Docker

Présentation de l'environnement

Ce document est une introduction au système de conteneurisation Docker.
Il s'appuie sur les versions 1.13 et ultérieures.

Après une présentation du concept, nous découvrons les commandes de base permettant de gérer des applications isolées dans leurs contextes.

Bonne lecture…

Pierre ROYER

Architecte systèmes d'information

https://pierreau.fr/pro/

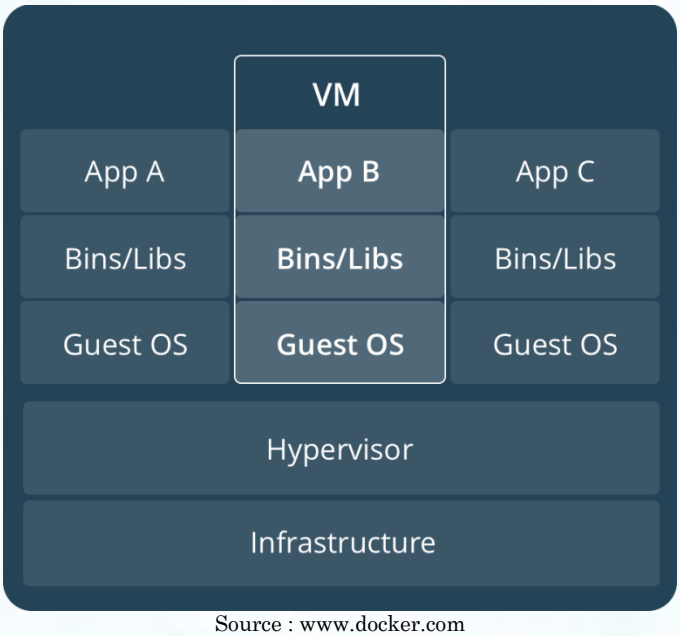
INDEX

PREAMBULE

I. Virtualisation systèmes

La virtualisation de serveurs est un concept qui existe depuis 1972 : IBM le pratiquait depuis 1972 ; avec des mainframes VM/370.

Les machines virtuelles partagent un même serveur physique, alors que les conteneurs partagent le même système d'exploitation.

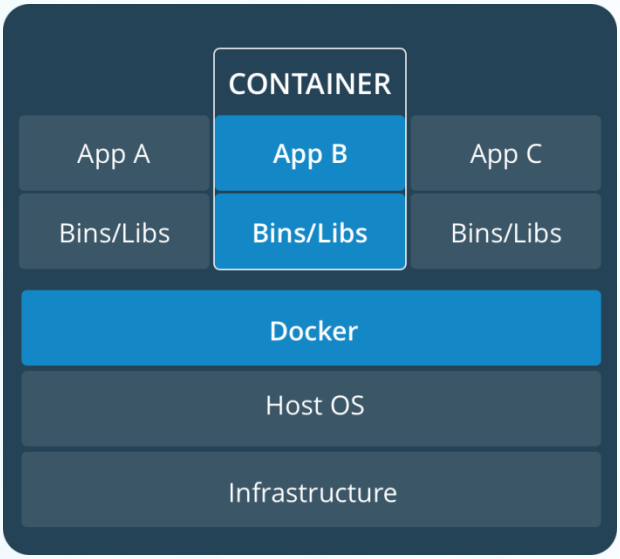


Source : www.docker.com

II. Virtualisation applicative

Contrairement aux machines virtuelles, les conteneurs Docker n'incluent pas de système d'exploitation, mais permettent d'isoler des applications dans des contextes.

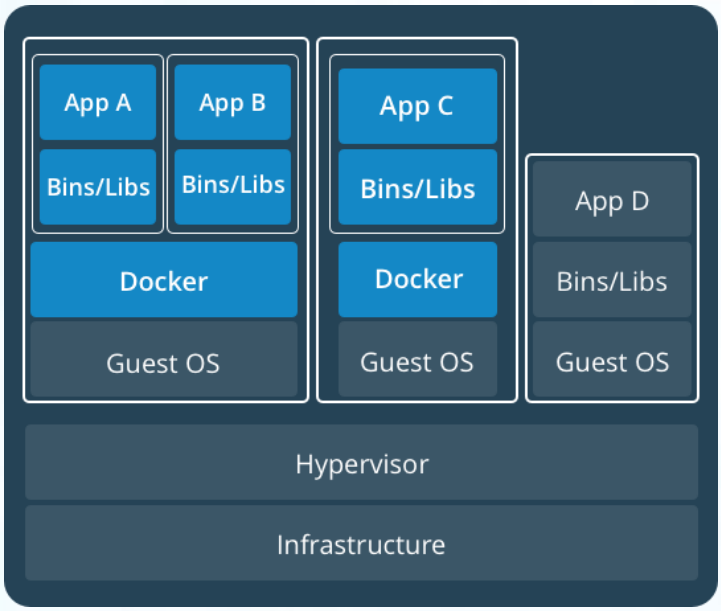
Ce principe d'isolation existe depuis plusieurs années, et a été implémenté entre autres sur Sun Solaris 10 (Oracle) en 2005. Les conteneurs Solaris sont appelés « zones ».



