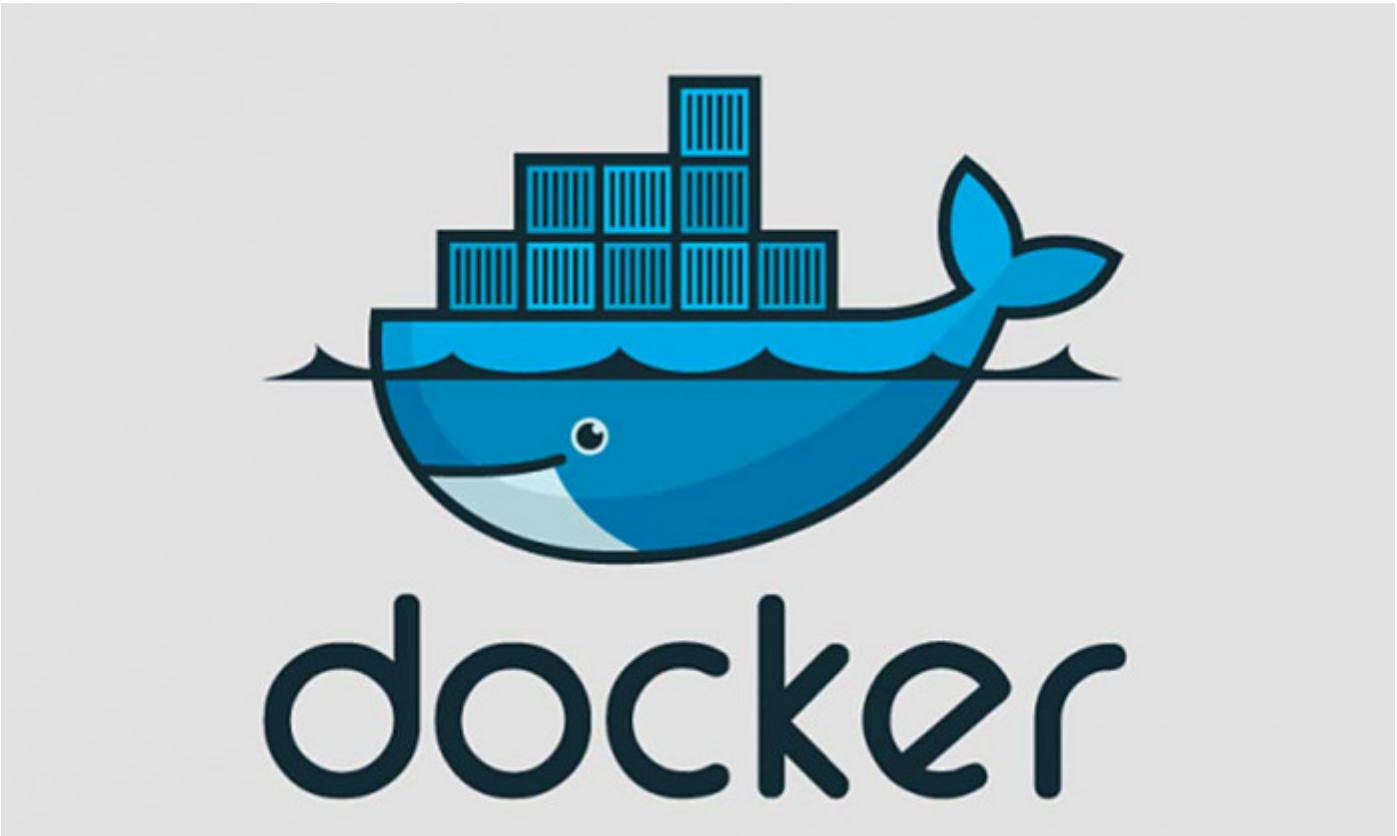
Introduction à Docker

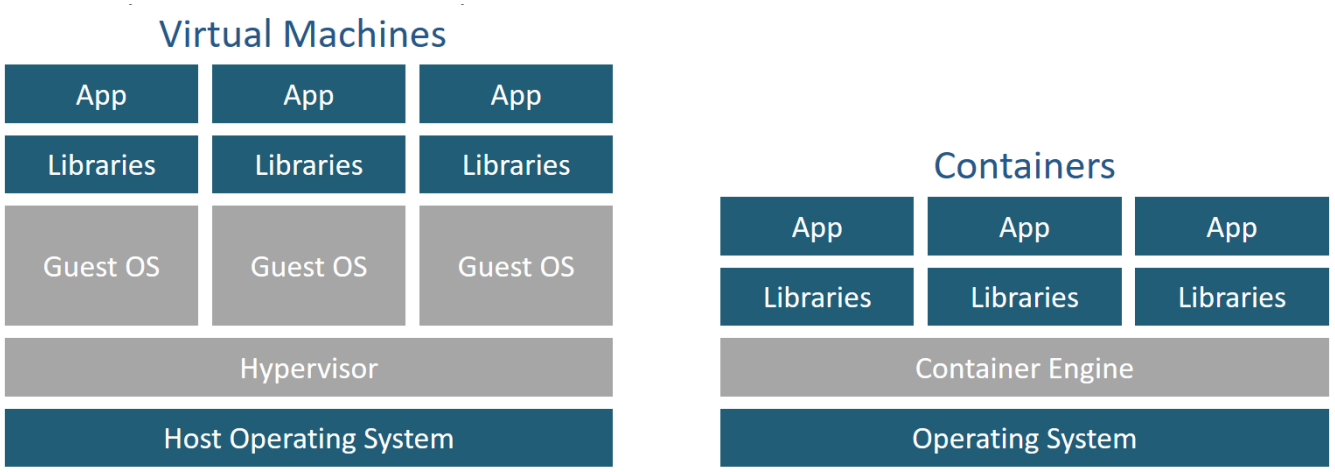
Travaux pratiques

Formation Concepteur et Développeur d'Applications



1. Introduction

Docker est un outil qui **simplifie le processus de gestion des processus d'application dans des conteneurs** (containers en anglais). Ils permettent d'**exécuter des applications dans des processus isolés des ressources**. Ils sont semblables à des machines virtuelles, mais les conteneurs sont plus transposables, respectueux des ressources et plus dépendants du système hôte.



Dans ce TP nous allons installer et utiliser Docker Community Edition (CE) sur une distribution Linux Ubuntu 20.04 Server.

1.1. Pré-requis

- Un compte actif sous Docker Hub.

1.2. Terminologie Docker

Images - Le système de fichiers et la configuration de notre application utilisés pour créer des conteneurs. Pour en savoir plus sur une image Docker, exécutez docker image inspect alpine.

Conteneurs - Exécution d'instances d'images Docker - les conteneurs exécutent les applications réelles. Un conteneur inclut une application et toutes ses dépendances. Il partage le noyau avec d'autres conteneurs et s'exécute comme un processus isolé dans l'espace utilisateur du système d'exploitation hôte.

Docker daemon - Service d'arrière-plan s'exécutant sur l'hôte qui gère la construction, l'exécution et la distribution des conteneurs Docker.

Client Docker - L'outil de ligne de commande qui permet à l'utilisateur d'interagir avec le démon Docker.

Docker Store - est, entre autres, un registre d'images Docker. Vous pouvez considérer le registre comme un répertoire de toutes les images Docker disponibles.

2. Installation de Docker

L'installation de Docker se réalise via la commande apt install docker.io

```
sudo apt update -y && sudo apt upgrade -y
sudo apt install docker.io
```

Installation de paquet permettant à apt d'utiliser les paquets via HTTPS

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Vérifiez que l'installation est correcte :

```
apt-cache policy docker.io
```

```
docker.io:
  Installé : 18.09.7-0ubuntu1~18.04.4
  Candidat : 18.09.7-0ubuntu1~18.04.4
 Table de version :
 *** 18.09.7-0ubuntu1~18.04.4 500
        500 http://fr.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages
        500 http://fr.archive.ubuntu.com/ubuntu bionic-security/universe amd64 Packages
        100 /var/lib/dpkg/status
     17.12.1-0ubuntu1 500
        500 http://fr.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages
```

Puis via la ligne de commande :

```
sudo systemctl status docker
```

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; disabled; vendor preset: enabled)
   Active: inactive (dead) since Tue 2020-02-18 19:45:37 UTC; 47s ago
     Docs: https://docs.docker.com
