Université de Picardie Jules Verne

Informatique – Master CCM

INSSET – Saint-Quentin

Conteneurs Applicatifs et Micro-Services M2

C. Drocourt

cyril.drocourt@u-picardie.fr

Cours 1 : Docker

V2023.1

Table des matières

1 - Installation

1.1 - Introduction

Docker est un système de conteneur, au même titre que LXC. La différence est que là où LXC démarre l'intégralité d'un système d'exploitation, Docker ne va exécuter que la tâche demandée dans le conteneur.

De plus, le système est basé sur des concepts supplémentaires :

- des containers pré-configurés sont disponibles dans des dépôts ,
- il est possible d'utiliser un conteneur pour diverses tâches,
- il est possible de modifier un conteur,
- il est possible de créer ses propres conteneurs personnalisés,

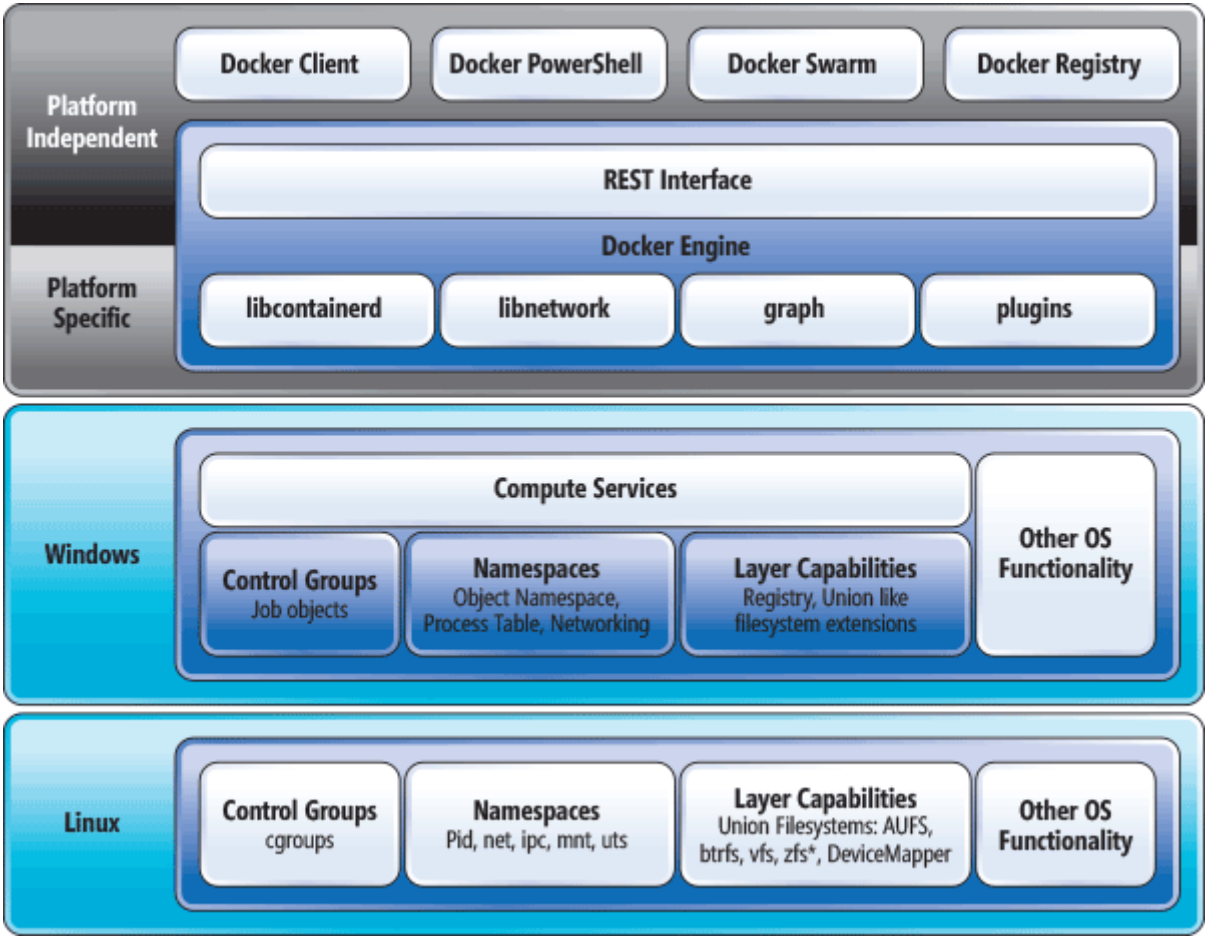
Dans la terminologie Docker, une image est un fichier alors qu'un conteneur est l'exécution de ce fichier.

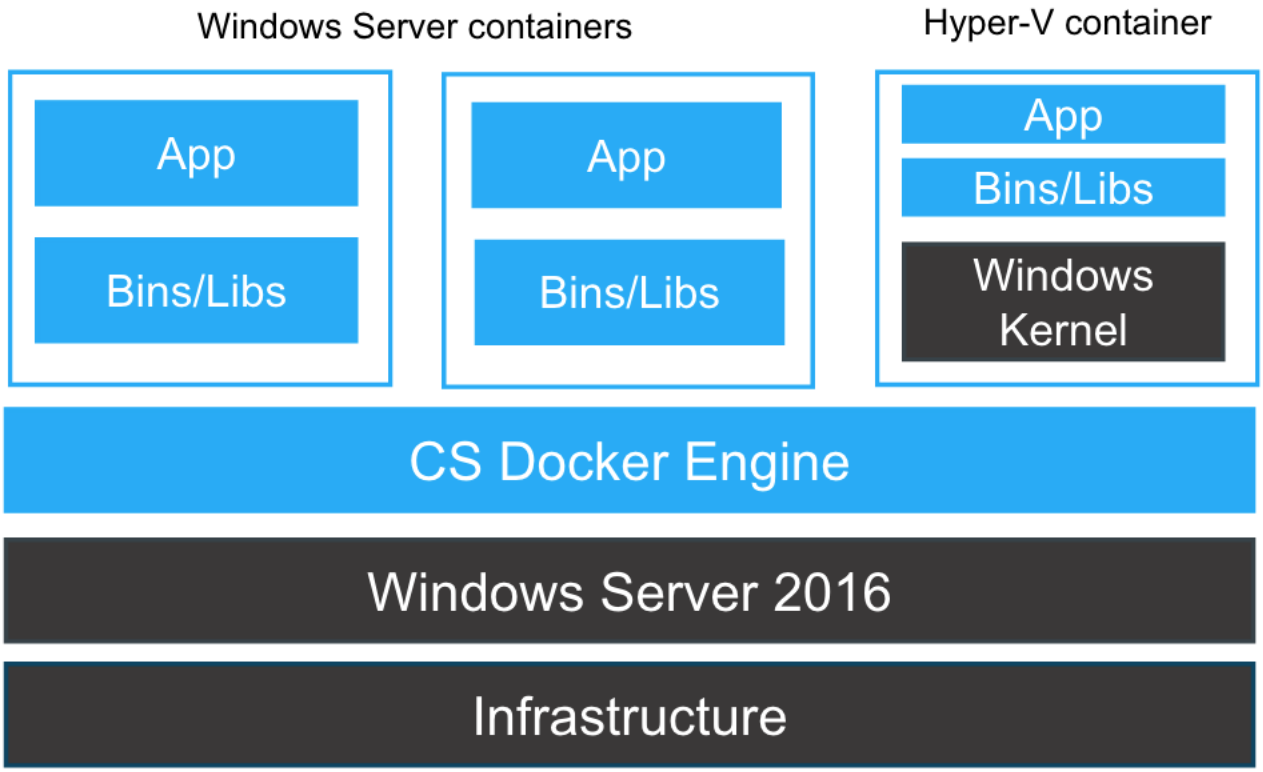
Afin de pouvoir isoler un ensemble de processus dans un système Linux, il a été décidé d'incorporer deux mécanismes :

