Conteneurisation d’une application web avec Docker

Maxime Lambert

Février 2024

Troisième TP de virtualisation pour le module R4.08, BUT Informatique deuxième année.

Travail individuel en salle de machines.

Savoirs de référence étudiés :

- Virtualisation d’applications
- Conteneurs

Pré-requis :

- Aisance sous un système Unix en ligne de commande
- CM du module R4.08 sur les conteneurs Linux
- Notions sur les réseaux TCP/IP
- Architecture client-serveur

Matériel ou logiciels utilisés :

- Accès au serveur Proxmox de l’Université depuis un navigateur web

1 Présentation du TP

L'objectif de ce TP est de vous initier à l'utilisation de Docker en exploitant concrètement des conteneurs.

1.1 Déroulement

Tout d'abord, vous allez installer Docker sur une nouvelle machine virtuelle.

Ensuite, vous vous familiariserez avec la *CLI*[1] de Docker en instanciant vos premiers conteneurs et en apprenant à écrire des images.

Finalement, vous mettrez en pratique les connaissances acquises au cours du TP en déployant un microservice[2] exposant une API REST[3].

1.2 Rappels du cours

Afin d'aborder sereinement ce TP, nous allons revoir la notion d'image et de conteneur.

1.2.1 Image

Une image Docker, est un modèle immuable contenant des instructions :

- Une collection ordonnée de modifications d'un système de fichiers

- Des paramètres d'exécution destinés à être utilisés lors de l'instanciation d'un conteneur

- Des méta-données (optionnelles) utiles à des fins de documentation

1.2.2 Conteneur

Un conteneur Docker est une instance exécutable d'une image Docker. Toutes les modifications apportées à un conteneur ne seront jamais répercutées sur son image de base car elles s'effectuent dans la « couche conteneur ».

Il possède son propre environnement d'exécution, distinct et isolé de celui des autres conteneurs et de son hôte.

Au démarrage d'un conteneur, des instructions préalablement définies dans l'image permettent d'exécuter une ou des commandes ;qui peuvent si l'utilisateur le souhaite, être surchargées au moment de l'instanciation du conteneur.

En fonction de votre besoin, il est possible d'isoler ou non certains aspects d'un conteneur (système de fichiers, réseau…).

[1] *CLI : Command-Line Interface*

[2] Microservice : Logiciel ayant la responsabilité d'une tâche unique et limité au sein d'un système plus étendu.

[3] API REST : Interface de programmation permettant à des logiciels de communiquer entre eux, selon un ensemble de règles s'appuyant sur les verbes du protocole HTTP.

1.2.3 Client et machine hôte

Comme indiqué en CM, Docker utilise une architecture client-serveur.

Le serveur, également qualifié de machine hôte, héberge le démon Docker. C'est ce démon, qui va télécharger les images, et gérer les conteneurs tout au long de leur cycle de vie.

Dans le cadre de ce TP, **nous utiliserons un client en ligne de commande qui est installé sur la même machine que le démon**. Gardez à l'esprit que ce n'est pas une obligation. Le client et l'hôte peuvent être deux machines différentes.

1.3 Des problèmes de proxy ?

A lire si le proxy vous bloque lors de ce TP.

- L'étape 2.2.2 a été laissée à titre informatif. Cette dernière a été réalisée par vos enseignants afin de vous éviter d'être bloqué dès le début du TP par le proxy.

- Il est normal d'obtenir une erreur liée à une absence de configuration du proxy dans les sections 3.2.2 et 3.4.2. La configuration vous est donnée dans les sections suivantes.

- Lorsqu'il est demandé d'utiliser le gestionnaire de paquets, si le proxy vous bloque : remplacez les instructions d'installation et de mise à jour de paquets par des créations de répertoires et des fichiers. L'idée étant de simplement générer de nouvelles couches dans votre image.

- Si vous n'arrivez pas à joindre le registre Docker Hub, pas de panique vos enseignants ont déployé un registre privé (sans authentification et non sécurisé...). Il est accessible à cette adresse : `prof:5000`[4]

Excepté pour la section 4 du TP, uniquement les images avec le tag `latest` seront disponibles. Exemple d'utilisation :

```
$ docker pull prof:5000/hello-world:latest
```

Si vous procédez de cette manière, il faudra toujours référencer ce registre au moment de l'instanciation de vos images.

[4]L'ip de la VM « enseignant » a été renseignée dans le fichier `etc/hosts` avant la création du *template*.

2 Configuration de l'environnement

Durée estimée : 15 minutes

2.1 Initialisation d'une nouvelle VM

→ Instanciez une nouvelle VM (*full clone*) à partir du patron `r509-template-docker-etudiant` (disponible sur le pve4). Choisissez le pve avec le plus de ressources disponibles.

→ Nommez la `r408-docker-<<mes initiales>>`, où `<<mes initales>>` est à remplacer avec vos initiales.

→ **SUPPRIMER l'étiquette « NE_PAS_SUPPRIMER »**

→ Démarrez-la, modifiez votre mot de passe (à l'aide `passwd`) et ouvrez une console.

Nom d'utilisateur : `user` Mot de passe : `pass`

Besoin de configurer le proxy pour le système et le gestionnaire de paquets ? Utiliser le script « Setup proxy » disponible sur le bureau de la VM (utilisez votre compte c3).