Source : www.docker.com

III. Virtualisation virtuelle

La virtualisation système et applicative sont deux concepts différents, mais qui peuvent être complémentaires. Il est ainsi possible de contenir des conteneurs dans un système virtualisé :



Source : www.docker.com

Si l'on pousse le raisonnement, il est tout à fait envisageable d'exécuter Docker… dans un autre conteneur Docker, lancé au sein d'une machine virtuelle qui est hébergée sur un serveur physique…

IV. Présentation de Docker

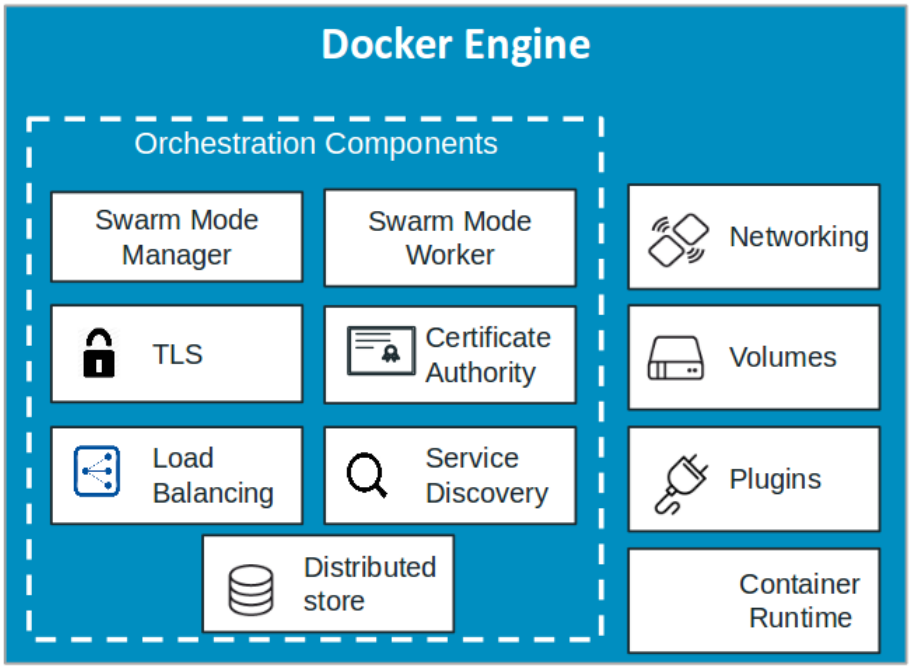
Docker est un logiciel sous licence Apache 2.0, qui a été distribué en tant que projet open source à partir de mars 2013.

Docker permet d'automatiser le déploiement d'applications dans des conteneurs logiciels. Il peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.

Docker utilise des fonctionnalités natives de Linux comme :

- son noyau
- ses Cgroups (control groups) : fonctionnalités pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.)
- une surcouche de LXC (LinuX Containers).

Outre le fait d'isoler un système de fichiers et des réseaux, Docker Engine intègre des outils d'orchestration, que nous aborderons dans le chapitre dédié à Swarm :



MAIS ALLONS DE CE PAS AU CŒUR DU SUJET…

INSTALLATION

Docker est distribué sous deux licences :

- Docker EE (Enterprise Edition) qui offre un support payant. Trois versions existent : Basic, Standard et Advanced.
- Docker CE (Community Edition), sans support.

Ce document se base sur Docker CE, et un système d'exploitation CentOS 7.5

I. Procédure

Paramétrage des dépôts de docker :

```
[root@CentOS ~]# yum install -y yum-utils
[root@CentOS ~]# yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
[root@CentOS ~]# yum-config-manager --enable docker-ce-edge
[root@CentOS ~]# yum makecache fast
```

Recherche de la dernière version :

```
[root@CentOS ~]# yum list docker-ce.x86_64  --showduplicates |sort -r
docker-ce.x86_64            18.03.1.ce-1.el7.centos             docker-ce-stable
```

Installation :

```
[root@CentOS ~]# yum install docker-ce-18.03.1.ce-1.el7.centos
[root@CentOS ~]# systemctl start docker
[root@CentOS ~]# systemctl enable docker
```

II. Post-installation

Vérifications de l'installation :

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

Vérification des interfaces et réseaux virtuels :

Lors de l'installation, une interface virtuelle sont créée :

```
[root@CentOS ~]# ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
```

Docker propose 3 types de réseaux par défaut, visibles via la commande suivante :

```
[root@CentOS ~]# docker network ls
NETWORK ID          NAME                DRIVER
7fca4eb8c647        bridge              bridge
9f904ee27bf5        none                null
cf03ee007fb4        host                host
```

Le réseau "bridge" utilise l'interface réseau « docker0 », qui est définie dans la machine hôte.

Les informations (Containers, Subnet, Gateway, etc…) du réseau « bridge » sont consultables via :

```
[root@CentOS ~]# docker network inspect bridge
```

PREMIERS PAS

I. Apperçu des commandes

```
[root@CentOS ~]# docker --help

Management Commands:
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  volume      Manage volumes

Commands:
  attach      Attach to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

II. Installation d'un conteneur

A. Récupération de l'image

Les images de conteneurs peuvent contenir aussi bien des applications, que d'autres distributions Linux. Les exemples suivants vont se baser sur une image Debian, ayant une réputation d'au moins 10 étoiles :

```
[root@CentOS ~]# docker search --filter stars=10 debian
[root@CentOS ~]# docker pull debian
```

Liste des images locales :