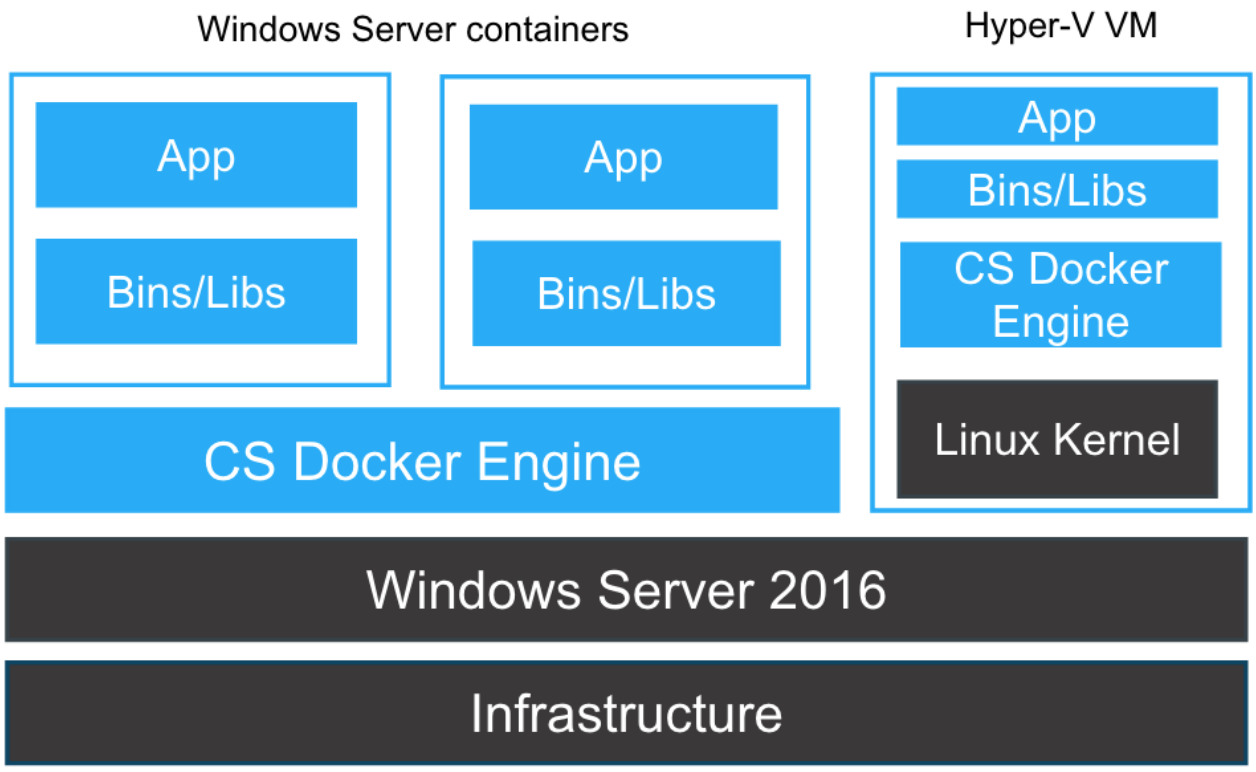
- **Namespaces** : Permettant d'offrir un espace de nommage spécifique pour un certain nombre de ressources, afin d'avoir une vision indépendante,
- **Cgroups** : Permettant d'enfermer des processus dans des groupes sur lesquels on pourra appliquer des limitations,

Ces deux éléments sont la base de l'isolation, de la conteneurisation, et par conséquents de Docker.

Ces deux notions (Namespaces et Cgroups) ont également été introduits dans Windows dans les dernières versions (à partir de Windows Server 2016), au travers le nouveau service HCS (Host Compute Service) de Hyper V.


Platform Independent
Docker Client
Docker PowerShell
Docker Swarm
Docker Registry
REST Interface
Docker Engine
Platform Specific
libcontainerd
libnetwork
graph
plugins
Windows
Compute Services
Control Groups
Job objects
Namespaces
Object Namespace, Process Table, Networking
Layer Capabilities
Registry, Union like filesystem extensions
Other OS Functionality
Linux
Control Groups
cgroups
Namespaces
Pid, net, ipc, mnt, uts
Layer Capabilities
Union Filesystems: AUFS, btrfs, vfs, zfs*, DeviceMapper
Other OS Functionality


Windows Server containers
Hyper-V container
App
Bins/Libs
App
Bins/Libs
App
Bins/Libs
Windows Kernel
CS Docker Engine
Windows Server 2016
Infrastructure


Windows Server containers
Hyper-V VM
App
Bins/Libs
App
Bins/Libs
App
Bins/Libs
CS Docker Engine
Linux Kernel
CS Docker Engine
Windows Server 2016
Infrastructure

1.2 - Pré-Installation

On change le nom de la machine, ce qui nous servira dans la suite.

```
root@skel:~# hostname master
root@skel:~# hostname
master
```

Egalement de manière définitive :

```
root@skel:~# hostname > /etc/hostname
root@skel:~# hostname -I
10.3.134.190
```

On l'ajoute également dans le système :

```
root@skel:~# echo "10.3.134.190 master" >> /etc/hosts
```

On se déconnecte et on se reconnecte ...

