Docker / URN-DSI 2021

tags: `docker` `dsi`

Docker (https://www.docker.com) est une plate-forme logicielle qui vous permet de concevoir, tester et déployer des applications rapidement. Docker intègre les logiciels dans des unités normalisées appelées conteneurs, qui rassemblent tous les éléments nécessaires à leur fonctionnement, dont les bibliothèques, les outils système, le code et l'environnement d'exécution. Avec Docker, vous pouvez facilement déployer et dimensionner des applications dans n'importe quel environnement, avec l'assurance que votre code s'exécutera correctement.

Cette présentation permet d'appréhender les concepts de base de Docker:

- les images et conteneurs
- les dockerfiles
- les réseaux
- les volumes
- docker-compose

Il vient en complément de la présentation d'introduction sur le fonctionnement de Docker.

Installation de Docker

Pour installer Docker, vous pouvez vous référer à la documentation officielle (https://docs.docker.com/engine/install/)

Utiliser la plateforme de formation

Vous disposez de 9 VMs : docker-01.univ-rouen.fr à docker-09.univ-rouen.fr

Pour vous connecter, vous devez utiliser le compte :

- login: formation
- password: dsi2021

le compte `formation` fait parti du group `docker`. Vous pouvez donc utiliser les commandes docker sans sudo.

Vérifier le status du service Docker

```
$ sudo systemctl status docker

● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
   Active: active (running) since Tue 2020-10-27 18:35:08 CET; 2min 42s ago
     Docs: https://docs.docker.com
 Main PID: 3252 (dockerd)
    Tasks: 10
   Memory: 50.6M
   CGroup: /system.slice/docker.service
           └─3252 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
```

Vérifier la version de Docker

```
 $ docker version

Client: Docker Engine - Community
 Version:           19.03.13
 API version:       1.40
 Go version:        go1.13.15
 Git commit:        4484c46d9d
 Built:             Wed Sep 16 17:02:55 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.13
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       4484c46d9d
  Built:            Wed Sep 16 17:01:25 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.3.7
  GitCommit:        8fba4e9a7d01810a393d5d25a3621dc101981175
 runc:
  Version:          1.0.0-rc10
  GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

Obtenir des infos sur la configuration Docker

```
$ docker system info

Client:
 Debug Mode: false

Server:
 Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
 Images: 1
 Server Version: 19.03.13
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 8fba4e9a7d01810a393d5d25a3621dc101981175
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.19.0-12-amd64
 Operating System: Debian GNU/Linux 10 (buster)
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 1.925GiB
 Name: debian
 ID: RV5D:BOAZ:K6V7:6DVE:2DGB:JTGK:YWDX:ZDBG:IQGB:NRBR:PYPU:DUTE
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
```

:::info
Cette commande retourne plusieurs informations intéressantes:

1: le driver de stockage est overlayFS: `Storage Driver: overlay2`
2: le dossier dans lequel Docker va stocker les images et les volumes: `Docker Root Dir: /var/lib/docker`
3: l'adresse de la registry pour télécharger les images: `Registry: https://index.docker.io/v1/`
:::

```
$ docker system df
docker system df
TYPE                TOTAL               ACTIVE              SIZE
RECLAIMABLE
Images              0                   0                   0B
0B
Containers          0                   0                   0B
0B
Local Volumes       0                   0                   0B
0B
Build Cache         0                   0                   0B
0B
```

:::success
`docker system df` permet d'obtenir un état précis de l'espace disque occupé par Docker.
:::

Gestion des images

Il existe 3 méthodes pour obtenir une image :

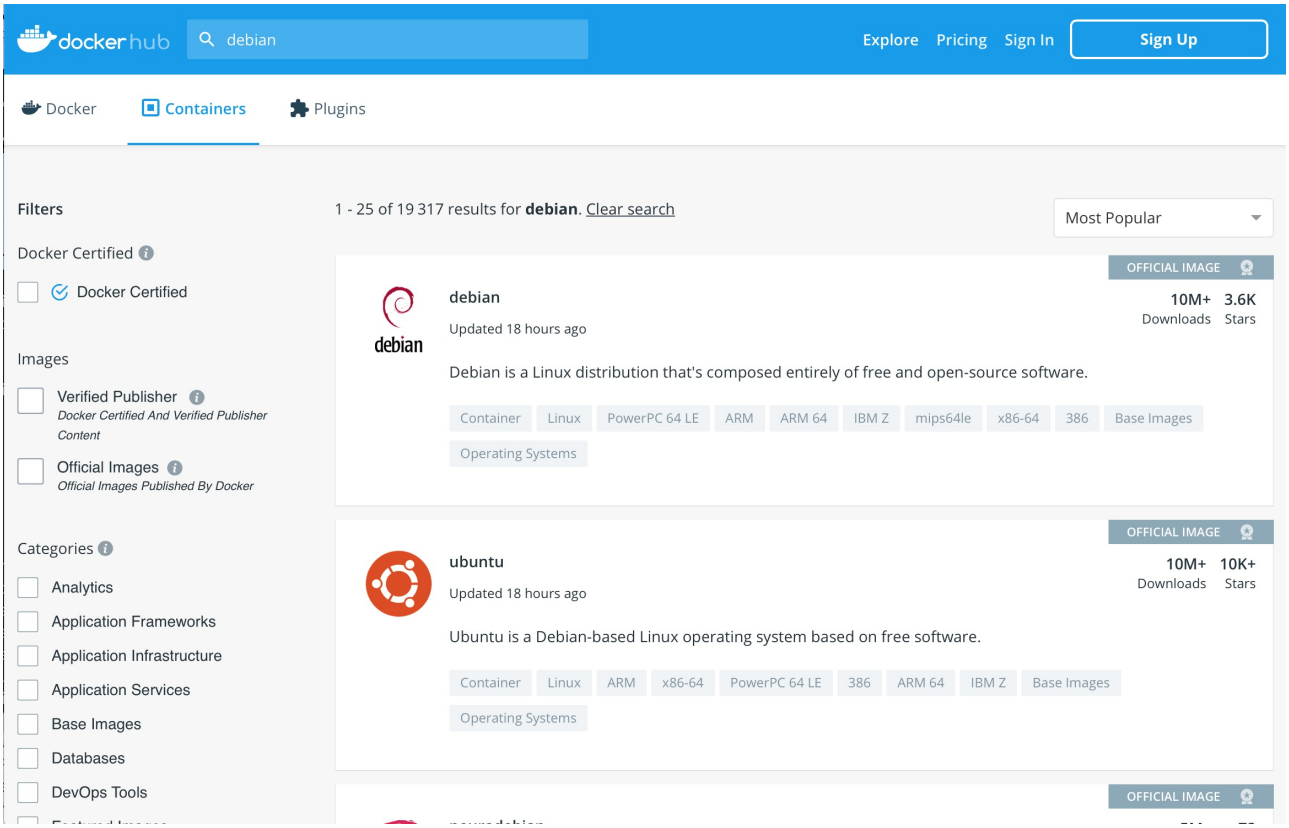
1. la télécharger depuis le Docker Hub ou un registre privé : **docker pull**
2. construire l'image à partir d'un dockerfile : **docker build**
3. lancer un conteneur (l'image est téléchargée si elle n'est pas présente localement) : **docker container run**.

Rechercher une image sur le Docker Hub : `docker search`