2.2 Installation de docker

Les instructions d'installation sont adaptées de la documentation officielle de Docker.

2.2.1 Les différentes options d'installation

Dans le cas présent, pour un serveur Ubuntu, il existe trois méthodes différentes d'installation :

- En installant le dépôt de paquets de Docker dans notre gestionnaire de paquets

- En téléchargeant et en installant manuellement les paquets

- En utilisant un script dit de « commodité »

Les deux premières options conviennent parfaitement dans un contexte de production.

La première solution est à privilégier de manière générale.

L'inconvénient de la seconde étant qu'il vous faudra télécharger et installer manuellement tous les paquets à chaque montée de version du *Docker Engine.*

La troisième est la plus simple du point de vue de l'installateur, il suffit de télécharger un script et de l'exécuter avec les droits d'administrateur afin d'installer la dernière version stable de Docker.

J'attire votre attention que l'exécution d'un script téléchargé depuis Internet et exécuté avec les privilèges administrateurs est un risque conséquent pour votre machine. C'est pour cette raison que cette méthode ne doit être utilisée que pour des environnements dit de développement.

Dans la section suivante, la troisième méthode, qui est appropriée à notre contexte éducatif, sera détaillée afin d'installer Docker. Vous êtes libre d'en choisir une autre.

2.2.2 Installation via le script de commodité

Afin de vous éviter une perte de temps à cause du proxy, cette étape a été réalisée par votre enseignant dans le patron de la vm.

→ Téléchargez le script à l'aide de la commande suivante :

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
```

→ Lancez l'installation avec la commande :

```
$ DRY_RUN=1 sudo sh ./get-docker.sh
```

`DRY_RUN=1` est une variable d'environnement utilisée par le script d'installation afin d'afficher les étapes au fur et à mesure de l'installation.

→ Pour vérifier que l'installation s'est correctement déroulée, veuillez exécuter la commande :

```
$ sudo docker --version
```

On obtient en retour la dernière version stable de Docker qui vient d'être installée sur notre système.

```
ml@r408-ml:~$ sudo docker --version
Docker version 20.10.22, build 3a2c30b
ml@r408-ml:~$
```

Figure 1: Version de Docker installée avec le script de commodité

2.3 Les étapes post-installation

Il s'agit d'une étape optionnelle, que je vous recommande tout de même d'effectuer dans le cadre de ce TP afin de travailler plus confortablement en ligne de commande avec Docker.

Le démon Docker est lié à un socket Unix. Par défaut, c'est l'utilisateur *root* qui détient ce socket et les autres utilisateurs y accède grâce à *sudo*.

Pour éviter de préfixer toutes les commandes par *sudo*, il est possible de créer un groupe `docker` et d'ajouter des utilisateurs à celui-ci. Lorsque le démon démarre, il va créer un socket accessible aux membres du groupe docker.

Attention, le groupe docker donne des privilèges *root* à votre utilisateur. Vous pouvez consulter les impacts que cela a sur la sécurité de votre système dans la documentation officielle de Docker (docker daemon attack surface).

→ Voici les commandes pour créer le groupe et ajouter votre nom d'utilisateur :

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

usermod modifie le compte d'un utilisateur. Grâce aux options **-aG** on va ajouter à l'utilisateur au groupe **docker**. **$USER** est une variable d'environnement contenant le nom de l'utilisateur de la session du terminal, elle est définie au moment de l'authentification (commande **login**).

→ Pour appliquer les changements au groupe **docker** sans redémarrer, utilisez la commande :

```
$ newgrp docker
```

→ Vous n'avez plus à préfixer vos commandes **docker** par **sudo**. Vous pouvez vérifier avec cette commande :

```
$ docker version
```

Les commandes qui vous seront fournies dans la suite du TP supposeront que cette étape a été appliquée. Vous êtes libre de continuer à préfixer par **sudo**.

3 A la découverte de Docker et de son interface de commande

Durée estimée : 1 heure

3.1 La commande *help*

→ Vous pouvez connaître les commandes qu'il est possible d'effectuer avec la *CLI* de Docker au moyen de la commande Docker `help` :

```
$ docker help
```

Pour avoir plus de détails sur une commande particulière, utilisez l'option `--help`

Par exemple, avec la commande `run` :

```
$ docker container run --help
```

3.2 Mon premier conteneur

3.2.1 Hello World ?

→ Il est de coutume lorsque l'on découvre un nouveau langage en informatique d'afficher hello-world dans une console. Il existe justement une image officielle se nommant `hello-world`. Nous allons instancier notre premier conteneur à partir de cette image :

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

Notre commande est équivalente à `docker container run hello-world:latest`. La commande `run` permet d'instancier un nouveau conteneur à partir d'une image, en l'occurrence l'image issue du dépôt `hello-world` en version `latest`. `latest` est un tag spécial utilisé par défaut si l'on en spécifie aucun.

3.2.2 Error response from daemon

Notre première tentative de création d'un conteneur s'est soldée par un échec… Vous devez obtenir un message d'erreur semble à celui-ci :