1.3 - Installation

Installation classique :

```
[root@master ~]# apt update
...
[root@master ~]# apt install docker.io
...
```

Puis on démarre et on vérifie le service associé :

```
[root@master ~]# systemctl start docker
[root@master ~]# systemctl enable docker
...
[root@master ~]# systemctl status docker
● docker.service - Docker Application Container Engine
 Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:enabled)
  Active: active (running) since ven. 2022-03-23 14:18:29 CET; 1min 58s ago
    Docs: https://docs.docker.com
```

1.4 - Commande « docker »

Pour connaître les commandes de « Docker » :

```
[root@drocourt ~]# docker
Usage: docker [OPTIONS] COMMAND [arg...]

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 from a container's filesystem to the
host path
  create    Create a new container
  diff      Inspect changes on a container's filesystem
  events    Get real time events from the server
  exec      Run a command in a running container
  export    Stream the contents of a container as a tar archive
  history   Show the history of an image
  images    List images
  import      Create a new filesystem image from the contents of a
tarball
  info      Display system-wide information
  inspect   Return low-level information on a container or image
  kill      Kill a running container
  load      Load an image from a tar archive
  login     Register or log in to a Docker registry server
```

```
  logout    Log out from a Docker registry server
  logs      Fetch the logs of a container
  port          Lookup the public-facing port that is NAT-ed to
PRIVATE_PORT
  pause     Pause all processes within a container
  ps        List containers
  pull        Pull an image or a repository from a Docker registry
server
  push      Push an image or a repository to a Docker registry server
  rename    Rename an existing container
  restart   Restart a running container
  rm        Remove one or more containers
  rmi       Remove one or more images
  run       Run a command in a new container
  save      Save an image to a tar archive
  search    Search for an image on the Docker Hub
  start     Start a stopped container
  stats         Display a stream of a containers' resource usage
statistics
  stop      Stop a running container
  tag       Tag an image into a repository
  top       Lookup the running processes of a container
  unpause   Unpause a paused container
  version   Show the Docker version information
  wait      Block until a container stops, then print its exit code
Run 'docker COMMAND --help' for more information on a command.
```

Pour obtenir des informations :

```
[root@drocourt ~]# docker info
Containers: 5
 Running: 0
 Paused: 0
 Stopped: 5
Images: 2
Server Version: 1.13.1
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 12
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd              version:                                (expected:
aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1)
runc version: N/A (expected: 9df8b306d01f59d3a8029be411de015b7304dd8f)
init version: N/A (expected: 949e6facb77383876aeff8a6944dde66b3089574)
```

```
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.4.0-87-generic
Operating System: Ubuntu 16.04.3 LTS
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 3.859 GiB
Name: ub16
ID: PUXJ:HXI3:V24U:35AZ:FOUM:3UNR:OFTV:ASCC:VRRH:MPP6:QLIA:KGJJ
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

2 - Les images

2.1 - Recherche d'images

Il est possible de rechercher des images spécifiques à l'aide de la commande « docker search <motif> », par exemple :

```
[root@drocourt ~]# docker search hello-world
NAME                                      DESCRIPTION                                     STARS   OFFICIAL  AUTOMATED
hello-world                               Hello World! (an example of minimal Docker...   478     [OK]
kitematic/hello-world-nginx               A light-weight nginx container that demons...   95
tutum/hello-world                         Image to test docker deployments. Has Apac...   48                [OK]
dockercloud/hello-world                   Hello World!                                    13                [OK]
hypriot/armhf-hello-world                 Hello World! (an example of minimal Docker...   5
marcells/aspnet-hello-world               ASP.NET vNext - Hello World                     5                 [OK]
armhf/hello-world                         Hello World! (an example of minimal Docker...   5
bonomat/nodejs-hello-world                a simple nodejs hello world container           3                 [OK]
crccheck/hello-world                      Hello World web server in under 2.5 MB          2                 [OK]
carinamarina/hello-world-app              This is a sample Python web application, r...   1                 [OK]
ppc64le/hello-world                       Hello World! (an example of minimal Docker...   1
infrastructureascode/hello-world          A tiny "Hello World" web server with a hea...   0                 [OK]
ansibleplaybookbundle/hello-world-apb     An APB which deploys a sample Hello World!...   0                 [OK]
ansibleplaybookbundle/hello-world-db-apb  An APB which deploys a sample Hello World!...   0                 [OK]
kevindockercompany/hello-world                                                            0
s390x/hello-world                         Hello World! (an example of minimal Docker...   0
gscrivano/hello-world                     hello world example system container            0                 [OK]
burdz/hello-world-k8s                     To provide a simple webserver that can hav...   0                 [OK]
hello-seattle                             Hello from DockerCon 2016 (Seattle)!            0       [OK]
lkungs/docker-hello-world                 Simple Hello World Example                      0                 [OK]
stumacsolutions/hello-world-container                                                     0
ashleybarrett/node-hello-world            Simple "hello world" image using node.          0
uniplaces/hello-world                                                                     0
sharor/hello-world                                                                        0
jensendw/hello-world                                                                      0
```

2.2 - Registry Docker

Depuis quelques temps, le trafic est limité sur le Docker Hub, c'est à dire le serveur officiel qui héberge les images par défaut. Si vous êtes à votre domicile, suivant la limite est largement suffisante, néanmoins sur le site de l'Université cela peut être bloquant.

En effet, la quantité de donnée téléchargée est calculée PAR ADRESSE IP, c'est donc une seule connexion Docker qui est comptabilisé pour le site de l'UPJV.

L'alternative est de se créer un compte sur Docker Hub et de l'utiliser pour manipuler les images, de cette manière, le quota sera par utilisateur, ce qui vous laisse de la marge :

```
[root@drocourt ~]# docker login -u rahan
Password:
Login Succeeded
```

Il est possible d'obtenir une erreur du type :

```
Cannot autolaunch D-Bus without X11 $DISPLAY
```

Dans ce cas, ls solution la plus simple (mais pas la plus sécurisée ni la plus élégante) est de renommer la commande « docker-credential-secretservice » :

```
[root@drocourt ~]# cd /usr/bin/
[root@drocourt ~]# mv docker-credential-secretservice
docker-credential-secretservice.orig
[root@drocourt ~]# docker login -u rahan
Password: XXXXX
WARNING! Your password will be stored unencrypted in
/root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/
#credentials-store

Login Succeeded
```

2.3 - Récupération d'images

Pour exécuter notre premier programme, il est possible de l'appeler directement, « docker » se chargeant de le télécharger automatiquement :

```
[root@drocourt ~]# docker run hello-world
Hello from Docker.
This message shows that your installation appears to be working correctly.

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 Hub.
 3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.
...
```

Il est cependant en général préférable de télécharger en amont l'image, et de le créer ensuite, par exemple pour « centos » :