```
$ docker search debian
NAME                                                DESCRIPTION
STARS               OFFICIAL            AUTOMATED
ubuntu                                              Ubuntu is a Debian-based
Linux operating sys…   11446                [OK]
debian                                              Debian is a Linux
distribution that's compos…   3627                [OK]
arm32v7/debian                                      Debian is a Linux
distribution that's compos…   66
itscaro/debian-ssh                                  debian:jessie
28                                  [OK]
```

ubuntu et debian sont des images officielles maintenues par les équipes de Docker (les autres ont un name de la formeuser_registry/image_name). La colonne STAR donne une indication sur la popularité de l'image.

Cette commande est équivalente à une recherche depuis l'interface web du [Docker Hub (https://hub.docker.com)](https://hub.docker.com):



:::warning
Même si une image officielle et populaire est un gage de qualité, cela ne dispense pas de rester vigilant sur la sécurité en auditant si possible le dockerfile.
:::

Exercice 1 :

Vous devez effectuer une recherche sur le `Docker Hub` pour localiser l'image `mariadb` maintenue par l'équipe de `linuxserver.io`.

Télécharger une image: `docker image pull`

```
$ docker image pull debian
Using default tag: latest
latest: Pulling from library/debian
3e17c6eae66c: Pull complete
Digest: sha256:26b2647845d66e20eeadf73d1c302a4ffd2cc9a74c39a52f2aced4f823484328
Status: Downloaded newer image for debian:latest
```

:::info
Ici, debian ne designe pas une image mais un dépôt sur le registre qui peut contenir plusieurs variantes d'une image identifiées par un TAG. Si celui-ci n'est pas précisé alors Docker

selectionne le tag `latest` (Using default tag: latest)
:::

Pour obtenir la liste des images sur l'hôte: `docker image ls`

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
debian              latest              1510e8501783        2 weeks ago
114MB
hello-world         latest              bf756fb1ae65        9 months ago
13.3kB
```

Une même image peut avoir plusieurs `TAG`

On précise la tag `buster`.

```
$ docker image pull debian:buster
docker pull debian:buster
buster: Pulling from library/debian
Digest: sha256:8414aa82208bc4c2761dc149df67e25c6b8a9380e5d8c4e7b5c84ca2d04bb244
Status: Downloaded newer image for debian:buster
docker.io/library/debian:buster

$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
debian              buster              1510e8501783        2 weeks ago
114MB
debian              latest              1510e8501783        2 weeks ago
114MB
```

Les images avec les TAG `latest` et `buster` ont le même `IMAGE ID`. Ils référencent donc la même image qui n'est stockée qu'une seule fois.

Dockerfile

Les images Docker sont créées à partir d'un fichier texte de description appelé `dockerfile`.

Sur votre serveur Docker, créez un fichier nommé `dockerfile` à la racine de votre répertoire personnel avec le contenu suivant:

```
FROM docker/whalesay
MAINTAINER prenom.nom@exemple.com
RUN apt-get -y update && apt-get install -y figlet
CMD figlet "Hy DSI Team" | cowsay -n
```

:::info
Instructions de base :

- **FROM** permet de définir l'image de base
- **MAINTAINER** permet de définir l'auteur de l'image
- **RUN** permet de lancer une commande

- **ADD** permet de copier un fichier depuis la machine hôte ou depuis une URL
- **EXPOSE** permet d'exposer un port du conteneur vers l'extérieur
- **CMD** détermine la commande qui sera exécutée lorsque le conteneur 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'environnement qui pourront ensuite être modifiées grâce aux paramètres de la commande `docker container run --env <key>=<value>`
- **VOLUMES** permet de créer un point de montage qui permettra de rendre les données persistantes
  :::

Pour construire l'image : `docker image build`

```
$ docker image build -t <image_name>:<tag> /path/to/dockerfile
```

```
$ docker image build -t my-image:v1 ./

Step 1/4 : FROM docker/whalesay
latest: Pulling from docker/whalesay

Step 2/4 : MAINTAINER prenom.nom@exemple.com
 ---> Running in 80494768cf94
Removing intermediate container 80494768cf94
 ---> 32a7f195feed

Step 3/4 : RUN apt-get -y update && apt-get install -y figlet
 ---> Running in b1cb6281e872
The following NEW packages will be installed:
  figlet
Unpacking figlet (2.2.5-2) ...
Setting up figlet (2.2.5-2) ...
update-alternatives: using /usr/bin/figlet-figlet to provide /usr/bin/figlet
(figlet) in auto mode
Removing intermediate container b1cb6281e872
 ---> 2af78a429bf1

Step 4/4 : CMD figlet "Hy MRIT Team" | cowsay -n
 ---> Running in a6ae37c91a49
Removing intermediate container a6ae37c91a49
 ---> 74590a5a19cb
Successfully built 74590a5a19cb
Successfully tagged my-image:v1
```

Pour des raisons de lisibilité, la sortie a été tronquée pour conserver les éléments essentiels. On distingue les 4 `steps` correspondants à chacune des instructions du dockerfile.

L'opération génère une nouvelle image

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
my-image            v1                  74590a5a19cb        5 minutes ago
274MB
debian              buster              1510e8501783        2 weeks ago
114MB
debian              latest              1510e8501783        2 weeks ago
114MB
```

Pour lancer un conteneur depuis cette image : `docker container run <image>:<tag>`

```
$ docker container run my-image:v1
 ________________________________________________________________
/  _   _          __  __ _____  ___ _____   _____                 \
| | | | |_   _  |  \/  |  _ \|_ _|_   _| |_   _|__  __ _ _ __ ___   |
| | |_| | | | | | |\/| | |_) || |  | |     | |/ _ \/ _` | '_ ` _ \  |
| |  _  | |_| | | |  | |  _ < | |  | |     | |  __/ (_| | | | | | | |
| |_| |_|\__, | |_|  |_|_| \_\___| |_|     |_|\___|\__,_|_| |_| |_| |
\         |___/                                                      /
 ----------------------------------------------------------------
    \
     \
      \
                    ##        .
              ## ## ##       ==
           ## ## ## ##      ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \         __/
          \____\_______/
```

Exercice 2 :

Vous devez construite une image basée sur la dernière version de l'OS `Alpine` (ne pas utiliser le tag `latest`). Cette image doit permettre au conteneur d'afficher la commande `htop` (vous devez utiliser les arguments `--it` que nous verrons un peu plus loin dans votre commande `docker container run`).

:::info
Alpine est très populaire sur Docker pour sa capacité à produire des images très légères (quelques dizaines de Mo)
:::

Exercice 3 :

Déposer l'image sur le Docker Hub ou un registre privé : `docker image push`

Pour pouvoir déposer une image sur le registre Docker Hub, vous devez lui attribuer un TAG spécifique de la forme `<Docker_ID>/<Repository>:<tag>`

- Docker_ID: votre identifiant sur le docker Hub

- Repository: dépôt dans le lequel vous allez placer l'image

Pour cela, vous devez d'abord créer un compte sur le [Docker Hub (https://hub.docker.com/signup)](https://hub.docker.com/signup) ainsi qu'un `repository`.

Ensuite, vous devez ajouter un tag à votre image pour indiquer votre repository.

`docker image tag <image>:<tag> <Docker_ID>/<Repository>:<tag>`

L'exemple ci-dessous reprend le cas de notre première image, vous devez l'adapter à votre besoin.

- my-image: nom de l'image locale
- levasbr1: Docker_ID
- cowsay: nom du repository sur le Docker Hub

