Introduction à Docker

Docker est un projet open source (Apache 2.0) écrit en GO et hébergé sur GitHub: https://github.com/docker (https://github.com/docker).

Initialement porté par la startup DotCloud (renommée depuis Docker) fondée par deux français anciens de l'Epitech.

Docker est composé de trois éléments :

- le daemon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (Containerd avec runC)
- une API de type REST qui permet de communiquer avec le daemon
- Le client en CLI (command line interface) : commande `docker`

Par défaut, le client communique avec le daemon Docker via un socket Unix (/var/run/docker.sock) mais il est possible d'utiliser un socket TCP.

Docker c'est aussi un dépôt d'images (aussi appelé registry) : https://store.docker.com (https://store.docker.com)
Il contient les images officielles maintenues par Docker mais aussi celles mises à disposition par d'autres contributeurs.

Quelques concepts:

- une image est un ensemble de fichiers inertes en read-only.
- Un conteneur est une instance une active (started) ou inactive (stopped) d'une image. L'execution d'un conteneur n'altère jamais une image.

Lexique

- **Conteneur** : Image exécutable d'un environnement complet incluant code, librairies, outils et configuration
- **Image** : template de conteneur en read-only contenant un systeme de base et une application.
- **Docker HUB** : Dépôt public d'images mises à disposition par Docker DockerHub (https://store.docker.com)
- **Dockerfile** : fichier texte de description d'une image

- **Docker Compose** : fichier texte (yaml) de description d'un ensemble de conteneurs
- **Docker Machine** : Outil de déploiement des hôtes Docker sur différentes plateformes (Mac, Windows) : https://docs.docker.com/machine/overview/ (https://docs.docker.com/machine/overview/)
- **Orchestrateur** : gère un pool de ressources serveurs (Swarm, Kubernetes, Mesos, Rancher…)
- **Registry** : Dépôt privé d'images Docker

Installation de Docker

Méthode d'installation officielle (https://docs.docker.com/install/linux/docker-ce/centos/)

Installer les prérequis (Centos 7.x amd64)

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

Configurer le dépôt officiel et installer Docker en version CE (Community Edition)

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/ce
sudo yum install docker-ce
```

Activer et démarrer le service

```
$ sudo systemctl enable docker
$ sudo systemctl start docker
```

Vérifier l'installation

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hu
    (amd64)
 3. The Docker daemon created a new container from that image which run
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Details des opérations réalisées par cette commande:
1: le client docker se connecte au daemon docker via le socket Unix
2: l'image "hello-world" n'etant pas présente localement, le daemon Docker la télécharge depuis la registery Docker Hub
3: le daemon Docker crée un nouveau conteneur depuis cette image dont la finalité est de produire le message ci-dessus.
4: le daemon Docker renvoi le message au client Docker pour afficher le résultat dans le terminal

Vérifier la version

```
 $ sudo docker version
Client:
 Version:           18.06.1-ce
 API version:       1.38
 Go version:        go1.10.3
 Git commit:        e68fc7a
 Built:             Tue Aug 21 17:23:03 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.06.1-ce
  API version:      1.38 (minimum version 1.12)
  Go version:       go1.10.3
  Git commit:       e68fc7a
  Built:            Tue Aug 21 17:25:29 2018
  OS/Arch:          linux/amd64
  Experimental:     false
```

Obtenir des infos sur la configuration Docker

```
$ sudo docker info
Containers: 1
 Running: 0
 Paused: 0
 Stopped: 1
Images: 1
Server Version: 18.06.1-ce
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: false
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 468a545b9edcd5932818eb9de8e72413e616e86e
runc version: 69663f0bd4b60df09991c08812a60108003fa340
init version: fec3683
Security Options:
 seccomp
  Profile: default
Kernel Version: 3.10.0-514.2.2.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 1.781GiB
Name: tp-docker.univ-rouen.fr
ID: GUWR:VBEF:77JQ:LZDZ:X2JG:E74O:3WJC:CRJN:VZOQ:XA5V:VNYF:UN5S
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

Cette commande retourne plusieurs informations intéressantes:
1: le driver de stockage est overlayFS: `Storage Driver: overlay2`
2: le dossier dans le quel Docker va stocker les images et les volumes: `Docker Root Dir: /var/lib/docker`
3: l'adresse du registre pour télécharger les images: `Registry: https://index.docker.io/v1/`

Utiliser Docker sans les droits "root"

Pour utiliser Docker sans les droits `root` , l'utilisateur doit appartenir appartenir au groupe `docker`

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

Il faut se déconnecter / reconnecter pour appliquer les modifications.