```
[root@drocourt ~]# docker search centos
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
centos                              The official build of CentOS.                   4121      [OK]
ansible/centos7-ansible             Ansible on Centos7                              105                  [OK]
jdeathe/centos-ssh                  CentOS-6 6.9 x86_64 / CentOS-7 7.4.1708 x8...   93                   [OK]
consol/centos-xfce-vnc              Centos container with "headless" VNC sessi...   49                   [OK]
imagine10255/centos6-lnmp-php56     centos6-lnmp-php56                              40                   [OK]
tutum/centos                        Simple CentOS docker image with SSH access      36
gluster/gluster-centos              Official GlusterFS Image [ CentOS-7 +  Glu...   25                   [OK]
centos/python-35-centos7            Platform for building and running Python 3...   19
kinogmt/centos-ssh                  CentOS with SSH                                 17                   [OK]
openshift/base-centos7              A Centos7 derived base image for Source-To...   17
openshift/jenkins-2-centos7         A Centos7 based Jenkins v2.x image for use...   10
centos/postgresql-96-centos7        PostgreSQL is an advanced Object-Relationa...   10
openshift/mysql-55-centos7          DEPRECATED: A Centos7 based MySQL v5.5 ima...   6
openshift/jenkins-1-centos7         DEPRECATED: A Centos7 based Jenkins v1.x i...   3
pivotaldata/centos-gpdb-dev         CentOS image for GPDB development. Tag nam...   3
darksheer/centos                    Base Centos Image -- Updated hourly             3                    [OK]
openshift/wildfly-101-centos7       A Centos7 based WildFly v10.1 image for us...   3
openshift/php-55-centos7            DEPRECATED: A Centos7 based PHP v5.5 image...   1
blacklabelops/centos                CentOS Base Image! Built and Updates Daily!     1                    [OK]
pivotaldata/centos                  Base centos, freshened up a little with a ...   1
pivotaldata/centos-mingw            Using the mingw toolchain to cross-compile...   1
openshift/wildfly-100-centos7       A Centos7 based WildFly v10.0 image for us...   1
pivotaldata/centos-gcc-toolchain    CentOS with a toolchain, but unaffiliated ...   0
smartentry/centos                   centos with smartentry                          0                    [OK]
jameseckersall/sonarr-centos        Sonarr on CentOS 7                              0                    [OK]
```

Exercice : Recherchez l'image « alpine »

Nous pouvons maintenant l'installer avec la commande « pull » de Docker :

```
root@ub22-60g:~# docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
213ec9aee27d: Pull complete
Digest:
sha256:bc41182d7ef5ffc53a40b044e725193bc10142a1243f395ee852a
8d9730fc2ad
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Pour lister les images de conteneurs présents sur le système :

```
[root@drocourt ~]# docker images
REPOSITORY      TAG             IMAGE ID        CREATED         SIZE
alpine          latest          9c6f07244728    4 weeks ago     5.54MB
hello-world     latest          f2a91732366c    4 months ago    1.85 kB
```

Pour obtenir des informations sur l'historique d'une image installé :

```
[root@drocourt ~]# docker history centos
IMAGE          CREATED       CREATED BY                                      SIZE
2d194b392dd1   2 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0 B
<missing>      2 weeks ago   /bin/sh -c #(nop)  LABEL name=CentOS Base ...   0 B
<missing>      2 weeks ago   /bin/sh -c #(nop) ADD file:8d83f3e2c14f39e...   195 MB
```

De même pour une autre distribution :

```
[root@drocourt ~]# docker pull <other>
latest: Pulling from <other>
a29f2b1e7978: Pull complete
432ac41e2bbf: Pull complete
```

2.4 - Exécution des conteneurs

Un conteneur est généralement définit par un cadre d'utilisation, et il exécute donc une instruction au démarrage (« Entrypoint » que nous verrons plus tard) :

```
[root@drocourt ~]# docker run -i -t alpine
[root@d20dadad2e73 /]# id
uid=0(root) gid=0(root) groups=0(root)
[root@d20dadad2e73 /]# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 sh
   16 ?        00:00:00 ps
[root@d20dadad2e73 /]# exit
exit
```

Les options « -i » et « -t » permettent respectivement d'exécuter le conteneur en mode interactif, et d'attacher le terminal courant sur les entrées/sorties de ce dernier.

Comme indiqué précédemment, un conteneur peut être utilisé pour réaliser son action par défaut, mais il est également possible en général de réaliser une autre action, par exemple :

```
[root@d…rt ~]# docker run -i -t alpine /bin/echo hello world
hello world
[root@drocourt ~]#
```

Ici, nous demandons l'exécution du conteneur « centos » pour réaliser l'exécution de la commande « /bin/echo » présente dans ce conteneur.

IMPORTANT :

Il est important de préciser à ce point que le conteneur se termine lorsque la commande demandée, ou le point d'entrée, est terminée.

Deuxième exemple :

```
[root@drocourt ~]# docker run -i -t alpine /bin/sh

[root@a4dc20f812b0 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt
proc  root  run  sbin  srv  sys  tmp  usr  var
[root@a4dc20f812b0 /]# pwd
/
[root@a4dc20f812b0 /]# ps ax
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh
    8 root      0:00 ps aux
[root@a4dc20f812b0 /]# exit
[root@drocourt ~]#
```

Il est possible de visualiser les conteneurs en cours avec la commande « ps » :

```
[root@drocourt ~]# docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED        STATUS        PORTS   NAMES
```

Mais aussi de visualiser tous les conteneurs du système avec l'option « -a » :

```
[root@drocourt ~]# docker ps -a
CONTAINER ID    IMAGE          COMMAND                  CREATED             STATUS                          PORTS     NAMES
6772e78be1a7    alpine         "/bin/sh"                51 seconds ago      Exited (0) 14 seconds ago                 flamboyant_einstein
2ef65ab39acb    alpine         "/bin/echo hello w..."   About a minute ago  Exited (0) About a minute ago             mystifying_carson
f96cd41a1929    alpine         "/bin/sh"                2 minutes ago       Exited (0) About a minute ago             eaceful_visvesvaraya
bd130973fd11    hello-world    "/hello"                 8 minutes ago       Exited (0) 8 minutes ago                  hardcore_poitras
```

Ainsi, nous voyons qu'il est possible d'identifier un conteneur par :

- Son « CONTAINER ID » ou « ID »,
- Son « NAMES », attribué automatiquement par Docker, mais qu'il est également possible de choisir au moment de l'instanciation par l'option « --name ».

2.5 - Environnement

Il est possible de positionner des variables d'environnement à la création du conteneur à l'aide de l'option « -e », ce qui permet par exemple de passer des arguments d'initialisation :

```
[root@drocourt ~]# docker run -it -e TOTO=ok alpine
[root@c933cdeaf05a /]# echo $TOTO
ok
[root@c933cdeaf05a /]# exit
exit
```

2.6 - Création de conteneur

Il est également possible de créer un conteneur sans le démarrer automatiquement à l'aide de la directive « create » :

```
[root@drocourt ~]# docker create -i -t --name alp1 alpine
3249e0973ed04321b349bffe19417ca9656e5c00c1373d5027d9213d104cdb00
```

On vérifie :

```
[root@drocourt ~]# docker ps
CONTAINERID  IMAGE         COMMAND     CREATED       STATUS PORTS NAMES
[root@drocourt ~]# docker ps -a
CONTAINERID  IMAGE         COMMAND     CREATED       STATUS PORTS NAMES
3249e0973ed0 alpine:latest "/bin/sh" 6 seconds ago               alp1
```

3 - Interactions

3.1 - Arrière plan

Un conteneur se termine dès que la commande associée l'est aussi. Par contre si cette dernière perdure dans le temps, il est préférables de placer le conteneur en arrière plan à l'aide de l'option « -d » :

```
[root@drocourt ~]# docker run -d debian /bin/sh -c "while true; do echo hello world; sleep 1; done"
5dc2eb77abf838484b7ed8bba18259ece4f6448ec44f90ecac44ecc87ddf2a79
```

On peut vérifier son exécution :

```
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND     CREATED       STATUS      PORTS NAMES
5dc2eb77abf8 debian:latest "/bin/sh -c 7 seconds ago Up 6 seconds      backstabbing_poincare
```

3.2 - Sortie

Il est possible de récupérer les derniers messages affichés sur la sortie standard, à l'aide du « ID » :

```
[root@drocourt ~]# docker logs 5dc2eb77abf8
hello world
hello world
hello world
[root@drocourt ~]#
```

Ou du « NAME » :

```
[root@drocourt ~]# docker logs backstabbing_poincare
hello world
hello world
hello world
[root@drocourt ~]#
```

Remarque : L'option « -f » permet de suivre les logs.