```
ml@r408-ml:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
docker: Error response from daemon: Get "https://registry-1.docker.io/v2/": net/http: request canceled while waiting for connect
ion (Client.Timeout exceeded while awaiting headers).
See 'docker run --help'.
ml@r408-ml:~$ _
```

Figure 3: Message d'erreur lors de la création du premier conteneur hello-world

Le registre contenant les images n'est pas joignable. Cette erreur réseau est symptomatique d'un problème de configuration du proxy.

3.2.3 Configuration du proxy pour le démon Docker

Sur notre *VM*, la configuration du proxy est effectuée au travers de variables d'environnement. Le démon docker n'utilise pas les variables d'environnement **`http_proxy`** et **`https_proxy`** de la machine hôte.

→ Pour configurer le proxy, veuillez créer un fichier à cet emplacement : **`/etc/systemd/system/docker.service.d/proxy.conf`**

→ Y renseigner les informations relatives au proxy de cette manière :

```
[Service]
Environment="http_proxy=http://<<user>>:<<password>>@192.168.0.2:3128"
Environment="https_proxy=http://<<user>>:<<password>>@192.168.0.2:3128"
Environment="no_proxy=localhost,127.0.0.1,prof"
```

Où **`<<user>>`** et **`<<password>>`** doivent être respectivement remplacés par votre identifiant et votre mot de passe pour accéder aux services numériques de l'Université.

→ Une fois le fichier enregistré, nous allons recharger la configuration du démon et relancer docker pour appliquer notre configuration :

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker.service
```

3.2.4 Hello World !

Maintenant que le proxy est configuré pour le démon, vous devez pourvoir télécharger l'image **`hello-world`** depuis le registre officiel de Docker.

→ Faites une nouvelle tentative d'instanciation d'un conteneur issu de l'image **`hello-world`**.

Si votre configuration est correcte, vous obtiendrez sur la sortie standard un message similaire à la capture d'écran de la figure 4.

```
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/
```

Figure 4: Résultat d'exécution d'un conteneur hello-world

3.3 Les commandes pour lister les images et conteneurs

3.3.1 Lister les images

Lorsque vous effectuez une commande `pull`, `create` ou `run`, le démon va dans un premier temps télécharger depuis un registre l'image indiquée.

→ Pour obtenir la liste des images présentes sur la machine hôte, utilisez la commande :

```
$ docker image ls
```

```
ml@r408-ml:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    feb5d9fea6a5   15 months ago   13.3kB
```

*Figure 5: Résultat de la commande **`docker images`** ou **`docker image ls`***

Nous retrouvons notre image hello-world avec le tag latest.

L'identifiant de l'image est un condensé (*digest*) c'est-à-dire une valeur générée à partir d'une fonction de hachage appliquée sur notre image.

Nous retrouvons également la date à laquelle l'image a été créée ainsi que la taille qu'elle occupe sur notre disque.

N.B : vous allez fréquemment faire référence à des images ou des conteneurs par leur identifiant. Vous n'êtes pas obligé de le taper en intégralité. Il suffit de taper le ou les premiers chiffres discriminants.

3.3.2 Lister les conteneurs

→ Sur le même principe, vous pouvez lister les conteneurs de cette manière :

```
$ docker container ls -a
```

L'option `-a` permet de lister tous les conteneurs. Par défaut, uniquement les conteneurs actifs sont affichés par cette commande.

```
ml@r408-ml:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
1fd1a3a81c09   hello-world   "/hello"   11 minutes ago   Exited (0) 2 minutes ago              amazing_ardinghelli
ee5446ef9e9b   hello-world   "/hello"   11 minutes ago   Exited (0) 11 minutes ago             pedantic_franklin
2977c484600b   hello-world   "/hello"   15 minutes ago   Exited (0) 15 minutes ago             ecstatic_meitner
4aafab1d81d3   hello-world   "/hello"   37 minutes ago   Exited (0) 37 minutes ago             pedantic_lalande
ml@r408-ml:~$
```

*Figure 6: Résultat d'exécution de la commande **`docker ps -a`** ou **`docker container ls -a`***

Sur la figure 6, on note que 4 conteneurs issus de l'image « hello-world » sont au statut « Exited » avec un code retour à 0. Cela veut dire que leur processus lancé au démarrage du conteneur s'est correctement achevé. Un conteneur peut tout aussi bien exécuter une tâche unique et rapide comme une tâche de longue durée.

3.4 Ubuntu avec Docker

3.4.1 Instanciation du conteneur ubuntu

Nous allons créer un conteneur un peu plus ambitieux embarquant une distribution Ubuntu complète.

→ Exécutez la commande suivante :

```
$ docker container run -it --name ubuntu-test ubuntu:20.04
```

L'option `-i` correspond au mode interactif, c'est-à-dire que nous conservons l'entrée standard attachée au conteneur. L'option `--name` permet de donner un nom porteur de sens à votre conteneur (par défaut, des noms exotiques sont générés aléatoirement).

Notez ici que nous avons indiqué le tag **20.04** afin d'obtenir une version spécifique d'Ubuntu.

