Premiers conteneurs Docker

Dans ce lab, vous allez exécuter un conteneur populaire, gratuit et léger et explorer les bases du fonctionnement des conteneurs, de la façon dont le moteur Docker s'exécute et isole les conteneurs les uns des autres.

Vous utiliserez une machine virtuelle Centos 7 pour exécuter ce Lab.

Concepts de cet exercice:

- Docker engine
- Conteneurs & images
- Registres d'images et Docker Hub
- Isolation des conteneurs

1. Exécuter votre premier conteneur

D'abord installez Docker

```
sudo yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine

sudo yum install -y yum-utils


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

sudo yum install docker-ce docker-ce-cli containerd.io \
docker-compose-plugin

sudo systemctl enable docker --now
```

Comme pour tout ce qui est technique, une application «bonjour le monde» est un bon point de départ.

Tout est en place pour déployer notre premier conteneur, la recherche permet de trouver notre conteneur hello-world.

```
[root@localhost ~]# docker search hello-world
INDEX       NAME                  DESCRIPTION           STARS     OFFICIAL   AUTOMATED
docker.io   docker.io/hello-world   Hello World!          1146      [OK]
…
```

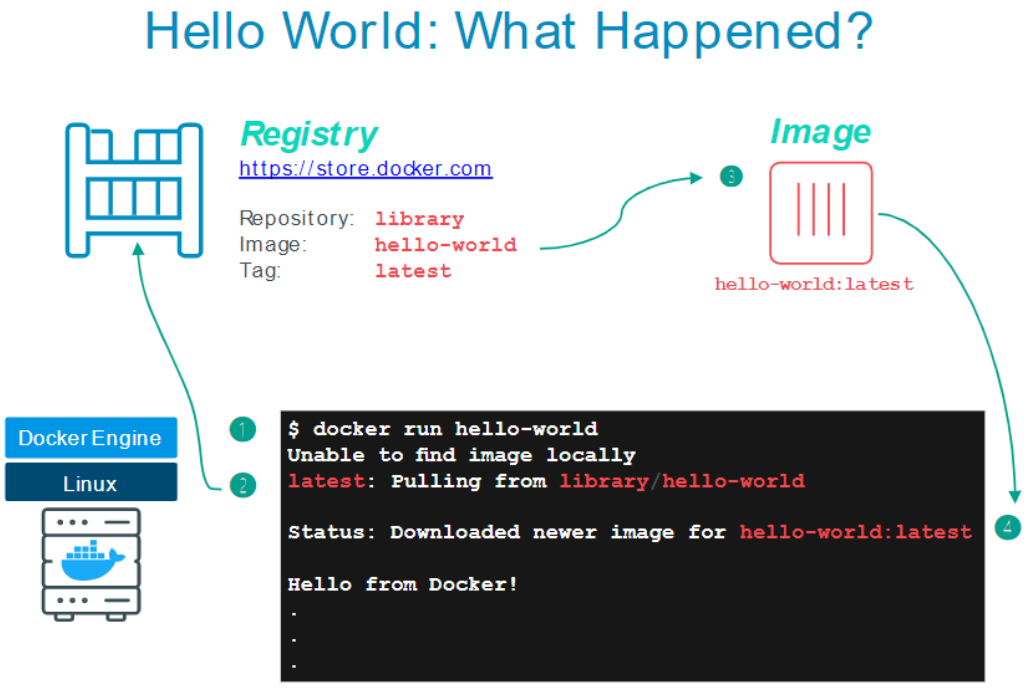
et commençons tout de suite par de la sensibilisation avec la colonne OFFICIAL.

Si vous voulez faire de la production, préférez partir d'une image officielle.