Tester sans sudo:

```
$ docker run hello-world
```

Les Images Docker

Faire une recherche sur le Docker Hub (commande : `docker search`)

```
$ docker search debian
NAME              DESCRIPTION                                     STARS
ubuntu            Ubuntu is a Debian-based Linux operating s...   6898
debian            Debian is a Linux distribution that's comp...   2356
google/debian                                                     52
armhf/debian      Debian is a Linux distribution that's comp...   29
```

Ici "ubuntu" et "debian" sont des images officielles (les autres sont de la forme user/nom_image). Elles sont maintenues par l'équipe docker et considérées comme plus "fiables".

La colonne STARS donne une indication sur la popularité de l'image (mise en favoris).

Télécharger une image (commande : `docker pull`)

```
$ docker pull debian
Using default tag: latest
latest: Pulling from library/debian
3e17c6eae66c: Pull complete
Digest: sha256:26b2647845d66e20eeadf73d1c302a4ffd2cc9a74c39a52f2aced4f8
Status: Downloaded newer image for debian:latest
```

Lancer un conteneur (commande : `docker run`)

La commande `docker run` qui permet de lancer un conteneur peut également télécharger l'image si elle n'est pas disponible localement

```
$ docker run debian:stretch
docker run debian:stretch
Unable to find image 'debian:stretch' locally
stretch: Pulling from library/debian
Digest: sha256:26b2647845d66e20eeadf73d1c302a4ffd2cc9a74c39a52f2aced4f8
Status: Downloaded newer image for debian:stretch
```

Connaitre l'historique de création de l'image (commande : `docker history`)

```
$ docker history debian
IMAGE               CREATED             CREATED BY
6d83de432e98        4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:a71e
```

Les TAGS

Lister les images présentent localement (commande : `docker images` ou `docker image ls`)

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
hello-world         latest              f2a91732366c        2 weeks ago
debian              latest              6d83de432e98        4 weeks ago
debian              stretch             6d83de432e98        4 weeks ago
```

Ici les images debian ont le même ID (6d83de432e98) : c'est la même image mais avec un TAG différent

Exemple d'images et de TAGS proposés sur le dépôt officiel Debian : https://hub.docker.com/r/library/debian/tags/ (https://hub.docker.com/r/library/debian/tags/)

Ajouter un tag à une image (commande : `docker image tag`)

```
$ docker image tag debian:stretch debian:levasbr1
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
debian              levasbr1            6d83de432e98        4 weeks ago
debian              stretch             6d83de432e98        4 weeks ago
```

Supprimer une image (commande : `docker rmi`)

Cette commande permet de supprimer une image à condition de ne pas avoir de conteneur lié.

```
$ docker rmi debian:latest
Untagged: debian:latest
Untagged: debian@sha256:26b2647845d66e20eeadf73d1c302a4ffd2cc9a74c39a52
Deleted: sha256:6d83de432e98210aa25bcc5556a641d60ec621b67786a2862cfeec5
Deleted: sha256:a75caa09eb1f7d732568c5d54de42819973958589702d415202469a

$ docker rmi f2a
Untagged: hello-world:latest
Untagged: hello-world@sha256:be0cd392e45be79ffeffa6b05338b98ebb16c87b25
Deleted: sha256:f2a91732366c0332ccd7afd2a5c4ff2b9af81f549370f7a19acd460
Deleted: sha256:f999ae22f308fea973e5a25b57699b5daf6b0f1150ac2a5c2ea9d7f
```

> On peut aussi utiliser l'ID abrégé (ex: f2A) pour désigner une image (en l'absence d'ambiguité)

Les conteneurs Docker

Lancer un conteneur à partir d'une image (commande : `docker run`)