  Process: 1669 ExecStart=/usr/bin/dockerd -H fd://
--containerd=/run/containerd/containerd.sock (code=exited, statu
 Main PID: 1669 (code=exited, status=0/SUCCESS)
```

Si le service est inactif, pensez à l'activer via la commande

```
sudo service docker start
```

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; disabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-02-18 19:46:48 UTC; 3s ago
     Docs: https://docs.docker.com
 Main PID: 1993 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─1993 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

2.1. Exécution de la commande Docker sans sudo

Par défaut, la commande docker ne peut être exécutée que par l'utilisateur root ou par un utilisateur du groupe docker, créé automatiquement lors du processus d'installation de Docker.

Si vous voulez éviter de taper sudo chaque fois que vous exécutez la commande docker, ajoutez votre nom d'utilisateur au groupe docker:

```
sudo usermod -aG docker username
```

Pour appliquer la nouvelle appartenance à un groupe, déconnectez-vous du serveur et reconnectez-vous, ou tapez ce qui suit:

```
su - ${USER}
```

Vous serez invité à entrer le mot de passe de votre utilisateur pour continuer.

Confirmez que votre utilisateur est maintenant ajouté au groupe docker en tapant:

```
id -nG
```

Vous devez avoir un résultat similaire à :

```
roger adm cdrom sudo dip www-data plugdev lxd docker
```

La suite de ce TP suppose que vous exécutez la commande docker en tant qu'utilisateur du groupe docker.

2.2. Utilisation de la commande Docker

Utiliser docker consiste à lui transmettre une chaîne d'options et de commandes suivie d'arguments. La syntaxe prend cette forme:

```
docker [option] [command] [arguments]
```

Pour afficher toutes les sous-commandes disponibles, tapez:

```
docker
```

A partir de Docker 18, la liste complète des sous-commandes disponibles comprend:

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

Pour afficher les options disponibles pour une commande spécifique, tapez:

```
sudo docker docker-subcommand --help
```

Pour afficher des informations sur Docker à l'échelle du système, utilisez:

```
docker info
```

Explorons certaines de ces commandes. Nous allons commencer par travailler avec des images.

2.3. Utilisation des images Docker

Les conteneurs Docker sont construits à partir d'images Docker. Par défaut, Docker extrait ces images de Docker Hub, un registre Docker géré par Docker, la société à l'origine du projet Docker. Tout le monde peut héberger ses images Docker sur Docker Hub. Ainsi, la plupart des applications et des distributions Linux dont vous aurez besoin auront des images hébergées ici.

Pour vérifier si vous pouvez accéder aux images et les télécharger à partir de Docker Hub, tapez:

```
docker run hello-world
```

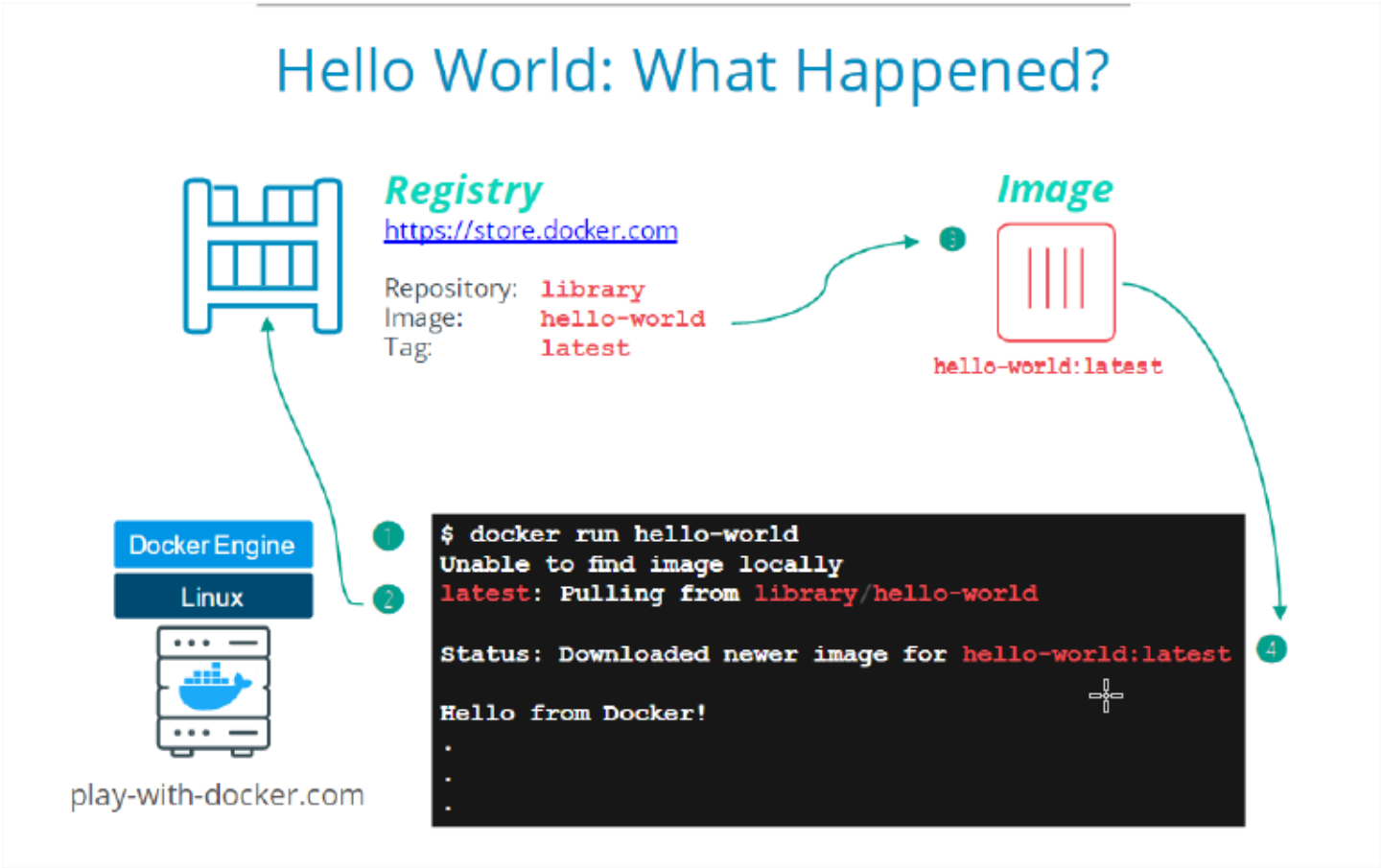
La sortie de données indiquera que Docker fonctionne correctement:

```
docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
Status: Downloaded newer image for 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.
```

Docker n'a pas pu trouver initialement l'image hello-world localement. Il a donc téléchargé l'image à partir de Docker Hub, le dépôt par défaut. Une fois l'image téléchargée, Docker a créé un conteneur à partir de celle-ci et de l'application exécutée dans le conteneur, affichant le message.



Vous pouvez rechercher des images disponibles sur Docker Hub en utilisant la commande docker avec la sous-commande search. Par exemple, pour rechercher l'image Ubuntu, tapez:

```
sudo docker search ubuntu
```

Le script analysera Docker Hub et renverra une liste de toutes les images dont le nom correspond à la chaîne de recherche. Dans ce cas, le résultat sera similaire à ceci:

```
NAME                                                     DESCRIPTION
STARS               OFFICIAL            AUTOMATED
ubuntu                                                   Ubuntu is a Debian-based Linux
operating sys...    10509               [OK]
dorowu/ubuntu-desktop-lxde-vnc                           Docker image to provide HTML5 VNC
interface ...    392                                   [OK]
rastasheep/ubuntu-sshd                                   Dockerized SSH service, built on top
of offi...    242                                   [OK]
consol/ubuntu-xfce-vnc                                   Ubuntu container with "headless" VNC
session...    210                                   [OK]
ubuntu-upstart                                           Upstart is an event-based replacement
for th...    105                 [OK]
neurodebian                                              NeuroDebian provides neuroscience
research s...    64                  [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5
ubuntu-16-nginx-php-phpmyadmin-mysql-5          50                                        [OK]
```