Commençons par l'installer.

```
[root@localhost ~]# docker container run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
1b930d010525: Pull complete
Digest:
sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752
Status: Downloaded newer image for docker.io/hello-world:latest

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.
    (amd64)
 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.

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/
```

C'est votre tout premier conteneur. La sortie du conteneur hello-world vous en dit essentiellement, le moteur Docker exécuté dans votre terminal a tenté de trouver une image nommée hello-world. Depuis que vous venez de commencer, aucune image n'est stockée localement (Unable to find image...), donc le moteur Docker va dans son registre Docker par défaut, qui est Docker Hub, pour chercher une image nommée «hello-world». Il y trouve l'image, la télécharge, puis l'exécute dans un conteneur. Et la seule fonction de hello-world est de sortir le texte que vous voyez dans votre terminal, après quoi le conteneur se ferme.



1.1 Images Docker

Dans ce qui suit, vous allez exécuter un conteneur Alpine Linux . Alpine est une distribution Linux légère, elle est donc rapide à déployer et à exécuter, ce qui en fait un point de départ populaire pour de nombreuses autres images.

```
[root@localhost ~]# docker image pull alpine
Using default tag: latest
Trying to pull repository docker.io/library/alpine ...
latest: Pulling from docker.io/library/alpine
c9b1b535fdd9: Pull complete
Digest:
sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for docker.io/alpine:latest
```

La commande pull récupère l'image alpine depuis Docker Hub et l'enregistre dans notre système.

Vous pouvez utiliser la commande docker image pour voir une liste de toutes les images sur votre système.

```
[root@localhost ~]# docker image ls
REPOSITORY              TAG           IMAGE ID            CREATED           SIZE
docker.io/alpine        latest        e7d92cdc71fe        7 weeks ago       5.59 MB
docker.io/hello-world   latest        fce289e99eb9        14 months ago     1.84 kB
```

Lorsque vous avez exécuté la commande *docker container run hello-world*, elle a également effectué une opération *docker image pull* en arrière-plan pour télécharger l'image de hello-world.

Pour en savoir plus sur une image Docker, exécutez *docker image inspect alpine*.

1.2 Exécution du conteneur Docker

Exécutez maintenant un conteneur Docker basé sur cette image. Pour ce faire, vous allez utiliser la commande docker container run.

```
[root@localhost ~]# docker container run alpine ls -l
total 8
drwxr-xr-x    2 root     root          4096 Jan 16 21:52 bin
drwxr-xr-x    5 root     root           340 Mar 10 02:43 dev
drwxr-xr-x    1 root     root            66 Mar 10 02:43 etc
drwxr-xr-x    2 root     root             6 Jan 16 21:52 home
drwxr-xr-x    5 root     root           185 Jan 16 21:52 lib
drwxr-xr-x    5 root     root            44 Jan 16 21:52 media
drwxr-xr-x    2 root     root             6 Jan 16 21:52 mnt
…
```

Lorsque vous appelez *run*, le client Docker recherche l'image (alpine dans ce cas), crée le conteneur, puis exécute une commande dans ce conteneur. Lorsque vous exécutez *docker container run alpine*, vous avez fourni une commande (ls -l), donc Docker a exécuté cette commande dans le conteneur pour lequel vous avez vu la liste des répertoires. Une fois la commande *ls* terminée, le conteneur s'est arrêté.

Essayons ce qui suit:

```
[root@localhost ~]# docker container run alpine echo "hello from alpine"
hello from alpine
```

Essayez encore une autre commande :

```
[root@localhost ~]# docker container run alpine /bin/sh
```

il ne s'est rien passé. Vous avez démarré une 3ème instance du conteneur alpin et il a exécuté la commande /bin/sh puis s'est arrêté. Il a simplement lancé le shell, quitté le shell, puis arrêté le conteneur.

Pour obtenir un shell interactif où vous pouvez taper quelques commandes, Docker a une option pour cela.

-i pour le mode interactif

-t pour avoir un pseudo TTY

Pour cet exemple, tapez ce qui suit:

```
[root@localhost ~]#  docker container run -it alpine /bin/sh
```

Vous êtes maintenant à l'intérieur du conteneur exécutant un shell Linux et vous pouvez essayer quelques commandes comme *ls -l*, *uname -a* et d'autres. Notez qu'Alpine est un petit système d'exploitation Linux, donc plusieurs commandes peuvent être manquantes. Quittez le shell et le conteneur en tapant la commande *exit*.