```
$ docker image tag my-image:v1 levasbr1/cowsayv1
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
levasbr1/cowsay       v1                  74590a5a19cb        About an hour ago
274MB
my-image            v1                  74590a5a19cb        About an hour ago
274MB
```

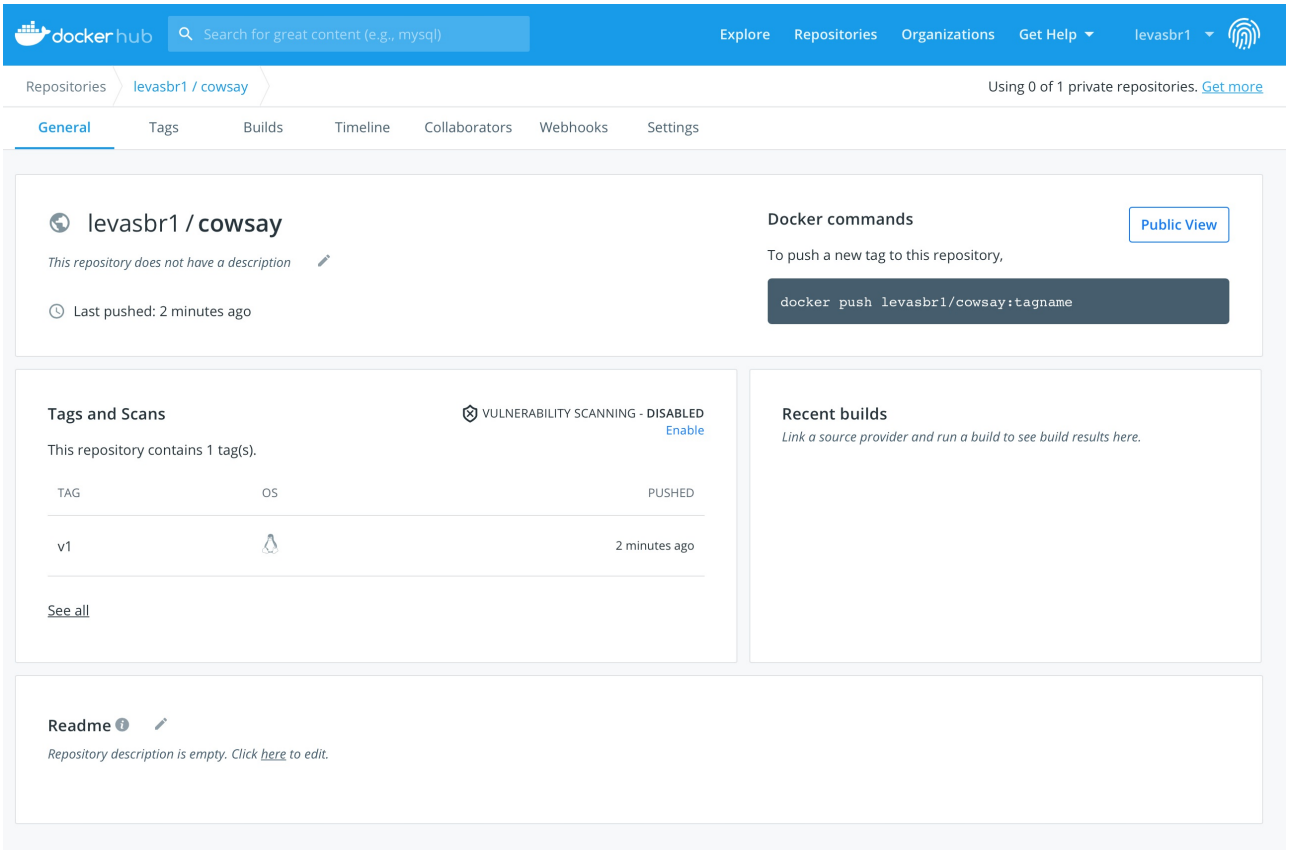
Enfin, il suffit de se connecter au registre pour pousser l'image.

```
$ docker login -u levasbr1
Password: *************

WARNING! Your password will be stored unencrypted in
/home/levasbr1/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded

$ docker image push levasbr1/htop:v1
The push refers to repository [docker.io/levasbr1/htop]
58aa4e5cd23f: Pushed
5f70bf18a086: Mounted from docker/whalesay
d061ee1340ec: Mounted from docker/whalesay
d511ed9e12e1: Mounted from docker/whalesay
091abc5148e4: Mounted from docker/whalesay
b26122d57afa: Mounted from docker/whalesay
37ee47034d9b: Mounted from docker/whalesay
528c8710fd95: Mounted from docker/whalesay
1154ba695078: Mounted from docker/whalesay
v1: digest:
sha256:cd4aeef14f4c7513beed8b474138d5c4113fd0a90c8bdd4401e45db872309670 size:
2614
```

L'interface web doit alors vous afficher:



:::info
Pour pousser une image sur un registre privé, vous devez indiquer l'adresse du registre et si besoin le port d'écoute du service dans le tag :
`docker image tag <image>:<tag> registry.exemple.com:<port>/<user>/<repository>:<tag>`
:::

Exercice 4 :

Partagez l'adresse votre repository avec un collègue et utilisez les commandes vues précédemment pour exécuter un conteneur à partir de son image.

Supprimer une image : `docker image rm <image_ID>`

Pour supprimer une image, on peut utiliser soit le repository soit l'image ID (abrégé).

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
levasbr1/cowsay     v1                  74590a5a19cb        2 hours ago
274MB
my-image            v1                  74590a5a19cb        2 hours ago
274MB


$ docker image rm 745
Error response from daemon: conflict: unable to delete 74590a5a19cb (must be
forced) - image is referenced in multiple repositories

docker image rm levasbr1/cowsay:v1
Error response from daemon: conflict: unable to remove repository reference
"levasbr1/image:v1" (must force) - container 0857115955b6 is using its
referenced image 74590a5a19cb
```

:::warning
Dans la 1ère tentative, Docker refuse de supprimer une image référencée par plusieurs repositories et dans la 2nde, Docker refuse naturellement de supprimer une image dont dépend un conteneur.
:::

Afficher l'historique d'une image : `docker image history <image_name>`

```
$ docker image history levasbr1/cowsay:v1
IMAGE               CREATED             CREATED BY
SIZE                COMMENT
74590a5a19cb        2 hours ago         /bin/sh -c #(nop)  CMD ["/bin/sh" "-c"
"figl…   0B
2af78a429bf1        2 hours ago         /bin/sh -c apt-get -y update && apt-get
inst…   26.8MB
32a7f195feed        2 hours ago         /bin/sh -c #(nop)  MAINTAINER
user@exemple.c…   0B
6b362a9f73eb        5 years ago         /bin/sh -c #(nop) ENV
PATH=/usr/local/bin:/u…   0B
<missing>           5 years ago         /bin/sh -c sh install.sh
30.4kB
<missing>           5 years ago         /bin/sh -c git reset --hard
origin/master       43.3kB
<missing>           5 years ago         /bin/sh -c #(nop) WORKDIR /cowsay
0B
<missing>           5 years ago         /bin/sh -c git clone
https://github.com/moxi…   89.9kB
<missing>           5 years ago         /bin/sh -c apt-get -y update && apt-get
inst…   58.6MB
<missing>           5 years ago         /bin/sh -c #(nop) CMD ["/bin/bash"]
0B
<missing>           5 years ago         /bin/sh -c sed -i 's/^#\s*\
(deb.*universe\)$…   1.9kB
<missing>           5 years ago         /bin/sh -c echo '#!/bin/sh' >
/usr/sbin/poli…   195kB
<missing>           5 years ago         /bin/sh -c #(nop) ADD
file:f4d7b4b3402b5c53f…   188MB
```

Gestion des conteneurs

Lancer un conteneur à partir d'une image : `docker container run`

```
$ docker container run debian:buster cat /etc/issue
Debian GNU/Linux 10 \n \l
```

Dans cet exemple, le conteneur:

1. démarre
2. exécute la commande passée en argument (cat /etc/issue)
3. affiche la sortie stdout de la commande
4. s'arrête

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

La commande `docker ps` ne retourne rien puisque le conteneur est arrété :

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

Pour obtenir la liste complète des conteneurs (quelque soit leur état) il faut utiliser l'option `-a` :

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
07f372e1689a        debian:buster       "cat /etc/issue"         3 minutes ago
Exited (0) 3 minutes ago                        admiring_grothendieck
0dd7be4c6231        74590a5a19cb        "/bin/sh -c 'figlet …"   2 hours ago
Exited (0) 2 hours ago                          stupefied_neumann
```

Le conteneur possède un identifiant unique, Container ID (CID), et un nom généré aléatoirement (admiring_grothendieck).

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

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

```
$ docker container run --name cmd_cat debian:latest cat /etc/issue
Debian GNU/Linux 10 \n \
```

Cette commande a créé un nouveau conteneur :

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
517729d68220        debian:buster       "cat /etc/issue"         12 seconds ago
Exited (0) 11 seconds ago                       cat_cmd
07f372e1689a        debian:buster       "cat /etc/issue"         3 minutes ago
Exited (0) 3 minutes ago                        admiring_grothendieck
```

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 container run -it  debian:buster /bin/bash
root@eae2cce2669d:/#
```

:::info
Le prompt reprend le "conteneur ID" (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 arrière plan.