```
ubuntu-debootstrap                                          debootstrap --variant=minbase
--components=m...   42                  [OK]
nuagebec/ubuntu                                             Simple always updated Ubuntu docker
images w...   24                                        [OK]
i386/ubuntu                                                 Ubuntu is a Debian-based Linux
operating sys...   18
1and1internet/ubuntu-16-apache-php-5.6                      ubuntu-16-apache-php-5.6
14                                      [OK]
1and1internet/ubuntu-16-apache-php-7.0                      ubuntu-16-apache-php-7.0
13                                      [OK]
ppc64le/ubuntu                                              Ubuntu is a Debian-based Linux
operating sys...   13
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10
ubuntu-16-nginx-php-phpmyadmin-mariadb-10       11                                          [OK]
1and1internet/ubuntu-16-nginx-php-5.6                       ubuntu-16-nginx-php-5.6
8                                       [OK]
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4           ubuntu-16-nginx-php-5.6-wordpress-4
7                                       [OK]
1and1internet/ubuntu-16-apache-php-7.1                      ubuntu-16-apache-php-7.1
6                                       [OK]
darksheer/ubuntu                                            Base Ubuntu Image -- Updated hourly
5                                       [OK]
1and1internet/ubuntu-16-nginx-php-7.0                       ubuntu-16-nginx-php-7.0
4                                       [OK]
pivotaldata/ubuntu                                          A quick freshening-up of the base
Ubuntu doc...   3
pivotaldata/ubuntu16.04-build                               Ubuntu 16.04 image for GPDB
compilation         2
smartentry/ubuntu                                           ubuntu with smartentry
1                                       [OK]
1and1internet/ubuntu-16-sshd                                ubuntu-16-sshd
1                                       [OK]
1and1internet/ubuntu-16-php-7.1                             ubuntu-16-php-7.1
1                                       [OK]
pivotaldata/ubuntu-gpdb-dev                                 Ubuntu images for GPDB development
1
```

Dans la colonne OFFICIAL, OK indique une image construite et prise en charge par la société derrière le projet. Une fois que vous avez identifié l'image que vous souhaitez utiliser, vous pouvez la télécharger sur votre ordinateur à l'aide de la sous-commande pull.

Exécutez la commande suivante pour télécharger l'image officielle ubuntu sur votre ordinateur:

```
sudo docker pull ubuntu
```

Vous obtenez un résultat similaire à :

```
Using default tag: latest
latest: Pulling from library/ubuntu
5c939e3a4d10: Pull complete
```

```
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
Digest: sha256:8d31dad0c58f552e890d68bbfb735588b6b820a46e459672d96e585871acc110
Status: Downloaded newer image for ubuntu:latest
```

Après le téléchargement d'une image, vous pouvez ensuite exécuter un conteneur en utilisant l'image téléchargée avec la sous-commande run. Comme vous l'avez vu avec l'exemple hello-world, si une image n'a pas été téléchargée lorsque docker est exécuté avec la sous-commande run, le client Docker télécharge d'abord l'image, puis exécute un conteneur en l'utilisant.

Pour voir les images téléchargées sur votre ordinateur, tapez:

```
docker images
```

La sortie de données devrait ressembler à ceci:

```
REPOSITORY                                    TAG                 IMAGE ID
CREATED             SIZE
<none>                                        <none>              d0afa2715b49        2 days
ago          459MB
registry.gitlab.com/roger.varnier/laravel-cicd   latest              1cd056ef3e24        2 days
ago          529MB
php                                           7.2                 9c91b6575468        2
weeks ago          398MB
ubuntu                                        latest              ccc6e87d482b        4
weeks ago          64.2MB
php                                           7.1                 060b5298208c        2
months ago         389MB
hello-world                                   latest              fce289e99eb9        13
months ago         1.84kB
```

2.4. Exécuter un conteneur Docker

Le conteneur hello-world que vous avez exécuté à l'étape précédente est un exemple de conteneur qui s'exécute et se ferme après avoir émis un message de test. Les conteneurs peuvent être beaucoup plus utiles que cela, et ils peuvent être interactifs. Après tout, ils ressemblent aux machines virtuelles, mais ils sont plus conviviaux.

Par exemple, exécutons un conteneur en utilisant la dernière image d'Ubuntu. La combinaison des commutateurs -i et -t vous donne un accès interactif au shell dans le conteneur:

```
docker run -it ubuntu
```

Votre invite de commande devrait changer pour refléter le fait que vous travaillez maintenant dans le conteneur et devrait prendre la forme suivante:

```
root@62681cdc60b2:/#
```

Notez l'ID de conteneur dans l'invite de commande. Dans cet exemple, il s'agit de d9b100f2f636. Vous aurez besoin de cet ID de conteneur plus tard pour identifier le conteneur lorsque vous souhaitez le supprimer.

Vous pouvez maintenant exécuter n'importe quelle commande dans le conteneur. Par exemple, mettons à jour la base de données de paquets à l'intérieur du conteneur. Vous n'avez pas besoin de préfixer n'importe quelle commande avec sudo car vous opérez dans le conteneur en tant qu'utilisateur root:

```
root@62681cdc60b2:/# apt update
```

Ensuite, installez n'importe quelle application. Installons par exemple Composer

```
root@62681cdc60b2:/# apt install composer
```

Cela installe Composer dans le conteneur à partir du dépôt officiel Ubuntu. Lorsque l'installation est terminée, vérifiez que Composer est installé.

