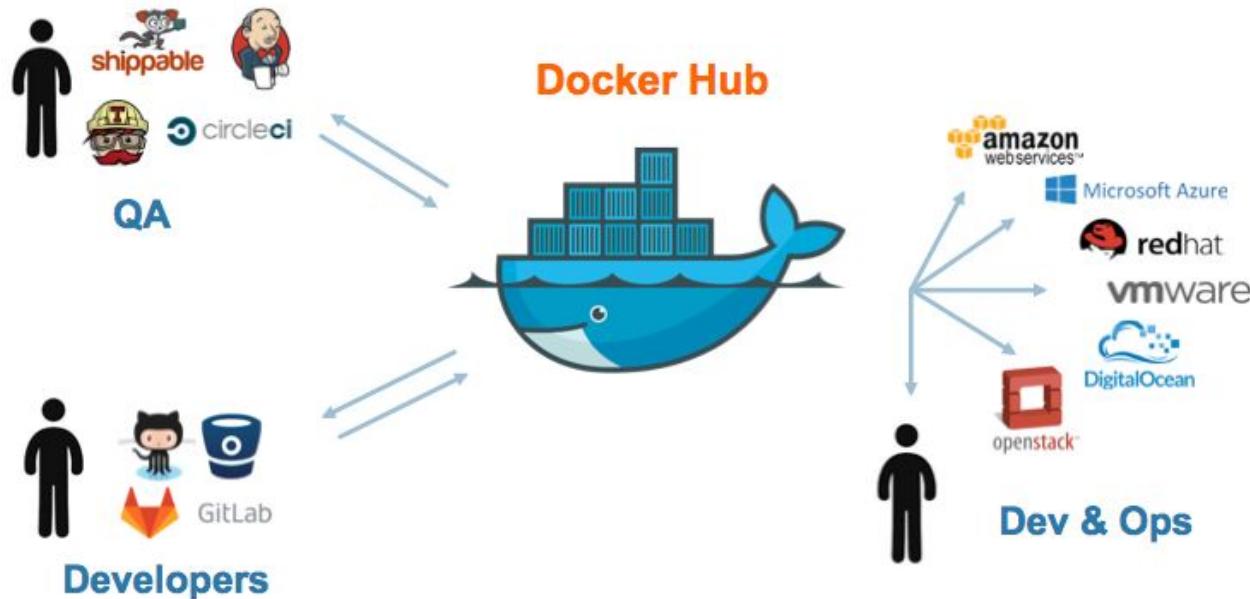
```
Digest: sha256:a1b18655c44407a8f366b0c01615920778c7f8e4c3d05070004edb7d3e16cf4c
```

```
Status: Downloaded newer image for lucj/labs:1.2
```

Docker Hub : CI/CD

- Au centre d'un pipeline de build automatique
- Création d'image déclenchée par le push de code ou HTTP Post
- Intégration avec des outils de versionning de code source
 - GitHub, Gitlab, BitBucket, ...
- Webhook pour l'intégration avec des outils tiers

Docker Hub : CI/CD



Exercices

Docker Open Source Registry

- https://hub.docker.com/_/registry/
- API HTTP Rest pour le management des images
- A la base du Docker Hub
- Configurable via un fichier de configuration
 - sécurisation avec TLS
 - méthode d'authentification
 - sélection du backend de stockage
 - <https://docs.docker.com/registry/configuration/#list-of-configuration-options>

Docker Open Source Registry

- Connexion depuis un Docker daemon
 - option `--insecure-registry` si communication non sécurisée
 - localhost exception
- Image préfixée par l'URL du registry
 - `HOST:PORT/NAME:PORT[:VERSION]`

Docker Open Source Registry

```
# Lancement du registry dans un container
$ docker container run -d -p 5000:5000 registry:2.6.1

// Liste des images présentes dans le registry
$ curl localhost:5000/v2/_catalog
{"repositories":[]}

// Download de l'image nginx depuis le Docker Hub
$ docker image pull nginx

// Tag de l'image nginx selon le format URL:PORT/NAME:VERSION
$ docker image tag nginx localhost:5000/nginx

// Upload de l'image dans le registry
$ docker image push localhost:5000/nginx

// Liste des images présentes dans le registry
$ curl localhost:5000/v2/_catalog
{"repositories":["nginx"]}
```

Exercices

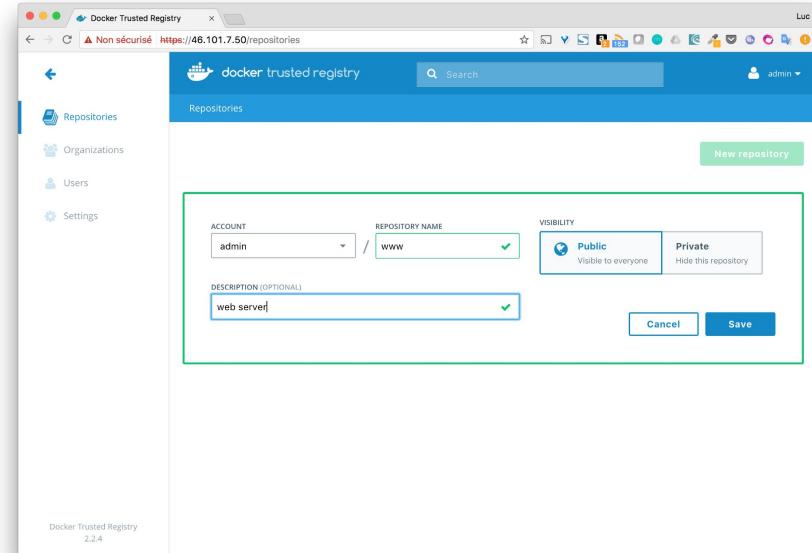
Docker Trusted Registry (DTR)

Composant de l'offre Docker Enterprise Edition

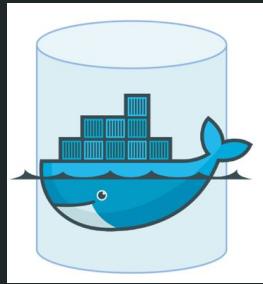


Docker Trusted Registry (DTR)

- Support commercial
- Installation on-prem
- Utilisateurs et teams
- Intégration LDAP / AD
- Scanning des images
- Sélection du backend de stockage
 - FS, S3, Azure, Google Cloud Storage, Swift



Stockage

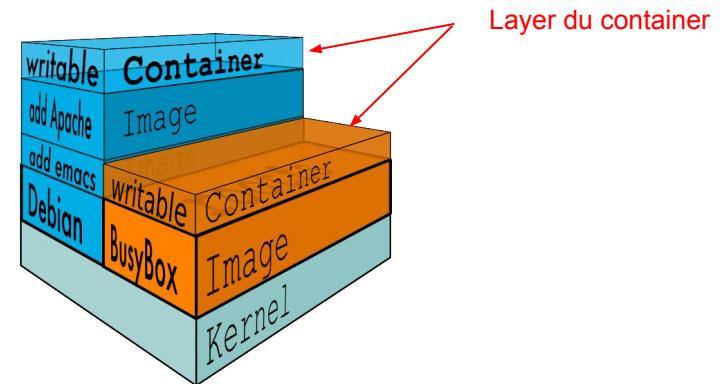


Sommaire

- Container et persistance des données
- Volume
- Bind-mount
- Les commandes de base
- Types de stockage
- Les drivers de stockage
- REX-Ray

Container et persistance des données

- Un container ne permet pas la persistance des données
- Changements dans la layer du container
 - read-write layer au dessus de l'image
 - créée et supprimée avec le container



- Les données doivent être gérées en dehors de l'union filesystem

Container et persistance des données

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

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

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

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

MYFILE est présent sur le filesystem de l'hôte
\$ find /var/lib/docker -name MYFILE
/var/lib/docker/overlay2/b80...401e/diff/MYFILE
/var/lib/docker/overlay2/b80...401e/merged/MYFILE

MYFILE est toujours présent
\$ find /var/lib/docker -name MYFILE
/var/lib/docker/overlay2/b80...401e/diff/MYFILE

MYFILE a été supprimé
\$ find /var/lib/docker -name MYFILE

Volume

- Répertoires / fichiers existant en dehors de l'union filesystem
- Utilisé pour découpler les données du cycle de vie d'un container
- Défini de différentes façons
 - instruction VOLUME dans le Dockerfile
 - options -v / --mount à la création d'un container
- Exemple de cas d'usage: base de données, logs, ...

Volume : définition dans le Dockerfile

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

```
$ docker container inspect -f '{{json .Mounts }}' mongo | python -m json.tool
```

```
[  
  {  
    "Name": "65a7aad62fb00b70b39901cd39cdea5064b7b6a310c5255cb7657dbeec66387b",  
    "Source":  
    "/mnt/sda1/var/lib/docker/volumes/65a7aad62fb00b70b39901cd39cdea5064b7b6a310c5255cb7657dbeec66387b/_data",  
    "Destination": "/data/configdb",  
    "Driver": "local",  
    ....  
  },  
  {  
    "Name": "87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13",  
    "Source":  
    "/mnt/sda1/var/lib/docker/volumes/87defba35fd0961f4c91736b1a6e533bac45101e4ae73b309dda080d10203c13/_data",  
    "Destination": "/data/db",  
    "Driver": "local",  
    ....  
  }  
]
```

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

Extrait du Dockerfile de mongo:3.2

Volume : définition dans le Dockerfile

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

```
$ docker container rm -f mongo
mongo
```

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



Le contenu du volume est présent
après la suppression du container

Volume : définition à l'exécution

- Option `-v` dans la commande `docker container run`

```
$ docker container run -v /usr/share/nginx/html nginx
```

- Même résultat que la définition dans le Dockerfile
- Crédit d'un volume et copie du contenu du répertoire du container

Volume : définition à l'exécution

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

#/ls /var/lib/docker/volumes/b1c6aa7103a4f1440c34f0cb7889df12bf6fe2760149c05c3a046a933871af65/_data
50x.html      index.html
← Le contenu du répertoire du container est
copié dans le volume

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

Bind-mount

- Répertoire ou fichier de l'hôte monté dans un container
- A la création d'un container avec l'option -v

```
$ docker container run -v HOST_PATH:CONTAINER_PATH ...
```

- Création automatique du folder sur l'hôte ou dans le container
- Cas d'usage
 - en développement: montage du code source dans un container
 - donner accès à la socket unix du daemon Docker
 - \$ docker container run -v /var/run/docker.sock:/var/run/docker.sock ...
 - utile pour écouter les évènements du Docker daemon / Swarm

Bind-mount

- Contenu du répertoire de l'hôte “cache” celui du répertoire du container

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

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

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

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

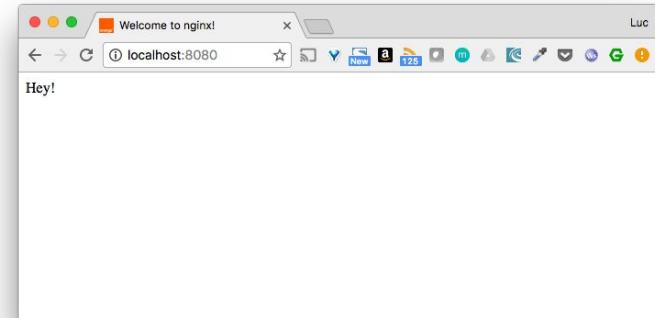
# Le fichier est visible dans le répertoire de l'hôte
$ ls /tmp/myfolder
file_from_host file_from_container
```

Bind-mount

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

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

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

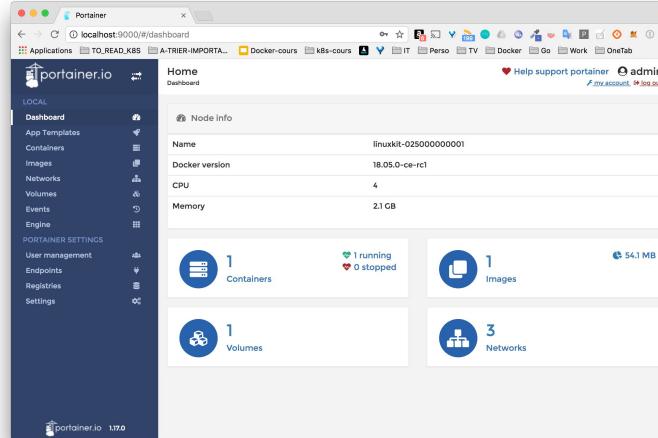


Bind-mount : la socket /var/run/docker.sock (1/3)

- Socket utilisée pour la communication entre le client et le daemon Docker
- Accès au daemon Docker depuis un container

```
# Exemple d'utilisation avec Portainer, une excellente application web de gestion d'hôtes Docker
```

```
$ docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```



Bind-mount : la socket /var/run/docker.sock (2/3)

```
# Lancement d'un container "admin" avec le bind mount de la socket Docker
$ docker container run --name admin -ti -v /var/run/docker.sock:/var/run/docker.sock alpine

# Création d'un container depuis le container admin
/ # curl -XPOST --unix-socket /var/run/docker.sock -d '{"Image":"nginx:1.12.2"}' -H 'Content-Type: application/json' http://localhost/containers/create
{"Id":"6b24af5f293bb0ee60ac432225b4bc4eb7d26c562b656d0ece7facda4268283b","Warnings":null}

# Lancement du container
/ # curl -XPOST --unix-socket /var/run/docker.sock http://localhost/containers/6b24...283b/start
```

Bind-mount : la socket /var/run/docker.sock (3/3)

```
# Lancement d'un container "admin" avec le bind mount de la socket Docker
$ docker container run --name admin -ti -v /var/run/docker.sock:/var/run/docker.sock alpine

# Suivi des évènements
$ curl --unix-socket /var/run/docker.sock http://localhost/events
{"status":"create","id":"9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2","from":"alpine","Type":"container","Action":"create","Actor":{"ID":"9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2","Attributes":{"image":"alpine","name":"vigilant_archimedes"}}, "scope": "local", "time": 1526470673, "timeNano": 1526470673161213763}
 {"status":"attach","id":"9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2","from":"alpine","Type":"container","Action":"attach","Actor":{"ID":"9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2","Attributes":{"image":"alpine","name":"vigilant_archimedes"}}, "scope": "local", "time": 1526470673, "timeNano": 1526470673163989653}
 {"Type": "network", "Action": "connect", "Actor": {"ID": "37602ad3c06b3856146ed5d450159d7e7e468a06e48182469ca86387f2d8ad9f", "Attributes": {"container": "9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2", "name": "bridge", "type": "bridge"}}, "scope": "local", "time": 1526470673, "timeNano": 1526470673183447422}
 {"status": "start", "id": "9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2", "from": "alpine", "Type": "container", "Action": "start", "Actor": {"ID": "9c146ad1ed6d0586c3d9167d0df8e449ece86446afe20c9da05a1526477f1ac2", "Attributes": {"image": "alpine", "name": "vigilant_archimedes"}}, "scope": "local", "time": 1526470673, "timeNano": 1526470673573369481}
 ...
```

Les commandes de base

```
$ docker volume --help
```

```
Usage:docker volume COMMAND
```

```
Manage volumes
```

```
Options:
```

```
  --help    Print usage
```

```
Commands:
```

```
  create      Create a volume
```

```
  inspect     Display detailed information on one or more volumes
```

```
  ls          List volumes
```

```
  prune       Remove all unused volumes
```

```
  rm          Remove one or more volumes
```

Les commandes de base

```
# Création du volume html en utilisant le driver par défaut
$ docker volume create --name html
html

# Liste des volumes existants
$ docker volume ls
DRIVER      VOLUME NAME
local       html

# inspection du volume html
$ docker volume inspect html
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/html/_data",
        "Name": "html",
        "Options": {},
        "Scope": "local"
    }
]
# Le volume est un répertoire vide de la machine hôte
$ ls /var/lib/docker/volumes/html/_data
```

Les commandes de base

```
# Le volume html est monté dans le container  
$ docker container run -d --name www -v html:/usr/share/nginx/html nginx
```

Le contenu du répertoire du container est visible dans le volume

```
$ ls /var/lib/docker/volumes/html/_data  
50x.html    index.html
```

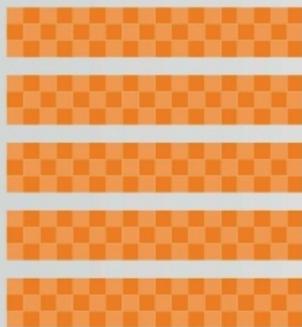
```
$ docker container run -ti alpine sh  
/ / # mount | grep '/tmp'  
/ #  
/ # exit  
  
# Le volume html est monté dans le container  
$ docker container run -ti -v html:/tmp alpine sh  
/ # mount | grep '/tmp'  
/dev/sda2 on /tmp type ext4 (rw,relatime,data=ordered)  
/ # touch test
```

```
$ ls /var/lib/docker/volumes/html/_data  
50x.html    index.html  test
```

Exercices

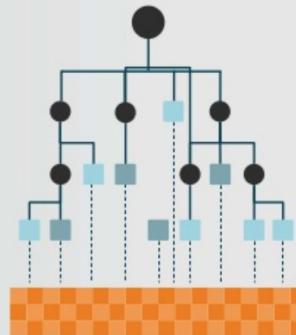
Les types de stockage

What are different storage options ?



BLOCK STORAGE

Physical storage media appears to computers as a series of sequential blocks of a uniform size.



FILE STORAGE

File systems allow users to organize data stored in blocks using hierarchical folders and files.