```
$ docker container run -d --name test_daemon nginx
4d81f9903afe1b777de6389954c762122b5aeea847f5b4f8953ad308bbc5203d

$ docker ps
CONTAINER ID        IMAGE                                     COMMAND
CREATED
4d81f9903afe        nginx                                     "nginx -g 'daemon
..."   51 seconds ago

$ docker stop test_daemon
```

:::info

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 fichiers 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>
```

Faire un diff entre les fichiers d'un conteneur et les fichiers de son image

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

Obtenir des stats CPU / RAM / Disk / Net des conteneurs

```
$ docker stats
```

Commiter un conteneur pour en faire une nouvelle image

```
$ docker commit <nom_conteneur> ou <CID> <user>/<image>:<tag>
```
:::

Exécuter une commande dans un conteneur démarré : `docker exec`

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

```
$ docker container 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  root  run  sbin
srv  sys  tmp  usr  var
root@331e1e904e1e:/# exit
exit
```

S'attacher à un conteneur pour interagir avec le processus racine : `docker attach`

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

:::info
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 abordée dans le § sur les réseaux Docker.
:::

```
$ docker container run -d -p 8000:80 --name test_attach nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
bb79b6b2107f: Pull complete
111447d5894d: Pull complete
a95689b8e6cb: Pull complete
1a0022e444c2: Pull complete
32b7488a3833: Pull complete
Digest: sha256:ed7f815851b5299f616220a63edac69a4cc200e7f536a56e421988da82e44ed8
Status: Downloaded newer image for nginx:latest
69d32bbaa69ce33566a26eae457604023b7d96f25f3b6dcacde3c81ea0695397


$ docker attach test_attach --sig-proxy=false
172.16.162.1 - - [28/Oct/2020:17:02:27 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
172.16.162.1 - - [28/Oct/2020:17:02:28 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
172.16.162.1 - - [28/Oct/2020:17:02:28 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
172.16.162.1 - - [28/Oct/2020:17:02:28 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
172.16.162.1 - - [28/Oct/2020:17:02:28 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
172.16.162.1 - - [28/Oct/2020:17:02:28 +0000] "GET / HTTP/1.1" 304 0 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:82.0) Gecko/20100101
Firefox/82.0" "-"
```

:::info
L'option `--sig-proxy=false` permet de se détacher du conteneur avec la séquence `CTRL-c` sans tuer le processsus racine qui provequerait l'arrêt du conteneur.
:::

Supprimer un conteneur arrêté : `docker rm`

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS                  NAMES
69d32bbaa69c        nginx               "/docker-entrypoint.…"   8 minutes ago
Up 8 minutes                0.0.0.0:8000->80/tcp   test_attach
d74cdcc2bdbb        debian:buster       "/bin/bash"              18 minutes ago
Exited (0) 16 minutes ago                          sweet_bardeen
07f372e1689a        debian:buster       "cat /etc/issue"         30 minutes ago
Exited (0) 30 minutes ago                          admiring_grothendieck
test_diff
```

Supprimer un conteneur par son nom ou CID abrégé :

```
$ docker stop test_attach
test_attach
$ docker rm test_attach
$ docker rm d74 07f  //permet de supprimer les conteneurs dont le CID commence
par d74 et 07f
```

Copier des fichiers depuis ou à destination d'un conteneur: `docker cp`

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

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

On copie le fichier index.html depuis le conteneur sur la machine hôte.
::: warning
Attention au "." en fin de ligne qui correspond au répertoire courant
:::

```
$ docker cp test_cp:/usr/share/nginx/html/index.html .
$ ls - l
-rw-r--r-- 1 levasbr1 levasbr1 138 Oct 28 15:06 dockerfile
-rw-r--r-- 1 levasbr1 levasbr1 612 Sep 29 16:12 index.html
```

On remplace le contenu du fichier :

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

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

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

On teste que le fichier a bien été modifié depuis un navigateur avec l'URL `http://docker-xx.univ-rouen.fr:8001`

les volumes Docker

Initialement prévu pour des services sans état (stateless), Docker permet néanmoins de rendre les données des conteneurs persistantes en utilisant la notion de [volume (https://docs.docker.com/storage/volumes/)](https://docs.docker.com/storage/volumes/).

Les volumes nommés :

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

```
$ docker container run -d -v myvol:/var/log --name test_volume debian:latest
/bin/sleep infinity
```

Ici l'option `-v myvol:/var/log` permet d'initialiser un volume (myvol) indépendant du conteneur, qui contiendra les logs du conteneur.

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` :

```
$ docker inspect test_volume | grep -A 10 "Mounts"
"Mounts": [
    {
        "Type": "volume",
        "Name": "myvol",
        "Source": "/var/lib/docker/volumes/myvol/_data",
        "Destination": "/var/log",
        "Driver": "local",
        "Mode": "z",
        "RW": true,
        "Propagation": ""
    }
```

On peut arrêter et supprimer le conteneur pour contrôler que les données sont toujours présentes sur l'hôte.

```
$ docker stop test_volume && docker rm test_volume
$ sudo ls /var/lib/docker/volumes/myvol/_data
apt  btmp  faillog  lastlog  wtmp
```

:::info
On peut créer un volume directement avec la commande `docker volume create --name <volume_name>` et lister tous les volumes créés avec la commande `docker volume ls`.
:::

Exercice 6 :

Créer un conteneur à partir de l'image officielle `mariadb` avec un volume nommé `mydb` permettant de rendre la database persistante. Vous devez utiliser la commande `docker volume create`.

:::warning
Vous devez spécifier une variable d'environnement (option `-e MYSQL_ROOT_PASSWORD=xxxxxx`) pour permettre l'initialisation du serveur mariadb.
:::

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

L'autre utilisation des volumes consiste à partager un dossier entre le conteneur et le système hôte.

```
$ mkdir projet-web
$ docker container run -d --name test_volume2  -p 8004:80 -v ~/projet-
web:/var/www/html php:rc-apache
```

Ici nous disposons d'un volume monté sur l'arborescence `/var/www/html` du conteneur et correspondant au dossier `/home/<user>/projet-web` 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://docker-xx.univ-rouen.fr:8004/phpinfo.php` que le fichier est bien accédé par le conteneur.

```
$ docker container inspect test_volume2
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/levasbr1/projet-web",
        "Destination": "/var/www/html",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
```

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.

:::danger
Dans cette méthode, le conteneur dispose d'un accès direct au file-system de l'hôte. Elle est à proscrire en production pour des raisons de sécurité.
:::

Les réseaux 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`).

```
$ 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":
"8dd02d7ad16f9e47a774f0ee5a652a606ecc23bcc15c126d3e6fbf6fd1c3465c",
       "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 obtiennent une adresse IP fournie par le daemon Docker (réseau privé : 172.17.0.0/16, gateway : 172.17.0.1).