N.B : pour sortir du conteneur, vous pouvez taper la commande `exit`

Comme vous pouvez le constater en consultant la liste des images, c'est une distribution relativement légère.

```
ml@r408-ml:~$ docker images
REPOSITORY    TAG          IMAGE ID       CREATED          SIZE
postgres      12-alpine    4dc174bd4291   7 days ago       237MB
busybox       latest       827365c7baf1   7 days ago       4.86MB
ubuntu        latest       6b7dfa7e8fdb   3 weeks ago      77.8MB
ubuntu        20.04        d5447fc01ae6   3 weeks ago      72.8MB
hello-world   latest       feb5d9fea6a5   15 months ago    13.3kB
```

Figure 7: Informations sur l'image ubuntu:20.04

3.4.2 Installation de paquets

Nous allons mettre à jour et installer quelques les paquets.

→ Veuillez mettre à jour les paquets et installez les commandes `curl`, `wget` et `nano`

N.B : si vous avez quitté le shell de votre conteneur avec la commande `exit`*, votre conteneur est à l'état «Stoppé». Vous pouvez au choix, démarrer le conteneur (dans ce cas pensez à attacher stdout/stderr avec l'option adéquate) ou en instancier un nouveau.*

Aidez-vous du cours sur les conteneurs p.18 ou de la commande `docker help`*.*

```
apt-get update
Err:1 http://archive.ubuntu.com/ubuntu focal InRelease
  Could not connect to archive.ubuntu.com:80 (185.125.190.39), connection timed out Could not connect to archive.ubuntu.com:80 (
91.189.91.38), connection timed out Could not connect to archive.ubuntu.com:80 (91.189.91.39), connection timed out Could not co
nnect to archive.ubuntu.com:80 (185.125.190.36), connection timed out
Err:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease
  Unable to connect to archive.ubuntu.com:http:
Err:3 http://archive.ubuntu.com/ubuntu focal-backports InRelease
  Unable to connect to archive.ubuntu.com:http:
Err:4 http://security.ubuntu.com/ubuntu focal-security InRelease
  Could not connect to security.ubuntu.com:80 (185.125.190.36), connection timed out Could not connect to security.ubuntu.com:80
 (91.189.91.39), connection timed out Could not connect to security.ubuntu.com:80 (185.125.190.39), connection timed out Could n
ot connect to security.ubuntu.com:80 (91.189.91.38), connection timed out
Reading package lists...
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal/InRelease  Could not connect to archive.ubuntu.com:80 (185.125.1
90.39), connection timed out Could not connect to archive.ubuntu.com:80 (91.189.91.38), connection timed out Could not connect t
o archive.ubuntu.com:80 (91.189.91.39), connection timed out Could not connect to archive.ubuntu.com:80 (185.125.190.36), connec
tion timed out
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal-updates/InRelease  Unable to connect to archive.ubuntu.com:http:
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/focal-backports/InRelease  Unable to connect to archive.ubuntu.com:htt
p:
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/focal-security/InRelease  Could not connect to security.ubuntu.com:80
 (185.125.190.36), connection timed out Could not connect to security.ubuntu.com:80 (91.189.91.39), connection timed out Could n
ot connect to security.ubuntu.com:80 (185.125.190.39), connection timed out Could not connect to security.ubuntu.com:80 (91.189.
91.38), connection timed out
W: Some index files failed to download. They have been ignored, or old ones used instead.
```

Figure 8: Echec de la mise à jour des paquets sur le conteneur Ubuntu

Encore une erreur réseau… Les dépôts ne sont pas joignables. Pourtant, nous avons déjà effectué la configuration du proxy pour Docker dans la section 3.2.3 ?

En réalité, nous avons configuré le proxy pour le démon s'exécutant sur l'hôte. Lors des rappels du cours dans la section 1.3.2, nous vous avions notifié que :

« Il [Le conteneur] possède son propre environnement d'exécution, distinct et isolé de celui des autres conteneurs et de son hôte. »

3.4.3 Les différentes options pour configurer un proxy pour les conteneurs

Plusieurs possibilités s'offrent à vous :

- Vous effectuez la configuration du proxy dans le conteneur ubuntu tel que vous l'avez fait avec la VM

- Vous créez un nouveau conteneur en passant la configuration du proxy via des variables d'environnement. Vous pouvez vous appuyez sur les options suivantes :

`-e NOM_ENV_VAR_A="VALEUR_ENV_VAR_A"` pour définir une variable d'environnement au démarrage d'un conteneur. Cette option est utilisable plusieurs fois.

`--env-file <<chemin-vers-env-file>>` pour charger au démarrage du conteneur un fichier contenant des variables d'environnement.

- Vous placez sur votre client Docker un fichier de configuration qui permettra d'appliquer la configuration du proxy sur tous vos futurs conteneurs.

Les deux premières méthodes sont tout à fait envisageables, mais vous devrez réitérer la configuration du proxy sur tous vos futurs conteneurs qui en auront besoin.

Choisissez la méthode qui vous convient le mieux. Dans la section suivante, nous détaillerons la dernière solution à partir du fichier de configuration.

3.4.4 Configuration du proxy pour les conteneurs avec un fichier côté client

→ Si ce n'est pas déjà fait, fermer le shell de votre conteneur (commande `exit`).