```
root@62681cdc60b2:/# composer -V
```

Vous devez avoir un résultat similaire à celui-ci :

```
Composer 1.6.3 2018-01-31 16:28:17
```

/!\ Attention toutes les modifications que vous apportez à l'intérieur du conteneur s'appliquent uniquement à celui-ci.

Pour quitter le conteneur, tapez exit.

2.5. Gestion des conteneurs Docker

Après avoir utilisé Docker pendant un moment, vous aurez de nombreux conteneurs actifs (en cours d'exécution) et inactifs sur votre ordinateur. Pour voir les active ones (conteneurs actifs), utilisez:

```
docker ps
```

Vous verrez une sortie de données semblable à celle-ci:

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
```

Dans ce TP vous avez démarré deux conteneurs. Un basé sur l'image hello-world et un autre basé sur l'image ubuntu. Les deux conteneurs ne sont plus en cours d'exécution, mais ils existent toujours sur votre système.

Pour afficher tous les conteneurs — actifs et inactifs, exécutez docker ps avec le commutateur -a:

```
docker ps -a
```

Vous devez avoir un résultat similaire à celui-ci :

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
62681cdc60b2        ubuntu              "/bin/bash"              9 minutes ago       Exited (0)
About a minute ago                       hopeful_kalam
9940660803a1        hello-world         "/hello"                 20 minutes ago      Exited (0)
20 minutes ago                           wonderful_torvalds
```

Pour afficher le dernier conteneur que vous avez créé, transmettez-le au commutateur -l:

```
docker ps -l
```

Vous devez avoir un résultat similaire à celui-ci :

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
62681cdc60b2        ubuntu              "/bin/bash"         10 minutes ago      Exited (0) 2
minutes ago                      hopeful_kalam
```

Pour démarrer un conteneur arrêté, utilisez docker start, suivi de l'ID du conteneur ou du nom du conteneur. Démarrons le conteneur basé sur Ubuntu avec l'ID de 62681cdc60b2

```
sudo docker start 62681cdc60b2
```

Pour arrêter un conteneur en cours d'exécution, utilisez docker stop, suivi de l'ID ou du nom du conteneur. Cette fois-ci, nous utiliserons le nom que Docker a attribué au conteneur, qui est hopeful_kalam

```
sudo docker stop hopeful_kalam
```

Une fois que vous avez décidé que vous n'avez plus besoin d'un conteneur, supprimez-le à l'aide de la commande docker rm en utilisant à nouveau l'ID du conteneur ou son nom. Utilisez la commande docker ps -a pour trouver l'ID ou le nom du conteneur associé à l'image hello-world et supprimez-le.

```
docker rm wonderful_torvalds
```

Vous pouvez démarrer un nouveau conteneur et lui donner un nom en utilisant le commutateur --name. Vous pouvez également utiliser le commutateur --rm pour créer un conteneur qui se supprime tout seul lorsqu'il est arrêté. Voir la commande docker run help pour plus d'informations sur ces options ainsi que d'autres.

Les conteneurs peuvent être transformés en images que vous pouvez utiliser pour créer de nouveaux conteneurs.

2.6. Valider des changements dans un conteneur à une image Docker

Lorsque vous démarrez une image Docker, vous pouvez créer, modifier et supprimer des fichiers. Les modifications que vous apportez ne s'appliqueront qu'à ce conteneur. Vous pouvez le démarrer et l'arrêter, mais une fois que vous l'avez détruit avec la commande docker rm, les modifications seront définitivement perdues.

Cette section explique comment enregistrer l'état d'un conteneur en tant que nouvelle image Docker.

Après avoir installé Composer dans le conteneur Ubuntu, vous disposez maintenant d'un conteneur exécutant une image, mais le conteneur est différent de l'image que vous avez utilisée pour la créer. Cependant vous voudrez peut-être réutiliser ce conteneur Composer comme base pour de nouvelles images plus tard.

Validez les modifications dans une nouvelle instance d'image Docker à l'aide de la commande suivante.

```
docker container commit -m "What you did to the image" -a "Author Name" container_id
repository/new_image_name
```

Le commutateur -m est destiné au message de validation qui vous aide, vous et les autres, à savoir les modifications que vous avez apportées, tandis que -a est utilisé pour spécifier l'auteur. Le container_id (ID du conteneur) est celui que vous avez noté précédemment dans le tutoriel lorsque vous avez démarré la session interactive de Docker. Sauf si vous avez créé des référentiels supplémentaires sur Docker Hub, le repository est généralement votre nom d'utilisateur Docker Hub.

Par exemple, pour l'utilisateur rogervar, avec l'ID de conteneur 62681cdc60b2, la commande serait:

```
docker container commit -m "Ajout Composer" -a "rogervar" 62681cdc60b2 rogervar/ubuntu-composer
```

Lorsque vous validez un conteneur, la nouvelle image est enregistrée localement sur votre ordinateur.