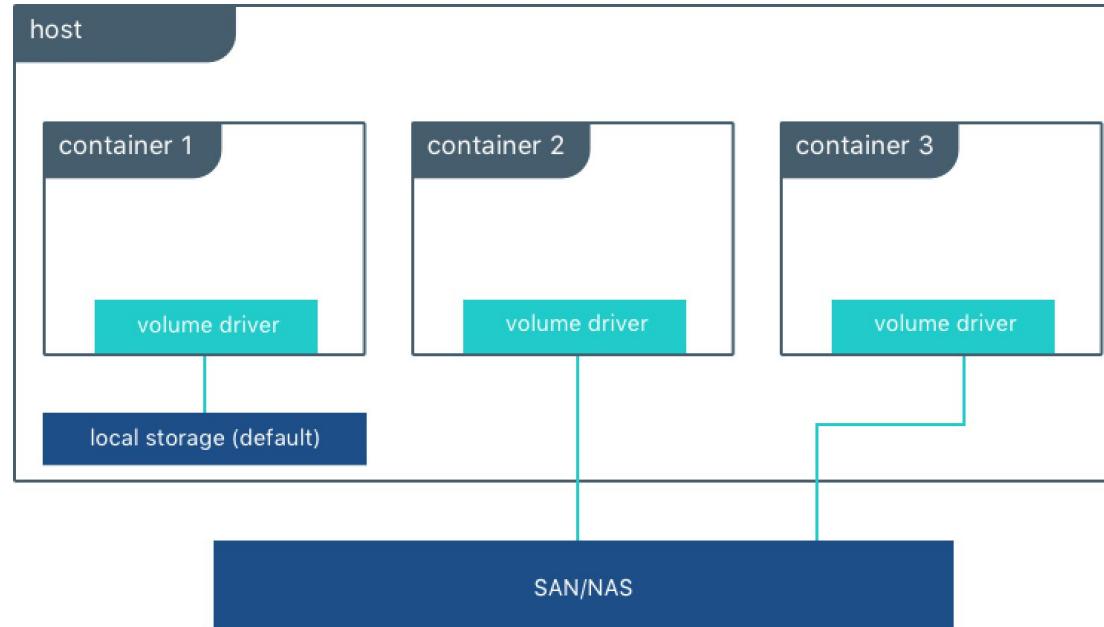
OBJECT STORAGE

Object stores distribute data algorithmically throughout a cluster of media, without a rigid structure.

Les drivers de stockage

- Permettent la création de volumes sur différents types de stockage
- Driver par défaut : “local”
 - répertoire de stockage sur la machine hôte
- Ajout de driver via l’installation de plugins
 - intégration avec des systèmes de stockage externes (Ceph, AWS, GCE, Azure, ...)
 - Legacy (avant 1.13)
 - https://docs.docker.com/engine/extend/legacy_plugins/#/volume-plugins
 - Managed Plugin System (à partir de 1.13)
 - install / start / stop / list / remove / ...
 - <https://docs.docker.com/engine/extend/#installing-and-using-a-plugin>

Les drivers de stockage



<https://success.docker.com>

Exemple de plugin : sshfs

- Stockage sur un système de fichier via ssh
- <https://github.com/vieux/docker-volume-sshfs>

```
# Installation du plugin
$ docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
- network: [host]
- mount: [/var/lib/docker/plugins/]
- mount: []
- device: [/dev/fuse]
- capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
52d435ada6a4: Download complete
Digest: sha256:1d3c3e42c12138da5ef7873b97f7f32cf99fb6edde75fa4f0bcf9ed277855811
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
```

Exemple de plugin : sshfs

```
# Création du répertoire sur le serveur ssh
$ ssh USER@HOST mkdir /tmp/data

# Création du volume avec le driver vieux/sshfs
$ docker volume create -d vieux/sshfs -o sshcmd=USER@HOST:/tmp/data -o password=PASSWORD data

# Liste des volumes
$ docker volume ls
vieux/sshfs:latest    data

# Utilisation du volume dans un container
$ docker run -it -v data:/data alpine
# touch /data/test

# Vérification sur le serveur ssh
$ ssh USER@HOST ls /tmp/data
test
```

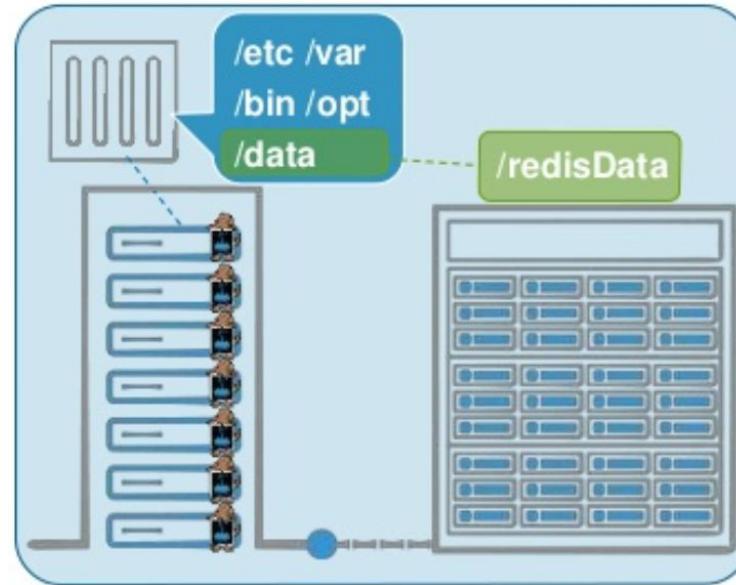
REX-Ray

- Solution d'orchestration de stockage distribué
- Persistance des données pour les containers stateful
- Utilisé avec Docker Swarm, Kubernetes, Mesos / Marathon
- Utilise le framework [libStorage](#)
- Multiples plugins disponibles

REX-Ray : les plugins

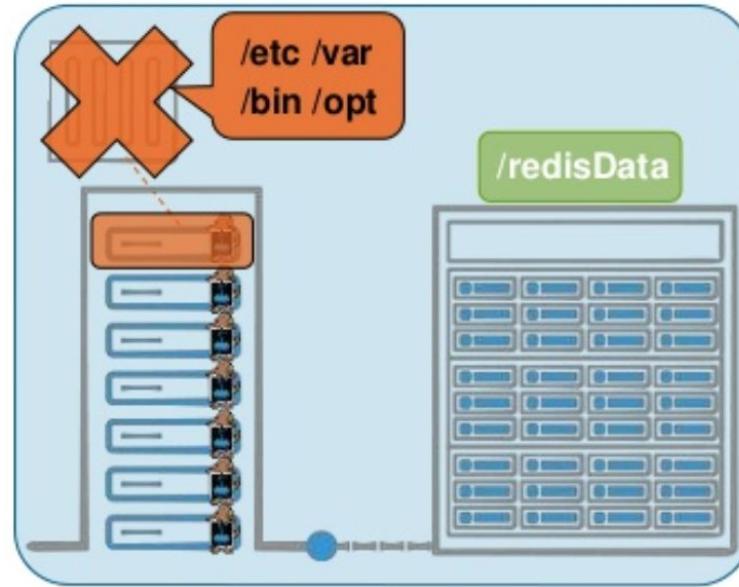
Provider	Storage Platform	Docker	CSI	Containerized
Amazon EC2	EBS	✓	✓	✓
	EFS	✓	✓	✓
	S3FS	✓	✓	✓
Ceph	RBD	✓	✓	✓
Local	CSI-BlockDevices		✓	✓
	CSI-NFS	✓	✓	✓
	CSI-VFS		✓	✓
Dell EMC	Ipsilon	✓	✓	✓
	ScaleIO	✓	✓	✓
DigitalOcean	Block Storage	✓	✓	✓
FittedCloud	EBS Optimizer	✓	✓	
Google	GCE Persistent Disk	✓	✓	✓
Microsoft	Azure Unmanaged Disk	✓	✓	
OpenStack	Cinder	✓	✓	✓
VirtualBox	Virtual Media	✓	✓	

REX-Ray : exemple d'utilisation



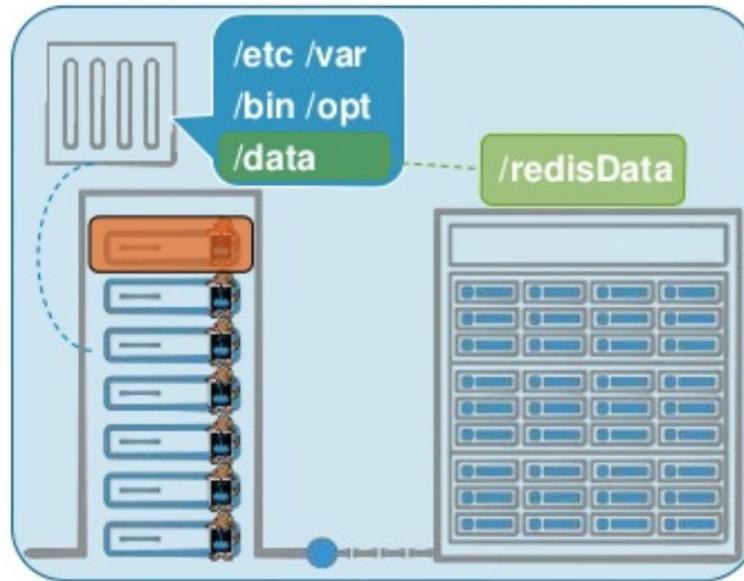
```
$ docker service create --mount type=volume,src=redisData,target=/data,volume-driver=rexray redis
```

REX-Ray : exemple d'utilisation



Les données persistent en dehors du cycle de vie du container du service

REX-Ray : exemple d'utilisation



Le volume est monté sur le container qui a été re-schédulé sur un autre node du cluster

Démo

Docker Machine



Sommaire

- Utilisation
- Installation
- Les commandes
- Création d'un hôte Docker
- Communication avec un hôte distant

Utilisation

- Création d'une VM ou utilisation d'une VM ou machine physique existante
- Installation de la plateforme Docker (client + daemon)
- Permet au client local de communiquer avec le daemon distant
- Création de certificats pour sécuriser la communication

Utilisation

- En local
 - Oracle Virtualbox
 - VMWare
- Sur une infrastructure cloud
 - DigitalOcean
 - Amazon Web Service
 - Microsoft Azure
 - Google Compute Engine
- Générique

Installation

- Docker for Mac / Docker for Windows / Docker Toolbox
 - Docker Machine installé par défaut
 - Mise à jour automatique
- Installation du binaire
 - <https://docs.docker.com/machine/install-machine/#/installing-machine-directly>
- Installation de VirtualBox pour la création d'hôtes en local

Les commandes

```
$ docker-machine --help
Usage: docker-machine [OPTIONS] COMMAND [arg...]
Options:
...
Commands:
  active          Print which machine is active
  config          Print the connection config for machine
  create          Create a machine
  env             Display the commands to set up the environment for the Docker client
  inspect         Inspect information about a machine
  ip              Get the IP address of a machine
  kill             Kill a machine
  ls              List machines
  regenerate-certs Regenerate TLS Certificates for a machine
  restart         Restart a machine
  rm              Remove a machine
  ssh             Log into or run a command on a machine with SSH.
  scp             Copy files between machines
...
Run 'docker-machine COMMAND --help' for more information on a command.
```

Les commandes

- Gestion du cycle de vie (create, rm, start, stop, restart, kill, upgrade)
- Information sur l'hôte (ip, url, config, status, version)
- Liste des hôtes créés (ls)
- Copie de fichiers (scp)
- Lancement d'un shell (ssh)

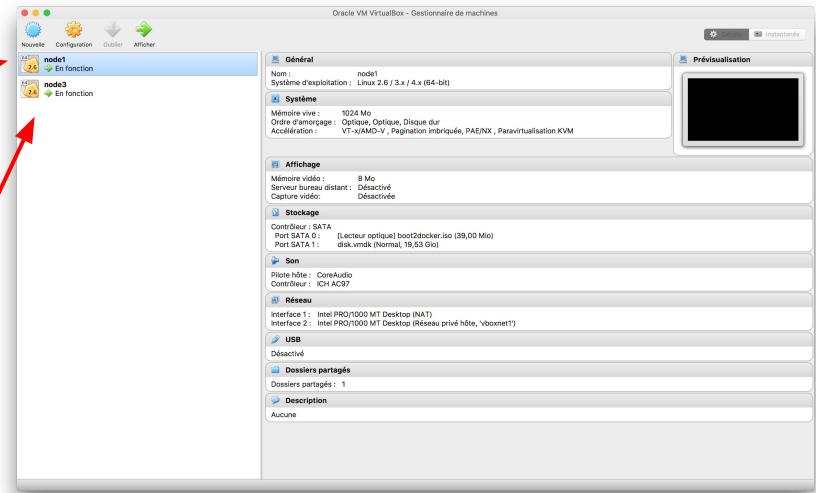
Création

- $\$ \text{ docker-machine create } [\text{OPTIONS}] [\text{arg...}]$
- Option $--\text{driver } \text{ DRIVER}$ à spécifier
- Options spécifiques au driver utilisé
 - identifiants du cloud provider
 - type d'instance (région, RAM, cpu, ...)
 - ...
- Options non spécifiques au driver utilisé
 - configuration du Docker daemon
 - configuration d'un cluster Swarm
 - ...

Exemple : création d'un hôte sur VirtualBox

- *Options par défaut*

```
$ docker-machine create --driver virtualbox node1
```



- *Options additionnelles*

```
$ docker-machine create --driver virtualbox \
--virtualbox-memory=2048 \
--virtualbox-disk-size=5000 \
node3
```

Interface de VirtualBox

Exemple : création d'un hôte sur DigitalOcean

- *Options par défaut*

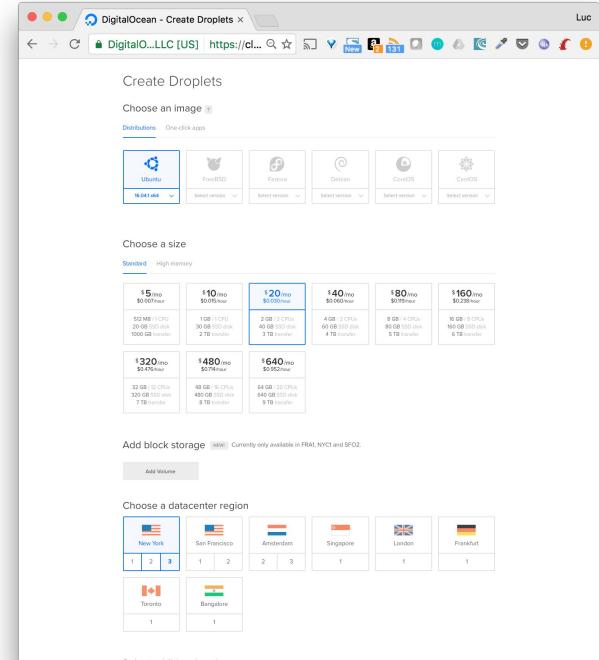
```
$ docker-machine create --driver digitalocean \
--digitalocean-access-token=$TOKEN \
node1
```

Un token d'identification doit être spécifié

- *Options additionnelles*

```
$ docker-machine create --driver digitalocean \
--digitalocean-access-token=$TOKEN \
--digitalocean-region=lon1 \
--digitalocean-size=1gb \
node2
```

Liste des options disponibles: <https://docs.docker.com/machine/drivers/digital-ocean/#/options>



Création d'un Droplet depuis
digitalocean.com

Exemple : création d'un hôte sur Amazon EC2

- *Options par défaut*

```
$ docker-machine create --driver amazonec2 \
--amazonec2-access-key=${ACCESS_KEY_ID} \
--amazonec2-secret-key=${SECRET_ACCESS_KEY} \
node1
```

Des clés d'identification doivent être spécifiées

- *Options additionnelles*

```
$ docker-machine create --driver amazonec2 \
--amazonec2-access-key=${ACCESS_KEY_ID} \
--amazonec2-secret-key=${SECRET_ACCESS_KEY} \
--amazonec2-region=eu-west-1 \
--amazonec2-ami=ami-ed82e39e \
--amazonec2-instance-type=t2.large \
node2
```

Liste des options disponibles: <https://docs.docker.com/machine/drivers/aws/#/options>

Démo

Création d'un hôte Docker

Communication avec un hôte distant

- Client local communique avec le daemon local par défaut (via unix socket)
- Variables d'environnement pour cibler un hôte distant

```
$ docker-machine env node1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.102:2376"
export DOCKER_CERT_PATH="/Users/luc/.docker/machine/machines/node1"
export DOCKER_MACHINE_NAME="node1"
```

- Définition des variables d'environnement

```
$ eval $(docker-machine env node1)
```

Communication avec un hôte distant

- Machine cible définie via la variable DOCKER_HOST

```
$ env | grep DOCKER
DOCKER_HOST=tcp://192.168.99.102:2376
DOCKER_MACHINE_NAME=node1
DOCKER_TLS_VERIFY=1
DOCKER_CERT_PATH=/Users/luc/.docker/machine/machines/node1
```

- Les commandes Docker ciblent l'hôte distant

```
$ docker image ls
```

- Suppression des variables pour cibler le daemon local

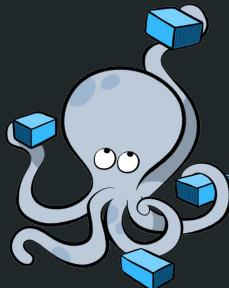
```
$ eval $(docker-machine env -u)
$ docker image ls
```

Démo

Communication avec un hôte distant

Exercice

Docker Compose



Sommaire

- Présentation
- Le fichier docker-compose.yml
- Le binaire docker-compose
- Service discovery
- Exemple avec la Voting App

Présentation

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

Présentation

- Hôte unique : application lancée avec le binaire docker-compose

```
$ docker-compose up
```

- Cluster Swarm : application lancée par le client Docker

```
$ docker stack deploy -c docker-compose.yml app
```

- Certaines options utilisables dans un seul contexte

- ex: *build* uniquement prise en compte par *docker-compose*
- ex: *deploy* uniquement prise en compte par *docker stack*

docker-compose.yml : structure

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

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

Application web décrite dans un fichier docker-compose.yml

docker-compose.yml : évolution du format

```
version: '1'  
web:  
  build: .  
  ports:  
    - "5000:5000"  
  volumes:  
    - ./code  
  links:  
    - redis  
redis:  
  image: redis
```



```
version: '2'  
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
    volumes:  
      - ./code  
    networks:  
      - front-tier  
      - back-tier  
  redis:  
    image: redis  
    volumes:  
      - data:/var/lib/redis  
    networks:  
      - back-tier  
  volumes:  
    data:  
  networks:  
    front-tier:  
    back-tier:
```



```
version: '3'  
services:  
  redis:  
    image: redis:alpine  
    volumes:  
      - data:/var/lib/redis  
  networks:  
    - back-tier  
  deploy:  
    replicas: 6  
    update_config:  
      parallelism: 2  
      delay: 10s  
    restart_policy:  
      condition: on-failure  
    placement:  
      constraints:  
        - node.role == manager  
    resources:  
      limits:  
        memory: 50M  
  volumes:  
    redis-data:  
  networks:  
    back-tier:
```

Version 1

Version 2.x

Version 3.x

docker-compose.yml : des instructions courantes

- Image utilisée
- Nombre de répliques
- Ports exposés
- Stratégie de redémarrage
- Contraintes de déploiement (Swarm)
- Configuration des mises à jour (Swarm)
- Health check



Instructions pouvant être utilisées pour chaque service

docker-compose.yml : Dev vs Prod

- Même fichier de base pour différents environnements
- Des contraintes et des options différentes
 - bind-mount du code applicatif
 - binding des ports
 - variables d'environnement
 - règles de redémarrage
 - services supplémentaires (tls, logs, monitoring, ...)
 - contraintes de placement
 - ...