3.3 - Attachement

Il est possible de s'attacher à un conteneur existant à l'aide de la commande « attach ». Toutefois, il est quasiment impossible de ressortir de cet attachement car le conteneur est considéré comme non interruptible !!

On prendra donc soin d'ajouter l'option « --sig-proxy=false » afin de pouvoir se détacher à l'aide d'un <ctrl><c> :

```
[root@drocourt ~]# docker attach --sig-proxy=false backstabbing_poincare
hello world
hello world
hello world
^C
```

Il est alors possible de constater que le conteneur est toujours en exécution à l'aide de la commande « docker ps ».

3.4 - Arrêt et démarrage

Nous avons vu qu'il était possible de créer un conteneur à l'aide de la commande « docker run ». Cependant, cette dernière commande va réellement créer un nouveau conteneur à chaque appel, pour démarrer un conteneur existant on utilisera « docker start » :

```
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND  ... NAMES
5dc2eb77abf8 debian:latest "/bin/sh ... backstabbing_poincare
[root@drocourt ~]# docker stop backstabbing_poincare
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND  ... NAMES
[root@drocourt ~]# docker start backstabbing_poincare
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND     ... NAMES
5dc2eb77abf8 debian:latest "/bin/sh    ... backstabbing_poincare
```

Il est également possible de supprimer un conteneur lorsque celui ci n'est plus nécessaire à l'aide de la directive « rm » :

```
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND     CREATED         STATUS      PORTS  NAMES
cfb671d0a6c4 debian:latest "/bin/bash" 13 minutes ago  Up 4 seconds       determined

[root@drocourt ~]# docker stop cfb671d0a6c4
cfb671d0a6c4

[root@drocourt ~]# docker rm cfb671d0a6c4
cfb671d0a6c4

[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND     CREATED         STATUS      PORTS  NAMES
```

3.5 - Interaction avec un conteneur existant

Il est possible d'exécuter des commandes dans un conteneur en cours d'exécution à l'aide de la commande « exec » à condition que celui ci soit démarré :

```
[root@drocourt ~]# docker exec alp1 ps
FATA[0000] Error response from daemon: Container cos7 is not
running
[root@drocourt ~]# docker start alp1
cos7
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE         COMMAND     ... NAMES
1e575cbc180f centos:latest "/bin/bash" ... cos7
[root@drocourt ~]# docker exec alp1 ps
  PID TTY          TIME CMD
   14 ?        00:00:00 ps
```

Pour lister simplement les processus, une commande plus simple existe :

```
[root@drocourt ~]# docker top cos7
```

```
[root@drocourt ~]# docker exec alp1 ip addr
...
15: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 02:42:ac:11:00:06 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.6/16 scope global eth0
...
```

Si la commande « ip » n'est pas installée :

```
[root@drocourt ~]# docker exec alp1 hostname -i
172.17.0.6
```

Il est même possible d'exécuter un shell à la condition de gérer correctement les attachements d'entrées/sorties (options « -i » et « -t ») :

```
[root@drocourt ~]# docker exec -it alp1 /bin/sh
```

3.6 - Démarrage automatique

Dans les versions de Docker > 1.2 il est possible de spécifier lors de la création d'un conteneur qu'il dit être démarré automatiquement à l'aide de l'option « --restart ».

Ainsi, pour reprendre l'un de nos exemples précédents :

```
[root@drocourt ~]# docker run -d --restart always --name dock1 alpine
```

3.7 - Suppression

Comme remarqué, un conteneur reste dans Docker même si il s'est arrêté. Il peut être nécessaire alors de le supprimer à l'aide de la directive « docker rm ».

Si le conteneur est temporaire et doit être effacé automatiquement, on peut utiliser l'option « --rm » lors de la création à l'aide de la commande « docker run ».

Pour supprimer complètement l'image qui est complètement indépendante du conteneur, on utilisera la commande « docker rmi ».

3.8 - Surveillance

Pour surveiller l'état des conteneurs lancés (ou tous avec « -a »), on peut utiliser :

```
[root@drocourt ~]# docker stats
```

4 - Le réseau

4.1 - Redirection de ports

Il est possible de rediriger un port local vers un port du conteneur à l'aide de l'option « -p ». Cela pourra être utile par exemple dans le cas d'un service comme Apache Httpd :

```
[root@drocourt ~]# docker search apache
NAME      DESCRIPTION                                     STARS  OFFICIAL   AUTOMATED
tomcat    Apache Tomcat is an open source implementa...   1757   [OK]
httpd     The Apache HTTP Server Project                  1583   [OK]
...
[root@drocourt ~]# docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
f2b6b4884fc8: Pull complete
...
0cc56b739fe0: Pull complete
Digest:
sha256:cf82f4031e4e9f20c50ebf155ba281e302f3ae07ae292b16b9bcf9a689c80b9
9
Status: Downloaded newer image for httpd:latest
```

Nous allons maintenant exécuter cette image en effectuant une redirection de port locale 8080 vers le port 80 du conteneur :

```
[root@drocourt ~]# docker run -d -p 8080:80 httpd
c5f23275b78fb987ae634b7f898fa15a8b851d93e320606ca0589956cb1856a4
```

On vérifie :

```
[root@drocourt ~]# docker ps
CONTAINER ID IMAGE  COMMAND        ...    PORTS                    NAMES
c5f23275b78f httpd "httpd-foreground" ...

                                0.0.0.0:8080->80/tcp   goofy_mcclintock
```

Et on test en local à l'aide de « curl » :

```
[root@drocourt ~]# curl -i http://localhost:8080
HTTP/1.1 200 OK
Date: Thu, 08 Oct 2015 06:16:01 GMT
Server: Apache/2.4.16 (Unix)
Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
ETag: "2d-432a5e4a73a80"
Accept-Ranges: bytes
Content-Length: 45
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>
```

Exercice : Créez le conteneur Apache mais en ajoutant l'option de suppression automatique et en le nommant « apache1 ».