```
$ docker run debian:latest cat /etc/issue
Debian GNU/Linux 9 \n \l
```

L'état d'un conteneur dépend de la commande qui est lancée. Ici, le conteneur execute la commande `cat` et s'arrête dès quelle est terminée.

Lister les conteneurs en cours d'exécution (commande : `docker ps`)

La commande `docker ps` qui permet de lister les conteneurs en cours d'executions ne retourne effectivement rien :

```
$ docker ps
CONTAINER ID    IMAGE   COMMAND   CREATED   STATUS    PORTS   NAMES
```

Pour obtenir la liste complete des conteneurs, il faut utiliser l'option `docker ps -a` :

```
$ docker ps -a
CONTAINER ID        IMAGE           COMMAND             CREATED     STA
396767b854a9        debian:latest   "cat /etc/issue"    44 minutes  Exi
```

Le conteneur possède un identifiant unique (96767b854a9) et un nom généré aléatoirement (quizzical_easley).

Nommer un conteneur (option : `--name` ou `-n`)

On peut utiliser l'option `--name` pour nommer un conteneur de manière plus explicite :

```
$ docker run --name cmd_cat debian:latest cat /etc/issue
Debian GNU/Linux 9 \n \
```

Cette commande a créé un nouveau conteneur :

```
docker ps -a
CONTAINER ID    IMAGE           COMMAND             CREATED
2770f96e4a3d    debian:latest   "cat /etc/issue"    About a minute ago
ef6b6f1c64a1    debian:latest   "cat /etc/issue"    4 minutes ago
```

Obtenir une session intéractive (option : `-it`)

On peut obtenir une session intéractive (option `-i`) en se branchant sur l'entrée standard du conteneur et en connectant un pseudo terminal TTY (option `-t`) :

```
$ docker run -it  debian:latest /bin/bash
root@eae2cce2669d:/#
```

Le prompt reprend le CID du conteneur (utiliser la commande `exit` pour quitter le conteneur).

Lancer un conteneur en mode daemon (option : `-d`)

On peut lancer un conteneur en mode daemon pour qu'il tourne en tâche de fond (le mode intéractif tourne au premier plan).

```
$ docker run -d --name test_daemon nginx
4d81f9903afe1b777de6389954c762122b5aeea847f5b4f8953ad308bbc5203d
// on affiche la liste des conteneurs en cours d'execution :
$ docker ps
CONTAINER ID        IMAGE                                       COMMAND
4d81f9903afe        nginx                                       "nginx -g 'da
// on stoppe ce conteneur
$ docker stop test_daemon
```

Cycle de vie des conteneurs

Obtenir la configuration détaillée d'un conteneur

```
$ docker inspect <nom_conteneur> ou <CID>
```

Récupérer la sortie standard d'un conteneur

```
$ docker logs <nom_conteneur> ou <CID>
```

Afficher les processus en cours dans un conteneur

```
$ docker top <nom_conteneur> ou <CID>
```

Suspendre (freezer) et réactiver un conteneur

```
$ docker pause / unpause <nom_conteneur> ou <CID>
```

Arrêter / démarrer / tuer / redémarrer un conteneur

```
$ docker stop / start / kill / restart <nom_conteneur> ou <CID>
```

Exporter l'ensemble du système de fichier d'un conteneur dans une archive TAR

```
$ docker export <nom_conteneur> ou <CID> > archive.tar
```

Afficher les ports réseaux exposés par un conteneur

```
$ docker port <nom_conteneur> ou <CID>
```

Afficher les changements effectués sur les fichiers d'un conteneur (A=Ajout, D=Delete, C=Modif)

```
$ docker run -it --name test_diff debian /bin/bash

root@c7d328b087eb:/# apt update && apt -y upgrade
Ign:1 http://deb.debian.org/debian stretch InRelease
Get:2 http://deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:3 http://deb.debian.org/debian stretch Release [118 kB]
Get:4 http://deb.debian.org/debian stretch Release.gpg [2434 B]
Get:5 http://security.debian.org stretch/updates InRelease [63.0 kB]
Get:6 http://deb.debian.org/debian stretch-updates/main amd64 Packages