Le binaire docker-compose

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

Le binaire docker-compose

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

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

Service discovery

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

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

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

Extrait d'un fichier docker-compose.yml

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

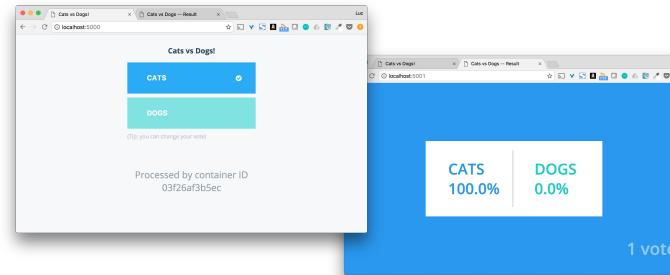
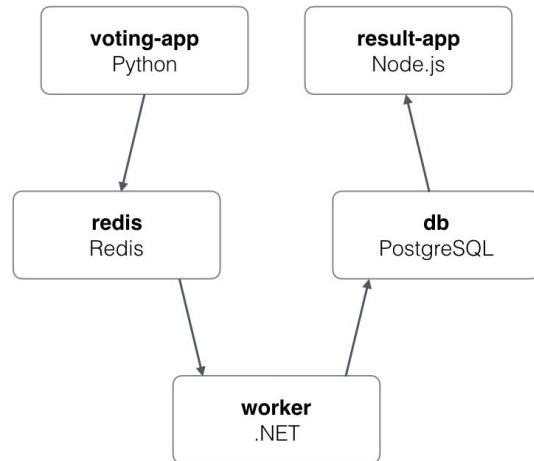
Extrait d'un code Node.js: connexion à la base de données

Démo

Service discovery

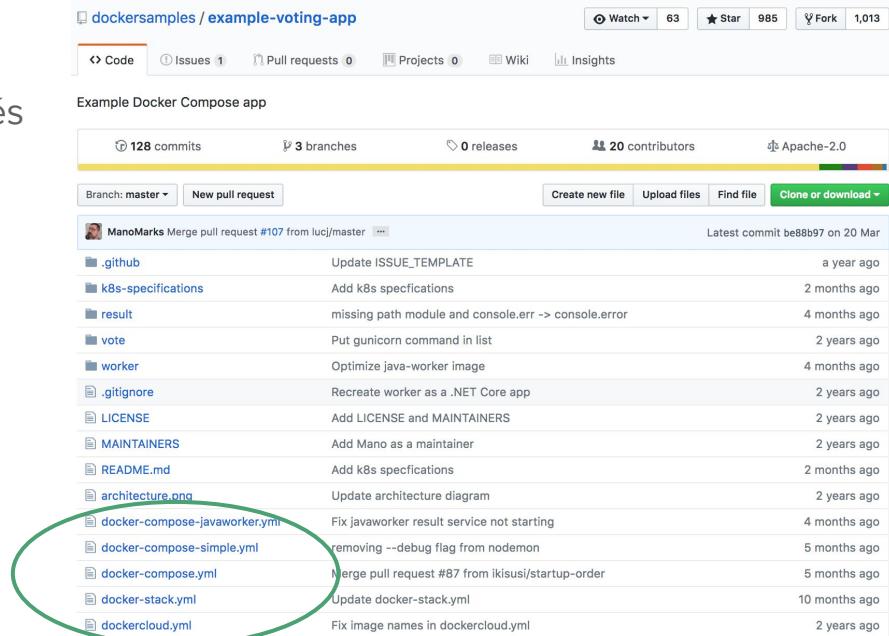
Voting App

- Application Open Source maintenue par Docker
- Utilisée pour des démos / présentations
- 5 services
 - différents langages
 - Node.js / Python / .NET
 - différentes bases de données
 - Redis / Postgres



Voting App

- <https://github.com/docker/example-voting-app>
- Plusieurs fichiers Compose
 - illustration de différentes complexités
 - worker .Net vs Java
 - environnement de dev vs prod
 - ...



Voting App : Dev vs Prod

```
version: "3"
services:
  vote:
    build: ./vote
    command: python app.py
    volumes:
      - ./vote:/app
    ports:
      - "5000:80"
  redis:
    image: redis:alpine
  worker:
    build: ./worker
  db:
    image: postgres:9.4
  result:
    build: ./result
    command: nodemon --debug server.js
    volumes:
      - ./result:/app
    ports:
      - "5001:80"
      - "5858:5858"
```

Instruction **build** pour la construction de l'image

Instruction **volume** pour le bind-mount du code applicatif

Redéfinition de la commande à lancer dans le container.
Nodemon permet de relancer l'application si changement du code depuis l'IDE

DEV

PROD

```
version: "3"
services:
  redis:
    image: redis:alpine
  networks:
    - frontend
  deploy:
    update_config:
      parallelism: 2
      delay: 10s
    placement:
      constraints:
        - node.role == manager
    restart_policy:
      condition: on-failure
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
  deploy:
    replicas: 2
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure
...
networks:
  frontend:
  backend:
  volumes:
    db-data:
```

Instruction **image** pour l'utilisation d'un image existante

Instruction **deploy** pour un déploiement sur un cluster Swarm

Réseaux pour l'isolation des services

Voting App : workflow

```
# Lancement de l'application
$ docker-compose -f docker-simple.yml up -d
... build des images ...
Creating examplevotingapp_vote_1    ... done
Creating examplevotingapp_redis_1   ... done
Creating examplevotingapp_worker_1  ... done
Creating examplevotingapp_db_1      ... done
Creating examplevotingapp_result_1 ... done

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

Voting App : workflow

```
# Logs de l'ensemble des containers de l'application
$ docker-compose -f docker-simple.yml logs -f
Attaching to examplevotingapp_db_1, examplevotingapp_vote_1, examplevotingapp_redis_1, examplevotingapp_worker_1,
examplevotingapp_result_1
db_1      | The files belonging to this database system will be owned by user "postgres".
db_1      | This user must also own the server process.
db_1      |
db_1      | The database cluster will be initialized with locale "en_US.utf8".
db_1      | The default database encoding has accordingly been set to "UTF8".
db_1      | The default text search configuration will be set to "english".
db_1      |
redis_1   | 1:C 11 May 20:20:45.953 # o000o000o000o Redis is starting o000o000o000o
vote_1    | * Serving Flask app "app" (lazy loading)
vote_1    | * Environment: production
vote_1    | * Debug mode: on
vote_1    | * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
vote_1    | * Restarting with stat
redis_1   | 1:C 11 May 20:20:45.953 # Redis version=4.0.9, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1   | 1:C 11 May 20:20:45.953 # Warning: no config file specified, using the default config. In order to specify a
config file use redis-server /path/to/redis.conf
...
...
```

Voting App : workflow

```
# Scale le nombre d'instances du service worker
$ docker-compose -f docker-compose-simple.yml up --scale worker=3 -d
examplevotingapp_vote_1 is up-to-date
examplevotingapp_redis_1 is up-to-date
Starting examplevotingapp_worker_1 ...
examplevotingapp_result_1 is up-to-date
Starting examplevotingapp_worker_1 ... done
Creating examplevotingapp_worker_2 ... done
Creating examplevotingapp_worker_3 ... done

# Suppression de l'application
$ docker-compose -f docker-compose-simple.yml down
Stopping examplevotingapp_worker_2 ... done
Stopping examplevotingapp_worker_3 ... done
Stopping examplevotingapp_db_1      ... done
Stopping examplevotingapp_worker_1 ... done
Stopping examplevotingapp_vote_1   ... done
Stopping examplevotingapp_result_1 ... done
Stopping examplevotingapp_redis_1  ... done
...
Removing network examplevotingapp_default
```

Voting App : environnement de développement

The screenshot shows a code editor interface with a sidebar labeled "EXPLORATEUR". The sidebar lists various files and folders related to the "EXAMPLE-VOTING-APP" project, including ".github", "k8s-specifications", "result", "tests", "views", "stylesheets", "app.js", "index.html", "socket.io.js", "Dockerfile", "package.json", "server.js", "vote", "worker", ".gitignore", "architecture.png", "LICENSE", "MAINTAINERS", and "README.md". The main panel displays the content of the "index.html" file, which is an Angular template for a voting application. The file includes HTML, CSS, and JavaScript code, such as:

```
<!DOCTYPE html>
<html ng-app="catsvsdogs">
  <head>
    <meta charset="utf-8">
    <title>Cats vs Dogs -- Result</title>
    <base href="/index.html">
    <meta name="viewport" content="width=device-width, initial-scale = 1.0">
    <meta name="keywords" content="docker-compose, docker, stack">
    <meta name="author" content="Docker">
    <link rel="stylesheet" href='/stylesheets/style.css' />
  </head>
  <body ng-controller="statsCtrl" >
    <div id="background-stats">
      <div id="background-stats-1">
        </div><!--
        --><div id="background-stats-2">
      </div>
    </div>
    <div id="content-container">
      <div id="content-container-center">
        <div id="choice">
          <div class="choice cats">
            <div class="label">Cats</div>
            <div class="stat">{{aPercent | number:1}}%</div>
          </div>
          <div class="divider"></div>
          <div class="choice dogs">
            <div class="label">Dogs</div>
            <div class="stat">{{bPercent | number:1}}%</div>
          </div>
        </div>
      </div>
    </div>
    <div id="result">
      <span ng-if="total == 0">No votes yet</span>
      <span ng-if="total == 1">{{total}} vote</span>
      <span ng-if="total >= 2">{{total}} votes</span>
    </div>
    <script src="socket.io.js"></script>
  </body>
</html>
```

The status bar at the bottom shows "master" and "Li 23, Col 36 (4 sélectionné)".

The terminal window shows the execution of several docker-compose commands:

```
4. ./tmp (docker-compose)
docker-compose.yml
example-voting-app $ docker-compose -f do
docker-compose-javaworker.yml docker-stack.yml
docker-compose-simple.yml dockercloud.yml
docker-compose.yml
example-voting-app $ docker-compose -f docker-com
docker-compose-javaworker.yml docker-compose.yml
docker-compose-simple.yml
example-voting-app $ docker-compose -f docker-compose-simple.yml up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm.
All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.
```

Creating network "example-voting-app_default" with the default driver
Building vote
Step 1/7 : FROM python:2.7-alpine
2.7-alpine: Pulling from library/python
81033e7c1d6a: Downloading [=====]
1.236MB/2.388MB: 0B/2.388MB
276.9kB/346.9kB: 0B/346.9kB
276.9kB/346.9kB: Pulling fs layer
100078176fb: Waiting

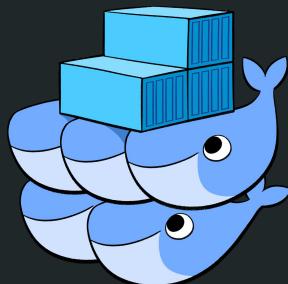
Lancement de l'application à partir du fichier docker-compose-simple.yml

Démo

Voting App

Exercice

Docker Swarm



Sommaire

- Historique
- Swarm mode
- Node
- Service
- Stack
- Secret
- Config
- Routing Mesh
- Interfaces web de gestion

Historique : le cluster Swarm avant Docker 1.12

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

Historique : le cluster Swarm à partir de Docker 1.12

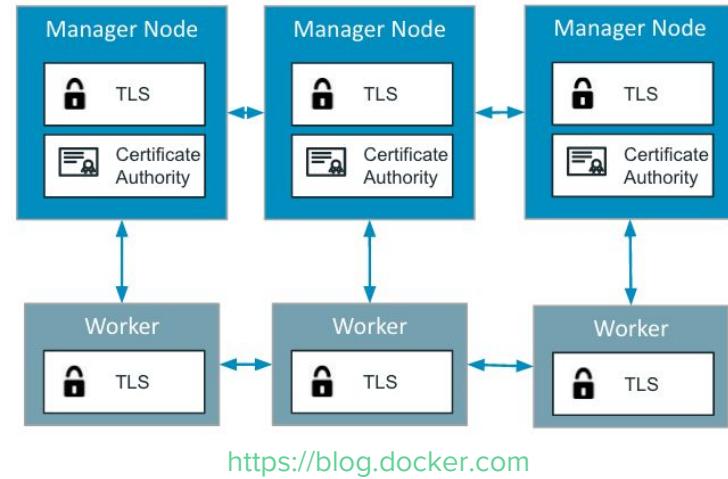
- Disponible depuis Juin 2016
- Orchestration intégrée au Docker daemon
- Swarm mode
- Sécurisé par défaut
- Simplicité de mise en place
- Accessible sur un poste de développement

Swarm mode

- Orchestrateur intégré au daemon Docker
- Permet le déploiement et la gestion des applications containerisées
- Des nouvelles primitives
 - Node : machine membre d'un cluster Swarm
 - Service : spécification des containers d'une application
 - Stack : groupe de services
 - Secret : données sensibles
 - Config : configuration de l'application

Swarm mode

- Communication sécurisée entre les nodes
- Algorithme de consensus Raft
- Boucle de reconciliation
- Rolling upgrade
- Load balancing
- Permet le déploiement d'application au format Docker Compose
- Autolock pour la sécurisation du cluster



Swarm mode : les commandes de base

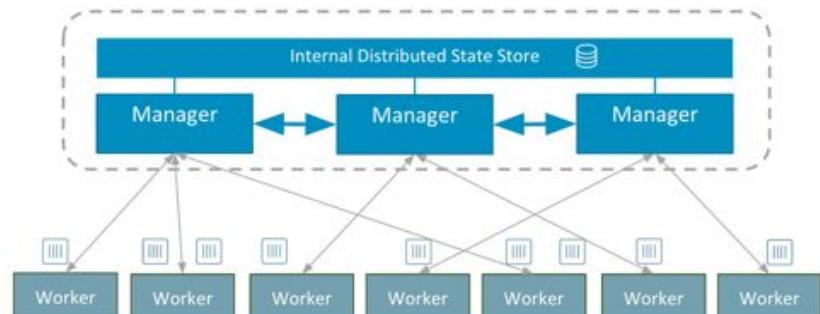
```
$ docker swarm --help
```

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

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

Node

- Machine faisant partie du Swarm
- Manager
 - schedule et orchestre les containers
 - gère l'état du cluster
 - algorithme de consensus RAFT
- Worker
 - exécute les containers
 - protocol Gossip entre les workers
- Par défaut, un manager est aussi un Worker



<https://blog.docker.com>

Node : plusieurs états possible (availability)

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

Node : les commandes de base

```
$ docker node --help
```

```
Usage:docker node COMMAND
```

```
Manage Docker Swarm nodes
```

```
Options:
```

```
  --help    Print usage
```

```
Commands:
```

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

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

Node : initialization du Swarm

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

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

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

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

option obligatoire si plusieurs interfaces réseau

Node : ajout d'un worker

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

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

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

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

Node : ajout d'un manager

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

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

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

docker@node1:~$ docker node ls
ID                                HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
1m9rdh9hlktiwaz3svweeppp        node3    Ready   Active        Reachable
lf2kfkj0442vplsxmgrp05lxjq *    node1    Ready   Active        Leader
pes38pa6w8xteze6dbj9lxx54       node2    Ready   Active
```

Node : promotion / destitution

- Commandes lancées depuis un manager

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

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

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

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
1m9rdh9hlkktiwaz3svweeppp	node3	Ready	Active	Leader
lf2kfkj0442vplsxmgrp05lxjq *	node1	Ready	Active	
pes38pa6w8xteze6dbj9lx54	node2	Ready	Active	Reachable

Node : availability

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

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

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

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

Node : labels

- Permet l'organisation des nodes
- Peut-être utilisé dans les contraintes de déploiement d'un service

```
# List des labels du node1
$ docker node inspect -f '{{ json .Spec.Labels }}' node1
{}

# Ajout du label Memcached avec la valeur true
$ docker node update --label-add Memcached=true node1
node1

# Liste des labels du node1
$ docker node inspect -f '{{ json .Spec.Labels }}' node1 | jq .
{
  "Memcached": "true"
}
```

Démo

Création d'un Swarm avec PWD

Démo

Création d'un Swarm avec Docker Machine

Service

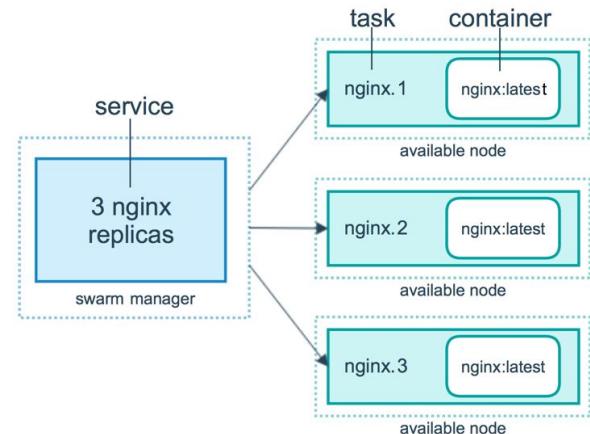
- Définition de la façon de lancer les containers d'une application
- Mode *répliqué* ou *global*
- Adresse IP virtuelle associée à chaque service
- Load balancer de niveau 4 (IPVS)
- Publication d'un port / Routing Mesh