→ Merci de créer un fichier à cet emplacement : `~/.docker/config.json`

→ Y renseigner les informations relatives au proxy au format json:

```
{
 "proxies":
 {
   "default":
   {
     "httpProxy": "http://<<user>>:<<password>>@192.168.1.12:3128",
     "httpsProxy": "http://<<user>>:<<password>>@192.168.1.12:3128",
     "noProxy": "127.0.0.0/8,localhost,prof"
   }
 }
}
```

3.4.5 Nouvelle instanciation d'un conteneur ubuntu

→ Nous pouvons maintenant créer un nouveau conteneur avec la commande :

```
$ docker container run -i --name linux-proxy ubuntu:20.04
```

Ne vous attachez pas trop à vos conteneurs… En cas d'erreur de configuration, il ne faut pas hésiter à le supprimer et à en instancier un nouveau.

→ Vous pouvez vérifier la bonne prise en compte de votre proxy grâce à la commande ***env***.

→ Reprenez la mise à jour et l'installation des paquets demandées dans la section 3.4.2.

→ Testez les commandes que vous venez d'installer.

En cas d'erreur avec `nano`, vous allez peut-être devoir exporter la variable d'environnement **`TERM`** avec la valeur `xterm`.

3.5 Le point d'entrée et la commande par défaut

Lors de la définition d'une image, un point d'entrée, une commande ou les deux sont spécifiés. L'objectif principal de l'instruction CMD est de définir une commande par défaut. Si uniquement des paramètres par défaut sont indiqués avec l'instruction CMD, alors une instruction ENTRYPOINT devra être présente.

Avec un conteneur hello-world, nous avions un texte qui s'affichait sur la sortie standard ; tandis que notre conteneur ubuntu exécutait un **shell**.

Vous pouvez connaître cette commande par défaut de plusieurs manières :

- En consultant la documentation de l'image (exemple : page d'un dépôt sur Docker Hub)

- En lisant le Dockerfile[5]

- En inspectant l'image avec la commande `inspect`

3.5.1 Inspection d'une image busybox

Exerçons-nous sur un nouveau dépôt, busybox. Il s'agit d'une image légère (entre 1 et 5 Mb sur le disque) qui contient de nombreux utilitaires UNIX.

→ Pour récupérer une image, sans créer, ni démarrer un conteneur dans la foulée, vous pouvez utiliser la commande **pull**.

```
$ docker image pull busybox
```

```
ml@r408-ml:~/.docker$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
busybox       latest    827365c7baf1   7 days ago       4.86MB
ubuntu        20.04     d5447fc01ae6   3 weeks ago      72.8MB
hello-world   latest    feb5d9fea6a5   15 months ago    13.3kB
ml@r408-ml:~/.docker$ docker inspect 82
```

Figure 9: Information sur l'image busybox:latest

→ Utilisez la commande **inspect** afin d'identifier la commande par défaut de cette image. Pour connaître comment celle-ci fonctionne, vous pouvez vous appuyer sur la commande **help**. Son fonctionnement est détaillé dans la section 3.1. Lorsque vous pensez avoir trouvé, demandez confirmation à l'enseignant chargé du TP.

3.5.2 Surcharge de la commande par défaut de busybox

Au moment d'instancier un conteneur, vous pouvez surcharger la commande par défaut.

→ Grâce aux commandes **run** ou **create**, essayez de surcharger la commande par défaut par quelques commandes communes sur Linux, par exemple **ls**, **env**…

Vous pouvez vous aider de la commande **help**.

3.6 La création d'images

Depuis le début de ce TP nous n'avons fait qu'utiliser des images existantes afin d'instancier nos conteneurs. Nous allons voir les deux méthodes qui existent pour créer des images.

3.6.1 Création d'une image à partir d'un conteneur

La création d'une image depuis un conteneur est utile pour le débogage ou exporter une « copie de travail » sur une autre machine.

[5]Dockerfile : Fichier contenant un ensemble d'instructions permettant de construire une image Docker

Dans la section 3.4.5, nous avons créé un conteneur et l'avons nommé **linux-proxy**. Pour rappel, vous pouvez consulter les conteneurs avec la commande `container ls :`

```
$ docker container ls -a
```

Une image est immuable. Les modifications que nous avons apportées à notre conteneur **linux-proxy** seront perdu en cas de suppression de ce dernier.

Nous allons y remédier en créant une image à partir de ce conteneur au moyen de la commande `commit`.

→ Veuillez créer une image Docker à partir du conteneur nommé « linux-proxy ». Vous la nommerez « ubuntu-tp3-commit » et lui attribuerez le tag « latest ».

→ Instanciez un nouveau conteneur et vérifiez que les modifications que nous avons effectuées dans la section 3.4.5 à l'image de base ubuntu sont bien présentes.

Attention, avez-vous noté que le proxy a été exporté dans les variables d'environnement ? **Cette image ne doit en aucun cas être partagée sur un registre,** car elle contient des informations confidentielles (nom d'utilisateur et mot de passe personnel de l'IUT).

Aucune information sensible ou spécifique à un environnement ne doit être présente dans une image !

3.6.2 Création d'une image à partir d'un Dockerfile

Cette seconde méthode est à privilégier, car les instructions pour construire l'image seront explicitement tracées. Cela facilite la documentation, la maintenance de vos images et surtout, vous obtenez un comportement reproductible et prédictible.