4.2 - Liens entre conteneurs

Il est possible de vouloir faire communiquer des conteneurs entre eux, mais par défaut cela ne fonctionne pas :

```
[root@drocourt ~]# docker run --rm -d --name alp2 alpine sleep 1000
4694dce05f427b19aefdf43fbb22c574c165c9541ebd3fb4804ee74bb36f1114
[root@drocourt ~]# docker run --rm -it alpine ping -c 5 alp2
ping: bad address 'alp2'
```

Pour cela il suffit d'utiliser l'option « --link » dont le format est le suivant :

```
--link <nom_reel>:<alias>
```

Exemple :

```
[root@drocourt ~]# docker run -d  --name alp3 centos sleep 1000
9ec7e61749bf94f96571e4397cf348d73207bf283ec7593800d0f45d0e80f2eb
[root@drocourt ~]# docker run -it --rm --link alp3:alp3 alpine
```

Dans le conteneur :

```
[root@d4a65c3e84c5 /]# ping -c 2 alp3
PING t1 (172.17.0.25) 56(84) bytes of data.
64 bytes from t1 (172.17.0.25): icmp_seq=1 ttl=64 time=0.069 ms
64 bytes from t1 (172.17.0.25): icmp_seq=2 ttl=64 time=0.045 ms

--- t1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.045/0.057/0.069/0.012 ms
```

```
[root@d4a65c3e84c5 /]# cat /etc/hosts
127.0.0.1 localhost
::1  localhost ip6-localhost ip6-loopback
fe00::0   ip6-localnet
ff00::0   ip6-mcastprefix
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters
172.17.0.4     alp3 59e70b0813d6
172.17.0.5   9c09e25597d3
```

Remarque : *le lien n'est nécessaire que lorsque les conteneurs sont sur le réseau par défaut (cf. paragraphe 4.3 -).*

4.3 - Réseaux internes

Par défaut, un conteneur sera automatiquement ajouté au réseau nommé « bridge », mais il est possible d'en créer d'autres, il est d'ailleurs conseillé de créer un ou plusieurs réseaux et de ne pas utiliser celui par défaut :

```
root@ub16:~# docker network create mynet1
4d606e7f13bf044c9134dfb49667b85f3b966887e354e8d58c85168bc1cd
6f09
```

Pour vérifier :

```
root@ub16:~# docker network list
NETWORK ID          NAME                DRIVER              SCOPE
c9ce1b5d0d05        bridge              bridge              local
8b4d1f905741        host                host                local
4d606e7f13bf        mynet1              bridge              local
cb38f3a01ded        none                null                local
```

Il est possible de vérifier l'existence de ce Bridge sous Linux :

```
root@ub16:~# brctl show
```

Nous allons créer un container avec le réseau par défaut :

```
root@ub16:~# docker run -it --rm --name c7-mynet1 centos
```

On vérifie les paramètres IPs :

```
[root@3ad5c5ab3065 /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 0.0.0.0
        inet6 fe80::42:acff:fe11:2  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 2313  bytes 14108413 (13.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 890  bytes 62101 (60.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
[root@3ad5c5ab3065 /]# exit
```

Maintenant en intégrant le réseau précédent :

```
root@ub16:~# docker run -t -d --rm --name c7-mynet1 --network mynet1 centos
```

```
root@ub16:~# docker exec c7-mynet1 hostname -I
172.18.0.2
```

On peut créer un deuxième conteneur dans le même réseau :

```
root@ub16:~# docker run -t -d --rm --name c7-mynet2 --network mynet1 centos
```

```
root@ub16:~# docker exec c7-mynet2 hostname -I
172.18.0.3
```

On peut tester la connexion entre les deux :

```
root@ub16:~# docker exec c7-mynet2 ping -c 3 c7-mynet1
PING c7-mynet1 (172.18.0.2) 56(84) bytes of data.
64 bytes from c7-mynet1.mynet1 (172.18.0.2): icmp_seq=1 ttl=64
time=0.070 ms
64 bytes from c7-mynet1.mynet1 (172.18.0.2): icmp_seq=2 ttl=64
time=0.079 ms
64 bytes from c7-mynet1.mynet1 (172.18.0.2): icmp_seq=3 ttl=64
time=0.081 ms

--- c7-mynet1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.070/0.076/0.081/0.011 ms
```

Remarque :

Lorsque les conteneurs sont dans le réseau par défaut, il ne se connaissent pas et il sera alors nécessaire d'indiquer l'option « --link <noeud> » sur la ligne de commande.

Exercice : Faites la manipulation avec l'image « alpine ».

Il est également possible de joindre un réseau après la création du container :

```
root@ub16:~# docker network connect bridge c7-mynet2
```

Pour tester que le lien a été ajouté :

```
root@ub16:~# docker exec c7-mynet2 hostname -I
172.18.0.3 172.17.0.4
```

Il est également possible de spécifier les caractéristiques du réseau :

```
root@ub16:~# docker network create --driver=bridge --
subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --
gateway=172.28.5.254 net2
a21b25f146f016d83133a74d4c2ddb5c5c4649e3eb0127fbc6542f2dd56f
```

On vérifie la création :

```
root@ub16:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c9ce1b5d0d05        bridge              bridge              local
8b4d1f905741        host                host                local
a21b25f146f0        net2                bridge              local
cb38f3a01ded        none                null                local
```

On va créer un conteneur dans ce réseau :

```
root@ub16:~# docker run -it --name c7-2 --network net2
centos
```

On vérifie les caractéristiques réseaux :

```
[root@3fcba93f132c /]# hostname -I
172.28.5.0
```

5 - Les volumes

5.1 - Introduction

Il est possible avec Docker de créer un conteneur qui accède à un espace disque externe :

- **Volume** : Désigne un partage de la machine locale géré par Docker,
- **Bind mounts** : Désigne un partage de la machine locale lié à un répertoire local spécifique,
- **tmpfs** : Désigne un partage depuis le conteneur, mais se situant dans le mémoire du « host »,

5.2 - Le partage de type « Volume »

Création

Pour créer un volume :

```
[root@drocourt ~]# docker volume create web1
```

Ce volume correspond en fait à un répertoire local :

```
[root@drocourt ~]# docker volume inspect web1
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/web1/_data",
        "Name": "web1",
        "Options": {},
        "Scope": "local"
    }
]
```

Pour lister les volumes existant :

```
[root@drocourt ~]# docker volume ls
DRIVER              VOLUME NAME
local               web1
```

Pour créer un conteneur qui utilise ce partage, deux solutions possibles :

- « -v » ou « --volume » : option d'origine, plus simple mais moins complète,
- « --mounts » : option plus récente qui permet de positionner d'avantage d'options,

Utilisation avec l'option « -v »

L'option « -v » demande 3 arguments séparés par le caractère « : » :

- Le nom du partage,
- Le point de montage dans le conteneur,
- Les options séparées par une virgule (si nécessaire),

Donc pour créer un nouveau conteneur avec « -v » :

```
[root@drocourt ~]# docker run -d -p 8081:80 --name=dock1 -v
web1:/usr/local/apache2/htdocs httpd
fc2ca5e318ad6cc2ba864d1d6ea5ee43e1f0c735bce2c3cb1b84d6bd4888
```

On vérifie :

```
[root@drocourt ~]# docker ps
CONTAINER ID        IMAGE               ... PORTS                    NAMES
fc2ca5e318ad docker.io/httpd:latest ... 0.0.0.0:8081->80/tcp   dock1
```

Pour avoir les infos des points de montages du conteneur :

```
[root@drocourt ~]# docker inspect dock1
...
   "Mounts": [
       {
           "Type": "volume",
           "Name": "web1",
           "Source": "/var/lib/docker/volumes/web1/_data",
           "Destination": "/usr/local/apache2/htdocs",
           "Driver": "local",
           "Mode": "z",
           "RW": true,
           "Propagation": ""
       }
   ],
...
```

On peut ensuite tester avec une petite commande « curl » et surtout vérifier s'il y a des fichiers dans le répertoire en question !!