Service

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

Service

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



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

Service : les commandes de base

```
$ docker service --help
```

```
Usage:docker service COMMAND
```

```
Manage services
```

```
Options:
```

```
  --help    Print usage
```

```
Commands:
```

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

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

Service : exemple d'un serveur HTTP

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

```
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE
sux9upku57t1	www	replicated	3/3	nginx:latest

```
$ docker service ps www
```

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

```
$ docker service scale www=1
```

```
www scaled to 1
```

```
$ docker service ps www
```

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

Service : exemple d'une base de données

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

```
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
zicokstoycg9	db	replicated	1/1	mongo:3.4	*:27017->27017/tcp

```
$ docker volume ls
```

DRIVER	VOLUME	NAME
local	86920eb910b7b346fd670efabbbd2e38851fa5ef1245bb40e395aaf6aff9777d	
local	data	

Volume créé par l'instruction mount

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

Service : rolling upgrade

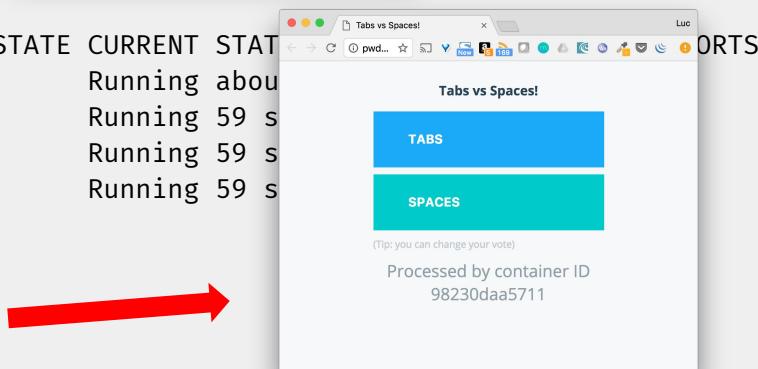
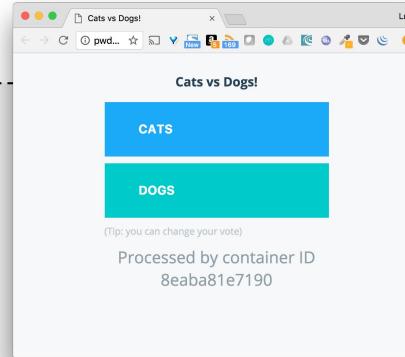
```
$ docker service create \
  --update-parallelism 2 \
  --update-delay 10s \
  --publish 8080:80 \
  --name vote \
  instavote/vote
```

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

```
$ docker service ps vote
```

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

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



Démo

Déploiement d'un service

Stack

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

```
$ docker stack --help
```

Usage:docker stack COMMAND

Manage Docker stacks

Options:

 --help Print usage

Commands:

 deploy Deploy a new stack or update an existing stack

 ls List stacks

 ps List the tasks in the stack

 rm Remove the stack

 services List the services in the stack

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

Stack : exemple avec la Voting App

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

```
docker@node1:~$ docker stack deploy -c docker-stack.yml vote
```

```
Creating network vote_backend
Creating network vote_default
Creating network vote_frontend
Creating service vote_worker
Creating service vote_visualizer
Creating service vote_redis
Creating service vote_db
Creating service vote_vote
Creating service vote_result
```

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

NAME	SERVICES
vote	6

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

ID	NAME	MODE	REPLICAS	IMAGE
0b7wliblok8l	vote_vote	replicated	2/2	dockersamples/examplevotingapp_vote:before
nwl97sw2xci5	vote_redis	replicated	2/2	redis:alpine
oa3k3lqhpkle	vote_visualizer	replicated	1/1	dockersamples/visualizer:stable
otmy5huvz333	vote_worker	replicated	1/1	dockersamples/examplevotingapp_worker:latest
oub9r3pwunqd	vote_db	replicated	1/1	postgres:9.4
w1aek7v8f059	vote_result	replicated	1/1	dockersamples/examplevotingapp_result:before

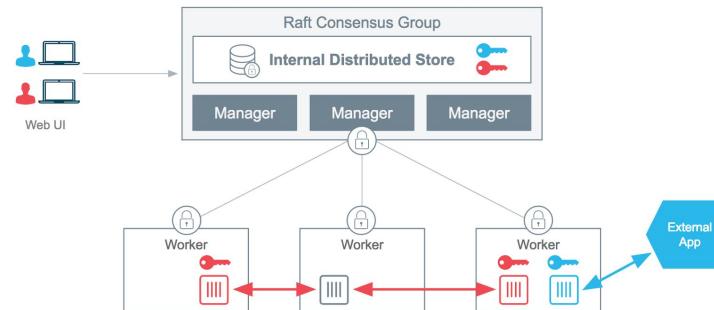
Démo

Déploiement d'une stack

Exercice

Secret

- Donnée sensible (clés SSH, login/password, ...)
- Implémentation dans l'API du Docker daemon depuis la version 1.13
- Sauvegardée dans les logs cryptés utilisés par Raft
- Utilisée par des services
- Disponible à l'exécution dans `/run/secrets/SECRET (tmpfs)`



Secret : exemple

```
docker@node1:~$ echo "a32feKeSE" | docker secret create password -  
n8bh3tv0mebu0nuof83o31ptk  
  
docker@node1:~$ docker service create --name=api --secret=password lucj/api  
sy4ic1zqw2ncqbogowwrcuhe  
  
docker@node1:~$ docker service ps api  
ID           NAME    IMAGE          NODE   DESIRED STATE  CURRENT STATE          ERROR      PORTS  
96wx64mvpvxa  api.1  lucj/api:latest  node2  Running     Running  42 minutes ago  
  
docker@node2:~$ docker exec -ti $(docker ps --filter name=api -q) sh  
# cat /run/secrets/password  
a32feKeSE  
  
docker@node1:~$ docker service update --secret-rm="password" api  
  
docker@node2:~$ docker exec -ti $(docker ps --filter name=api -q) sh  
# cat /run/secrets/password  
cat: /run/secrets/password: No such file or directory  
#
```

Démo

Création et utilisation d'un Secret

Config

- Donnée non sensible, fichiers de configuration
- Non cryptés dans les logs de Raft
- Utilisée par un service

```
version: '3.3'
services:
  proxy:
    image: nginx:1.12.2
    configs:
      - source: server_config
        target: /etc/nginx/nginx.conf
        mode: 0444
        uid: '33'
        gid: '33'
      ...
  configs:
    server_config:
      file: ./nginx.conf
    ...
  ...
```

Démo

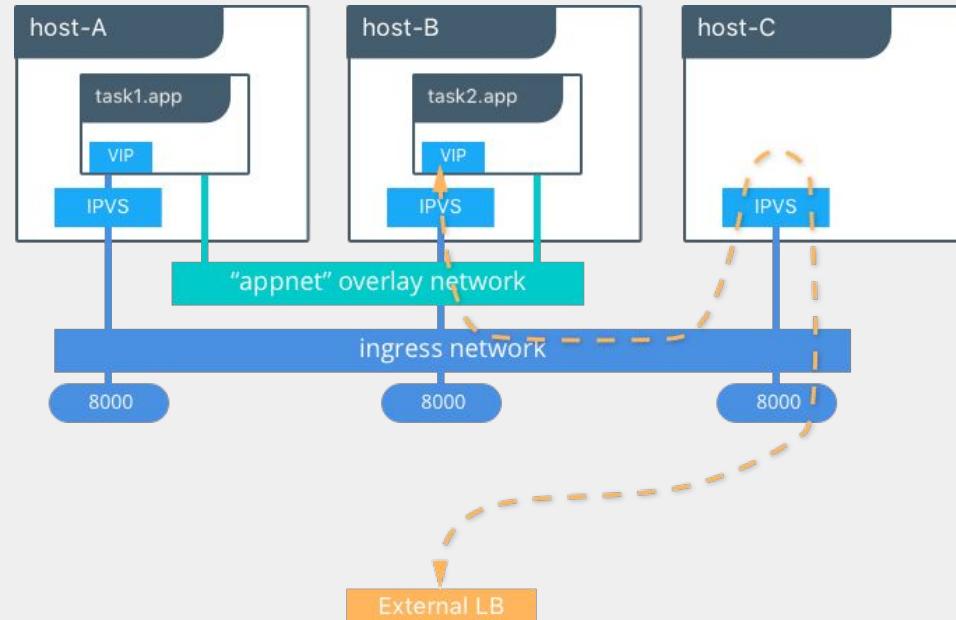
Utilisation d'une Config

Routing Mesh

- Exposition de services à l”extérieur
- Disponible depuis Docker 1.12
- Basé sur ipvs et iptables
- Load balancer L4
- Chaque node accepte des requêtes sur le port publié par un service

Routing Mesh

```
$ docker service create --name app --replicas 2 --network appnet -p 8000:80 nginx
```



<https://success.docker.com/Architecture>

Exercice

Interface web de gestion : Portainer

- Open source
- <https://portainer.io>
- Gestion d'hôtes Docker et de clusters Swarm

```
$ docker volume create portainer_data
$ docker service create \
--name portainer \
--publish 9000:9000 \
--replicas=1 \
--constraint 'node.role == manager' \
--mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
--mount type=volume,src=portainer_data,dst=/data \
portainer/portainer -H unix:///var/run/docker.sock
```

Interface web de gestion : Portainer

The screenshot shows the Portainer web interface running on a Mac OS X system. The title bar indicates the window is titled "Portainer". The address bar shows the URL "localhost:9000/#/dashboard". The top right corner shows the user "Luc" and navigation links for "Help support portainer", "admin", "my account", and "log out".

The left sidebar has two main sections: "PRIMARY" and "PORTAINER SETTINGS". Under PRIMARY, the "Dashboard" tab is selected, along with other options like App Templates, Stacks, Services, Containers, Images, Networks, Volumes, Configs, Secrets, and Swarm. Under PORTAINER SETTINGS, there are User management, Endpoints, Registries, and Settings.

The main content area is the "Home Dashboard". It displays "Node info" for a node named "linuxkit-02500000001" running Docker version "18.05.0-ce-rcl". It also shows that this node is part of a Swarm cluster with a "Manager" role and one node in the cluster.

Below this, there are six cards providing a summary of the Docker environment:

- Stacks: 1
- Services: 6
- Containers: 30 (21 running, 8 stopped)
- Images: 74
- Volumes: 73
- Networks: 10

The bottom left of the dashboard shows the Portainer logo and the version "1.17.0".

Démo

Portainer

Interface web de gestion : Swarmpit

- Open source
- <https://swarmpit.io/>
- Autodeploy

```
$ git clone https://github.com/swarmpit/swarmpit
$ docker stack deploy -c swarmpit/docker-compose.yml swarmpit
Creating network swarmpit_net
Creating service swarmpit_app
Creating service swarmpit_db
Creating service swarmpit_event-collector
```

Interface web de gestion : Swarmpit

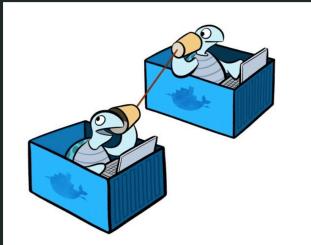
The screenshot shows the Swarmpit web interface running in a browser window titled "Stacks :: swarmpit". The URL is "localhost:888/#/stacks". The top navigation bar includes a logo for "swarmpit 1.5-36f0cc", a search bar, and a user dropdown for "admin". On the left, a sidebar lists categories: APPLICATIONS (Stacks, Services, Tasks), INFRASTRUCTURE (Networks, Nodes), DATA (Volumes, Secrets, Configs), DISTRIBUTION (Dockerhub, Registry), and ADMIN (Users). The main content area is titled "Stacks" and contains a table for "Search stacks". The table has columns: Name, Services, Networks, Volumes, Configs, and Secrets. It lists two stacks: "swampit" (Services: 3, Networks: 1, Volumes: 1, Configs: 0, Secrets: 0) and "vote" (Services: 6, Networks: 3, Volumes: 1, Configs: 0, Secrets: 0).

Name	Services	Networks	Volumes	Configs	Secrets
swampit	3	1	1	0	0
vote	6	3	1	0	0

Démo

Swarmpit

Network



Sommaire

- Les networks de Docker
- Les différents drivers
- Les commandes de base
- Container Network Model (CNM)
- Bridge par défaut
- “User defined” bridge
- Overlay
- Macvlan
- Plugins

Les networks de Docker

- 3 networks créés lors de l'installation

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

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

- Création de l'interface bridge0 sur la machine hôte

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

Les networks de Docker

- Bridge
 - représenté par l'interface docker0 créée sur la machine hôte
 - network auquel les containers sont attachés par défaut
 - permet la communication des containers sur un même hôte
- Host
 - stack réseau de la machine hôte
- None
 - pas de connexion sur l'extérieur
 - une seule interface: localhost

Les différents drivers

- Définit le type de network

```
$ docker network ls
```

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

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

Les différents drivers

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

```
# Driver de type bridge utilisé par défaut
docker@node02:~$ docker network create bgnet
e706c80e070c792c5e85659d03dbbdb35def2d1176ddcd182db41b1b701c2ab9
```

```
# Création d'un network de type macvlan
docker@node02:~$ docker network create --driver macvlan mvnet
b39891e44e677ea05ab3b70f7ccf528508f1e681e886164aeddb10199e74279c
```

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

Démo

Networks et drivers par défaut

Les commandes de base

```
$ docker network --help
```

```
Usage: docker network COMMAND
```

```
Manage networks
```

```
Options:
```

```
--help Print usage
```

```
Commands:
```

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

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

Les commandes de base : exemples

```
$ docker network create mynet
```

```
b38e6d31bc92ff9f57a916ffffe9e9288fccb79dd8a1280dccadc0481cfeead6a
```

```
$ docker network ls
```

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

```
$ docker network inspect -f '{{ json .IPAM.Config }}' mynet | jq
```

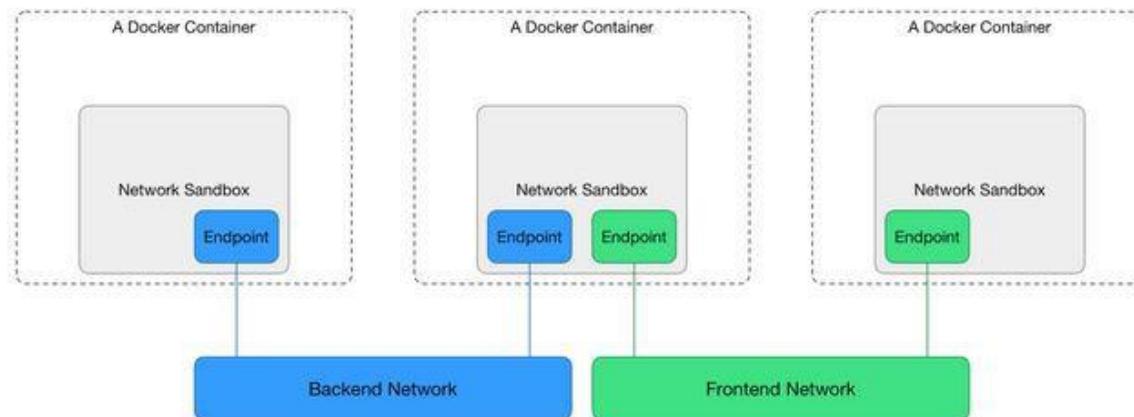
```
[{  
    "Subnet": "172.19.0.0/16",  
    "Gateway": "172.19.0.1"  
}]
```

```
$ docker network rm mynet
```

```
mynet
```

Container Network Model (CNM)

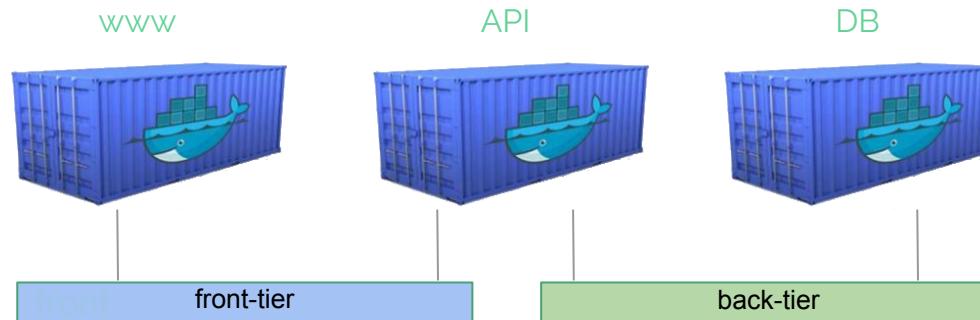
- Sandbox: stack réseau d'un container
- Endpoint: interface réseau
- Network: un ensemble de Endpoint



<https://i1.wp.com/blog.docker.com/media/2015/04/cnm-model.jpg?ssl=1>

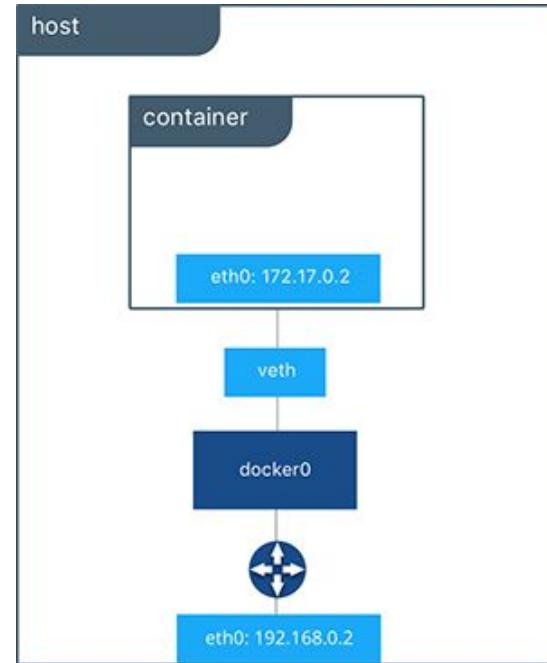
Container Network Model (CNM)