→ Créez un nouveau répertoire, définissez-le comme votre répertoire courant et ajoutez un fichier se nommant « Dockerfile ». Attention, la première lettre **doit** être en majuscule (sinon le client ne trouvera pas votre fichier).

→ Compléter ce Dockerfile pour construire une image identique à celle que nous avons créé dans la section précédente (ubuntu-tp3-commit).

Si vous avez besoin d'aide pour la syntaxe du fichier Dockerfile. Vous disposez d'un exemple de Dockerfile dans le CM n°3 sur les conteneurs Linux ou vous pouvez consulter la documentation officielle du Dockerfile.

Quelques indices pour vous aider dans votre tâche :

- Nous sommes partis d'une image ubuntu avec le tag 22.04

- Nous avons mis à jour et installé les paquets suivants : **`curl wget nano`**.

- Nous avons exporté une variable d'environnement **TERM** ayant pour valeur **xterm**.

→ Lorsque que votre Dockerfile est prêt, faites valider votre fichier par votre enseignant et construisez votre image à l'aide de la commande :

```
$ docker build --tag ubuntu-tp3-dockerfile:latest .
```

Ici, nous indiquons construire une image **ubuntu-tp3-dockerfile** avec le tag **latest**.

Le « **.** » est le chemin vers le répertoire de contexte de construction de l'image.

La commande `build` recherche dans ce contexte de construction un fichier se nommant **D**ockerfile. Si vous l'avez nommé autrement, vous pouvez utiliser l'option `--file`.

N.B : si vous n'avez pas appliqué la configuration proxy au niveau de votre client docker, vous risquez d'avoir des échecs lors de la construction de votre image. Utilisez alors les options `--build-arg http_proxy=… --build-arg https_proxy=….`

→ Instanciez un conteneur depuis votre nouvelle image et vérifiez que vous disposez des mêmes fonctionnalités que l'image ubuntu-tp3-commit.

3.7 Un peu de ménage…

3.7.1 Suppression sélective d'images et conteneurs

→ Affichons les conteneurs et les images que nous avons créé depuis le début du TP. Vous devriez obtenir un résultat similaire à la figure 10.

```
ml@r408-ml:~$ docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
latest                  latest    ff7475f973fc   32 minutes ago   126MB
ubuntu-tp3-dockerfile   latest    07e3b2195b41   32 minutes ago   126MB
ubuntu-tp3-commit       latest    07e3b2195b41   32 minutes ago   126MB
postgres                latest    a26eb6069868   7 days ago       379MB
busybox                 latest    827365c7baf1   8 days ago       4.86MB
ubuntu                  22.04     6b7dfa7e8fdb   3 weeks ago      77.8MB
ubuntu                  20.04     d5447fc01ae6   3 weeks ago      72.8MB
hello-world             latest    feb5d9fea6a5   15 months ago    13.3kB
ml@r408-ml:~$ docker ps -a
CONTAINER ID   IMAGE                   COMMAND                  CREATED              STATUS                          PORTS     N
AMES
96b465a12859   ubuntu-tp3-commit       "bash"                   About a minute ago   Exited (0) About a minute ago             m
agical_jennings
30da76ec49ad   ubuntu-tp3-dockerfile   "bash"                   2 minutes ago        Exited (0) 2 minutes ago                  r
ice_brattain
91ef4aa165e7   ubuntu-tp3-dockerfile   "bash"                   2 minutes ago        Exited (0) 2 minutes ago                  s
tupefied_mcnulty
9aaef3ec2b76   hello-world             "/hello"                 3 minutes ago        Exited (0) 3 minutes ago                  e
lated_poitras
cbcaf3a91ced   6b7dfa7e8fdb            "/bin/sh -c 'apt upd…"   32 minutes ago       Exited (1) 32 minutes ago                 e
xciting_haibt
6eb06734c35d   busybox                 "sh"                     6 hours ago          Exited (0) 6 hours ago                    c
harming_shtern
5d1f090bb102   ubuntu:20.04            "bash"                   6 hours ago          Exited (0) About an hour ago              l
inux-proxy
ml@r408-ml:~$ _
```

Figure 10: Résultats des commandes ***`docker images`*** *et* ***`docker ps -a`***

Lorsque l'on utilise docker, nous pouvons vite nous retrouver avec une multitude de conteneurs et d'images.

→ Essayez de supprimer quelques images et conteneurs avec les commandes suivantes :

```
$ docker container rm <<identifiant(s)>>
$ docker image rm <<identifiant(s)>>
```

3.7.2 Docker prune

Vous pouvez effectuer un nettoyage plus radical grâce aux commandes `prune`.

→ Par exemple, nous supprimons ici les conteneurs arrêtés et les images flottantes[6] :

```
$ docker container prune
$ docker image prune
```

→ Comme vous avez expérimenté et que des identifiants trainent dans les variables d'environnement, nous allons tout effacer.

```
$ docker system prune --all
```

Cette approche peut vous interpeller. Toutefois, c'est relativement courant dans le milieu professionnel. La persistance d'un état ou d'informations doit s'effectuer en dehors des conteneurs, par exemple en montant des volumes. Nous aborderons ce sujet dans un autre TP.