En répertoriant à nouveau les images Docker, vous verrez apparaître la nouvelle image, ainsi que l'ancienne dont elle est issue:

```
docker images
```

Vous devez avoir un résultat similaire à celui-ci :

```
REPOSITORY                                      TAG                 IMAGE ID
CREATED              SIZE
rogervar/ubuntu-composer                        latest              b85bacc2031a        28
minutes ago      243MB
<none>                                          <none>              d0afa2715b49        2 days
ago          459MB
registry.gitlab.com/roger.varnier/laravel-cicd  latest              1cd056ef3e24        2 days
ago          529MB
php                                             7.2                 9c91b6575468        2
weeks ago          398MB
```

```
ubuntu                                  latest              ccc6e87d482b        4
weeks ago           64.2MB
php                                     7.1                 060b5298208c        2
months ago          389MB
hello-world                             latest              fce289e99eb9        13
months ago          1.84kB
```

Dans cet exemple, ubuntu-composer est la nouvelle image, dérivée de l'image ubuntu existante de Docker Hub. La différence de taille reflète les modifications apportées et, dans cet exemple, le changement est l'ajout de Composer. Ainsi, la prochaine fois que vous devrez exécuter un conteneur en utilisant Ubuntu avec Composer pré-installé, vous pourrez simplement utiliser cette nouvelle image.

Vous pouvez également créer des images à partir d'un Dockerfile, ce qui vous permet d'automatiser l'installation de logiciels dans une nouvelle image. Ce que nous verrons dans un autre TP.

2.7. Transmettre des images Docker vers un référentiel Docker

La prochaine étape logique après la création d'une nouvelle image à partir d'une image existante consiste à la partager avec quelques amis, le monde entier sur Docker Hub ou un autre registre Docker auquel vous avez accès. Pour envoyer une image vers Docker Hub ou tout autre registre Docker, vous devez avoir un compte là-bas.

Pour transmettre votre image, connectez-vous d'abord à Docker Hub.

```
docker login -u docker-registry-username
```

Vous serez invité à vous authentifier à l'aide de votre mot de passe Docker Hub. Si vous avez spécifié le bon mot mot de passe, l'authentification devrait réussir.

Ensuite, vous pouvez transmettre votre propre image en utilisant:

```
docker push docker-registry-username/docker-image-name
```

Pour transmettre l'image ubuntu-composer au dépôt rogervar, la commande serait la suivante:

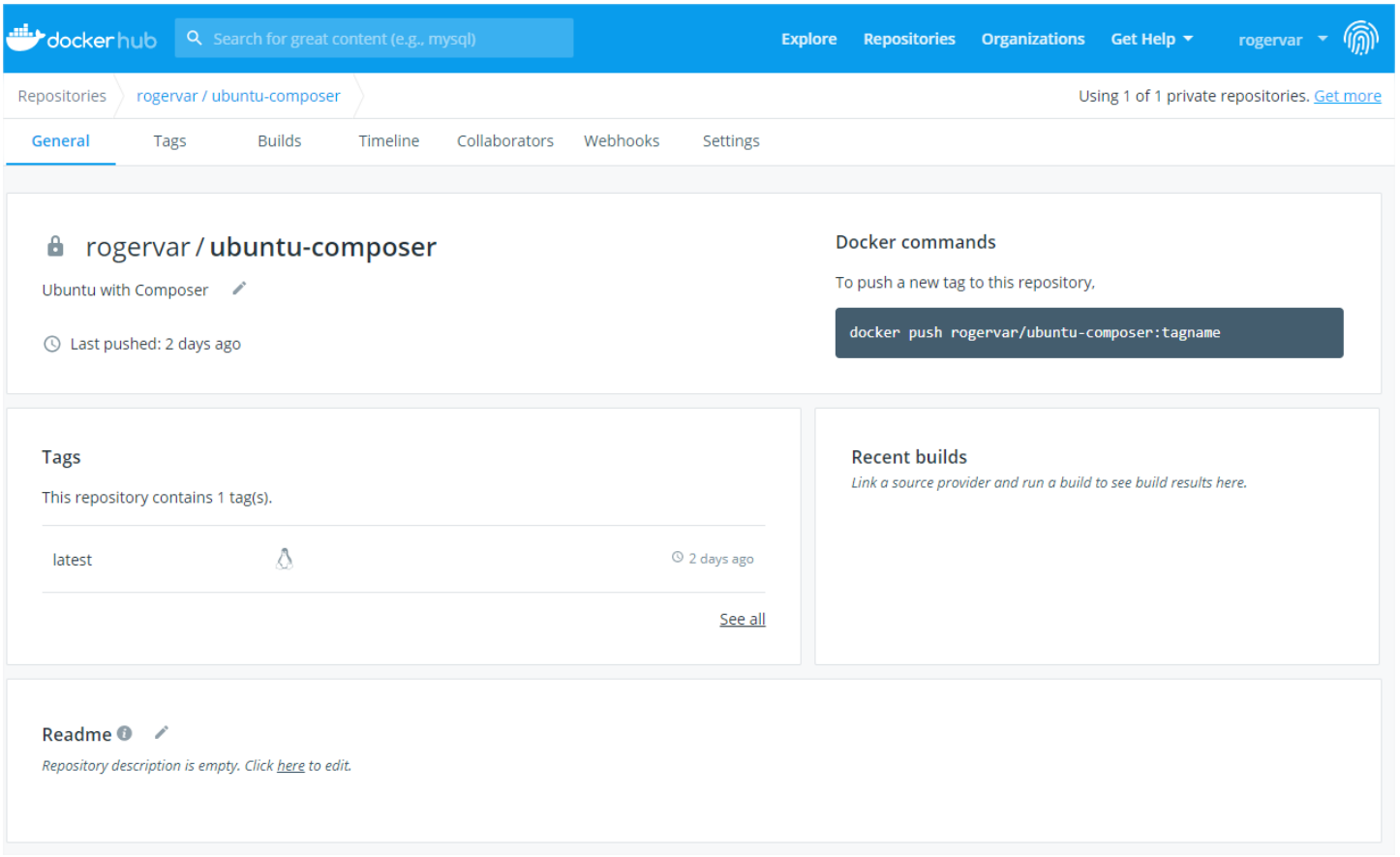
```
docker push rogervar/ubuntu-composer
```