```
[root@CentOS ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest              978d85d02b87        2 weeks ago         123 MB
```

B. Création d'un conteneur

Créer un conteneur à partir de l'image Debian, nommé « DebCont0 », accessible avec un terminal (-t) interactif (-i)

```
[root@CentOS ~]# docker create --name DebCont0 debian
```

Vérification, en visualisant la liste des conteneurs :

```
[root@CentOS ~]# docker ps -a
CONTAINER ID   IMAGE    COMMAND       CREATED             STATUS          PORTS       NAMES
2e2d14b89c2b   debian   "/bin/bash"   4 seconds ago       Created                     DebCont0
```

C. Démarrage de conteneurs

Démarrer un conteneur déjà existant en attachant les signaux STDIN/OUT ERR en mode interactif :

```
[root@CentOS ~]# docker start -a -i DebCont0
```

Renommer un conteneur :

```
[root@CentOS ~]# docker rename DebCont0 ContDeb0
```

Démarrer un conteneur nommé « ContDeb1 » en tache de fond (option -d), depuis l'image « debian » avec un terminal en mode interactif :

```
[root@CentOS ~]# docker run -dti --name ContDeb1 debian
```

L'option **--rm** permet de détruire le conteneur une fois utilisé et stoppé

```
[root@CentOS ~]# docker run --rm --name ContDeb2 debian df -h
```

Exécution d'une commande dans le conteneur Debian déjà démarré

```
[root@CentOS ~]# docker exec -ti ContDeb1 cat /etc/issue
Debian GNU/Linux 8 \n \l
```

Démarrer un conteneur et accéder à son terminal

```
[root@CentOS ~]# docker run -ti debian
root@8fcceab47ae2:/# cat /etc/issue
Debian GNU/Linux 8 \n \l
```

Accéder à un conteneur démarré

```
[root@CentOS ~]# docker attach ContDeb1
```

Quitter un conteneur

```
root@8fcceab47ae2:/# <CTRL>p <CTRL>q
```

D. Arrêt / suppression de conteneurs / images

Accéder au shell d'un conteneur déjà démarré

```
[root@CentOS ~]# docker exec -ti ContDeb1 /bin/bash
```

Arrêt normal d'un conteneur

```
[root@CentOS ~]# docker stop ContDeb1
```

Arrêt forcé d'un conteneur

```
[root@CentOS ~]# docker kill ContDeb1
```

Effacer un conteneur

```
[root@CentOS ~]# docker rm ContDeb1
```

Effacement forcé d'un conteneur (même démarré)

```
[root@CentOS ~]# docker rm -f ContDeb1
```

Suppression de tous les conteneurs

```
[root@CentOS ~]# docker rm $(docker ps -a -q)
```

Effacer une image

```
[root@CentOS ~]# docker rmi -f debian
```

Suppression de toutes les images

```
[root@CentOS ~]# docker rmi $(docker images -q)
```

E. Paramètres de démarrage

Création d'un conteneur nginx nommé ContNgx1, limité à 1Go de swap, 512 Mo de RAM, avec mappage du dossier host local vers le dossier du conteneur /usr/share/nginx/html, et mappage du port externe 8080 vers le port interne 80

```
[root@CentOS ~]# docker run -dti --name ContNgx1 --memory-swap 1024m -m 512m -v $(pwd):/usr/share/nginx/html -p 8080:80 nginx
```

Nous pouvons vérifier que nginx répond bien à partir du serveur hôte (http://localhost:8080/)

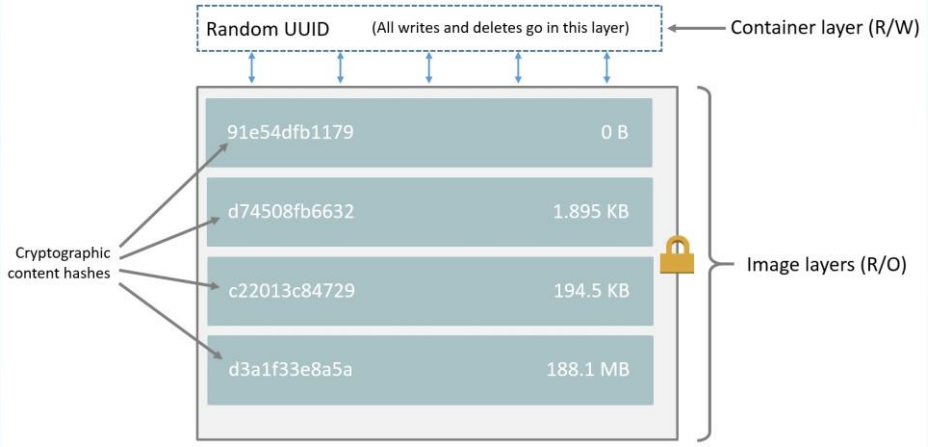
Modification de quelques paramètres, à la volée, une fois le conteneur démarré :