$ docker diff test_diff
C /root
A /root/.bash_history
C /tmp
C /var/lib/apt/lists
A /var/lib/apt/lists/deb.debian.org_debian_dists_stretch-updates_InRele
A /var/lib/apt/lists/deb.debian.org_debian_dists_stretch-updates_main_b
A /var/lib/apt/lists/deb.debian.org_debian_dists_stretch_Release
A /var/lib/apt/lists/deb.debian.org_debian_dists_stretch_Release.gpg
A /var/lib/apt/lists/deb.debian.org_debian_dists_stretch_main_binary-am
A /var/lib/apt/lists/lock
```

Commiter un conteneur pour obtenir une nouvelle image

Une solution pour “persister” les données ou les configurations consiste à commiter le conteneur pour créer une nouvelle image (clairement pas une bonne pratique).
Dans cet exemple, l’image est associée à l’utilisateur “levasbr1” correspondant à un compte sur le docker-hub pour faciliter la publication des images sur la plate-forme public.

```
$ docker commit test_diff  <user>/<image>:<tag>
$ docker commit test_diff levasbr1/debian:latest
```

Pousser une image locale sur une registry privée

Pour pouvoir utiliser une registry non sécurisée en https, il faut ajouter une exception dans la configuration de Docker.
Ajouter le fichier: `/etc/docker/daemon.json`

```
{
  "insecure-registries" : ["registry.univ-rouen.fr:5000"]
}
```

et relancer de daemon docker pour prendre en compte les modifications: `sudo systemctl restart docker`

On commence par “tagger” l’image locale pour la registry privée. Dans l’exemple le serveur qui heberge la registry se nomme “registry”

```
$ docker tag  <user>/debian:latest registry.domain.tld:<port/<user>/deb
$ docker tag levasbr1/debian:latest registry.univ-rouen.fr:5000/levasbr
$ docker images
REPOSITORY                        TAG                 IMAGE ID
registry.univ-rouen.fr/levasbr1/debian   latest              55f2b1db89
```

Pousser l’image dans la registry :

```
$ docker push registry.univ-rouen.fr:5000/levasbr1/debian:latest
The push refers to repository [registry.univ-rouen.fr:5000/levasbr1/deb
6737a6f8cc10: Pushed
e1df5dc88d2c: Pushed
latest: digest: sha256:56bbd19006e1d1166d56715e01771cf22fb2ca67335009b7
```

Pour créer une registry, vous pouvez tout simplement utiliser un conteneur Docker:
`$ docker run -d -p 5000:5000 --restart=always --name registry registry:2`

Pour récuperer l'image depuis la registry privée:

on commence par supprimer l'image locale:

```
$ docker rmi registry.univ-rouen.fr:5000/levasbr1/debian:latest
```

ensuite on récupère l'image sur la registry privée:

```
$ docker pull registry.univ-rouen.fr:5000/levasbr1/debian:latest
```

La démarche est identique pour pousser une images sur le DockerHub avec les commandes `docker login -u identifiant-docker-hub` et `docker push nom_image` (il faut disposer d'un compte sur docker-hub)

Exécuter une commande dans un conteneur démarré

Dans l'exemple, on lance un conteneur nginx en mode daemon et on utilise la commande `docker exec` pour s'y connecter :

```
$ docker run -d --name test_exec nginx
$ docker exec -it test_exec /bin/bash
root@331e1e904e1e:/# ls
bin  boot  dev    etc  home  lib    lib64  media  mnt  opt   proc  ro
root@331e1e904e1e:/# exit
exit
```

S'attacher à un conteneur démarré pour visualiser ou interagir avec le procesus racine

Dans l'exemple, on lance un conteneur nginx en mode daemon et on utilise la commande attach pour visualiser la sortie standard du processus ngnix (il faut ouvrir un navigateur avec l'URL : `http://<ip-server-docker>.univ-rouen.fr:8000` pour générer des logs).

> L'option -p 8000:80 qui permet d'exposer le port 80 du conteneur et de le joindre sur le port 8000 de l'hôte est abordé dans le § les links Docker