Le processus peut prendre un certain temps, mais une fois terminé, le résultat ressemblera à ceci:

```
The push refers to repository [docker.io/rogervar/ubuntu-composer]
a3f6ff31d8ae: Pushed
f55aa0bd26b8: Mounted from library/ubuntu
1d0dfb259f6a: Mounted from library/ubuntu
21ec61b65b20: Mounted from library/ubuntu
43c67172d1d1: Mounted from library/ubuntu
latest: digest: sha256:1c3ca5caf9880ec3bd4fa1e411fd5b7396a2c534e5416147d6c62d8bb000a49e size:
```

```
1364
```

Après avoir transmis une image dans un registre, celle-ci doit être répertoriée dans le tableau de bord de votre compte, comme indiqué dans l'image ci-dessous.



Si une tentative de transmission entraîne une erreur de ce type, vous ne vous êtes probablement pas connecté:

```
The push refers to a repository [docker.io/rogervar/ubuntu-composer]
e3fbbfb44187: Preparing
5f70bf18a086: Preparing
a3b5c80a4eba: Preparing
7f18b442972b: Preparing
3ce512daaf78: Preparing
7aae4540b42d: Waiting
unauthorized: authentication required
```

Connectez-vous avec login docker et répétez la tentative de transmission. Ensuite, vérifiez que l'image existe sur sur votre dépôt Docker Hub.

Vous pouvez maintenant utiliser docker pull rogervar/ubuntu-composer pour extraire l'image et l'envoyer sur une nouvelle machine et l'utiliser pour exécuter un nouveau conteneur.

3. Création de notre première image Docker

Intéressons maintenant à la création de notre première image Docker via un fichier Dockerfile. Dans votre répertoire /home/user créer un nouveau répertoire nommé DockerImages. Puis créez dans celui-ci un fichier nommé Dockerfile contenant le code suivant :

```
FROM alpine:3.11 #Défini la distribution Linux sur laquelle l'image va se créer
MAINTAINER test #Auteur de l'image, vous en sommes
```

Ensuite, construisons notre image via la commande suivante :

```
sudo docker build .
```

Docker va télécharger puis exécuter les étapes définies dans le fichier Dockerfile. Si tout se passe bien, le script se termine sur une phrase similaire à :

```
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM alpine:3.11
3.11: Pulling from library/alpine
c9b1b535fdd9: Pull complete
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for alpine:3.11
 ---> e7d92cdc71fe
Step 2/2 : MAINTAINER Roger
 ---> Running in a9516a489c37
Removing intermediate container a9516a489c37
 ---> 3f540abcb7c8
Successfully built 3f540abcb7c8
```

Ajoutons maintenant l'installation d'Apache2 en mettant à jour la liste des paquets et en installant Apache2. Pour cela ajouter les lignes suivantes au fichier Dockerfile :

```
RUN apk update
RUN apk add apache2
RUN apk add php7-apache2
RUN apk add openrc --no-cache
```

La commande RUN nous permet de lancer des commandes lors de la création d'une image et le caractère "\" permet d'écrire la commande sur plusieurs lignes.

```
Exemple alternatif :
RUN apk update && \
apk add apache2 && \
apk add php7-apache2 && \
```

```
apk add openrc --no-cache
```

Reconstruisons notre image suite à cette modification et donnons lui un nom et un tag :

```
sudo docker build -t alpine-apache:8000 .
```

A l'issue du traitement de la commande, vous devez avoir un résultat similaire à celui-ci :

```
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM alpine:3.11
 ---> e7d92cdc71fe
Step 2/3 : MAINTAINER Roger
 ---> Using cache
 ---> 3f540abcb7c8
Step 3/3 : RUN apk update && apk add apache2
 ---> Running in 4d9583e87f07
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/community/x86_64/APKINDEX.tar.gz
v3.11.3-73-g095aa9b9d4 [http://dl-cdn.alpinelinux.org/alpine/v3.11/main]
v3.11.3-72-g77ec45c4dc [http://dl-cdn.alpinelinux.org/alpine/v3.11/community]
OK: 11262 distinct packages available
(1/6) Installing libuuid (2.34-r1)
(2/6) Installing apr (1.7.0-r0)
(3/6) Installing expat (2.2.9-r1)
(4/6) Installing apr-util (1.6.1-r6)
(5/6) Installing pcre (8.43-r0)
(6/6) Installing apache2 (2.4.41-r0)
Executing apache2-2.4.41-r0.pre-install
Executing busybox-1.31.1-r9.trigger
OK: 9 MiB in 20 packages
Removing intermediate container 4d9583e87f07
 ---> f4f65e40ed01
Successfully built f4f65e40ed01
Successfully tagged alpine-apache:test
```

Si l'on affiche la liste des images docker présentes sur notre serveur via la commande docker images, on constate bien que la dernière version de l'image est créée :

```
roger@ubuntu-preprod:~/dockerimages$ docker images
REPOSITORY                                  TAG                 IMAGE ID
CREATED             SIZE
alpine-apache                               8000                f4f65e40ed01        54
seconds ago      41.2MB
```

Au fur et à mesure que l'on manipule les images Docker il se peut qu'on ai besoin de créer une image sans utiliser le cache. Pour cela utiliser l'option --no-cache comme ceci :

```
docker build --no-cache .
```

Lançons maintenant un conteneur avec Apache2. Pour cela il faut utiliser la commande suivante :

```
docker run -d -p 8000:80 alpine-apache:8000 /usr/sbin/httpd -D FOREGROUND
```

A quoi servent ces paramètres ?