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



Bridge par défaut

- Containers attachés à ce network par défaut
- Représentée par l'interface bridge Docker0
 - créé à l'installation de Docker
- Création d'un paire d'interfaces virtuelles (veth pair)
 - une interface dans chaque network namespace
 - network namespace du container (ethX)
 - root network namespace (machine hôte)



<https://success.docker.com>

Bridge par défaut

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

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

Bridge par défaut

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
2f00ffa53dd0	bridge	bridge	local
8af0e6c63253	host	host	local
f81df072273d	none	null	local

```
$ docker run -d --name=nginx nginx
```

```
0eee3f18...
```

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

```
172.17.0.2
```

```
$ docker run -ti --name=box busybox sh
```

```
/ # ping -c3 nginx
```

```
ping: bad address 'nginx'
```

```
/ # ping -c3 172.17.0.2
```

```
PING 172.17.0.2 (172.17.0.2): 56 data bytes
```

```
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.064 ms
```

```
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.080 ms
```

```
...
```

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

```
$ docker network inspect 2f00ffa53dd0
```

```
...
```

```
Containers:
```

```
{
```

```
    "0eee3f18...": {  
        "IPv4Address": "172.17.0.2/16",  
        "Name": "nginx",  
        ...  
    },
```

```
    "f4088270...": {  
        "IPv4Address": "172.17.0.3/16",  
        "Name": "box",  
        ...  
    }  
...
```

```
}
```

```
...
```

Bridge par défaut

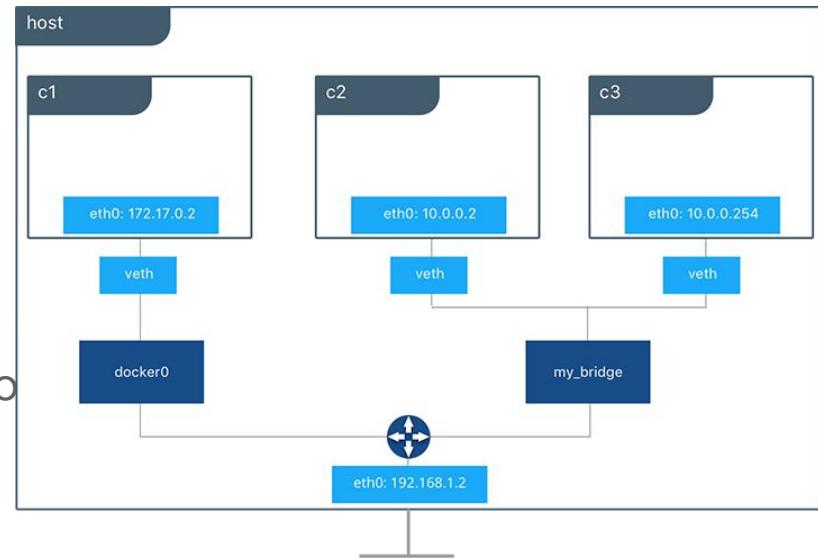
```
# Inspect the nginx network settings
$ docker container inspect -f '{{json .NetworkSettings }}' nginx
{
  "Bridge": "",
  "SandboxID": "4969d4507fab66c072ed4c0625f4b300795fcade2ab5220de324f9e3f67c83f",
  "EndpointID": "11f6a79d2d11f5f1772d7a5843ac0d7815e0ad3f8d35446e06c259b54a1d9745",
  "Gateway":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

* une partie du résultat a été supprimée pour la lisibilité

```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
2f00ffa53dd0    bridge    bridge      local
ef947e210d07    host      host      local
52db6ce3687a    none      null      local
```

“User defined” bridge

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



<https://success.docker.com>

“User defined” bridge

```
$ docker network create --driver=bridge newnet
```

```
49d1e5ee6356...
```

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
689cf99b52ef	bridge	bridge	local
c7695fcd2124	host	host	local
49d1e5ee6356	newnet	bridge	local
2328341f0c45	none	null	local

```
$ docker run -d --name=nginx --network=newnet nginx
```

```
fe46ce155793...
```

```
$ docker run -ti --network=newnet busybox sh
```

```
/ # ping -c3 nginx
```

```
PING nginx (172.18.0.2): 56 data bytes
```

```
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.057 ms
```

```
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.084 ms
```

```
...
```

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

```
$ docker network inspect 49d1e5ee
```

```
...
```

```
Containers:
```

```
{
```

```
    "fe46ce155791...": {  
        "IPv4Address": "172.17.0.2/16",  
        "Name": "nginx",  
        ...  
    },
```

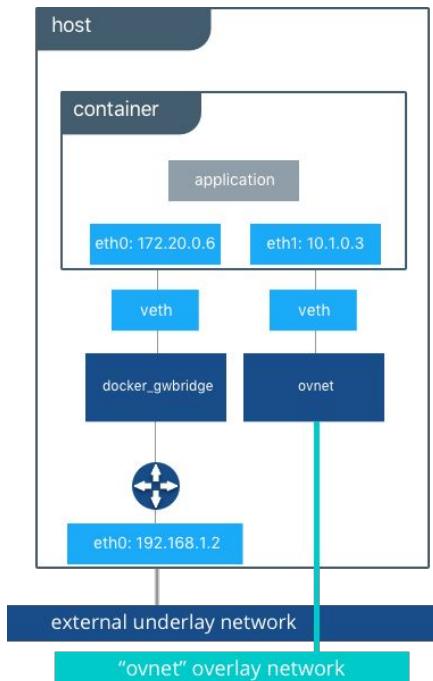
```
    "e43c2872afe4...": {  
        "IPv4Address": "172.17.0.3/16",  
        "Name": "box",  
        ...  
    }  
...
```

```
}
```

```
...
```

Overlay

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



<https://success.docker.com>

Overlay

```
root@node01:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen1000
    link/ether 42:ef:4b:71:82:85 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen1000
    link/ether 16:32:9d:23:8a:35 brd ff:ff:ff:ff:ff:ff
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:eb:94:c3:7a brd ff:ff:ff:ff:ff:ff

root@node01:~# docker swarm init

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

Overlay

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

```
Ptk11nhj82v5j4b2m6kas4d2p
```

```
root@node01:~# docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
efb245433581	bridge	bridge	local
55b9f342458b	docker_gwbridge	bridge	local
5615defdbf54	host	host	local
lk2s876fiaas	ingress	overlay	swarm
993636a4a90a	none	null	local
ptk11nhj82v5	ovnet	overlay	swarm

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

```
pynw2rd27xrdz9sv34kubfk1n
```

```
root@node01:~# ip link
```

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

Overlay

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



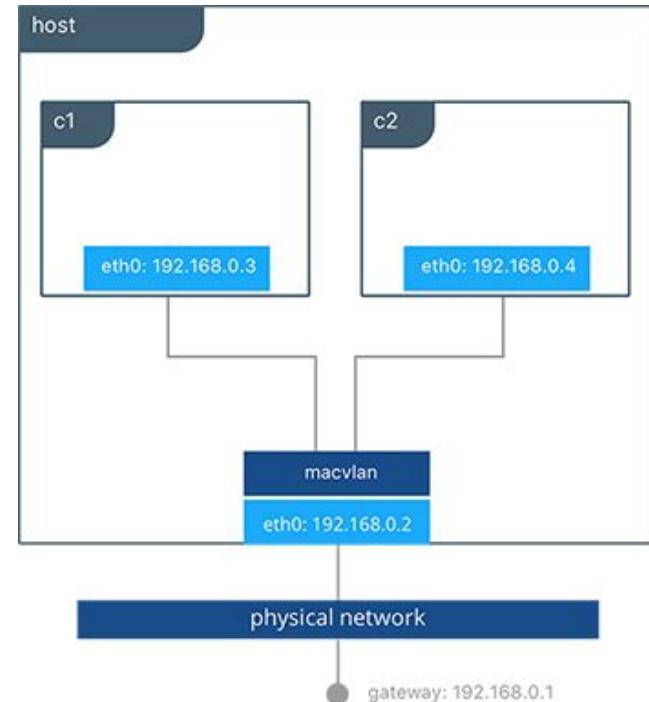
Interface permettant la connection au network overlay



Interface permettant la connection au docker_gwbridge

Macvlan

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



<https://success.docker.com>

Macvlan

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

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

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

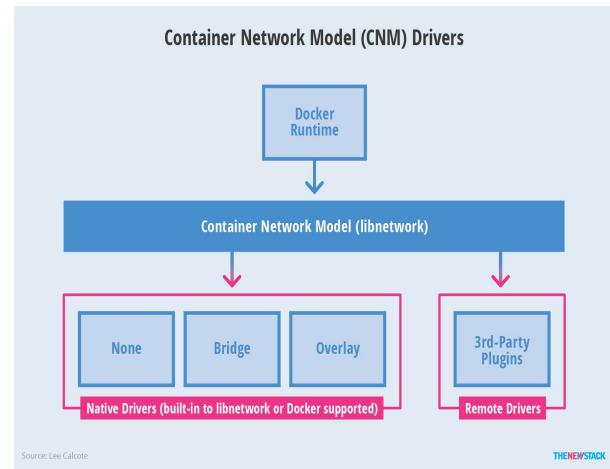
Démo

Containers et networks

Exercice

Plugins

- Permet d'étendre le fonctionnement du daemon
- Ajout de fonctionnalités
- Utilisable via des **drivers**
- Exemple de plugins réseau
 - Weave Network
 - Kuryr Network (OpenStack)
 - https://docs.docker.com/engine/extend/legacy_plugins/#network-plugins



Source: Lee Calore

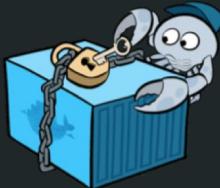
<https://thenewstack.io/>

Plugins

```
# Installation du plugin Weave
$ docker plugin install weaveworks/net-plugin:latest_release
Plugin "weaveworks/net-plugin:latest_release" is requesting the following privileges:
- network: [host]
- mount: [/proc/]
- mount: [/var/run/docker.sock]
- mount: [/var/lib/]
- mount: [/etc/]
- mount: [/lib/modules/]
- capabilities: [CAP_SYS_ADMIN CAP_NET_ADMIN CAP_SYS_MODULE]
Do you grant the above permissions? [y/N] y
latest_release: Pulling from weaveworks/net-plugin
34165d7d46d9: Download complete
Digest: sha256:5016b9e5596df4c700546421ffea86a8a6d9e8c65cdf5306afb14adeedee102a
Status: Downloaded newer image for weaveworks/net-plugin:latest_release
Installed plugin weaveworks/net-plugin:latest_release

# Création d'un network utilisant le driver Weave
$ docker network create --driver=weaveworks/net-plugin:latest_release weavenet
j5rnxdyyymwe45yjlz33xf37x9
```

Sécurité

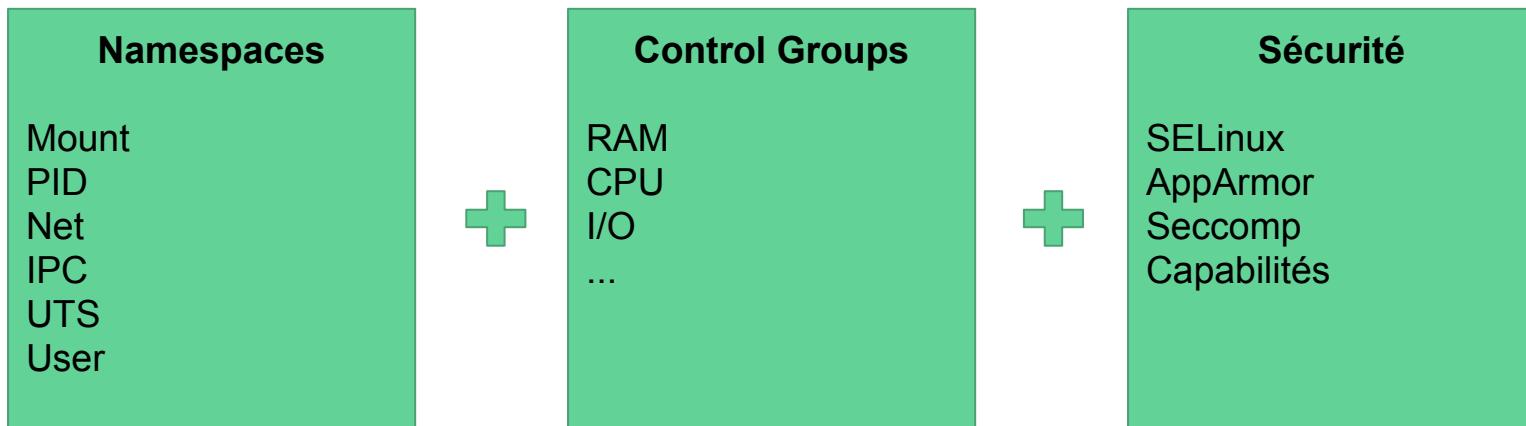


Sommaire

- Quelques rappels
- Exemples de menaces
- Hardening
- Isolation (namespaces)
- Limitation des ressources (cgroups)
- Linux Security Modules
- Capabilities
- Seccomp
- Scan de vulnérabilités
- Content Trust
- La sécurité dans un Swarm

Quelques rappels

- Un container est composé d'un ensemble de fonctionnalités d'isolation du Kernel Linux



- Travaux très actifs sur la sécurité des containers

Exemples de menaces

- Compromission de l'hôte par un container
 - épuisement des ressources
 - accès au filesystem de l'hôte
- Compromission d'un container par un autre container
 - utilisation massive des ressources
 - problème dans un environnement multi-tenants
- Corruption de l'image utilisée
- Compromission de l'application tournant dans un container

```
# Exemple de fork bomb
$ :(){ :|:& };;
Warning: n'essayez pas chez vous :)
```

Hardening

“In computing, hardening is usually the process of securing a system by reducing its surface of vulnerability.” - Wikipedia



Hardening : Center for Internet Security (CIS) benchmark

- <https://learn.cisecurity.org/benchmarks>
- Configuration de la machine hôte
- Configuration du daemon Docker
- Fichiers de configuration du daemon Docker
- Images et Dockerfile
- Environnement d'exécution d'un container
- Opérations

Linux Security Modules

- AppArmor / SELinux
- Implémentation du MAC (Mandatory Access Control)
- Etend le DAC (Discretionary Access Control)



```
-rw-r--r-- 1 root root 3106 Aug 22 2017 .bashrc
```

A red box highlights the first three characters of the permissions column, which are "rw-", indicating read and write permissions for the owner.

- Renforce la sécurité en ajoutant des droits d'accès supplémentaires

Linux Security Modules : AppArmor

- Autorisation basée sur le chemin d'accès aux ressources
- Un profil de sécurité par application pour limiter les droits
- Spécifie quels fichiers une application peut lire / écrire / exécuter
- Disponible par défaut sur les distribution Ubuntu, OpenSuse



https://sharewiz.net/secure_server_apparmor_security.html

Linux Security Modules : AppArmor

Profil par défaut utilisé par Docker

```
...
network,
capability,
file,
umount,
deny @{PROC}/* w,    # deny write for all files directly in /proc (not in a subdir)
# deny write to files not in /proc/<number>/** or /proc/sys/**
deny @{PROC}/{[^1-9],[^1-9][^0-9],[^1-9s][^0-9y][^0-9s],[^1-9][^0-9][^0-9][^0-9]*}/** w,
deny @{PROC}/sys/[^k]** w,  # deny /proc/sys except /proc/sys/k* (effectively /proc/sys/kernel)
deny @{PROC}/sys/kernel/{?,??,[^s][^h][^m]**} w,  # deny everything except shm* in /proc/sys/kernel/
deny @{PROC}/sysrq-trigger rwklx,
deny @{PROC}/mem rwklx,
deny @{PROC}/kmem rwklx,
denied @{PROC}/kcore rwklx,  
----->
deny mount,
deny /sys/[^f]/** wklx,
deny /sys/f[^s]/** wklx,
deny /sys/fs/[^c]/** wklx,
deny /sys/fs/c[^g]/** wklx,
deny /sys/fs/cg[^r]/** wklx,
deny /sys/firmware/** rwklx,
deny /sys/kernel/security/** rwklx,
...
...
```

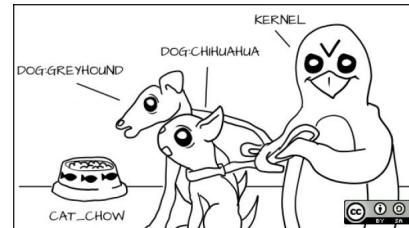
[AppArmor template.go](#)

```
# Container confined in default AppArmor profile
$ docker run -ti alpine sh
/ # cat /proc/kcore
cat: can't open '/proc/kcore': Permission denied

# Unconfined container
$ docker run -ti --security-opt
apparmor:unconfined alpine sh
/ # cat /proc/kcore
/ #
```

Linux Security Modules : SELinux

- Appliqué sur le système entier
- Défini des attributs étendus sur le système de fichiers
- Contexte de sécurité associé aux sujets et aux objets
 - sujet: utilisateur, application, processus
 - objet: fichier, répertoire, device, interface, ...
- Règles définissant les objets auquel un sujet a accès



<https://opensource.com/business/13/11/selinux-policy-guide>

Linux Security Modules : SELinux

```
# Contexte de sécurité des binaires Docker
$ ls -Z /usr/bin/docker*
-rwxr-xr-x. root root system_u:object_r:docker_exec_t:s0 /usr/bin/docker
-rwxr-xr-x. root root system_u:object_r:docker_exec_t:s0 /usr/bin/dockerd
-rwxr-xr-x. root root system_u:object_r:docker_exec_t:s0 /usr/bin/docker-containerd
-rwxr-xr-x. root root system_u:object_r:docker_exec_t:s0 /usr/bin/docker-runc
...
...
# Contexte de sécurité du fichier index.html servi par Nginx
$ ls -Z /usr/share/nginx/html/index.html
-rw-r--r--. root root system_u:object_r:httpd_sys_content_t:s0 index.html