3.8 Docker Hub

Docker Hub est un registre hébergé par la société Docker. C'est ici que sont publiées des images maintenues par la communauté et des entreprises. Par défaut, Docker Engine pointe vers ce registre.

En tant qu'utilisateur de la version gratuite, nous sommes limités à des dépôts publics et à 200 pull d'images toutes les 6 heures. Il est sous licence Apache 2.0, vous pouvez donc déployer le vôtre.

La recherche d'image avec le client Docker est possible, mais un site internet simple et efficace existe : https://hub.docker.com/

Toutes les images que nous avons utilisées précédemment ont été recherchées et téléchargées depuis ce registre.

La figure 11 montre un exemple de recherche pour une image ubuntu.

[6]Image flottante : *dangling image* en anglais. Il s'agit d'une image associée à aucun dépôt.

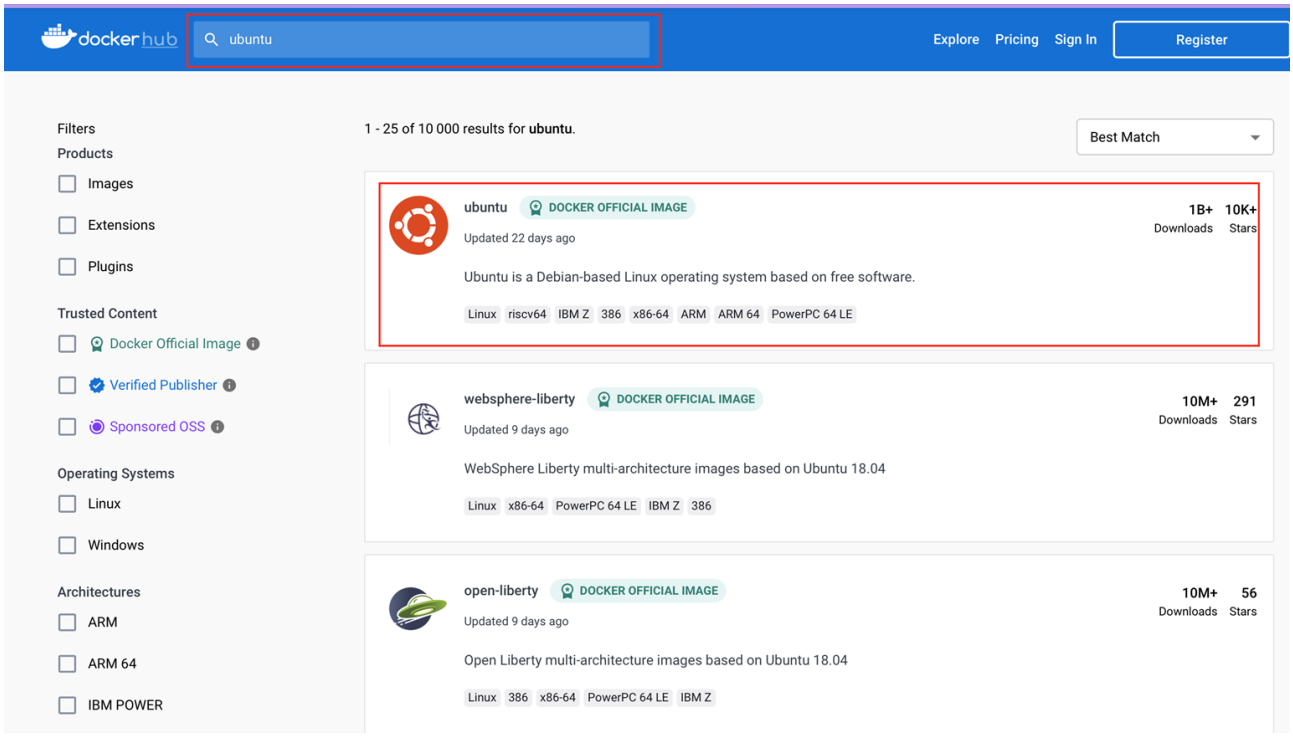

Figure 11: Recherche du dépôt 'ubuntu' sur docker hub

Certains dépôts bénéficient d'une étiquette « DOCKER OFFICIAL IMAGE ». Cela veut dire qu'ils contiennent des images vérifiées et publiées par les employés de la société Docker.

Généralement, la documentation qui leur est associée est de très bonne qualité et les mises à jour de sécurité sont appliquées fréquemment. Lorsque l'on débute avec Docker, il est recommandé de privilégier ce type de dépôt.

→ Prenez un peu de temps pour explorer et consulter quelques images disponibles sur Docker Hub ainsi que la documentation associée à leur image.

4 Exploitation d'un service web avec Docker

Durée estimée : 45 minutes

4.1 Présentation du service web « score »

Vos enseignants mettent à votre disposition un service web écrit en java empaqueté dans un jar[7].

Pour l'exécuter, vous aurez besoin d'un environnement d'exécution Java (JRE) en version 11.

À son démarrage, le serveur écoute les requêtes HTTP sur le port 8080.

Il expose une API HTTP REST, dont la documentation est donnée à l'annexe n°1.