- -d : permet de lancer le conteneur en mode détaché, ce qui libère votre terminal. Plutôt pratique sur un serveur uniquement en ligne de commande.
- -p : permet de définir le port à appeler pour qu'il atteigne le port 80 sur le conteneur. Pratique car nous pouvons à terme avoir un conteneur par port par exemple.
- -D FOREGROUND : permet de lancer le processus dans le conteneur et d'attacher la console au processus d'entrée classique.

Normalement, vous pouvez accéder à la page par défaut dans un navigateur en tapant http://adresse_ip_de_votre_serveur:8000.

Vous pouvez à présent vérifier que le conteneur est bien lancé avec la commande docker ps.

https://wiki.alpinelinux.org/wiki/Tutorials_and_Howtos#HTTP

4. Création d'une seconde image Docker

4.1. Insérer des fichiers sans son conteneur

Maintenant que nous avons vu comment créer une image, nous allons en réaliser une seconde en intégrant des fichiers de configurations issus de notre environnement.

De manière générale, on pourrait créer l'ensemble de la configuration d'une image Docker dans le fichier Dockerfile correspond. Cependant, il existe une méthode plus simple et à privilégier.

Dans notre répertoire DockerImages, créer un nouveau répertoire nommé docker où nous allons créer deux fichiers. Le premier sera nommé host-apache2.conf et contient les informations suivantes :

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/localhost/htdocs/test
</VirtualHost>
```

Le second est un fichier index.html qui contient le texte ci-dessous :

```
Docker, suis-je à bon port ?
```

Ensuite nous allons éditer notre fichier Dockerfile afin de copier les deux fichiers créés précédemment dans la nouvelle image.

```
FROM alpine:3.11
MAINTAINER Roger
RUN apk update && \
apk add apache2 && \
apk add php7-apache2 && \
apk add openrc --no-cache
ADD docker/host-apache2.conf /etc/apache2/conf.d/vhost.conf
ADD docker/index.html /var/www/localhost/htdocs/test/index.html
```

Construisez ensuite votre image via la commande :

```
sudo docker build -t alpine-apache:8001 .
```

Puis créer le conteneur via la commande :

```
docker run -d -p 8001:80 alpine-apache:8001 /usr/sbin/httpd -D FOREGROUND
```

En fonction des conteneurs qui tournent sur votre VM, vous constaterez que le port 8000 est toujours accessible si vous n'avez pas couper le conteneur de l'exercice précédent.

4.2. Lancer un script au lancement du conteneur

Le fichier Dockerfile permet de lancer des scripts lors du lancement du conteneur comme on le fait lors de la commande docker run…

Pour lancer un script nous allons ajouter la ligne suivante à la fin du fichier Dockerfile :

```
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

Votre fichier Dockerfile doit ressembler à celui-ci :

```
FROM alpine:3.11
MAINTAINER Roger
RUN apk update && \
apk add apache2 && \
apk add php7-apache2 && \
apk add openrc --no-cache
ADD docker/host-apache2.conf /etc/apache2/conf.d/vhost.conf
ADD docker/index.html /var/www/localhost/htdocs/test/index.html
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

En construisant une nouvelle image et en lançant un nouveau conteneur sur le port 8002, vous devez toujours avoir la page qui s'affiche. Cependant le lancement d'Apache est réalisé directement dans le Dockerfile.

```
sudo docker build -t alpine-apache:test3 .
sudo docker run -d -p 8002:80 alpine-apache:test3
```

Il existe d'autres fonctionnalités utiles dans Dockerfile, comme :

- WORKDIR [path] déplacer le curseur dans un répertoire
- ENV [nom] [valeur] permet de créer des variables d'environnement que vous pouvez récupérer via la variable ${nom}
- VOLUME [path] permet de créer un point de montage comment entre l'hôte et le conteneur
- USER [nom] permet d'indiquer l'utilisateur à utiliser

N'hésitez pas à consulter la documentation officielle pour en savoir plus et découvrir d'autres fonctionnalités moins utilisées.

La commande docker run revient à lancer la commande docker create + docker start !

5. Allez plus loin avec Docker

5.1. Exercice complémentaire

A partir de l'image Alpine, construisez une nouvelle image et un conteneur permettant d'héberger un projet Symfony et rendre ce projet accessible via un navigateur.

5.2. Liens utiles

Lien vers la documentation officielle : https://docs.docker.com/engine/reference/builder/

Lien vers la documentation d'Alpine : https://wiki.alpinelinux.org/wiki/Main_Page

Exemple complexe de Dockerfile :

https://git.ademe.fr/docker/httpd/blob/63612be79b45636cbb19c940bbf2b36ae872955d/alpine/Dockerfile