```
$ docker run -d -p 8000:80 --name test_attach nginx
5d7d6f3108532971fef27434fc3504ef5b42c741d7923913e556b9629de6c30c

$ docker attach test_attach --sig-proxy=false
10.197.1.22 - - [18/Dec/2017:15:37:28 +0000] "GET / HTTP/1.1" 200 612 "
10.197.1.22 - - [18/Dec/2017:15:37:28 +0000] "GET /favicon.ico HTTP/1.1
2017/12/18 15:37:28 [error] 5#5: *2 open() "/usr/share/nginx/html/favic
```

> L'option `--sig-proxy=false` permet de se detacher du conteneur avec la sequence `CTRL-c` sans killer le processsus racine.

Supprimer un conteneur (il doit être arrêté…)

```
$ docker ps -a
docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATE
5d7d6f310853        nginx               "nginx -g 'daemon ..."   29 min
331e1e904e1e        nginx               "nginx -g 'daemon ..."   About
c7d328b087eb        debian              "/bin/bash"              About
```

Supprimer un conteneur

```
$ docker rm test_attach
```

Supprimer plusieurs conteneurs en utilisant les CID abrégés

```
$ docker rm 331 c7d
```

Copier des fichiers depuis ou à destination d'un conteneur

Dans cet exemple, on récupere un fichier depuis un conteneur puis on le ré-importe après modification.

```
$ docker run -d -p 8001:80 --name test_cp nginx
```

On copie le fichier index.html depuis le conteneur sur la machine hôte

> Attention au "." en fin de ligne qui représente le répertoire courant

```
$ docker cp test_cp:/usr/share/nginx/html/index.html .
```

On remplace le contenu du fichier

```
$ echo "Hello World" > index.html
```

On copie le fichier modifié depuis l'hôte vers le conteneur

```
$ docker cp index.html test_cp:/usr/share/nginx/html/
```

On teste avec un navigateur avec l'URL : `http://<ip-server-docker>.univ-rouen.fr:8001`

Afficher des informations sur les conteneurs exécutés par Docker (équivalent à un top sous Linux)

```
$ docker stats
CONTAINER           CPU %               MEM USAGE / LIMIT     MEM %
776035a48a44        0.00%               1.766MiB / 1.958GiB   0.09%
```

Afficher les événements qui se produisent sur le bus du moteur Docker (équivalent à un tailf sur un fichier de log)

Pour tester cette commande, il faut générer quelques actions sur un conteneur et visualiser en même temps les évènements sur un autre terminal :

```
$ docker run -d --name test_events nginx
$ docker pause test_events
$ docker unpause test_events
$ docker stop test_events
```

Observer les événements depuis un autre terminal :

```
$ docker events
2017-12-18T16:54:15.866668426+01:00 container pause 331e1e904e1e21e963a
2017-12-18T16:54:24.678680900+01:00 container unpause 331e1e904e1e21e96
2017-12-18T16:54:29.942202432+01:00 container kill 331e1e904e1e21e963ac
```

Les DockerFiles

Un dockerfile est un fichier texte (donc versionable) de description qui permet de générer une image. Il est basé sur une image standard auquel on ajoute les éléments propres à l'application que l'on veut déployer.

Instructions de bases :

- **FROM** permet de définir depuis quelle base votre image va être créée
- **LABEL maintainer** permet de définir l'auteur de l'image
- **RUN** permet de lancer une commande, mais aura aussi pour effet de créer une image intermédiaire.
- **ADD** permet de copier un fichier depuis la machine hôte ou depuis une URL
- **EXPOSE** permet d'exposer un port du container vers l'extérieur
- **CMD** détermine la commande qui sera exécutée lorsque le container démarrera
- **ENTRYPOINT** permet d'ajouter une commande qui sera exécutée par défaut
- **WORKDIR** permet de définir le dossier de travail pour toutes les autres commandes (par exemple RUN, CMD, ENTRYPOINT et ADD)
- **ENV** permet de définir des variables d'environnements qui pourront ensuite être modifiées grâce au paramètre de la commande `run --env <key>=<value>`
- **VOLUMES** permet de créer un point de montage qui permettra de persister les données. On pourra alors choisir de monter ce volume dans un dossier spécifique en utilisant la commande `run -v`

Nous allons créer les 2 fichiers suivants (dans votre homedir):