```
[root@CentOS ~]# docker update --memory-swap 2048m -m 1024m ContNgx1
```

IMAGES & LAYERS & CONTAINER

I. Modèle en couches

Un conteneur Docker est en réalité une couche (layer) contenant un système de fichiers modifiable, s'appuyant sur plusieurs autres couches en lecture seule, qui se superposent.

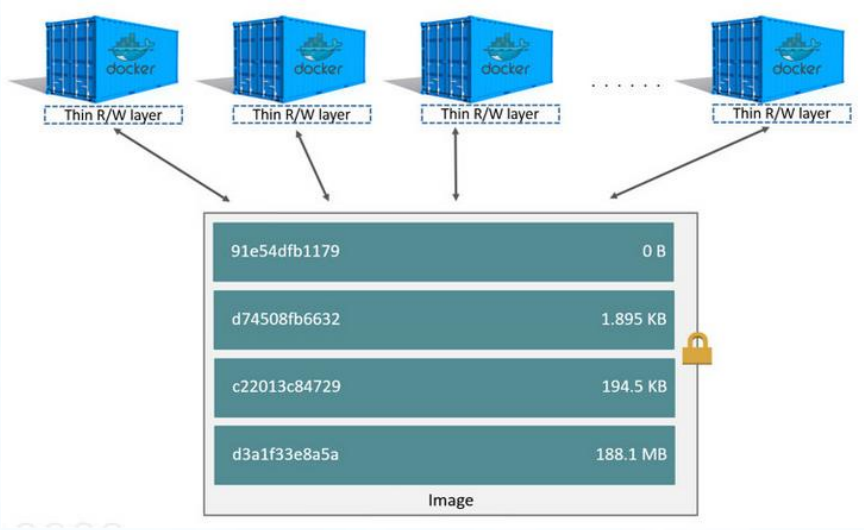


Source : www.docker.com

La liste de l'ensemble des couches formant une image est affichée via

```
[root@CentOS ~]# docker history <image>
```

A chaque création d'un nouveau conteneur basé sur une même instance d'image, seule la dernière couche en lecture/écriture est créée, ce qui optimise ainsi les ressources en stockage, et la rapidité d'instanciation d'un nouveau conteneur :



Source : www.docker.com

II. Pilotes et systèmes de fichiers

La gestion de ces couches est assurée par un driver Docker, qui est associé à un type de filesystem.

Il existe plusieurs types de drivers, qui sont pris en charge pas différents systèmes hôtes. Parmi les plus fréquents :

Driver	(sup)porté par

overlay / overlay2 (Linux kernel >= 4.0) / AUFS	ext4 / xfs
Btrfs	Btrfs
devicemapper	LVM
zfs	zfs

Le choix de ces pilotes peut être spécifié soit :

1/ dans la variable DOCKER_OPTS définie dans le fichier /etc/default/docker

2/ avec la commande :

```
[root@CentOS ~]# dockerd --storage-driver=devicemapper &
```

III. Conversion d'un conteneur en image

Imaginons que :

- l'on récupère une image Debian
- on créé un conteneur nommé « debian_nginx.updated »
- on le mette à jour, en installant Nginx
- on veuille créer une image nommée « debian/nginx:updated ».

La séquence sera la suivante :

```
[root@CentOS ~]# docker pull debian:latest
[root@CentOS ~]# docker run -dti --name debian_nginx.updated debian:latest
[root@CentOS ~]# docker exec -ti debian_nginx.updated apt-get -y update
[root@CentOS ~]# docker exec -ti debian_nginx.updated apt-get install -y nginx
```

On contrôle l'existence d'un différentiel entre l'image d'origine et le conteneur :

```
[root@CentOS ~]# docker diff debian_nginx.updated |head -10
...
[root@CentOS ~]# docker commit debian_nginx.updated  debian/nginx:updated
```

On contrôle la création de l'image :

```
[root@CentOS ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian/nginx        updated             f3234c725502        6 seconds ago       195 MB
debian              latest              8cedef9d7368        6 weeks ago         123 MB
```

Il est possible de renommer une image :

```
[root@CentOS ~]# docker tag debian/nginx:updated debian/nginx:latest
```

IV. Options de conversion

Il est possible d'intégrer des options de création d'image :

```
[root@CentOS ~]# docker commit -c "EXPOSE 80" debian_nginx.updated  debian/nginx:updated
```

…lors du lancement du conteneur construit à partir de cette image, le port 80 sera exposé.

V. Archivage d'une image

Les images peuvent être archivées (et compressées) :

```
[root@CentOS ~]# docker save debian/nginx:updated | gzip > debian_nginx:updated.tar.gz
```

…puis restaurées :

```
[root@CentOS ~]# docker load < debian_nginx:updated.tar.gz
```

DOCKERFILE

Les Dockerfiles sont des scripts permettant d'automatiser et d'industrialiser la création de conteneurs, à partir d'une image.

Exemple de script
```
[root@CentOS ~]# vi Dockerfile
```

```
# Image de référence
 FROM          debian:latest

# Données méta
 LABEL  fr.pierreau.dept="R&D" \
               fr.pierreau.version="1.0" \
               fr.pierreau.environment="production" \
               fr.pierreau.released="November 28, 2016"

# Exécution d'une commande dans le conteneur après sa création
 RUN           apt-get –y update
 RUN           apt-get install -y nginx

# Copie de fichiers du host vers le conteneur
 COPY   nginx.conf /etc/nginx/nginx.conf
 COPY   service_start.sh /home/docker/script/service_start.sh
 RUN           chmod 744 /home/docker/script/service_start.sh

# Affectation d'une valeur à une variable d'environnement
ENV            HOME    /root

# Répertoire utilisé lors des commandes ENTRYPOINT et/ou CMD
 WORKDIR       /home/docker

# Commande (non surchargeable par docker run) lancée au démarrage du conteneur
 ENTRYPOINT /home/docker/script/service_start.sh