Sous linux, Docker utilise netfilter (iptables) pour implémenter :

- du NAT, pour permettre aux conteneurs de sortir sur le réseau en utilisant l'adresse IP de l'hôte
- du port forwarding, pour accéder aux conteneurs depuis l'adresse IP de l'hôte

:::info
Il est possible d'utiliser un réseau Overlay (vxlan) pour permettre aux conteneurs de communiquer entre eux même si ils sont exécutés sur des hôtes différents.
:::

```
$ docker container 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.

Réseaux définis par l’utilisateur

On utilise aujourd'hui les `user-defined network`, qui apportent une meilleure isolation entre les conteneurs et un mécanisme interne de résolution DNS.

On peut créer autant de réseaux qu'il est nécéssaire et connecter / déconnecter les conteneurs à la volée.

Exercice 7

Dans cet exercice, on va utiliser le mecanisme interne de résolution DNS pour permettre la connexion entre une base mysql et un frontend PHPMyAdmin.

On commence par déclarer un nouveau réseau en utilisant le driver de type bridge (on peut utiliser également un driver MACVLAN ou OVERLAY).

```
$ docker network create -d bridge mynet
```

On contrôle le nouveau réseau `mnynet` :

```
$ docker network inspect mynet
[
    {
        "Name": "mynet",
        "Id":
"f5d8b3b0ef90654c06662afecbcee70b9dad4fb8dec54e4d3d2fccd8013ddc4e",
        "Created": "2018-11-11T17:45:17.124793619+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
```

On affiche la liste des réseaux :

```
docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
87adfb041868        bridge              bridge              local
f946a0bd82f7        host                host                local
b17e5a6fc39a        mynet               bridge              local
4ade9b530900        none                null                local
```

On démarre le conteneur avec une database MySQL dans ce réseau :

```
$ docker container run --name db --network mynet -e MYSQL_ROOT_PASSWORD=bonjour
-d mysql:5.7
```

:::info
Docker utilise le nom du conteneur pour créer une entrée DNS pointant sur l'adresse IP attribuée au conteneur. Le nom db est utilisé par défaut dans la configuration de l'image PHPMyAdmin comme adresse pour joindre la base de donnée.
:::

On démarre ensuite un conteneur avec l'interface PHPmyadmin:

```
$ docker container run --name phpmyadmin  --network mynet -d -p 8080:80
phpmyadmin/phpmyadmin
```

On teste la connexion entre les 2 conteneurs : http://<ip-address-server>:8080.
On peut également tester la résolution DNS db sur le réseau mynet.

```
 $ docker run --network=mynet --name=test-dig sequenceiq/alpine-dig dig db
 ;; ANSWER SECTION:
db.         600 IN  A   172.18.0.2
```

L'alias db est résolu en `172.18.0.2`. Pour vérifier qu'il s'agit bien de l'adresse IP du conteneur mysql :

```
$ docker inspect db | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.18.0.2",
```

:::warning
L'option `--lynk` qui permet de peupler dynamiquement le fichier `hosts` pour relier des conteneurs est considéré comme obsolète (deprecated)
:::

Docker-Compose

Docker-compose est un outil pour simplifier le déploiement d'applications complexes (avec de nombreux conteneurs) en utilisant un simple fichier texte de description au format yaml (https://docs.ansible.com/ansible/2.9/reference_appendices/YAMLSyntax.html). Il reprend l'ensemble des options qui seraient normalement à fournir à un `docker container run`.

Pour installer cet outil, vous pouvez vous référer à la documentation (https://docs.docker.com/compose/install/)

Dans l'exemple retenu, nous allons deployer la solution d'édition collaborative CodiMD (utilisée pour ce support).

Vous allez créer un fichier `docker-compose.yml` dans votre répertoire personnel :

```
version: '3'
services:
  database:
    image: postgres:9.6-alpine
    container_name: codi_db
    environment:
      - POSTGRES_USER=hackmd
      - POSTGRES_PASSWORD=hackmdpass
      - POSTGRES_DB=hackmd
    volumes:
      - database:/var/lib/postgresql/data
    networks:
      backend:
    restart: always

  app:
    image: hackmdio/hackmd:1.2.1
    container_name: codi_app
    environment:
      - CMD_DB_URL=postgres://hackmd:hackmdpass@database:5432/hackmd
    ports:
      - "3000:3000"
    networks:
      backend:
    restart: always
    depends_on:
      - database

networks:
  backend:
volumes:
  database:
```

:::warning
Attention, le yaml est très sensible à l'indentation...
:::

- **restart** : permet de définir l'état du conteneur au démarrage du serveur
- **container_name** : équivalent à l'option `--name`
- **networks** : équivalent à `docker network create`
- **volumes** : équivalent à `docker volume create`
- **depends_on** : permet de définir l'ordre de démarrage (ici le service database démarre avant le service app)

Dans l'exemple proposé, docker-compose va lancer 2 conteneurs automatiquement :

- **app**, qui contient le serveur d'application
- **database**, qui contient un serveur PostgreSQL

Pour lancer les conteneurs, on utilise la commande : `docker-compose up -d`.

```
$ docker-compose up -d
Creating network "formation_backend" with the default driver
Creating volume "formation_database" with default driver
Pulling database (postgres:9.6-alpine)...
9.6-alpine: Pulling from library/postgres
4fe2ade4980c: Pull complete
08cf8c12f47e: Pull complete
a3e76355a10e: Pull complete
a8839f4153ad: Pull complete
Pulling app (hackmdio/hackmd:1.2.1)...
1.2.1: Pulling from hackmdio/hackmd
f189db1b88b3: Pull complete
3d06cf2f1b5e: Pull complete
687ebdda822c: Pull complete
3dd088865e63: Pull complete
108825449826: Pull complete
Creating formation_database_1_798dfc64f458 ... done
Creating formation_app_1_5fe390eb2e7e      ... done
```

:::info
Comme pour la commande `docker container run`, l'option `-d` permet de faire tourner les conteneurs en arrière plan.
:::

- docker-compose supporte les options suivantes:
    - up: pour créer les services (conteneneurs, volumes, réseaux...)
    - down: pour détruire les services
    - pause / unpause / stop / start et restart

On teste le bon fonctionnement de l'application avec l'URL `http://docker-xx.univ-rouen.fr:3000`.

Exercice 8 :

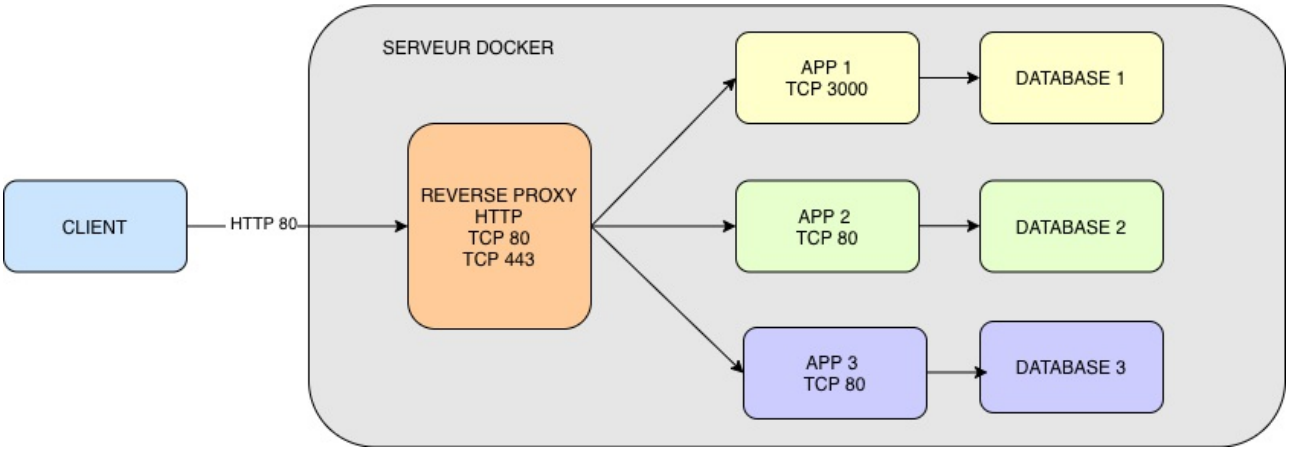
A partir de l'exemple fourni sur le Docker Hub, vous devez composer votre propre docker-compose pour lancer le CMS Wordpress avec les éléments suivants :

- des volumes nommés: wp_db_vol et wp_app_vol
- un réseau nommé: wp_lan
- des conteneurs nommés: wp_db et wp_app
- accessible depuis le port 8888

Reserse proxy

Nous avons vu qu'il était possible de joindre un conteneur depuis une adresse de port forwarding. En production, il n'est cependant pas souhaitable d'avoir à définir un numéro de port non standard pour joindre un service.

La solution consiste à mettre en place un reverse proxy HTTP:



Exemple avec la solution de [Jwilder (https://github.com/nginx-proxy/nginx-proxy)]

Le reverse proxy basé sur NGINX proposé par Jwilder est une solution bien connue et facile à mettre en place pour obtenir un reverse proxy automatique.

La configuration est assurée en interrogeant directement le Daemon Docker (le conteneur accède au socket via un bind mount) et en utilisant des variables d'environnements.

On commence par créer un réseau dédié au reverse proxy.