4.2 Création d'une image encapsulant le service web (JRE11)

→ Veuillez télécharger le jar du service avec la requête suivante :

```
$ curl -LJO http://prof/score-jre11.jar
```

→ Recherchez sur Docker Hub le dépôt « eclipse-temurin », il s'agit d'un dépôt offrant des images en différentes versions de JDK et JRE maintenues par la fondation Eclipse. Si le proxy bloque le téléchargement, vous pouvez utiliser le tag `xx-jre` (où `xx` correspond à la version de Java. e.g. `11-jre`) sur notre registre privé (`prof:5000`).

→ Écrivez un Dockerfile permettant d'exécuter le jar précédemment téléchargé. Vous devriez trouver toute l'aide nécessaire sur la page du dépôt sur Docker Hub.

→ Construisez une image à partir de votre Dockerfile, vous lui appliquerez le tag « score:jre11 »

4.3 Déploiement du service (JRE11) sur votre machine hôte

→ Vérifiiez que votre machine hôte comporte bien une variable d'environnement no_proxy avec à minima les valeurs suivantes "localhost, 127.0.0.0/8". Sinon le proxy bloquera vos requêtes.

→ Instanciez maintenant un conteneur à partir de votre nouvelle image grâce à cette commande :

```
$ docker container run -d -p 8080:8080 --name ws-score-jre11 score:jre11
```

L'option `-d` permet d'exécuter le conteneur en tâche de fond. Un serveur étant un processus sans fin, on ne souhaite pas bloquer la console de notre hôte.

[7]jar : archive contenant des fichiers java compilés

L'option `-p <<host:conteneur>>` permet de mapper le port TCP 8080 du conteneur sur le port TCP 8080 de l'hôte. De cette manière, vous pourrez transmettre des requêtes HTTP au service dans son conteneur avec l'url de base « http://localhost:8080 ».

→ A partir de la documentation disponible en annexe 1, essayez maintenant d'exécuter des requêtes HTTP sur votre conteneur, voici un exemple :

```
$ curl -X POST http://localhost:8080/points
```

→ Consulter les logs du conteneur avec la commande Docker `logs`. Les mettre en relations avec ce que vous avez obtenu en réponse de vos requêtes **`curl`**.

4.4 Déploiement du service en JRE17 sur votre machine hôte

Les développeurs à l'initiative du service « score » viennent de passer sur la version 17 de Java car il s'agit de la dernière version LTS[8].

→ Vous récupérerez le jar en version 17 grâce à la commande :

```
$ curl -LJO http://prof/score-jre17.jar
```

→ Déployez simultanément sur votre machine hôte les deux versions du service :

- pour le conteneur avec un JRE11 sur le port 8080 (déjà fait dans la section 4.3)
- pour le conteneur avec un JRE17 sur le port 8081

[8] *LTS : Long Terme Support*. Version d'un logiciel qui va bénéficier de mises à jour sur un cycle plus long que celui de développement. Dans un contexte de production, il faut les privilégier afin de bénéficier des mise à jour adressant des failles de sécurités ou des problèmes de performances.

5 Fin de séance

Pensez à éteindre votre VM pour libérer les ressources mobilisées.

Ne supprimez pas votre VM, vous pourrez la réutiliser lors des prochains TD et TP.

6 Conclusion

Au cours de ce TP, vous avez appris à installer Docker et acquis les bases pour manipuler les conteneurs et les images.

Vous vous êtes également confronté à la difficulté de travailler derrière un proxy avec Docker. En effet, la configuration est à effectuer entre le *démon*, les conteneurs et les contextes de construction d'images.

Vous avez aussi déployé votre premier service web à l'aide des conteneurs. Pour cette séance, nous nous sommes contenté d'utiliser un fichier exécutable. Par la suite, vous apprendrez à construire ce fichier exécutable depuis une image directement à partir des sources.

Vous avez pu constater la rapidité de déploiement des solutions à bases de conteneurs. Que ce soit sur la partie opérationnelle (faible temps de démarrage d'un conteneur ubuntu par exemple) ou dans la conception des images servant à instancier les conteneurs.

Enfin, dans la partie 4.4, vous avez déployé simultanément 2 applications Java écoutant sur le même port TCP (8080) et sous deux versions différentes d'un JRE (11 et 17). Créer deux machines virtuelles distinctes pour adresser ces problématiques semble un peu excessif vu la simplicité de ces applications. C'est notamment sur ce cas d'usage que la conteneurisation tire son épingle du jeu par rapport à d'autres solutions de virtualisation.

Annexe n°1 : Documentation de l'api du service web « score »

```yaml
info:
  title: webservice - score
  port: 8080
paths:
  /points:
    POST:
      summary: Incrémente d'un point le compteur du service
      responses:
        "200":
          description: Retourne le nombre de points après incrémentation
          content:
            application/json:
              schema:
                type: number
    GET:
      summary: Retourne le nombre de points dans le compteur du service
      responses:
        "200":
          description: Le nombre de points au compteur
          content:
            application/json:
              schema:
                type: number
    DELETE:
      summary: Réinitialise le nombre de points du compteur à 0
      responses:
        "204":
          description: Retourne le nombre de points après incrémentation
```

N.B : POST, GET, et DELETE sont des verbes HTTP.