# Le fichier index.html ne peut pas être modifié par un processus Docker
$ docker run -ti -v /:/host alpine sh
sh-4.3# echo "test" > /host/usr/share/nginx/html/index.html
sh: can't create /host/usr/share/nginx/html/index.html: Permission denied

# Le fichier index.html peut être modifié si le labeling SELinux est désactivé
$ docker run -ti -v /:/host --security-opt label:disable alpine sh
/ # echo "test" > /host/usr/share/nginx/html/index.html
/ #
```

L'filesystem de l'hôte est monté dans sur /host dans le container

Capabilités

- Contrôle d'accès granulaire qui améliore la dichotomie root vs non root
- Différents groupes d'accès
 - CAP_CHOWN: permet la modification des uid/gid
 - CAP_SYS_ADMIN: permet des opérations administratives côté système
 - CAP_NET_ADMIN: permet des opérations administratives côté network
 - CAP_NET_BIND_SERVICE: permet d'affecter un port privilégié (<1024) à un processus
 - ...
- Peuvent être ajoutées / supprimées au lancement d'un container

Capabilités : exemples

```
$ docker run -ti alpine sh  
/ # hostname foo  
hostname: sethostname: Operation not permitted
```

```
$ docker run -ti --cap-add=SYS_ADMIN alpine sh  
/ # hostname foo  
/ # hostname  
foo
```

La capacité SYS_ADMIN est nécessaire pour modifier le hostname (appel à la fonction sethostname)

```
$ docker run alpine ping 8.8.8.8  
PING 8.8.8.8 (8.8.8.8): 56 data bytes  
64 bytes from 8.8.8.8: seq=0 ttl=37 time=0.394 ms  
...
```

```
$ docker run --cap-drop=NET_RAW alpine ping 8.8.8.8  
ping: permission denied (are you root?)  
PING 8.8.8.8 (8.8.8.8): 56 data bytes
```

La capacité NET_RAW est nécessaire pour utiliser la commande *ping*

Seccomp

- Secure Computing Mode
- Contrôle les appels aux fonctions système (mount, mkdir, ptrace, reboot, ...)
- Profil par défaut qui désactive 44 appels (sur +300 pour un Linux x86-64)
- Un profil custom peut-être fourni au lancement d'un container
- Utilisation de l'outil **strace** pour lister les appels système et affiner le profil
- Intersection avec les capacités

Seccomp : exemple

```
$ docker container run -it alpine sh  
/ # mkdir test  
/ # exit
```

```
$ cat policy.json  
{  
    "defaultAction": "SCMP_ACT_ALLOW",  
    "syscalls": [  
        {  
            "name": "mkdir",  
            "action": "SCMP_ACT_ERRNO"  
        }  
    ]  
}
```

```
$ docker run -it --security-opt seccomp:policy.json alpine sh  
/ # mkdir test  
mkdir: can't create directory 'test': Operation not permitted
```

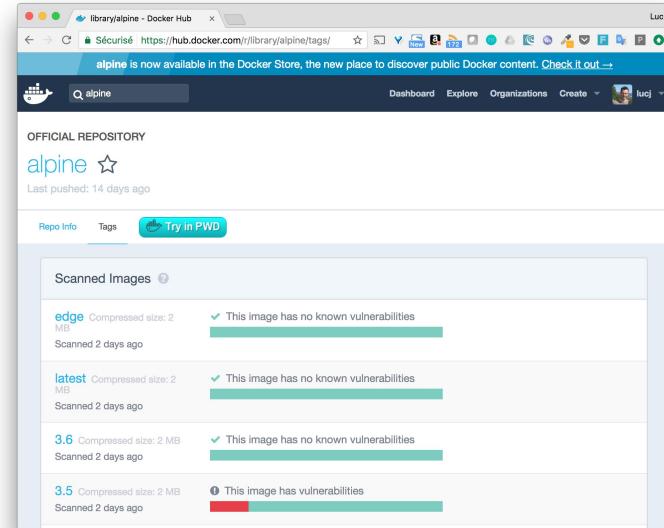
L'appel à *mkdir* est autorisé avec le profil par défaut

Fichier json définissant un profil custom qui interdit l'appel à *mkdir*

Un container utilisant ce profil n'est pas autorisé à faire un appel à *mkdir*

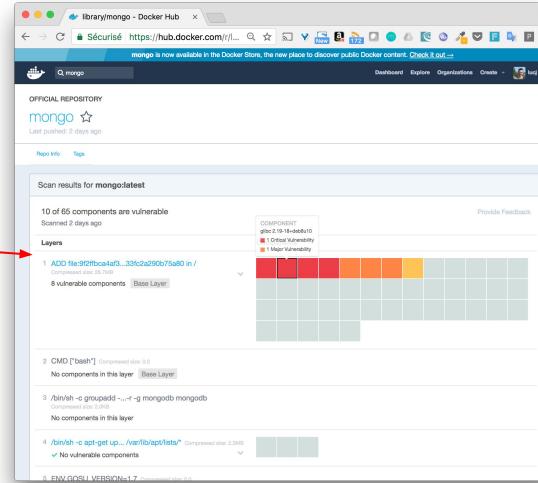
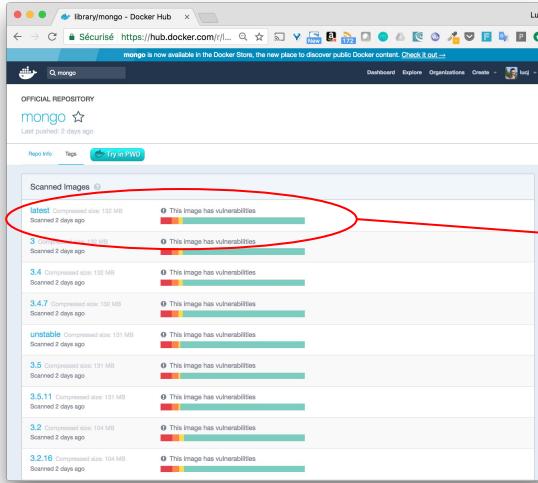
Scan de vulnérabilités

- Docker Security Scanning
- Analyse des images
- Au niveau du registry
- Quay: project open source de CoreOS
- CVE: Common Vulnerabilities and Exposures
- Images basées sur alpine recommandées



Scan de vulnérabilité : Docker Security Scanning

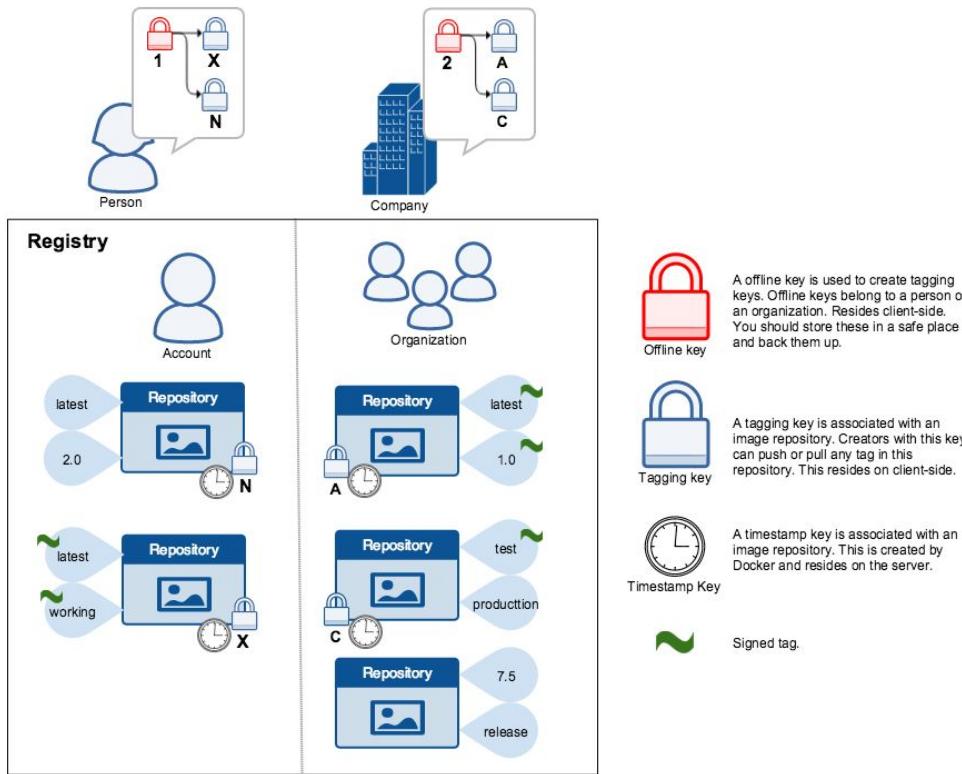
- Analyse de chaque layer au niveau binaire



Content Trust

- Basé sur The Update Framework (TUF)
 - sécurité des systèmes lors de mises à jour logicielles
- Utilisation de clés pour le chiffrement au niveau du tag de l'image
- Signature d'une image lors du **push**
- Vérification de l'intégrité et de son origine lors du **pull**
- Intervient au niveau du tag de l'image
- Activé par une variable d'environnement
 - `export DOCKER_CONTENT_TRUST=1`

Content Trust



Content Trust

```
# Build et push d'une image avec le tag 1.0
node1:~$ docker image build -t lucj/app:1.0 .
node1:~$ docker image push lucj/app:1.0
```

```
# Activation de Docker Content Trust
node1:~$ export DOCKER_CONTENT_TRUST=1
```

```
# Build et push d'une image avec le tag 2.0
node1:~$ docker image build -t lucj/app:2.0 .
node1:~$ docker image push lucj/app:2.0
```

The push refers to a repository [docker.io/lucj/app]

latest: digest: sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe size: 528

Signing and pushing trust metadata

You are about to create a new root signing key passphrase. This passphrase will be used to protect the most sensitive key in your signing system. Please choose a long, complex passphrase and be careful to keep the password and the key file itself secure and backed up. It is highly recommended that you use a password manager to generate the passphrase and keep it safe. There will be no way to recover this key. You can find the key in your config directory.

Enter passphrase for new root key with ID 2252dc6:

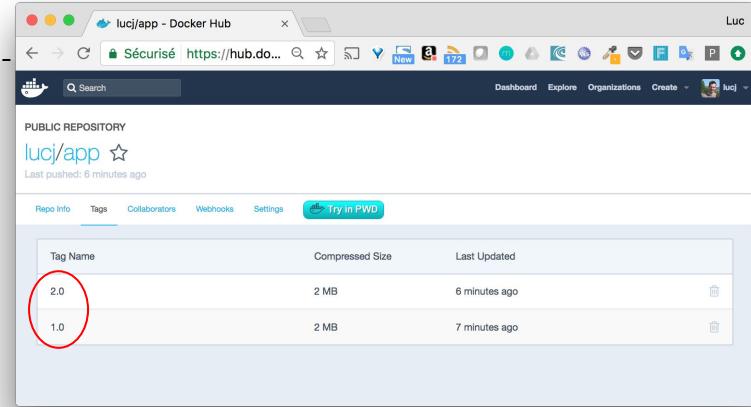
Repeat passphrase for new root key with ID 2252dc6:

Enter passphrase for new repository key with ID 53f07ca (docker.io/lucj/app):

Repeat passphrase for new repository key with ID 53f07ca (docker.io/lucj/app):

Finished initializing "docker.io/lucj/app"

Successfully signed "docker.io/lucj/app":2.0



Content Trust

```
# Activation de Docker Content Trust sur une autre machine
node2:~$ export DOCKER_CONTENT_TRUST=1

# Download de la version 1.0 de l'image (tag non signé)
node2:~$ docker image pull lucj/app:1.0
No trust data for 1.0

# Download de la version 2.0 de l'image (tag signé)
node2:~$ docker image pull lucj/app:2.0
Pull (1 of 1): lucj/app:2.0@sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe: Pulling from lucj/app
88286f41530e: Pull complete
Digest: sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
Status: Downloaded newer image for
lucj/app@sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
Tagging lucj/app@sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe as lucj/app:2.0
```

Docker Security Bench

- Vérification des bonnes pratiques autour du déploiement des containers Docker en production
- Outil basé sur les recommandations du [CIS Docker 1.13 Benchmark](#)
- Plusieurs domaines couverts
 - configuration de la machine hôte
 - configuration du daemon Docker
 - ...
- <https://github.com/docker/docker-bench-security>

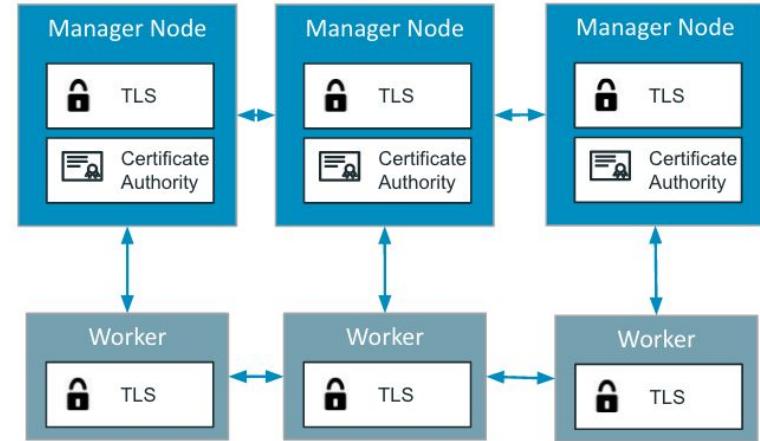
Docker Security Bench

```
$ docker run -it --net host --pid host --cap-add audit_control \
-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security

...
[INFO] 1 - Host Configuration
[WARN] 1.1 - Ensure a separate partition for containers has been created
[NOTE] 1.2 - Ensure the container host has been Hardened
[PASS] 1.3 - Ensure Docker is up to date
      * Using 17.06.0 which is current
[INFO]      * Check with your operating system vendor for support and security maintenance for Docker
[INFO] 1.4 - Ensure only trusted users are allowed to control Docker daemon
      * docker:x:50:docker
[WARN] 1.5 - Ensure auditing is configured for the Docker daemon
[WARN] 1.6 - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.7 - Ensure auditing is configured for Docker files and directories - /etc/docker
...
```

La sécurité dans un Swarm

- Sécurisé par défaut
- TLS entre chaque nodes
- Certificats générés automatiquement
- Rotation des certificats
- Gestion des secrets
- Encryption “at rest” dans les logs de l’algorithme de consensus Raft
- Autolock feature



Monitoring



Importance de la gestion des metrics

- Anticiper les problèmes potentiels
- Comprendre les performances d'une application
- Aide au dimensionnement de l'infrastructure
- Aide au debug
- Un challenge ⇒ collecter et consolider ces metrics dans un environnement cloud évolutif
- Ne pas confondre avec la gestion des logs !

Prometheus

- Projet open source
- Initié par 2 ex-Googlers
- Monitoring de systèmes distribués et gestion d'alertes
- Second projet incubé par la CNCF (après Kubernetes)
- Pull des metrics et les sauvegarde en timeseries

Prometheus stack : les composants

- node-exporter : metrics de la machine hôte
- cAdvisor : metrics des containers
- docker-exporter: metrics Docker (expérimental)
- Prometheus: scrape / pull les metrics exposées
- AlertManager : déclenche des alertes / notifications
- Grafana : interface de visualisation

Prometheus stack : exemples d'implémentations

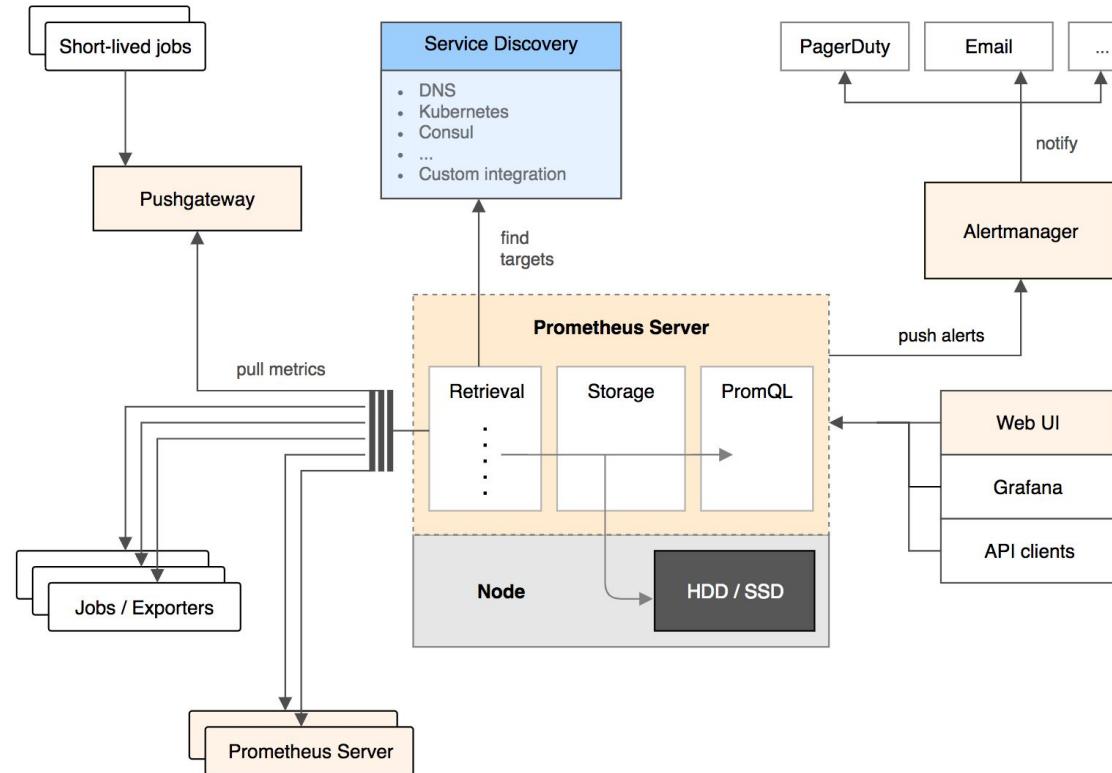
- Projet open source
- Gestion d'un hôte unique

<https://github.com/stefanprodan/dockprom>

- Gestion d'un Swarm

<https://github.com/stefanprodan/swarmprom>

Prometheus stack



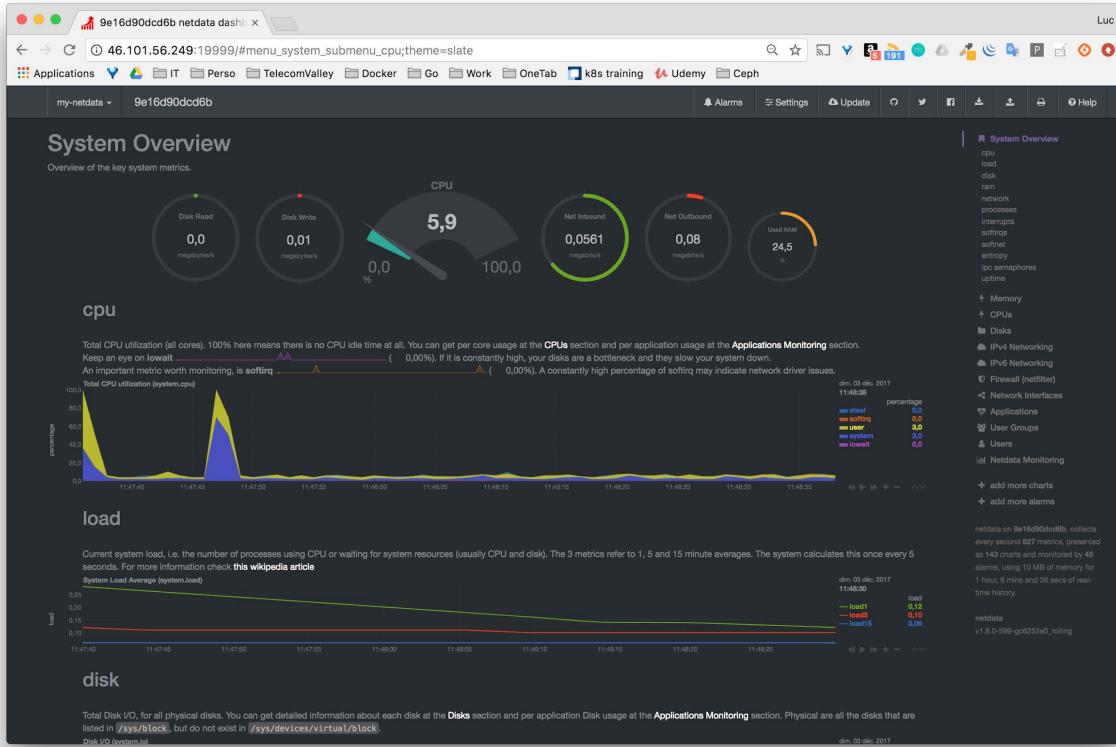
Démo

Netdata

- Daemon collectant des metrics en temps réel
- Plusieurs images Docker dont [titpetric/netdata](#)
 - metrics de la machine hôte
 - metrics des containers
 - gestion des alarmes