Nous allons maintenant faire une deuxième manipulation. Pour cela nous allons créer un autre volume et y placer une page web que l'on va utiliser dans le conteneur :

```
[root@drocourt ~]# docker volume create web2
[root@. ~]# vi /var/lib/docker/volumes/web2/_data/index.html
<html> <body>
Super
</body></html>
```

Créez maintenant un conteneur « Apache » qui utilise ce volume et faites une requête « curl », alors ? Conclusion par rapport à la manipulation précédente ?

Utilisation avec l'option « --mount »

L'option « --mount » demande 4 arguments sous la forme « clef=valeur », séparés par le caractère « , » :

- type : Le type de partage à savoir « volume » (défaut), « bind » et tmpfs,
- source ou src : La source du montage,
- destination, dst ou target : La destination,
- Les options séparées par des virgules,

```
[root@drocourt ~]# docker run -d -p 8081:80 --name=dock1 --
mount src=web,dst=/usr/local/apache2/htdocs httpd
```

Création automatique

Il est également possible de partager un répertoire entre deux conteneurs sans passer par la création explicite du volume, il sera alors créé directement dans le répertoire « /var/lib/docker/volumes/ » sous un nom spécifique :

```
docker run -d -v /partage <cont1>
```

Pour l'utiliser dans le second :

```
docker run -d --volumes-from=<ID cont1> <cont2>
```

5.3 - Le partage de type « bind »

Création

Pour cela on peut simplement créer un répertoire local dans lequel on placera les fichiers à utiliser :

```
[root@drocourt ~]# mkdir /var/tmp/apache
```

On va créer une page web que l'on va utiliser dans le conteneur :

```
[root@.t ~]# vi /var/tmp/apache/index.html
<html> <body>
Extra
</body></html>
```

La aussi deux options possibles :

- « -v » ou « --volume » : option d'origine, plus simple mais moins complète,
- « --mounts » : option plus récente qui permet de positionner d'avantage d'options,

Utilisation avec l'option « -v »

Il suffit ensuite d'utiliser le répertoire en indiquant son chemin :

```
[root@drocourt ~]# docker run -d -p 8082:80 --name=dock2 -v /var/tmp/apache:/usr/local/apache2/htdocs httpd
fc2ca5e31cc2ba8df4a64d1d6ea5ee43e1f0c735bce2c3cb1b84d6bd4888
```

On vérifie :

```
[root@drocourt ~]# docker ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS   PORTS                    NAMES
80d17cb7f2c3  httpd  "httpd"  57 sec   Up 56 s  0.0.0.0:8082->80/tcp   dock2
8e86cf7ab756  httpd  "httpd"  17 hours Up 15 h  0.0.0.0:8081->80/tcp   dock1
```

On peut vérifier le partage :

```
[root@drocourt ~]# docker inspect dock2
...
   "Mounts": [
       {
           "Type": "bind",
           "Source": "/var/tmp/apache",
           "Destination": "/usr/local/apache2/htdocs",
           "Mode": "",
           "RW": true,
           "Propagation": "rprivate"
       }
   ],
...
```

Utilisation avec l'option « --mount »

```
[root@drocourt ~]# docker run -d -p 8082:80 --name=dock2 --
mount type=bind,src=web,dst=/usr/local/apache2/htdocs httpd
```

5.4 - Le partage de type « tmpfs »

Spécificité

Le type « tmpfs » travaille en mémoire, les contraintes sont donc :

- Contrairement aux volumes, on ne peut les partager entre différents conteneurs,
- Cette fonctionnalité n'est disponible que pour Docker sous Linux,

Utilisation avec l'option « --tmpfs »

```
[root@drocourt ~]# docker run -d --name dock3 --tmpfs
/usr/local/apache2/htdocs -p 8083:80 httpd
```

On peut vérifier le partage :

```
[root@drocourt ~]# docker inspect dock3
...
    "Tmpfs": {
        "/usr/local/apache2/htdocs": ""
    },
```

Utilisation avec l'option « --mount »

```
[root@drocourt ~]# docker run -d -p 8083:80 --name=dock3 --
mount type=tmpfs,dst=/usr/local/apache2/htdocs httpd
```

Remarque :

Avec l'option « --mount » il est possible d'utiliser les options suivantes :

- *tmpfs-size : pour la taille du système de fichier,*
- *tmpfs-mode : pour le mode de création,*

5.5 - Exercice

1. Créer un volume nommé « portainer_data »,
2. Installez l'image « portainer/portainer-ce »,
3. Créez un container basé sur l'image précédente qui :

- se nomme « portainer »,
- redémarre automatiquement,
- exporte le port local 8000 sur le port externe 8000,
- exporte le port local 9443 sur le port externe 9443,
- exporte le répertoire « /data portainer/portainer-ce » sur le volume « portainer_data »,
- exporte le fichier « /var/run/docker.sock » vers le fichier « /var/run/docker.sock »,

Connectez vous en https sur le port 9443 et créez un utilisateur « admin » avec un mot de passe que vous m'enverrez par mail,

6 - Exercice récapitulatif

1. Installez l'image wordpress:latest,
2. Installez l'image mariadb:latest,
3. Créez un réseau net6,
4. Créez un volume vol6_bd,
5. Créez un conteneur basé sur « mariadb » qui :
   - Exporte le chemin vol_bd sur le répertoire du conteneur « /var/lib/mysql »,
   - Initialise les variables MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, MYSQL_USER et MYSQL_PASSWORD,
   - porte le nom dock6-bd,
6. Vérifiez à l'aide de la commande « docker ps »,
7. Quels ports sont exposés ? Pourquoi ?

8. Créez un conteneur basé sur « wordpress:latest » qui :
   - Porte le nom « dock6-wp »,
   - Puisse accéder au conteneur précédent,
   - Exporte son port 80 sur le port 8013 de la machine locale,
   - Initialise les variables WORDPRESS_DB_HOST, WORDPRESS_DB_USER, WORDPRESS_DB_PASSWORD et WORDPRESS_DB_NAME (la première est de la forme HOST:PORT),
9. Vérifiez avec la commande « docker ps »,
10. Connectez vous avec un navigateur sur la machine port 8013 et installez Wordpress.