Chacune de nos commandes ci-dessus a été exécutée dans une instance de conteneur distincte qui c’est ensuite terminée.

Puisqu'aucun conteneur n'est en cours d'exécution, vous pouvez observer ces conteneurs avec :

```
[root@localhost ~]# docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
8b24af792f32        alpine              "/bin/sh"                22 hours ago        Exited
(0) 4 minutes ago                       jovial_euclid
8bd7acd259a2        alpine              "/bin/sh"                23 hours ago        Exited
(0) 23 hours ago                        distracted_bartik
22e008786238        alpine              "echo 'hello from ..."   23 hours ago        Exited
(0) 23 hours ago                        inspiring_lamport
8a5ea713d3e7        alpine              "ls -l"                  32 hours ago        Exited
(0) 32 hours ago                        upbeat_darwin
e31c2c982206        hello-world         "/hello"
```

La commande *docker container ls* en elle-même vous montre tous les conteneurs en cours d'exécution et n'affichera donc rien :

```
[root@localhost ~]# docker container ls
CONTAINER ID      IMAGE        COMMAND       CREATED         STATUS        PORTS       NAMES
```

1.3 Isolement des conteneurs

Pourquoi y a-t-il autant de conteneurs répertoriés s'ils proviennent tous de l' image alpine ?

Il s'agit d'un concept de sécurité critique dans Docker. Même si chaque commande *docker container run* utilisait la même image alpine, chaque exécution était un conteneur séparé et isolé. Chaque conteneur a un système de fichiers distinct et s'exécute dans un espace de noms différent; par défaut, un conteneur n'a aucun moyen d'interagir avec d'autres conteneurs, même ceux de la même image. Essayons un autre exercice pour en savoir plus sur l'isolement.

```
[root@localhost ~]# docker container run -it alpine /bin/ash
/ # echo hello > hello.txt
/ # ls
bin         dev         etc         hello.txt   home        lib         media
mnt         opt         proc        root        run         sbin        srv
sys         tmp         usr         var
/ # exit
```

Pour montrer comment fonctionne l'isolement, exécutez ce qui suit:

```
[root@localhost ~]# docker container run alpine ls
bin         dev         etc         home        lib         media       mnt
opt         proc        root        run         sbin        srv         sys
tmp         usr         var
```

C'est la même commande *ls* que nous avons utilisée dans le shell interactif ash du conteneur, mais cette fois, avez-vous remarqué que votre fichier "hello.txt" est manquant? C'est l'isolement! Votre commande a été exécutée dans une nouvelle instance distincte, même si elle est basée sur la même image.

Gestion de l'instance

```
[root@localhost ~]# docker container ls -a
CONTAINER ID      IMAGE     COMMAND    CREATED          STATUS      PORTS     NAMES
ff54e33f1ffc      alpine    "ls"       5 minutes ago    Exited (0)            hardcore_gold
cdf1a27b0fa9      alpine    "/bin/ash" 7 minutes ago    Exited (0)            nifty_lalande
```

Le conteneur dans lequel nous avons créé le fichier «hello.txt» est le même que celui où nous avons utilisé le shell /bin/ash, que nous pouvons voir répertorié dans la colonne «COMMAND». Le numéro d' ID de conteneur de la première colonne identifie de manière unique cette instance de conteneur particulière. Dans l'exemple de sortie ci-dessus, l'ID du conteneur est cdf1a27b0fa9. Nous pouvons utiliser une commande légèrement différente pour dire à Docker d'exécuter cette instance de conteneur spécifique. (Conseil: au lieu d'utiliser l'ID complet du conteneur, vous pouvez utiliser uniquement les premiers caractères, à condition qu'ils soient suffisants pour identifier de manière unique un conteneur.) Essayez de taper:

```
[root@localhost ~]# docker container start cdf1a
cdf1a
[root@localhost ~]# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
cdf1a27b0fa9        alpine              "/bin/ash"          11 minutes ago
Up 6 seconds          nifty_lalande
```