```
$ docker run -d --cap-add SYS_PTRACE \
-v /proc:/host/proc:ro \
-v /sys:/host/sys:ro \
-v /var/run/docker.sock:/var/run/docker.sock \
-p 19999:19999 titpetric/netdata
```

Netdata



Démo

Docker Enterprise Edition

Sommaire

- Présentation
- Installation
- Architecture HA
- Démo

Présentation

- Plateforme CaaS (Container as a Service)
- Solution pour la gestion du cycle de vie des applications
 - traditionnelles et micro-services
 - développées sous Linux ou Windows
- Disponible sur différentes infrastructures
 - On-premises (Linux, Windows Server 2016, IBM Z)
 - Cloud (Amazon AWS, Microsoft Azure)
- Plateforme partagée par les développeurs et les opérateurs

Présentation : les fonctionnalités clés

- Création d'images
 - scanning de sécurité
 - automatisation
- Distribution
 - Content Trust
 - cache
- Execution de containers
 - RBAC
 - interface web de gestion

Présentation : différents orchestrateurs

- Swarm “Classic”
- Swarmkit (Swarm mode)
- Kubernetes

Présentation : les différentes offres / fonctionnalités

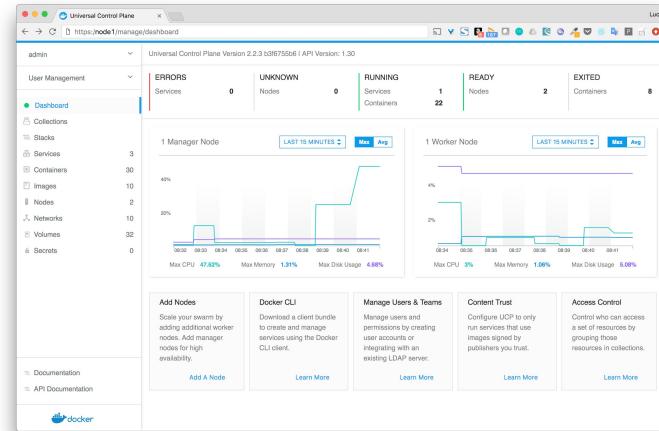
	EE BASIC SUPPORTED CONTAINER RUNTIME	EE STANDARD ENTERPRISE MANAGEMENT FEATURES FOR SINGLE TEAM USE	EE ADVANCED ADDITIONAL CAPABILITIES FOR MULTI-TEAM USE, POLICY-BASED AUTOMATION
Secure container engine with networking, security, and storage	✓	✓	✓
Docker Certified infrastructure, plugins and containers	✓	✓	✓
Private image registry with caching		✓	✓
Integrated app and cluster management across Swarm and Kubernetes		✓	✓
Enhanced RBAC, LDAP / AD support		✓	✓
Integrated secrets management, image signing policy		✓	✓
Secure multi-tenancy with node-based isolation			✓
Policy-based, automated image promotions			✓
Image mirroring across registries			✓
Image security scanning and continuous vulnerability scanning			✓

Release tous les 3 mois

Support d'un an pour chaque release

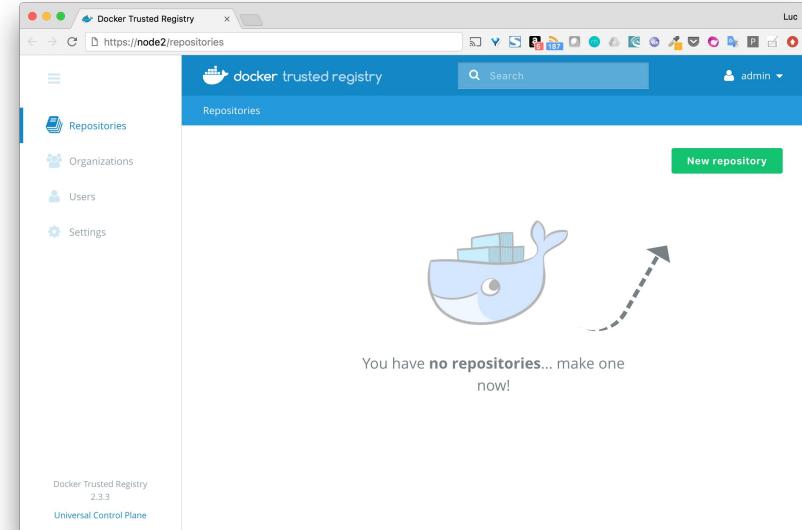
Présentation : UCP

- Universal Control Plane
- Ensemble de composants au dessus d'un Swarm
- Fournit une interface pour la gestion unifiée de l'ensemble des ressources
 - Node, Stack, Service, Secret, Config, ...
- Gestion des utilisateurs
 - organisation / team / users
- Gestion des applications



Présentation : DTR

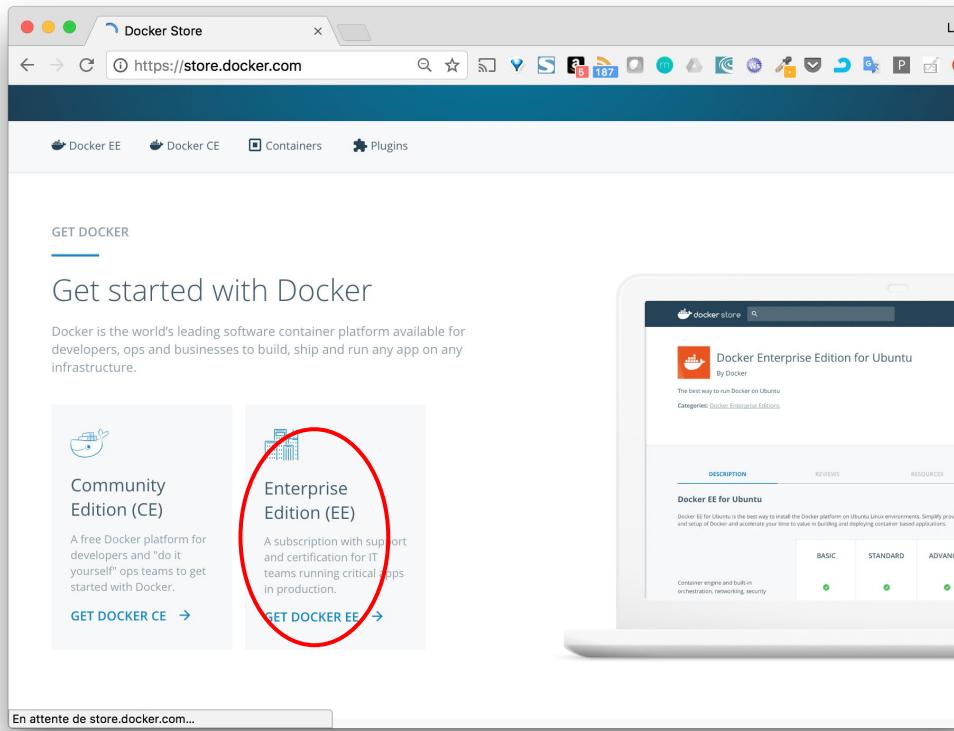
- Registry privé
- Stockage d'images Windows et Linux
- Gestion du backend de stockage
- Garbage collection
- Gestion des utilisateurs
 - organisation / team / users
- Docker Security Scanning
 - niveau avancé



Installation : différentes options

- Manuelle sur des serveurs Linux
 - Ubuntu, CentOS, SUSE, Oracle Linux, IBM Z, ...
 - <https://docs.docker.com/datacenter/install/linux/>
- Automatisée sur Amazon AWS
 - <https://docs.docker.com/datacenter/install/aws/>
- Automatisée sur Microsoft Azure
 - <https://docs.docker.com/datacenter/install/azure/>

Installation : depuis le Docker store



<https://store.docker.com>

Installation : multi plateforme, OS, architecture

PLATFORMS

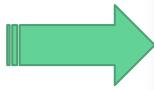
- Cloud
- Server

OPERATING SYSTEMS

- Linux
- Windows

ARCHITECTURES

- ARM32
- ARM64
- IBM POWER
- IBM Z
- x86-64



Explore - Docker Store

Cloud

Docker Enterprise Edition for AWS
By Docker

A one click template to quickly deploy Docker on Amazon EC2.

Edition Docker Certified Linux x86-64

Docker Enterprise Edition for Azure
By Docker

A one click template to quickly deploy Docker on Microsoft Azure.

Edition Docker Certified Linux x86-64

Explore - Docker Store

Server

Docker Enterprise Edition for CentOS
By Docker

The best way to run Docker on CentOS

Edition Docker Certified Linux x86-64

Docker Enterprise Edition for IBM POWER
By Docker

The best way to run Docker on IBM POWER

Edition Docker Certified Linux IBM POWER

Docker Enterprise Edition for IBM Z
By Docker

The best way to run Docker on IBM Z

Edition Docker Certified Linux IBM Z

Docker Enterprise Edition for Oracle Linux
By Docker

The best way to run Docker on Oracle Linux

Edition Docker Certified Linux x86-64

Docker Enterprise Edition for Ubuntu
By Docker

The best way to run Docker on Ubuntu

Edition Docker Certified Linux x86-64

Docker Enterprise Edition for Red Hat Enterprise Linux
By Docker

The best way to run Docker on Red Hat Enterprise Linux

Edition Docker Certified Linux IBM Z

Docker Enterprise Edition for SUSE Linux Enterprise Server
By Docker

The best way to run Docker on SUSE Linux Enterprise Server

Edition Docker Certified Linux x86-64

Docker Enterprise Edition for Windows Server 2016
By Docker

Native Docker containers on Windows Server 2016

Edition Docker Certified Windows x86-64

Installation : repository Docker EE

The screenshot shows a web browser window titled "Setup Instructions - Docker Store". The URL in the address bar is <https://store.docker.com/profiles/luci/content/sub-a0739a38-7fd1-4227-a980-0d885b94ad23>. The page header includes the "docker store" logo, a search bar, and navigation links for "Explore", "Publish", and "Feedback". A user profile for "luci" is visible on the right.

The main content area displays "My Content > Setup Instructions" for "Docker EE - Mon Nov 6 2017". The left sidebar lists "Get Docker EE" sections for "Docker EE on CentOS", "Docker EE on Oracle Linux", and "Docker EE on RHEL". Each section contains step-by-step instructions and shell commands for configuring the Docker EE repository. The "Docker EE on Oracle Linux" section is highlighted with a red oval.

The right sidebar is titled "Resources" and includes links for "License Key", "Download CVE Vulnerability Database for DTR version 2.2.5 or lower", and "Download CVE Vulnerability Database for DTR version 2.2.6 or higher". Below these are "Getting Started" and "User Guide" links. At the bottom, there is a box with the text "Copy and paste this URL to download your Edition:" followed by a URL: <https://storebits.docker.com/ee/m/su>.

A red arrow points from the text "URL du Docker EE repository associé au trial ou à la subscription" to the copied URL in the screenshot.

URL du Docker EE repository associé au trial ou à la subscription

Installation : pré-requis

- Daemon Docker
 - Entreprise Edition version >= 17.06.2-ee-8
- Spécification minimale de chaque node
 - Linux kernel version >= 3.10
 - 4G de RAM
 - 3G de disque
- Système d'exploitation
 - CentOS 7
 - Red Hat Enterprise Linux 7.0, 7.1, 7.2, ou 7.3
 - Ubuntu 14.04 LTS ou 16.04 LTS
 - SUSE Linux Enterprise 12
 - Oracle Linux 7.3

<https://docs.docker.com/ee/ucp/admin/install/system-requirements/>

Installation sur Ubuntu

Les étapes

- Souscription à une licence Docker Enterprise
- Installation de Docker-EE sur chaque node
- Installation de UCP sur l'un des nodes
- Configuration de la licence
- Ajout de nodes supplémentaires (managers / workers)
 - 3 managers au total pour un environnement HA
- Installation de la registry DTR
- Ajout de replicas supplémentaires
 - 3 replicas DTR au total pour un environnement HA

Installation de Docker EE

```
$ sudo apt-get update  
  
$ sudo apt-get install apt-transport-https ca-certificates curl software-properties-common  
  
$ curl -fsSL ${DOCKER_EE_URL}/ubuntu/gpg | sudo apt-key add -  
  
$ sudo add-apt-repository "deb [arch=amd64] ${DOCKER_EE_URL}/ubuntu $(lsb_release -cs) stable-17.06"  
  
$ sudo apt-get update  
  
$ sudo apt-get install docker-ee
```

Installation de UCP (Universal Control Plane)