7 - Versioning et Sauvegarde

7.1 - Version

Un conteneur représente un contexte d'exécution d'une image présente, ainsi si nous détruisons notre conteneur, nous avons perdu les traces de l'exécution et de la configuration de ce dernier.

Il est donc possible de demander à Docker de réaliser un enregistrement à un instant précis à l'aide de la commande « docker commit » afin de les réutiliser plus tard :

```
[root@drocourt ~]# docker ps -a
CONTAINER ID  IMAGE          COMMAND      ...  STATUS PORTS  NAMES
1e575cbc180f  debian:latest  "/bin/bash"  ...  Exited        deb1
[root@drocourt ~]# docker commit deb1 rahan/debian:latest
91f98c48edd6a83622f99182586be548f859fd53075bbf3d0f056084988d78d8
[root@drocourt ~]# docker images
REPOSITORY     TAG       IMAGE ID       CREATED          VIRTUAL SIZE
rahan/debian   latest    91f98c48edd6   3 seconds ago    123 MB
debian         latest    432ac41e2bbf   9 days ago       123 MB
```

Il est également possible de placer un « tag » sur une version d'image afin d'affecter un nom supplémentaire à l'image à l'aide de la commande « docker tag » :

```
[root@drocourt ~]# docker images
REPOSITORY     TAG       IMAGE ID       CREATED          VIRTUAL SIZE
rahan/debian   latest    91f98c48edd6   3 seconds ago    123 MB
debian         latest    432ac41e2bbf   9 days ago       123 MB
[root@drocourt ~]# docker tag rahan/debian:latest
rahan/debian:8.1
[root@drocourt ~]# docker images
REPOSITORY     TAG           IMAGE ID      CREATED          VIRTUAL SIZE
rahan/debian   latest    91f98c48edd6   About a minute ago   123 MB
rahan/debian   8.1       91f98c48edd6   About a minute ago   123 MB
debian         latest    432ac41e2bbf   9 days ago           123 MB
```

Remarque :

Il est habituel de nommé les images de la forme « dépôt/contenu:version », où le dépôt est soit le nom d'une personne soit celui d'une entreprise. De plus, un tag « latest » existe généralement sur la dernière version.

7.2 - Sauvegarde

Il faut bien garder à l'esprit qu'un conteneur est l'exécution mémoire d'une image à partir du disque. Il peut alors être intéressant de pouvoir sauvegarder les images et les conteneurs.

Il existe pour cela 2 directives « export » et « save », « export » est pour sauvegarder un conteneur alors que « save » est pour sauvegarder une image.

De la même manière, on utilisera les directives « import » et « load » pour restaurer respectivement un conteneur ou une image.

Exemple :

```
[root@drocourt ~]# docker export deb1 > deb1.tar
```

Pour le restaurer sur un autre système :

```
[root@drocourt ~]# cat deb1.tar | docker import - deb2
e80eef6fa48d21355cbcb0fb1bab2a20dce5473c1b8bb4140944557a287f
e0eb
```

On vérifie :

```
[root@drocourt ~]# docker images
REPOSITORY    TAG      IMAGE ID       CREATED         VIRTUAL SIZE
deb2          latest   e80eef6fa48d   11 seconds ago  123 MB
madebian1     latest   3de0a901ef3f   2 hours ago     123 MB
debian        latest   432ac41e2bbf   24 hours ago    123 MB
hello-world   latest   7a5a2d73abce   3 days ago      1.84 kB
```

7.3 - Registry Docker

Il est possible de sauvegarder une image sur un serveur distant offrant un service de « registry ». Si aucun serveur n'est précisé celui de « docker hub » est utilisé, à condition d'avoir créé un compte auparavant, et de s'être connecté :

```
[root@drocourt ~]# docker login -u rahan
Password:
Email: rahan@u13.org
```

On consulte les images possibles :

```
[root@drocourt ~]# docker images
REPOSITORY          TAG     IMAGE ID      CREATED       VIRTUAL SIZE
debian        8.1     91f98c48edd6  5 days ago    123 MB
```

Il est nécessaire de réaliser un tag en fonction de son nom de login, par exemple :

```
[root@drocourt ~]# docker tag debian:8.1 rahan/debian:8.1
```

On consulte les images possibles :

```
[root@drocourt ~]# docker images
REPOSITORY          TAG     IMAGE ID      CREATED       VIRTUAL SIZE
debian              8.1     91f98c48edd6  5 days ago    123 MB
rahan/debian        latest  91f98c48edd6  5 days ago    123 MB
```

On utilise la commande « push » pour sauvegarder :

```
[root@drocourt ~]# docker push rahan/debian:8.1
The push refers to a repository [rahan/debian] (len: 1)
91f98c48edd6: Image already exists
432ac41e2bbf: Image successfully pushed
a29f2b1e7978: Pushing [=====>          ] 23.59 MB/51.36 MB
```

On peu retrouver facilement ses images avec :

```
[root@drocourt ~]# docker search rahan/
```

7.4 - Registry personnel

Il est possible de mettre en place son propre « registry », soit en installant l'ensemble des services associés, soit en utilisant un conteneur de registry pré-existant. On récupère le conteneur :

```
[root@drocourt ~]# docker pull registry:2
```

On l'exécute :

```
[root@drocourt ~]# docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

On peut maintenant créer un nouveau conteneur :

```
[root@drocourt ~]# docker run --name cos7 -it centos
```

Une fois le conteneur créé, on installe le paquet « net-utils » à l'aide de la commande « yum » puis on quitte le conteneur.

On peut ensuite faire un « commit » :

```
[root@drocourt ~]# docker commit cos7 rahan/mycos7
```

Il est nécessaire ensuite d'effectuer un « tag » spécifique pour notre serveur :

```
[root@drocourt ~]# docker tag rahan/mycos7 localhost:5000/
mycos7
```

Nous pouvons maintenant faire un « push » :

```
[root@drocourt ~]# docker push localhost:5000/mycos7
```

Et enfin supprimer toute trace :

```
[root@drocourt ~]# docker rm cos7
[root@drocourt ~]# docker image rm centos
[root@drocourt ~]# docker image rm rahan/mycos7
[root@drocourt ~]# docker image rm localhost:5000/mycos7
```

Maintenant, il est possible de l'installer à partir de notre « registry » :

```
[root@drocourt ~]# docker pull localhost:5000/mycos7
[root@drocourt ~]# docker images
REPOSITORY                  TAG                 IMAGE ID
CREATED             SIZE
localhost:5000/mycos7   latest              4220126e97b3
46 minutes ago      300 MB
```

Il n'est pas possible a priori d'effectuer une recherche classique sur le « registry » local, mais on peut récupérer la liste par une simple commande « curl » :

```
[root@drocourt ~]# curl localhost:5000/v2/_catalog
```

7.5 - Exercice

1. Faites un tag de l'image alpine:latest puis un push dans le docker hub,
2. Créez un conteneur registry en ayant créé au préalable un volume qui sera utilisé sur /var/lib/registry,
3. Placer les images précédentes dans le registry,
4. Vérifiez en local dans le volume créé,