# Points de montage créé dans le conteneur
 VOLUME         ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d",
"/var/log/nginx", "/var/www/html"]

# Port réseau ouvert à l'extérieur du conteneur
 EXPOSE 80

# Commande (surchargeable par docker run) lancée au démarrage du conteneur
 CMD           ping pierreau.fr
```

En considérant que le fichier Dockerfile est dans le répertoire courant, la création de l'image nommée « debian:nginx » via ce fichier s'effectue avec la commande

```
[root@CentOS ~]# docker build -t 'debian:nginx' .
```

On peut spécifier le chemin du dossier d'un autre dockerfile de cette manière

```
[root@CentOS ~]# docker build -f /[path]/Dockerfile .
```

L'image créée sera constituée de plusieurs couches, visibles avec la commande

```
[root@CentOS ~]# docker history debian:nginx
```

VOLUMES

Un volume permet à des conteneurs de stocker des informations persistantes. Il peut être partagé entre plusieurs containers.

I. Création d'un volume

Création d'un volume :

```
[root@CentOS ~]# docker volume create DataVolume1
```

Vérifications :

```
[root@CentOS ~]# docker volume ls
DRIVER              VOLUME_NAME
local               DataVolume1

[root@CentOS ~]# docker volume inspect DataVolume1
```

Partage de ce volume dans deux dossiers « ExtVolume » montés dans 2 conteneurs « DebCont1 » (effacé automatiquement lors de son arrêt), et « DebCont2 » :

```
[root@CentOS ~]# docker run -ti --rm --name DebCont1 -v DataVolume1:/ExtVolume debian
root@6c780805a52c:/# ls ExtVolume
root@6c780805a52c:/# touch ExtVolume/test
root@6c780805a52c:/# ls ExtVolume
test
<CTRL P> <CTRL Q>
[root@CentOS ~]# docker stop DebCont1

[root@CentOS ~]# docker run -ti --name DebCont2 -v DataVolume1:/ExtVolume debian
root@9825b015920a:/# ls ExtVolume
test
```

II. Création à la volée

Il est possible de créer un volume persistant dès la création ou le démarrage d'un conteneur :

```
[root@CentOS ~]# docker create --name DebCont0 -t -i -v DataVolume1:/ExtVolume debian

[root@CentOS ~]# docker run -ti -name DebCont1 -v DataVolume1:/ExtVolume debian
```

III. Droits sur les conteneurs

La lecture seule sur un volume est assurée par le paramètre read-only **:ro**

```
[root@CentOS ~]# docker run -ti -name DebCont1 -v DataVolume1:/ExtVolume:ro debian
```

IV. Gestion des volumes

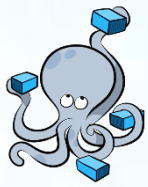
Supprimer un volume :

```
[root@CentOS ~]# docker volume rm DataVolume1
```

Suppression des volumes non utilisés par des conteneurs :

```
[root@CentOS ~]# docker volume prune
```

DOCKER COMPOSE



Docker Compose est un outil pour définir et exécuter des applications Docker multi-conteneurs interdépendants. Il utilise :

- La définition de l'environnement via un Dockerfile
- la définition des services contenus dans un fichier « docker-compose.yml »
- la commande « docker-compose up » pour démarrer l'environnement isolé

Pour utiliser Docker Compose, il faut sans doute l'installer :

I. Installation

Identifier la dernière version en date sur l'URL https://github.com/docker/compose/releases/

Définir la variable d'environnement (optionnel) en fonction de la dernière version :

```
[root@CentOS ~]# dockerComposeVersion="1.13.0"
```

Installer Docker Compose :

```
 [root@CentOS ~]# curl -L
https://github.com/docker/compose/releases/download/$dockerComposeVersion/docker-compose-
`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

Définir les droits d'execution :

```
[root@CentOS ~]# chmod +x /usr/local/bin/docker-compose
```

II. Définition de l'environnement

A. Dockerfile

```
[root@CentOS ~]# vi Dockerfile
```

```
# Image de référence
 FROM          nginx:alpine

# Copie de fichiers du host vers le conteneur
 COPY   nginx.conf /etc/nginx/nginx.conf
```

B. docker-compose.yml

```
[root@CentOS ~]# vi docker-compose.yml
```

```
version: "3"
services:
    reverseproxy:
        image: reverseproxy
        ports:
            - "8080:8080"
            - "8081:8081"
        restart: always

    nginx:
        depends_on:
            - reverseproxy
        image: nginx:alpine
        restart: always

    apache:
        depends_on:
            - reverseproxy
        image: httpd:alpine
```

```
        restart: always
```

Le nombre d'instances de l'application est fixé à 4 :

```
[root@CentOS ~]# docker-compose scale app=4
```

Liste des conteneurs :

```
[root@CentOS ~]# docker-compose ps
```

Liste des images utilisées :

```
[root@CentOS ~]# docker-compose images
```

Arrêt des conteneurs gérés par Docker-Compose :

```
[root@CentOS ~]# docker-compose stop
```

Supprimer l'ensemble des conteneurs crées avec docker-compose :

```
[root@CentOS ~]# docker-compose rm
```

DOCKER MACHINE

Docker-machine est un outil qui permet de provisionner des machines physiques ou virtuelles, afin d'héberger des conteneurs. Ces machines auront les rôles de « Manager » et « Worker ».