```
$ sudo docker container run --rm -it --name ucp \
-v /var/run/docker.sock:/var/run/docker.sock \
docker/ucp:3.0.1 install \
--host-address ${NODE_IP} \
--interactive
Unable to find image 'docker/ucp:3.0.1' locally
INFO[0000] Your engine version 17.06.2-ee-11, build 06fc007 (4.4.0-116-generic) is compatible with UCP 3.0.1 (2f7c7e1)
Admin Username: admin
Admin Password:
Confirm Admin Password:
INFO[0184] Pulling required images... (this may take a while)
INFO[0184] Pulling docker/ucp-compose:3.0.1
INFO[0195] Pulling docker/ucp-swarm:3.0.1
INFO[0197] Pulling docker/ucp-hyperkube:3.0.1
INFO[0206] Pulling docker/ucp-interlock-extension:3.0.1
...
INFO[0068] Login to UCP at https://89.145.162.164:443
INFO[0068] Username: admin
INFO[0068] Password: (your admin password)
```

IP exposée aux autres noeuds du swarm

Le container docker/ucp lance les différents composants dans des containers

Accès à l'interface de gestion de UCP

The screenshot shows the Docker Enterprise Edition (EE) 2.0 dashboard for managing a Universal Container Platform (UCP) cluster. The URL is <https://ucp.techwhale.io/manage/dashboard>. The interface includes a sidebar with navigation links like Admin, Dashboard, User Management, Shared Resources, Kubernetes, and Swarm. The main content area displays Manager Nodes (1 Ready, 0 Errors, 0 Warnings) and Worker Nodes (0 Ready, 0 Errors, 0 Warnings). It also shows a timeline chart for CPU, Memory, and Disk usage over the last minute. Below this, there's a section for 0 Worker Nodes stating "There is no data for workers". The right side of the dashboard provides links for adding nodes, using Docker CLI, managing users & teams, configuring content trust, and setting access control.

Docker Enterprise Edition (EE) 2.0

Non sécurisé | <https://ucp.techwhale.io/manage/dashboard>

admin

Dashboard

User Management

Shared Resources

Kubernetes

Swarm

Manager Nodes

Ready 1 Errors 0
Warnings 0 Pending 0

Worker Nodes

Ready 0 Errors 0
Warnings 0 Pending 0

1 Manager Node

LAST MINUTE

30%
25%
20%

09:22:00 PM 09:22:15 PM 09:22:30 PM 09:22:45 PM

Max CPU **16.35%** Max Memory **16.15%** Max Disk Usage **31.87%**

0 Worker Nodes

LAST 15 MINUTES

Add Nodes

Scale your swarm by adding additional worker nodes. Add manager nodes for high availability.

Add a Node

Docker CLI

Download a client bundle to create and manage services using the Docker CLI client.

Learn More

Manage Users & Teams

Manage users and permissions by creating user accounts or integrating with an existing LDAP server.

Learn More

Content Trust

Configure UCP to only run services that use images signed by publishers you trust.

Learn More

Access Control

Control who can access a set of resources by grouping those resources in collections.

Learn More

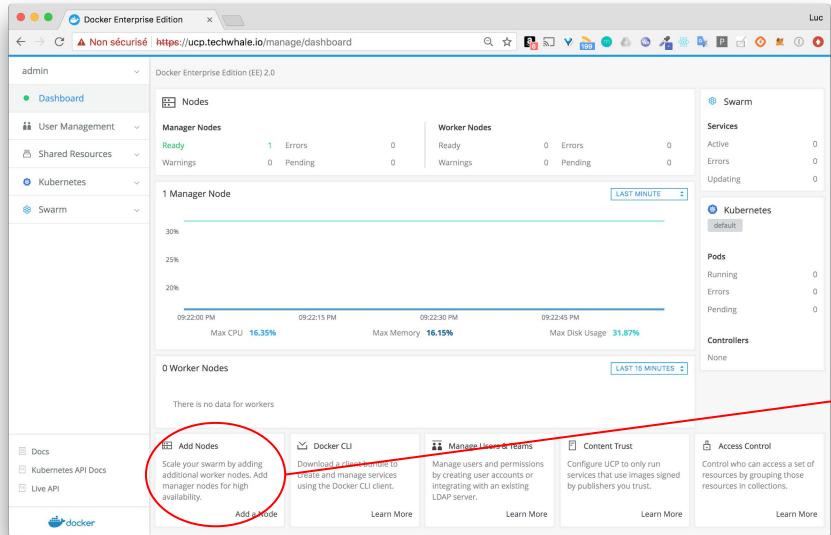
Docs

Kubernetes API Docs

Live API

docker

Ajout d'un node



Docker Enterprise Edition (EE) 2.0

Nodes

Manager Nodes

Ready	Errors	Pending	Warnings
1	0	0	0

Worker Nodes

Ready	Errors	Pending	Warnings
0	0	0	0

Swarm

Kubernetes

Pods

Controllers

LAST MINUTE

Max CPU **16.35%** Max Memory **16.15%** Max Disk Usage **31.87%**

0 Worker Nodes

There is no data for workers

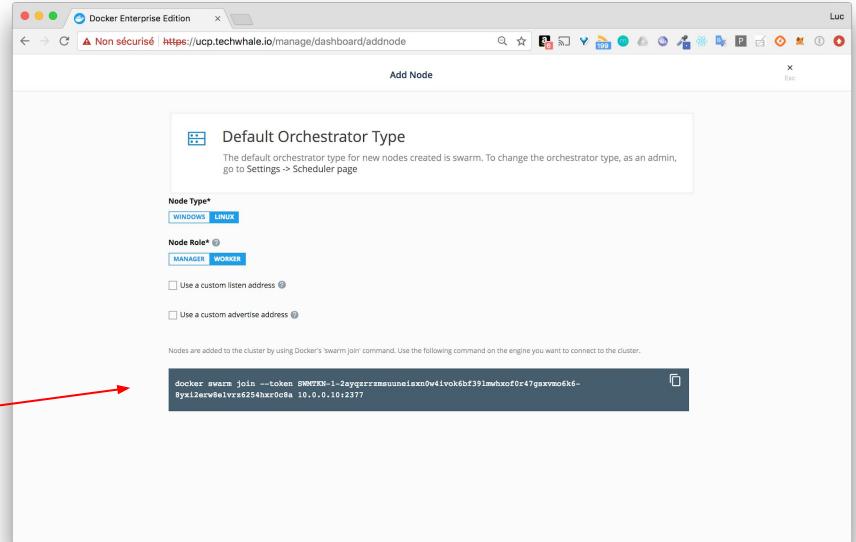
Add Nodes Scale your swarm by adding additional worker nodes. Add manager nodes for high availability. **Add Node**

Docker CLI Download a command-line tool to create and manage services using the Docker CLI client. **Learn More**

Manage Users & Teams Manage users and permissions by creating user accounts or integrating with an existing LDAP server. **Learn More**

Content Trust Configure UCP to only run services that use images signed by publishers you trust. **Learn More**

Access Control Control who can access a set of resources by grouping those resources in collections. **Learn More**



Add Node

Default Orchestrator Type

The default orchestrator type for new nodes created is swarm. To change the orchestrator type, as an admin, go to Settings->Scheduler page.

Node Type* **WINDOWS** **LINUX**

Node Role* **MANAGER** **WORKER**

Use a custom listen address

Use a custom advertise address

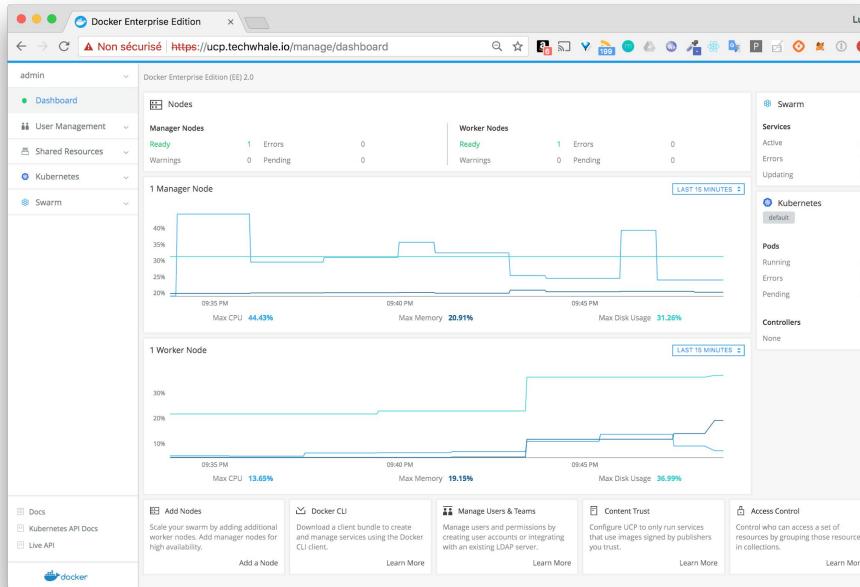
Notes are added to the cluster by using Docker's `swarm join` command. Use the following command on the engine you want to connect to the cluster.

```
docker swarm join --token SWMTKN-1-2zygurrrzmsuueisxn0w4ivok6bf391mhxf0r47gxvmo6k6-8yxi2erw8elvrs6254hxrc8a 10.0.0.10:2377
```

Ajout d'un node

```
ubuntu@ucp2:~$ sudo docker swarm join --token  
SWMTKN-1-2ayqzrrzmsuuneisxn0w4ivok6bf39lmwhxof0r47gsxvmo6k6-8yxi2erw8e1vrz6254hx  
r0c8a \  
10.0.0.10:2377
```

This node joined a swarm as a worker.



Installation de DTR (Docker Trusted Registry)

Docker Enterprise Edition

Non sécurisé | <https://ucp.techwhale.io/manage/settings/>

Admin Settings

Docker Trusted Registry

This UCP cluster does not yet have a Docker Trusted Registry installed.

DTR EXTERNAL URL optional

UCP NODE ucp2

Assign a DTR replica ID

Disable TLS verification for UCP

Use a PEM-encoded TLS CA certificate for UCP

Run the following command against UCP using a client bundle to install DTR:

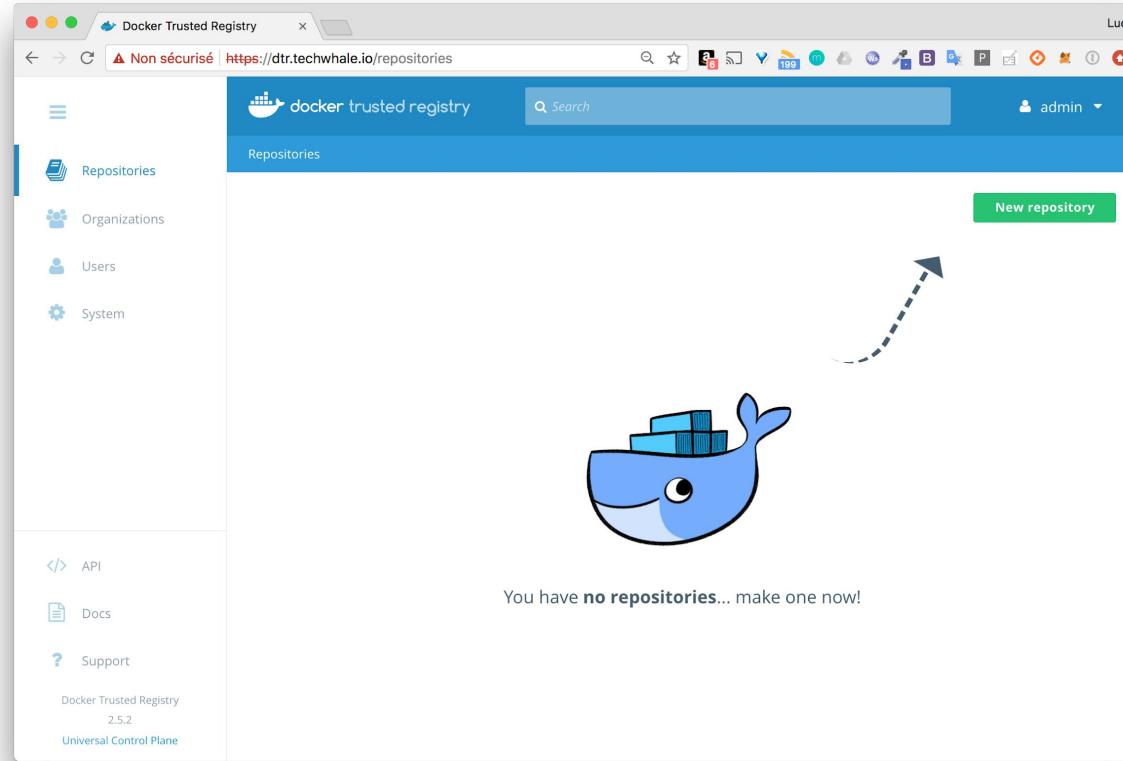
```
docker run -it --rm docker/dtr install \
--ucp-node ucp2 \
--ucp-username admin \
--ucp-url https://ucp.techwhale.io
```

[Learn How To Install DTR](#) [Copy To Clipboard](#)

Cancel Save

```
$ docker container run -it --rm \
docker/dtr install \
--ucp-node ${NODE_HOSTNAME} \
--ucp-insecure-tls
```

Accès à l'interface de gestion de DTR



Démo

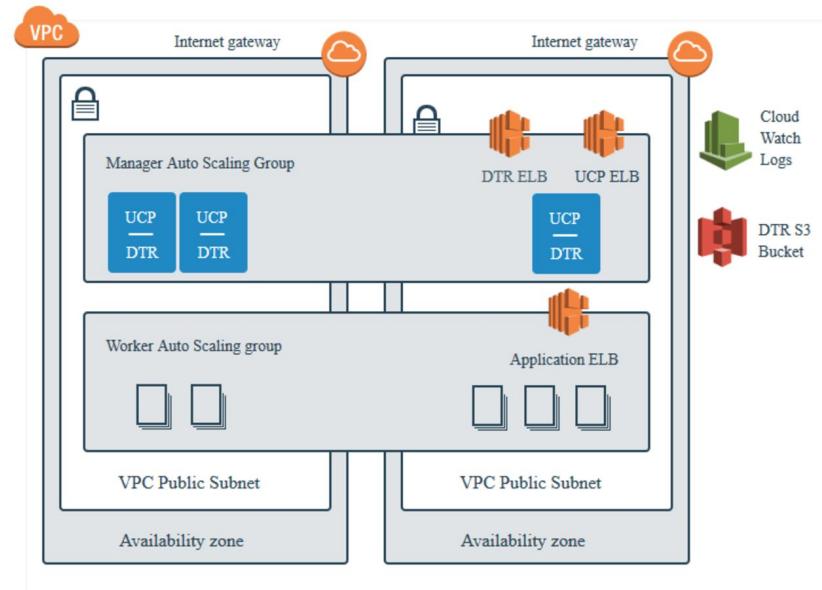
Installation : sur Amazon AWS

The screenshot shows the Docker store interface. At the top, there's a navigation bar with 'Explore', 'Publish', 'Feedback', and a user profile. Below it, a product card for 'Docker Enterprise Edition for AWS' by Docker is displayed. The card includes a small Docker logo icon, the product name, and a brief description: 'A one click template to quickly deploy Docker on Amazon EC2.' Below the description are buttons for 'EE Basic' and 'EE Std/Adv'. At the bottom of the card, there are tabs for 'DESCRIPTION', 'REVIEWS', and 'RESOURCES'. A note in a box states: 'You must subscribe to this product before you can review it.'

The screenshot shows the AWS Marketplace page for 'Docker EE for AWS (Standard/Advanced) - BYOL'. It features a large Docker logo. Below it, the product name and a brief description: 'Docker EE for AWS (Standard/Advanced) is an integrated end-to-end platform for agile application development and management security. Docker Enterprise Edition provides a simple and low cost way to build, ship and run distributed applications at scale in production in a highly scalable Amazon AWS'. A 'Continue' button is visible on the right. The page also displays sections for 'Customer Rating', 'Latest Version', 'Operating System', 'Delivery Methods', 'Pricing Information', 'For Region', 'Delivery Methods', and 'AWS Services Required'.

Installation : sur Amazon AWS

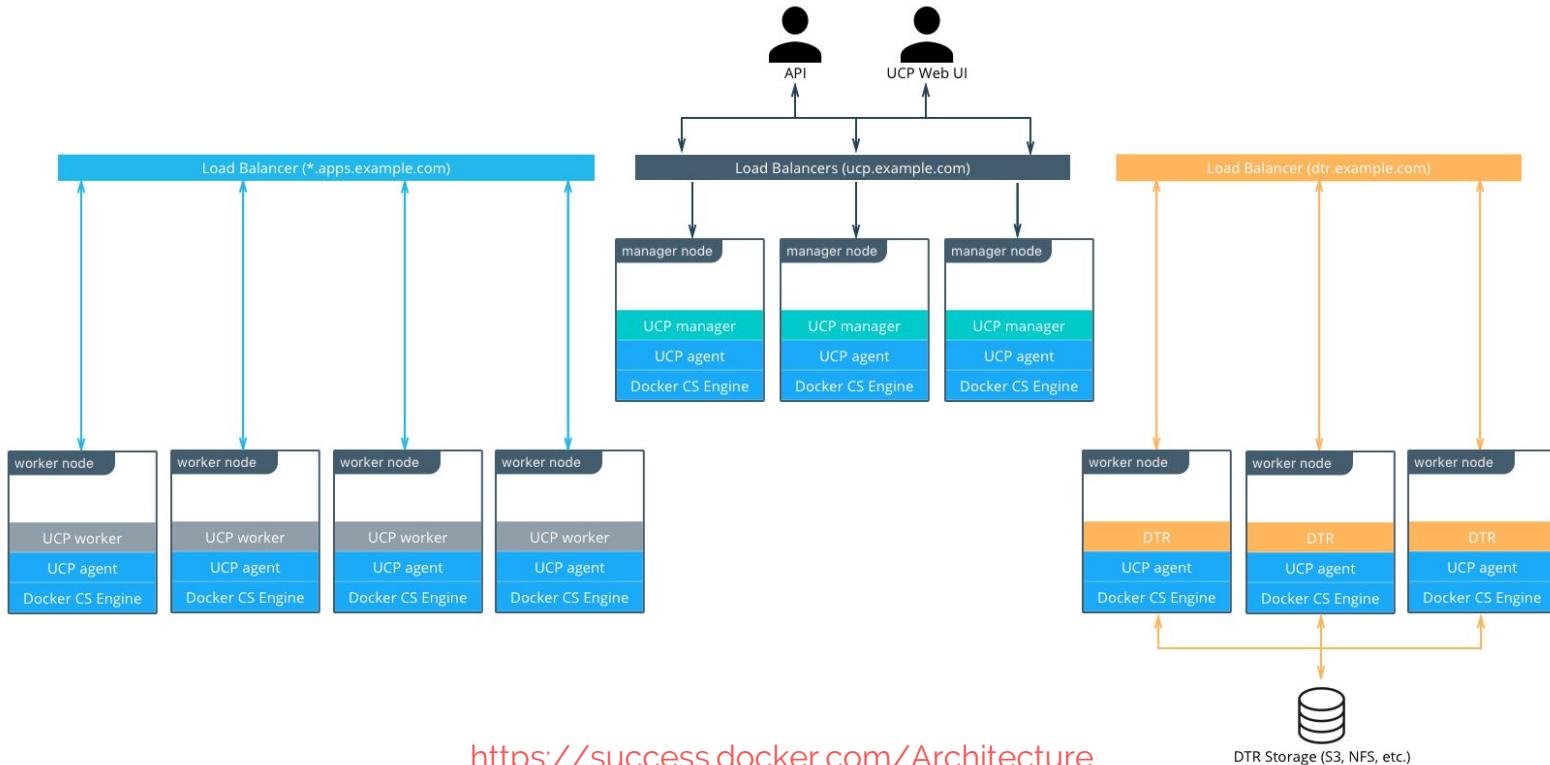
- Template CloudFormation (IaC)
- Intégration avec les produits AWS
 - AZ, EC2, VPC, ELB, CloudWatch, S3, ...
- Architecture HA
 - 3 managers UCP
 - 3 replicas DTR



Architecture HA : préconisations

- 3 managers UCP
- 3 replicas DTR
- 3 load balancers
 - managers UCP
 - DTR
 - applications
- Un nombre de worker en fonction de la charge applicative

Architecture HA de production



<https://success.docker.com/Architecture>

Démo

Démo en ligne: <https://dockertrial.com>

Un mois d'essai gratuit <https://store.docker.com/editions/enterprise/docker-ee-trial>

L'écosystème