```
$ docker network create proxy
```

Puis on lance le reverse proxy (attention ce conteneur doit être le seul à utiliser le port 80 de la machine hôte).

```
$ docker container run -d -p 80:80 -v /var/run/docker.sock:/tmp/docker.sock:ro
--name proxy --network proxy jwilder/nginx-proxy
$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED
STATUS              PORTS                  NAMES
3bb26a10d084        jwilder/nginx-proxy    "/app/docker-entrypo…"   5 seconds
ago       Up 4 seconds        0.0.0.0:80->80/tcp    proxy
```

Et enfin, on utilise un conteneur de test HTTP (https://hub.docker.com/r/traefik/whoami) avec une variable d'environnement pour indiquer au reverse proxy comment gérer les requêtes HTTP.

```
$ docker container run --name test_proxy --network proxy -d -e
VIRTUAL_HOST=docker-xx.univ-rouen.fr traefik/whoami
```

Dans cet exemple, toutes les requêtes Get HTTP avec un host défini à docker-xx.univ-rouen.fr seront renvoyées au conteneur `test_proxy`.

Pour vérifier le bon fonctionnement, vous pouvez utiliser un navigateur à l'adresse : http://docker-xx.univ-rouen.fr ou simplement avec la commande `curl` dans un terminal

```
$ curl docker-00.univ-rouen.fr
Hostname: a7b7b8e873e9
IP: 127.0.0.1
IP: 172.21.0.3                    <- @IP du conteneur
RemoteAddr: 172.21.0.2:58124      <- @IP du reverse proxy
GET / HTTP/1.1
Host: docker-00.univ-rouen.fr     <- champ Host du Get HHTP
User-Agent: curl/7.64.1
Accept: */*
Connection: close
X-Forwarded-For: 10.0.192.27      <- @IP du client (curl)
X-Forwarded-Port: 80
X-Forwarded-Proto: http
X-Forwarded-Ssl: off
X-Real-Ip: 10.0.192.27
```

Exercice 9 :
Faire les opérations de l'exemple puis modifier votre docker-compose dédié à Wordpress pour intégrer le reverse-proxy. Il devra comporter un réseau dédié à la connexion entre le reverse proxy et le serveur Web et un second dédié à la connexion entre le serveur Web et la base de données.

Exercice 10 :

Mettre en place une solution de reverse proxy similaire reposant sur le produit Traefik Proxy (https://doc.traefik.io/traefik/). Créer un docker-compose dédié pour Traefik et modifier votre docker-compose wordpress en conséquence.

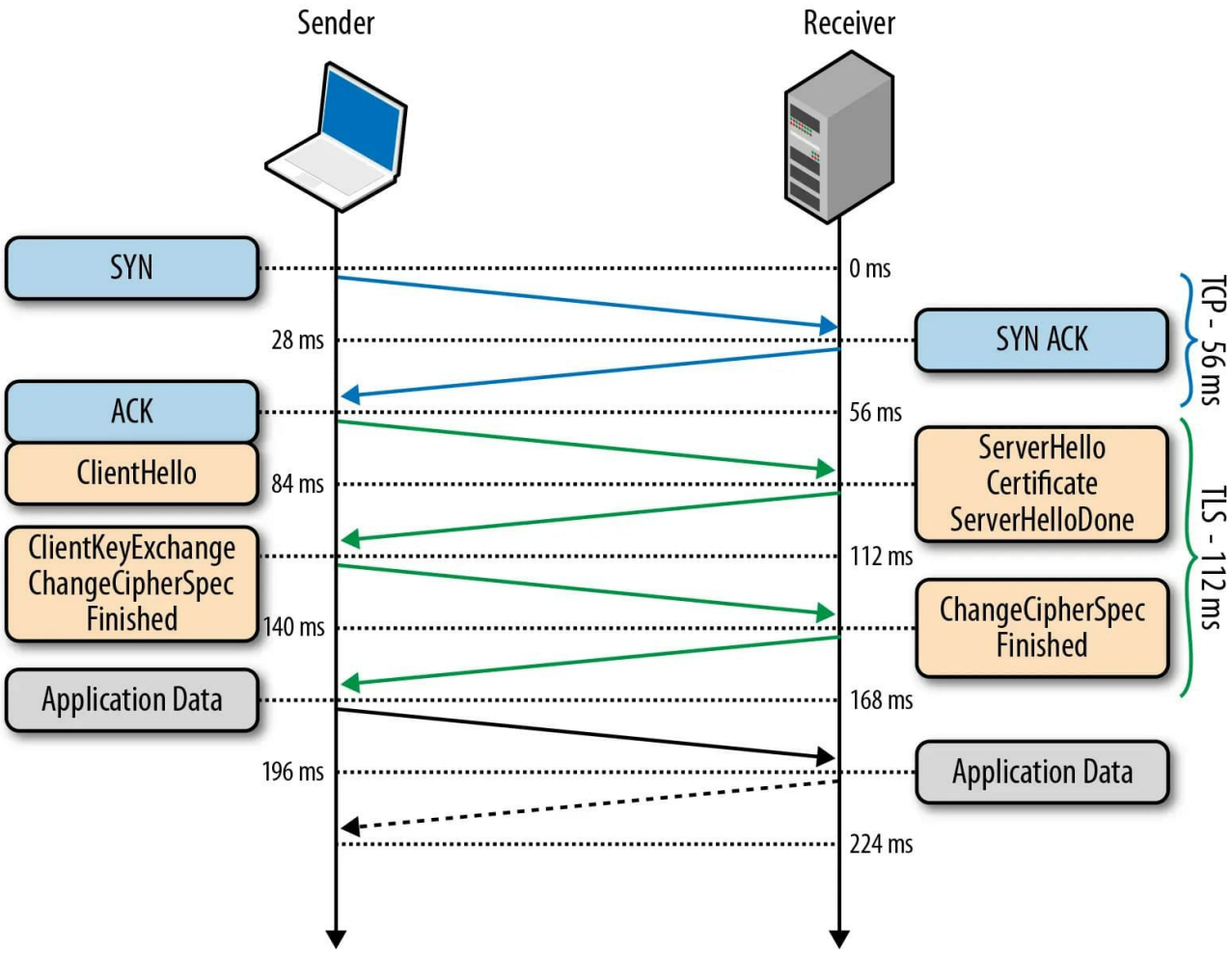
Ajouter une terminaison TLS

L'objectif de cette partie est de mettre en place un accès HTTPS à une applicaition conteneurisée. Comme nous utilisons un reverse-proxy, la terminaison TLS sera portée par le conteneur Traefik qui exposera le port 443 de la machine hôte.