Nous pouvons envoyer une commande à exécuter dans le conteneur en utilisant la commande *exec*, comme suit:

```
[root@localhost ~]# docker container exec cdf1a ls
bin        dev        etc        hello.txt  home       lib        media
mnt        opt        proc       root       run        sbin       srv
sys        tmp        usr        var
```

Pour retourner dedans

```
[root@localhost ~]# docker container attach cdf1a
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/ash
    6 root      0:00 ps aux
/ #
```

Nous sommes à présent dans l'OS. Mais attention au Ctrl+D car ça va tuer l'instance en cours (en tuant le processus n°1 ash). Pour sortir en laissant le conteneur tourner il faut faire un **Ctrl+P** et ensuite **Ctrl+Q**.

Si vous vous reconnectez au conteneur avec la commande suivante :

```
[root@localhost ~]# docker container exec -ti cdf1a /bin/ash
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/ash
    7 root      0:00 /bin/ash
   12 root      0:00 ps aux
/ # ^D
```

à partir de là, vous pouvez quitter le conteneur avec un Ctrl+D sans tuer l'instance car cela tuera le shell en cours de PID 7.

```
[root@localhost ~]# docker container ls
CONTAINER ID  IMAGE   COMMAND     CREATED       STATUS    PORTS       NAMES
cdf1a27b0fa9  alpine  "/bin/ash"  4 hours ago   Up 3 hours    nifty_lalande
```

Si on veut tuer l'instance en cours :

```
[root@localhost ~]# docker stop cdf1a
```

Ou

```
[root@localhost ~]# docker kill cdf1a
```

Fondamentalement, la différence est subtile, mais décrite dans la référence de la ligne de commande :

- ✓ docker stop : arrête un conteneur en cours d'exécution (envoie SIGTERM, puis SIGKILL après la période de grâce) [...] Le processus principal à l'intérieur du conteneur recevra SIGTERM, et après une période de grâce, SIGKILL.
- ✓ docker kill : tuer un conteneur en cours d'exécution (envoyer SIGKILL, ou le signal spécifié). Le processus principal à l'intérieur du conteneur sera envoyé SIGKILL, ou tout signal spécifié avec l'option --signal.

Une alternative plus élégante pour ne pas avoir à vous souvenir de l'ID du conteneur serait d'attribuer un nom unique à chaque conteneur que vous créez en utilisant l'option --name sur la ligne de commande, comme dans l'exemple suivant

```
[root@localhost ~]# docker container run -it --name test1  alpine /bin/ash
/ # ^P^Q
[root@localhost ~]# docker container ls
CONTAINER ID   IMAGE   COMMAND     CREATED         STATUS     PORTS   NAMES
cda56c9ec50e   alpine  "/bin/ash"  12 seconds ago  Up 11 seconds      test1
```

Ensuite, en utilisant le nom que vous avez attribué au conteneur, vous pouvez manipuler le conteneur (start, stop, rm, top, stats) plus simplement en adressant son nom, comme dans les exemples ci-dessous :

```
[root@localhost ~]# docker container top test1
UID      PID      PPID      C   STIME      TTY     TIME            CMD
root     25158    25143     0   00:22      pts/1   00:00:00        /bin/ash

[root@localhost ~]# docker container stats test1
CONTAINER     CPU %    MEM USAGE / LIMIT    MEM %  NET I/O  BLOCK I/O  PIDS
b6f4cb566cf3  0.00%   132 KiB / 3.84 GiB   0.00%   0 B / 0 B  0 B / 0 B   1
```

Enfin si le conteneur ne sert plus vous pouvez le supprimer :

```
[root@localhost ~]# docker container rm test1
Error response from daemon: You cannot remove a running container
cda56c9ec50eb32600a966084b5e561b0a724a8c538ae109dca8be9101312356. Stop the
container before attempting removal or use -f
```

Biensur après avoir arrêté l'instance

```
[root@localhost ~]# docker container stop test1
test1
[root@localhost ~]# docker container rm test1
test1
```