I. Installation

Vérifier la dernière version sur Github : https://github.com/docker/machine/releases/

```
[root@CentOS ~]# base=https://github.com/docker/machine/releases/download/v0.15.0
[root@CentOS ~]# curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine
[root@CentOS ~]# install /tmp/docker-machine /usr/local/bin/docker-machine
```

Un outil d'aide aux commandes (completion) peut aussi être intallé :

```
[root@CentOS ~]# curl -L
https://raw.githubusercontent.com/docker/machine/v0.15.0/contrib/completion/bash/docker-
machine.bash -o /etc/bash_completion.d/docker-machine
[root@CentOS ~]# exec $SHELL -l
```

On vérifie la version installée :

```
[root@CentOS ~]# docker-machine version
```

La liste des options disponibles s'affiche avec la commande :

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

II. Création d'un cluster VirtualBox

A. Installer virtualbox

https://www.virtualbox.org/wiki/Linux_Downloads

```
[root@CentOS ~]# cd /etc/yum.repos.d/
[root@CentOS ~]# wget http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo
[root@CentOS ~]# yum update
[root@CentOS ~]# yum install VirtualBox-5.2
[root@CentOS ~]# VirtualBox

[root@CentOS ~]# yum install kernel-devel gcc make perl
[root@CentOS ~]# /sbin/vboxconfig
```

B. Création de 3 machines

L'opération consiste à mettre à disposition 1 Manager et 2 Workers :

```
[root@CentOS ~]# docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-
memory "2048" --virtualbox-disk-size "25000" manager
[root@CentOS ~]# docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-
memory "2048" --virtualbox-disk-size "25000" worker1
[root@CentOS ~]# docker-machine create -d virtualbox --virtualbox-cpu-count "2" --virtualbox-
memory "2048" --virtualbox-disk-size "25000" worker2
```

Contrôle :

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

Mises à jour des machines :

```
[root@localhost ~]# docker-machine upgrade manager
[root@localhost ~]# docker-machine upgrade worker1
[root@localhost ~]# docker-machine upgrade worker2
```

Récupérer l'adresse IP du manager :

```
[root@CentOS ~]# docker-machine ip manager
IP-Manager
```

Cette adresse va être utilisée pour initialiser un cluster Swarm, en se connectant en SSH à la machine « Manager » :

```
[root@CentOS ~]# docker-machine ssh manager
docker@manager:~$ docker swarm init --advertise-addr IP-Manager
```

Cette commande créée un « Node », défini le rôle de la machine « manager », et renvoie un token à utiliser pour déployer les workers. Copier la ligne générée avec la commande « docker swarm join… »

Se connecter sur les Workers, et coller cette commande afin de définir leurs rôles :

```
[root@CentOS ~]# docker-machine ssh worker1
docker@worker1:~$ docker swarm join --token abcd1234 IP-Manager:2377
docker@worker1:~$ exit

[root@CentOS ~]# docker-machine ssh worker2
docker@worker2:~$ docker swarm join --token abcd1234 IP-Manager:2377
docker@worker2:~$ exit
```

Déployer un conteneur nginx dans une machine :

```
[root@CentOS ~]# docker run --name web01 -d --hostname worker1 nginx
[root@CentOS ~]# docker ps
```

Arrêt d'une machine

```
[root@CentOS ~]# docker-machine stop worker1
```

Démarrage

```
[root@CentOS ~]# docker-machine start worker1
```

Définir la machine active

```
[root@CentOS ~]# docker-machine env worker1
```

Connaître la machine active

```
[root@CentOS ~]# docker-machine active
```

Suppression d'une machine

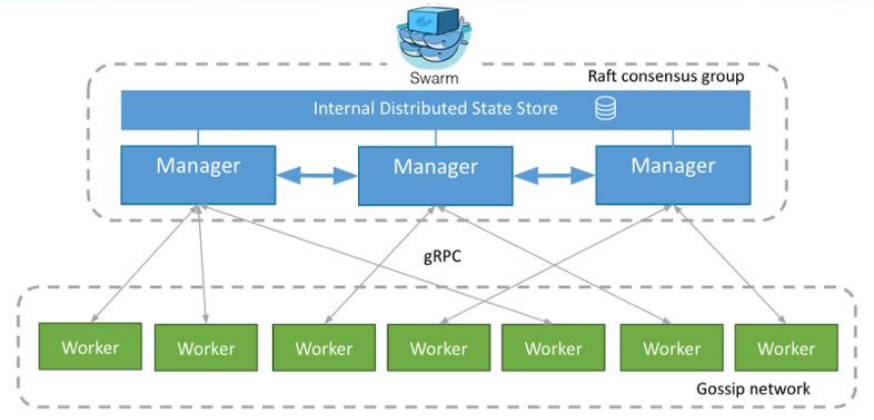
```
[root@CentOS ~]# docker-machine rm worker1
```

SWARM

I. Vue d'ensemble

Swarm est l'orchestrateur intégré nativement dans Docker depuis la version 1.12. Swarm repose sur les notions de nœuds « Manager », Manager « leader » (ou primaire), et « Worker ».

Le Manager maintient l'état du cluster et orchestre les tâches (tasks) sur les noeuds de type worker
Le Worker reçoit les tâches envoyées par les noeuds de type manager, et exécute les tâches.