```
$ cat dockerfile
#
# Simple example of a Dockerfile
# Build Debian / Nginx with demo file
#
FROM debian:latest
LABEL maintainer="prenom.nom@domain.tld"

RUN apt-get update
RUN apt-get install -y nginx

ADD demo.html /var/www/html/demo.html
CMD ["nginx", "-g", "daemon off;"]
EXPOSE 80
STOPSIGNAL SIGTERM

CMD ["nginx", "-g", "daemon off;"]
# Then run the following command for build new image
# $ docker build -t identifiant_docker_hub/nginx:latest .
# $ docker run -d -p 8002:80 identifiant_docker_hub/nginx:latest
```

et

```
$ cat demo.html
<html>
  <header><title>Hello world</title></header>
  <body>
    Hello world
  </body>
</html>
```

Générer une image à partir d'un dockerfile (commande : `docker build`)

> Attention au "." en fin de première ligne qui représente le répertoire courant

```
$ docker build -t levasbr1/nginx:latest .
Sending build context to Docker daemon  6.144kB
Step 1/9 : FROM debian:latest
 ---> da653cee0545
Step 2/9 : MAINTAINER "prenom.nom@domaine.tld"
 ---> Running in 5680fbaa6dce
 ---> 7139f3b99829
Removing intermediate container 5680fbaa6dce
Step 3/9 : RUN apt-get update
 ---> Running in c849b83991b7
---------8<-------------
```

On peut ensuite lancer un conteneur depuis cette nouvelle image

```
$ docker run -d -p 8002:80 --name test_build  levasbr1/nginx:latest
```

On teste avec un navigateur avec l'URL : `http://<ip-server-docker>:8002/demo.html`

les volumes Docker

Par essence, les conteneurs Docker sont éphémères. Ils doivent être conçus pour pouvoir être supprimés sans perdre les données. Pour assurer cette persistance, on utilise les volumes Docker.

- Ils sont initialisés à la création du container. Si l'image de base contient des données au point de montage spécifié, celles-ci sont copiés dans le volume lors de son initialisation
- Ils peuvent être partagés entre plusieurs containers (exemple : pour créer des clusters de container)
- Les changements apportés aux données du volume sont pris en compte immédiatement
- Les données restent sur votre hôte même si le container vient à disparaître

Initialiser un volume (option : `-v`)

L'utilisation la plus simple est d'initialiser un volume à la création du conteneur :

```
$ docker run -d -v /var/log --name test_volume debian:latest  /bin/slee
```

Ici l'option -v /var/log permet d'initialiser un volume indépendant du conteneur qui contiendra les logs du conteneur :

```
$ docker inspect test_volume
----------------8<--------------------
"Mounts": [
    {
        "Type": "volume",
        "Name": "e566163a0978a34dc1357cb4df7c9d86ddbb308de1c0ba26393e56
        "Source": "/var/lib/docker/volumes/e566163a0978a34dc1357cb4df7c
        "Destination": "/var/log",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
----------------8<-------------
```

La commande inspect permet de localiser le volume sur l'arborescence locale de l'hôte ("Source") et le point de montage dans le conteneur ("Destination")

```
$ sudo ls /var/lib/docker/volumes/e566163a0978a34dc1357cb4df7c9d86ddbb3
apt  btmp  faillog  lastlog  wtmp
```

On constate que les fichiers de logs du conteneur sont accessibles dans ce volume

Partager un dossier entre l'hôte et un conteneur (bind mounts)

L'autre utilisation des volumes est de partager un dossier entre le conteneur et le systeme hôte (exemple : pour qu'un développeur PHP puisse éditer directement les fichiers depuis la machine hôte).