```
NAVIGATEUR <-------------> TRAEFIK <-------------> APPLICATION
               HTTPS      TCP 443      HTTP         TCP 80
```

Rappels TLS

TLS (Transport Layer Security) est une évolution du protocole SSL qui permet entre autre de sécuriser les échanges avec un site en HTTPS. Il repose sur le principe de la poignée de main (handshake) et permet à la fois de chiffrer les échanges mais aussi d'identifier les partenaires.



1. **TLS** démarre après l'établissement d'une connexion TCP (syn/syn ack/ack)
2. **ClientHello:** le client initie le handshake en envoyant un message « Hello » au serveur. Le message inclut la version TLS prise en charge par le client, les suites de chiffrement (cypher suite) prises en charge et une chaîne d'octets aléatoires (client random).
3. **ServerHelloDone:** le serveur envoie un message contenant le certificat SSL du serveur, la suite de chiffrement choisie par le serveur et le « Server random », une autre chaîne aléatoire d'octets qui est générée par le serveur.
4. **ClientKeyExchange** et **ChangeCipherSpec** le client vérifie le certificat SSL du serveur auprès de l'autorité de certification qui l'a émis. Si il est correcte, il génère une clé symétrique qu'il chiffre avec la clé publique du certificat du serveur
5. **encrypted pre-master key:** le client envoie une autre chaîne d'octets aléatoire chiffrée avec la clé publique du serveur
6. **Clé privée utilisée:** le serveur déchiffre le secret pré-maître avec sa clé privée
7. **Clés de session créées:** le client et le serveur génèrent des clés de session à partir du client-random, du server-random et du secret pré-maître. Ils devront arriver aux mêmes résultats.
8. **Le client est prêt:** le client envoie un message « Client Finished » qui est chiffré avec une clé de session.

9. **le serveur est prêt:** le serveur envoie un message « Server Finished » chiffré avec une clé de session.

10. Chiffrement symétrique sécurisé atteint : Le handshake est terminé et la communication se poursuit à l'aide des clés de session.

Pour voir en vidéo:

{%youtube _UpuZ0Y3k-c %}

Certificats TLS

Pour mettre en place une liaison HTTPS avec une application conteneurisée, il nous faut un certificat serveur (et une clé privée) validé par une autorité de certification reconnue par les navigateurs. Dans le monde de l'entreprise, on peut recourir à un prestataire privé (ex: DigiCert, Sertigo, GlobalSign..), se tourner vers des solutions gratuites comme Let's Encrypt (https://letsencrypt.org) ou bien mettre en place sa propre PKI (Public Key Infrastructure).

Pour les besoins du TP, nous allons tout simplement utiliser OpenSSL (https://www.openssl.org) pour générer un certificat racine à intégrer dans le navigateur et un certificat serveur ainsi que sa clé privée.

Sur le serveur Docker

- on s'assure qu'openSSL est bien installé

```
$ openssl version
OpenSSL 1.1.1d  10 Sep 2019

# dans le cas contraire:
$ sudo apt install openssl
```

- on crée un répertoire dédié

```
$ mkdir rootCA && cd rootCA
```

- on crée une clé privée (rootCA.key) protégée avec un mot de passe pour le certificat racine

```
$ openssl genrsa -des3 -out rootCA.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................++++
.....................................................++++
e is 65537 (0x010001)
Enter pass phrase for rootCA.key:
Verifying - Enter pass phrase for rootCA.key:

$ ls
rootCA.key
```

- on crée un certificat racine (rootCA.crt) auto-signé avec la clé obtenue précédemment (rootCA.key) avec une validité d'un an

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 365 -out
rootCA.crt
Enter pass phrase for rootCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Normandie
Locality Name (eg, city) []:Mont-Saint-Aignan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Universite de Rouen
Organizational Unit Name (eg, section) []:DSI
Common Name (e.g. server FQDN or YOUR name) []: univ-rouen.fr
Email Address []:prenom.nom@univ-rouen.fr

$ ls
rootCA.crt  rootCA.key
```

- on génére une clé privée pour chaque application (ici pour docker-xx.univ-rouen.fr)

```
$ openssl genrsa -out docker-xx.univ-rouen.fr.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................+++++
.....................+++++
e is 65537 (0x010001)

$ ls
rootCA.crt  rootCA.key  docker-xx.univ-rouen.fr.key
```

- on génére une CSR (certificate signing request) qui permet de spécifier les détails du certificat que vous souhaitez générer. Cette demande sera traitée par l'autorité racine pour générer le certificat.

```
$ openssl req -new -key docker-xx.univ-rouen.fr.key -out docker-xx.univ-
rouen.fr.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Normandie
Locality Name (eg, city) []:Mont-Saint-Aignan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Universite de Rouen
Organizational Unit Name (eg, section) []:DSI
Common Name (e.g. server FQDN or YOUR name) []:docker-xx.univ-rouen.fr
Email Address []:prenom.nom@univ-rouen.fr

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:secret-password
An optional company name []:

$ ls
rootCA.crt  rootCA.key  docker-xx.univ-rouen.fr.csr  docker-xx.univ-
rouen.fr.key
```

- enfin, il suffit de générer le certificat serveur signé par l'autorité racine à partir de la CSR

```
openssl x509 -req -in docker-xx.univ-rouen.fr.csr -CA rootCA.crt -CAkey
rootCA.key -CAcreateserial -out docker-xx.univ-rouen.fr.crt -days 365 -sha256
Signature ok
subject=C = FR, ST = Normandie, L = Mont-Saint-Aignan, O = Universite de Rouen,
OU = DSI, CN = docker-xx.univ-rouen.fr = prenom.nom@univ-rouen.fr
Getting CA Private Key
Enter pass phrase for rootCA.key:

$ ls
rootCA.crt  rootCA.key  rootCA.srl  docker-xx.univ-rouen.fr.crt  docker-
xx.univ-rouen.fr.csr  docker-xx.univ-rouen.fr.key
```

Pour récapituler, nous avons:

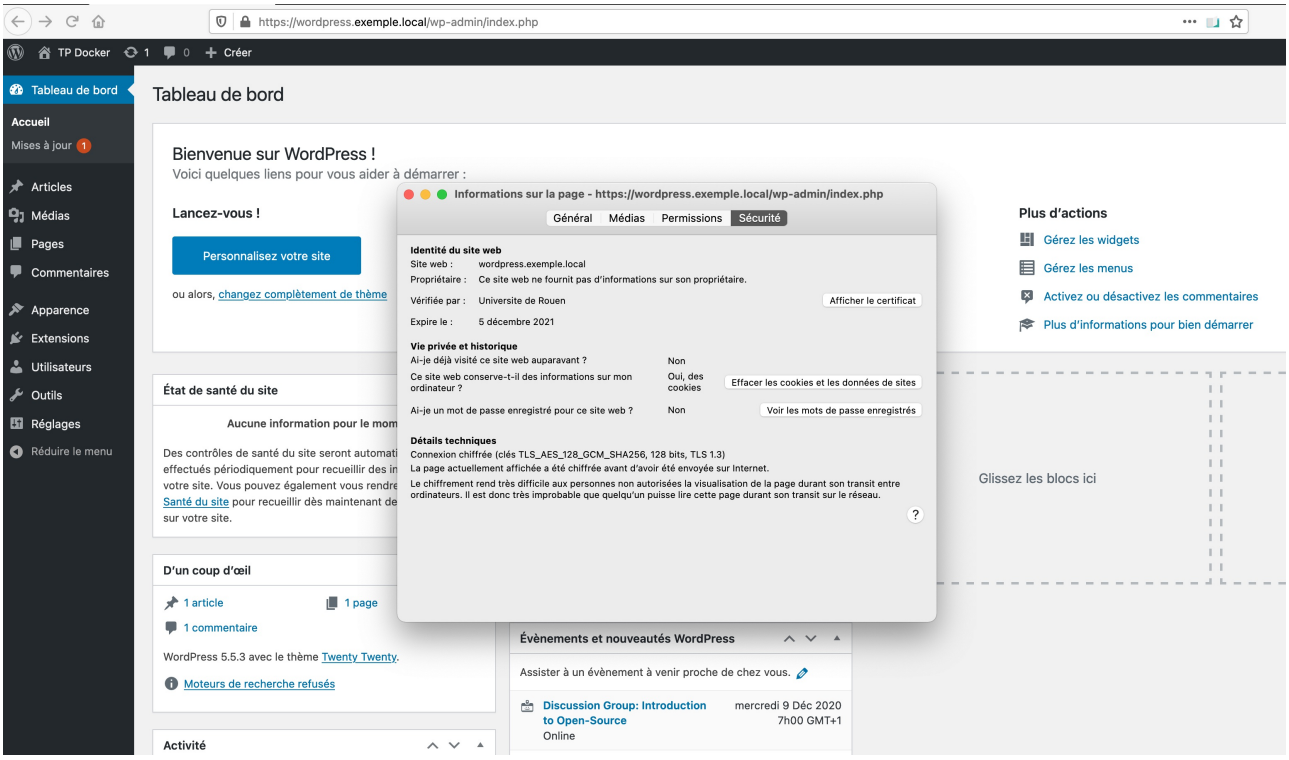
- **rootCA.crt** le certificat racine qu'il faut importer dans Firefox
- **docker-xx.univ-rouen.fr.crt** le certificat server
- **docker-xx.univ-rouen.fr.key** la clé privée du certificat

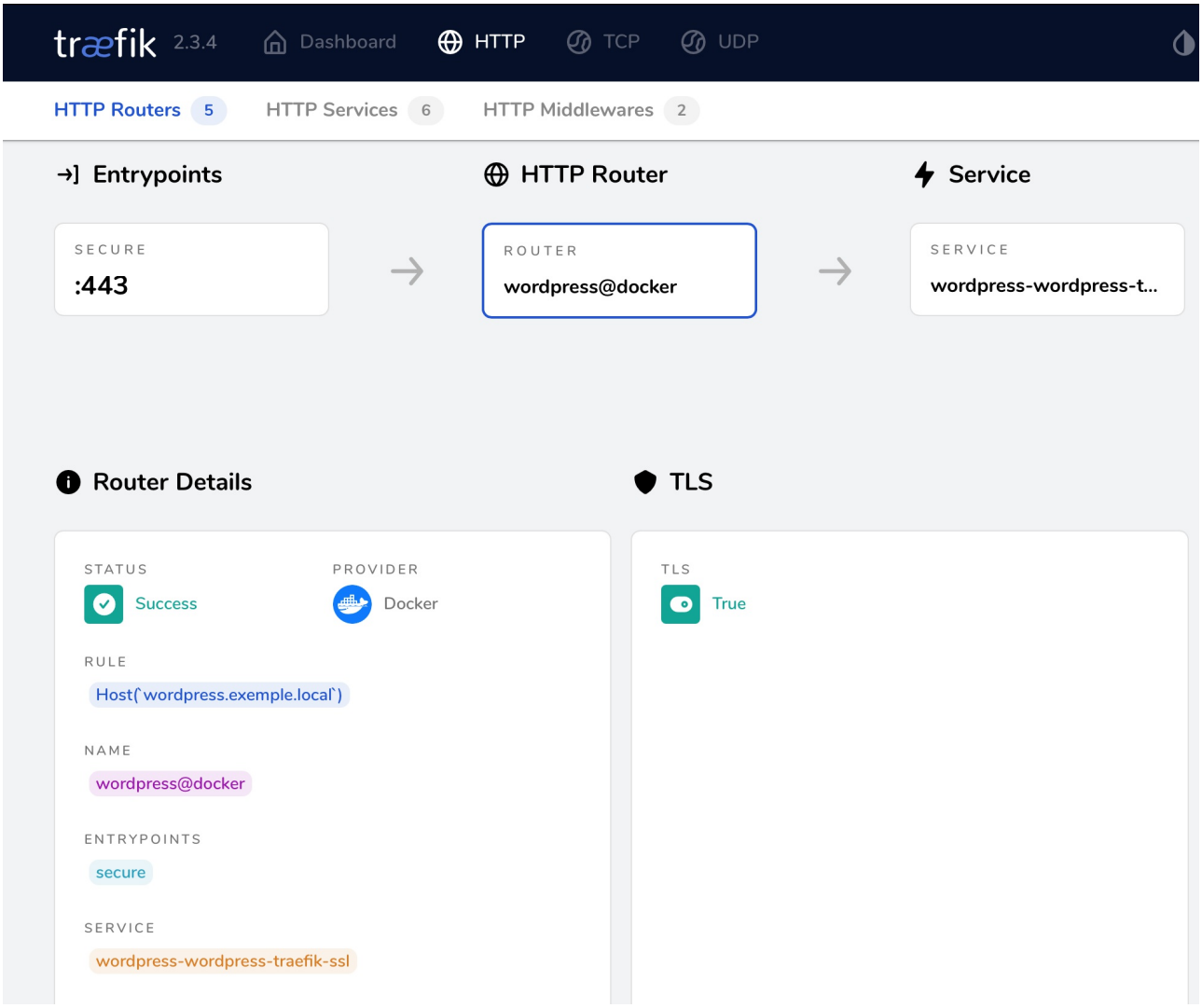
Exercice 12

A l'aide de l'exemple ci-dessus, vous devez créer les certificats et intégrer le certificat racine dans Firefox.

Ensuite, vous devez modifier votre configuration Traefik et Wordpress pour mettre en place un accès HTTPS (si besoin, vous pouvez vous référer à [cet article (https://traefik.io/blog/traefik-2-tls-101-23b4fbee81f1/)](https://traefik.io/blog/traefik-2-tls-101-23b4fbee81f1/))

Exemples de resultats:





La boite à outils Docker

- Portainer (https://www.portainer.io) : une interface Web pour gérer des serveurs Docker
- ctop (https://github.com/bcicen/ctop) : un équivalent à la commande `top` dédié aux conteneurs
- cAdvisor (https://github.com/google/cadvisor) : une solution de monitoring qui peut être couplé avec Prometheus et graphana
- Weave Scope (https://github.com/weaveworks/scope) : une alternative à Portainer orientée monitoring

Quelques astuces

- stopper tous les conteneurs : `docker stop $(docker ps -q -a)`
- supprimer tous les conteneurs : `docker rm $(docker ps -q -a`
- supprimer toutes les images : `docker rmi -f $(docker images -q`
- purger les volumes orphelins : `docker volume rm $(docker volume ls -qf dangling=true`
- purger Docker : `docker system prune -a` et `docker volume prune`
- vérifier l'espace disque utilisé : `docker system df`
- disposer d'un bac à sable : Docker en ligne (https://labs.play-with-docker.com/)
- quelques one-liners

(http://www.commandlinefu.com/commands/matching/docker/ZG9ja2Vy)
- formation en ligne (https://docs.docker.com/get-started/)
- quelques exemples (https://gitlab.univ-rouen.fr/bruno.levasseur/docker-compose) de docker-compose pour faire du self-hosting

Lexique

- **conteneur** : instance active d'une image
- **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
- **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/
- **orchestrateur** : gère un pool de ressources serveurs (Swarm, Kubernetes, Mesos, Rancher...)
- **registry / registre** : dépôt d'images Docker