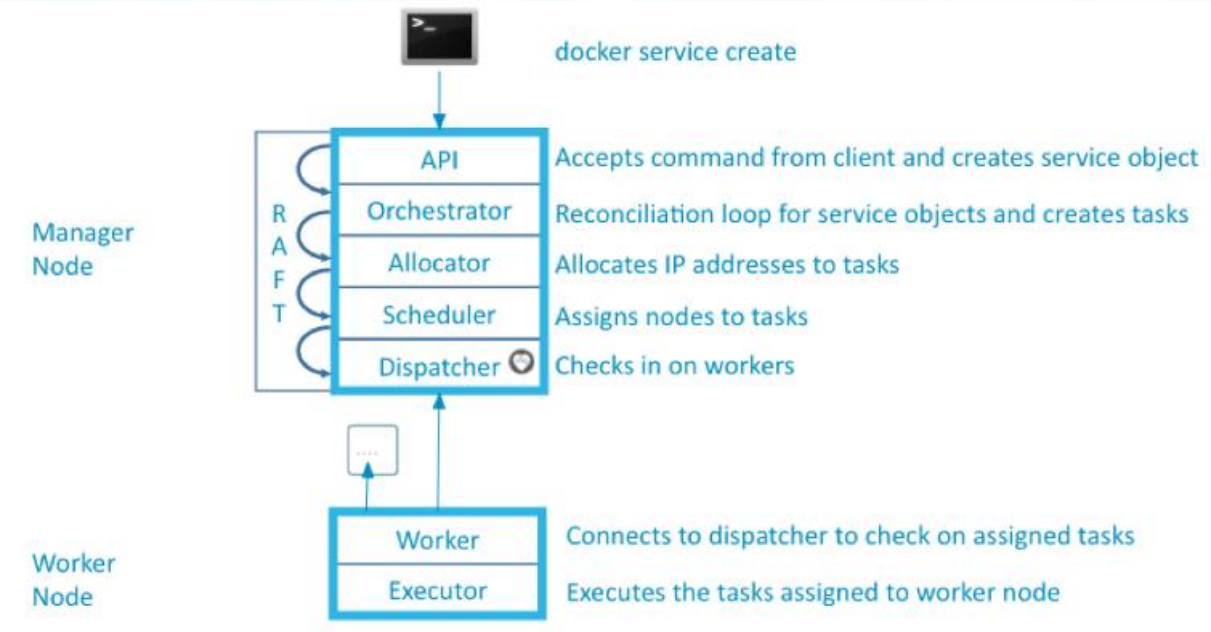
Le groupe de consensus **Raft** est un quorum permettant la haute disponibilité entre les nœuds managers. Les noeuds worker communiquent avec le(s) nœud(s) Manager(s) en utilisant le protocole **gRPC**, basé sur HTTP/2. Cette communication permet au Manager la gestion du versionning des conteneurs (un même niveau de version n'est pas obligatoire), la progression de leurs tâches, et un contrôle de disponibilité des Workers.

Le rôle des noeuds étant dynamique, si un Manager s'arrête de fonctionner, un Worker peut automatiquement basculer en Manager afin d'en assurer une redondance.

II. Briques techniques

Dans le vocabulaire Docker, un service correspond à un conteneur déployé dans un cluster. Le Manager gère le cycle de vie de ces services (déploiement / exécution / résilience).

Gestion des services

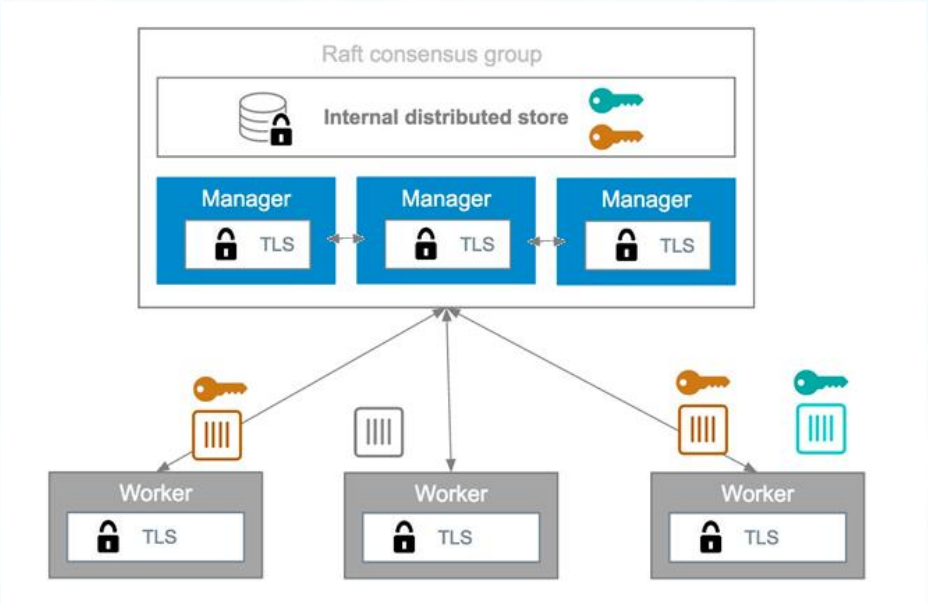


En cas de perte d'un nœud,

- Le module Dispatcher le détecte
- l'Orchestrator réconcilie l'état du cluster avec le nombre d'instances prévu initialement
- le module Allocator alloue des ressources nécessaires pour instancier le nouveau nœud.