```
$ mkdir projet-web
$ docker run -d --name test_volume2  -p 8004:80 -v ~/projet-web:/var/ww
```

Ici nous disposons d'un volume monté sur l'arborescence /var/www/html du conteneur et correspondant au dossier /home/user (~) de l'hôte

```
$ echo "<?php phpinfo(); ?>" > ~/projet-web/phpinfo.php
```

on crée un fichier phpinfo.php directement sur l'arborescence du système hôte. On teste depuis un navigateur avec l'URL : `http://<ip-server-docker>:8004/phpinfo.php` que le fichier est bien accédé par le conteneur

```
$ docker inspect test_volume2
---------------8<--------------------
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/formation/projet-web",
        "Destination": "/var/www/html",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
---------------8<--------------------
```

la commande `inspect` permet de visualiser cette configuration du conteneur. "Source" correspond à l'arborescence de l'hôte et "Destination" au point de montage dans le conteneur

> Dans cette méthode, le conteneur dispose d'un accès direct au filesysteme de l'hôte. Elle est à proscrire en production pour des raisons de sécurité.

Partager un volume de données entre plusieurs conteneurs (option : `--volumes-from`)

On démarre un conteneur qui "exporte" son arborescence /usr/share/nginx/html dans un volume

```
$ docker run -d -p 8005:80 -v /usr/share/nginx/html  --name test_volume
```

On démarre ensuite un conteneur en mode intéractif qui partage le même volume de données

```
$ docker run -it --volumes-from test_volume3 debian:latest  /bin/bash
```

Depuis ce conteneur on crée un fichier de test (exit pour quitter le conteneur)

```
root@e232d8d9d527:/# echo bonjour > /usr/share/nginx/html/test.html
```

On teste depuis un navigateur avec l'URL http://docker-x.univ-rouen.fr:8005/test.html (http://docker-x.univ-rouen.fr:8005/test.html) que le fichier est bien accessible depuis le conteneur test_volume3

> Docker ne supprime pas les volumes à la suppression des conteneurs. Pour éviter l'accumulation de volumes orphelins, il faut utiliser la commande `rm` avec l'option `-v` `docker rm -v nom_conteneur ou CID` ou faire une purge régulière avec la commande `docker volume rm $(docker volume ls -qf dangling=true)`

Les links Docker

L'installation de Docker crée par défaut un réseau nommé `bridge` (interface Docker0). Il est utilisé par les conteneurs si aucun réseau n'est spécifié au lancement (option `--network=mon_reseau`)

> Par défaut, tous les conteneurs peuvent communiquer entre eux sur le réseau bridge. Pour plus de sécurité, il est possible de désactiver ce comportement (option `DOCKER_OPTS="-icc=false"` de la configuration de daemon).

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8dd02d7ad16f        bridge              bridge              local
72244137afae        host                host                local
f7ab4027acf8        none                null                local

$ docker network inspect bridge
{
       "Name": "bridge",
       "Id": "8dd02d7ad16f9e47a774f0ee5a652a606ecc23bcc15c126d3e6fbf6fd
       "Created": "2017-12-05T16:44:54.821533922+01:00",
       "Scope": "local",
       "Driver": "bridge",
       "EnableIPv6": false,
       "IPAM": {
           "Driver": "default",
           "Options": null,
           "Config": [
               {
                   "Subnet": "172.17.0.0/16",
                   "Gateway": "172.17.0.1"
               }
```

Les conteneurs uilisent une adresse IP fournie par le daemon Docker (réseau privé non routable: 172.17.0.0/16, gateway: 172.17.0.1).

Sous linux, Docker utilise netfilter (iptables) pour implémenter les fonctions de NAT qui permettent aux conteneurs de communiquer avec l'extérieur en utilisant l'adresse IP de l'hôte et les fonctions de DNAT pour joindre les conteneurs depuis l'extérieur en utilisant un port particulier.

```
$ docker run -d -p 8006:80 --name test_port nginx
```

Ici l'option -p permet d'exposer le port 80 du conteneur et de joindre le conteneur sur le port 8006 et l'adresse IP de l'hôte

Faire communiquer deux conteneurs (option : `--link`)

On démarre un conteneur avec une database MySQL

```
$ docker run --name mysql -e MYSQL_ROOT_PASSWORD=bonjour -d mysql:5.7
```

On démarre ensuite un conteneur avec l'interface PHPmyadmin, qui permet d'administrer une base de données MySQL. L'option --link permet de renseigner le fichier host du conteneur phpmyadmin pour qu'il puisse se connecter à la base de données

```
$ docker run --name phpmyadmin  -d --link mysql:db -p 8080:80 phpmyadmi
```