III. Réseau virtuel et sécurité

L'ensemble des échanges dans un cluster Docker est sécurisé nativement, par la création et le déploiement automatique de certificats TLS entre les conteneurs.



Swarm va créer un réseau virtuel spécifique afin d'interagir avec ses conteneurs :

```
[root@CentOS ~]# docker network ls

NETWORK ID          NAME                DRIVER              SCOPE
969b95ed19c0        bridge              bridge              local
ee931dcac4ec        docker_gwbridge     bridge              local
7eb375f5aa6f        host                host                local
670641df8c2e        none                null                local

[root@CentOS ~]# ifconfig -a
```

```
docker_gwbridge: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.18.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
```

IV. Mise en oeuvre

L'exercice suivant consiste à installer un cluster de serveurs nginx

A. Ouverture des ports

```
[root@CentOS ~]# firewall-cmd --add-port=2376/tcp --permanent
[root@CentOS ~]# firewall-cmd --add-port=2377/tcp --permanent
[root@CentOS ~]# firewall-cmd --add-port=7946/tcp --permanent
[root@CentOS ~]# firewall-cmd --add-port=7946/udp --permanent
[root@CentOS ~]# firewall-cmd --add-port=4789/udp --permanent
[root@CentOS ~]# firewall-cmd --add-port=80/tcp --permanent
[root@CentOS ~]# firewall-cmd --reload
[root@CentOS ~]# systemctl restart docker
```

B. Initialisation d'un noeud

Swarm nécessite au moins un « manager » pour gérer les nœuds du cluster. L'exemple suivant met en place un manager accessible sur l'adresse IP ***MANAGER-IP*** (port par défaut : 2377). Cette adresse IP correspond à une adresse du réseau local *192.168.x.y*.

```
[root@CentOS ~]# docker swarm init --advertise-addr <MANAGER-IP>
Swarm initialized: current node (tl9ps5zgym09ior21isb1zcmy) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-339oq9s8kgc2sw4onxsspyieq3v9oblfh81so04elvz59546hh-806pfeeis0atmfyukoz50qwhf \
    MANAGER-IP:2377
```

La commande swarm init génère deux tokens : un pour l'ajout de manager, l'autre pour l'ajout d'un worker :

```
[root@CentOS ~]# docker swarm join-token manager
[root@CentOS ~]# docker swarm join-token worker
```

Vérification du noeud :

```
[root@CentOS ~]# docker node ls
ID                            HOSTNAME                STATUS  AVAILABILITY  MANAGER STATUS
tl9ps5zgym09ior21isb1zcmy *   localhost.localdomain   Ready   Active        Leader

[root@CentOS ~]# docker node inspect localhost.localdomain --pretty
```

Nous pouvons modifier l'état d'un nœud de cette manière :

```
[root@CentOS ~]# docker node update --availability pause localhost.localdomain
[root@CentOS ~]# docker node update --availability active localhost.localdomain
```

C. Création d'un service

Un service est la formalisation de tâches qui seront exécutées par les noeuds worker.

Le service web sera à l'écoute sur le port 80 :

```
[root@CentOS ~]# docker service create -p 80:80 --name web nginx:latest
```

Un navigateur web permet de vérifier que le serveur est fonctionnel : http://MANAGER-IP/

Les informations du service sont visibles avec :

```
[root@CentOS ~]# docker service inspect web
```

D. Scalabilité

La scabilité permet la continuité d'un service dans de bonnes conditions lors d'une montée en charge. Swarm permet de dupliquer des conteneurs afin d'effectuer un paraléllisme, et d'assurer ainsi une fluidité lors d'une forte sollicitation.

La commande suivante va créer 3 instances du conteneur ngnix :

```
[root@CentOS ~]# docker service scale web=3
web scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
```

Le contrôle de l'état du service s'effectue via :

```
[root@CentOS ~]# docker service ps web
ID             NAME    IMAGE          NODE                    DESIRED STATE     CURRENT STATE
elg4dqob1ux1   web.1   nginx:latest   localhost.localdomain   Running           Running 3 minutes ago
ouvv0mw1jdi8   web.2   nginx:latest   localhost.localdomain   Running           Running about a minute ago
v1if8y8fkhw2   web.3   nginx:latest   localhost.localdomain   Running           Running about a minute ago
```

SUPERVISION

Plusieurs commandes Docker nous permettent une surveillance des isolateurs.

Affichez l'information sur l'ensemble du système

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

Conteneurs

```
[root@CentOS ~]# docker inspect <container>
[root@CentOS ~]# docker container logs <container>
```

Evènements : la commande suivante s'exécute sur un terminal libre

```
[root@CentOS ~]# docker events
```

Etat des process

```
[root@CentOS ~]# docker top <container>
```

Journaux système

```
[root@CentOS ~]# docker logs --tail 0 -f <container>
```

Ports mappés

```
[root@CentOS ~]# docker port <container>
```

Statistiques du conteneur

```
[root@CentOS ~]# docker stats ContDeb1
CONTAINER     CPU %     MEM USAGE / LIMIT      MEM %     NET I/O           BLOCK I/O     PIDS
ContDeb1      0.00%     476 KiB / 3.844 GiB    0.01%     648 B / 648 B     0 B / 0 B     1
```

RESOURCES

Documentation
https://docs.docker.com

Images gratuites
https://hub.docker.com/

Kubernetes
https://kubernetes.io/