On consulte le fichier hosts du conteneur phpmyadmin:

```
$ docker exec phpmyadmin cat /etc/hosts
# cat /etc/hosts
127.0.0.1    localhost
172.17.0.3   db d72489c44026 mysql
172.17.0.4   7d5ed10b026e
```

Ici on constate que les alias db et mysql pointent vers l'ip 172.17.0.3 du serveur MySQL

```
$ docker inspect mysql | grep IP
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
                    "IPAMConfig": null,
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
```

La commande inspect permet de confirmer l'adresse IP du conteneur MySQL. On teste depuis un navigateur avec l'URL : `http://<ip-server-docker>:8080` l'utilisation de phpmyadmin pour administrer la base

L'option -e (dans l'exemple ci-dessus `-e MYSQL_ROOT_PASSWORD=bonjour`) permet de passer des variables d'environnement au conteneur (ici on définit le password root de la base de données).

Docker-Compose

Docker-compose est un outil officiel Docker qui permet de simplifier le déploiement d'applications complexes (avec de nombreux conteneurs) en utilisant un simple fichier texte de description au format yaml. Il reprend l'ensemble des options qui seraient normalement à fournir à un `docker run`

Installer

```
$ sudo pip install docker-compose
$ docker-compose -v
docker-compose version 1.22.0, build f46880f
```

Tester

Pour tester docker-compose, vous pouvez créer un fichier `docker-compose.yml` dans votre répertoire personnel

```
version: '2'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: bonjour
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Dans l’exemple proposé, docker-compose va lancer 2 conteneurs automatiquement (mysql et wordpress).

Pour lancer les conteneurs, on utilise la commande `docker-compose up` ou `docker-compose up -d` pour lancer en arrière plan.
La commande doit être executée depuis le répertoire contenant le fichier yaml

```
$ docker-compose up
Creating network "levasbr1_default" with the default driver
Creating volume "levasbr1_db_data" with default driver
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
Digest: sha256:1f95a2ba07ea2ee2800ec8ce3b5370ed4754b0a71d9d11c0c35c934e
Status: Downloaded newer image for mysql:5.7
Pulling wordpress (wordpress:latest)...
latest: Pulling from library/wordpress
e7bb522d92ff: Already exists
75651f247827: Pull complete
93b186f8edb2: Pull complete
77c007e2f556: Pull complete
bf4da9c43c0b: Pull complete
11843d906ebb: Pull complete
e03cc73ddbff: Pull complete
```

On teste le bon fonctionnement depuis un navigateur avec l'URL : `http://<ip-server-docker>:8000`

On utilise la sequence CTRL-c permet de stopper les conteneurs ou la commande `docker-compose down` si le docker-compose est lancé avec l'option `-d`

Exercice

Utiliser une interface web d'administration Docker : Portainer

Quelques snippets:

- Pour stopper tous les conteneurs: `docker stop $(docker ps -q -a)`
- Pour supprimer tous les conteneurs: `docker rm $(docker ps -q -a)`
- Pour supprimer toutes les images: `docker rmi -f $(docker images -q)`
- Pour purger les volumes orphelins: `docker volume rm $(docker volume ls -qf dangling=true)`
- Pour avoir une interface web pour Docker: https://github.com/portainer/portainer (https://github.com/portainer/portainer)
- Pour avoir un reverse proxy automatique pour Docker: https://github.com/jwilder/nginx-proxy (https://github.com/jwilder/nginx-proxy)
- Pour disposer d'un bac à sable Docker en ligne: https://labs.play-with-

docker.com/ (https://labs.play-with-docker.com/)

- Quelques one-liners: http://www.commandlinefu.com/commands/matching/docker/ZG9ja2Vy (http://www.commandlinefu.com/commands/matching/docker/ZG9ja2Vy)
- Formation officielle en ligne Docker: https://docs.docker.com/get-started/ (https://docs.docker.com/get-started/)
- Purger Docker : `docker system prune -a` et `docker volume prune` vérification avec `docker system df`