**Docker**



# Introduction

**Docker** is a tool used to create, deploy, and run applications by using containers

Docker est une plateforme de conteneurisation, utilisé pour déployer une application sur une machine, ou bien plusieurs sans qu’il y ait un souci de dépendances.

Il est plus léger qu'une machine virtuelle n’ayant pas une couche de virtualisation du matériel

Il faut voir Docker, en gros, comme un outil pour rendre portable votre application, sans les contraintes d’une machine virtuelle.

Docker est une plateforme qui va vous permettre d'exécuter votre code à l'intérieur d'un conteneur indépendamment de la machine sur laquelle vous êtes ! Un conteneur ressemble à une machine virtuelle sauf qu'il n'embarque pas tout un système d'exploitation avec lui ce qui lui permet de s'exécuter en quelque secondes et d'être beaucoup plus léger.

# Installation and Configuration

On installe Docker en fonction de la version de notre Linux

Pour résumer il faut :

1. Update
2. Installer les prérequis
3. Ajout de la clé GPG
4. Ajout du repository Docker aux sources APT
5. Faire un Update
6. Installer la dernière version de Docker

1ère commande : **docker images**

Afin de voir les images docker présentes.

# The Docker Hub

Docker Hub is the world's largest library and community for container images.

Users get access to free public repositories for storing and sharing images or can choose subscription plan for private repos.

Dans le docker hub puis Explore tu as un listing des images officiels, pour en récupérer une tu tapes par exemple la commande : **docker pull** debian

# Base Images

Here we will pull down a couple of base images and show how we can get some information on what they are and how they were built.

We need an image to run a container

We can do a search from the terminal with the command: docker search, ex:

**docker search** centos

**docker pull** hello-world

When you type docker run hello-world:latest, docker will run a container whose only goal is to display the message : “ hello world”.

An image has a name and an ID



So we can type either: docker run hello-world or docker run fce289e99eb9

Afin d’avoir plus d’infos sur une image on tape par exemple: **docker inspect** debian

# Running containers

**docker ps** : To know which container is running

**docker ps -a** : To know which container had been started

En tapant ifconfig tu auras une carte docker0 avec une adresse IP : 172.17.0.1

Run a container connecting to the current terminal:

**docker run -it centos /bin/bash**

Then, you are not working and your host OS but on your container OS as a root.

You can type a **yum upgrade**

**docker run -d centos /bin/bash**

Là ça va lancer le conteneur en background mais le fermer aussitôt.

**docker run -it –rm ubuntu**

Le paramètre ‘-t’ permet d’avoir un pseudo-terminal (pour exécuter des commandes dans le conteneur une fois lancé). Le paramètre ‘-i’ permet d’activer le mode interactif, qui redirige tous les messages sur les sorties standards (permet de voir les logs). Le paramètre ‘–rm’ permet de supprimer automatiquement le conteneur à la fin de l’exécution.

Quand tu démarres un nginx, pas la peine de mettre un /bin/bash car il démarre automatiquement une commande :

**docker run -d nginx**

Avec un docker ps tu verras ton conteneur avec nginx démarré.

Avec docker inspect NomContainer tu pourras voir l’ip du conteneur et ensuite tu peux pinguer dessus.

**eLinks**

eLinks est un navigateur web en mode texte très léger et rapide avec de nombreuses fonctionnalités.

Basé sur Links, il supporte la navigation par onglets et avec la souris, les marque-pages, les cookies et les pages en cache, les menus déroulants et les champs de texte.

Pour l’installer : apt install elinks

Tu peux voir que nginx est bien installée en tapant :sudo tel

elinks <http://IPConteneur> ex : elinks <http://172.17.0.2>

**Telnet** est un protocole conçu pour accéder aux ordinateurs à distance, en mode client-serveur. Il est moins sûr que SSH car non crypté. Son port par défaut est 23 mais on peut le spécifier en paramètre.

Tester si le port http est ouvert et que son service écoute :

telnet localhost 80

Lorsque l’on veut tester une connexion sur un port d’un hôte distant, le premier réflexe du SysAdmin est de faire un “telnet” dessus.

Telnet, pour Terminal Network ou Telecommunication Network est un protocole de communication TCP/IP permettant de communiquer de façon bi-directionnelle avec un serveur distant. Son problème, c’est qu’il n’est absolument pas sécurisé, toutes les informations y transitent en clair. Du coup des protocoles chiffrés comme SSH ont vu le jour pour remplacer TelNet. Il n’est donc pas rare que TelNet ne soit pas installé sur des serveurs d’entreprises.

Telnet is an old network protocol that is used to connect to remote systems over a TCP/IP network. It connects to servers and network equipment over port 23. Let’s take a look at Telnet command usage.

1. Telnet is not a secure protocol and is thus **NOT RECOMMENDED!**. This is because data sent over the protocol is unencrypted and can be intercepted by hackers.
2. Instead of using telnet, a more preferred protocol to use is **SSH** which is encrypted and more secure

## Using telnet to check for open ports

Telnet can also be used to check if a specific port is open on a server. To do so, use the syntax below.

$ telnet server-IP port

For example, to check if port 22 is open on a server, run

$ telnet 38.76.11.19 22

**Stop a container**

docker stop ContainerName

Attribuer un nom en lançant un container :

docker run -d --name=Webserver nginx

# The Container Lifecycle

We need to understand the lifecycle of our containers, so we talk about how to start, stop, restart and display their state.

First docker run …

We can start or restart a container by type: docker start ID/NAME or docker restart ID/NAME

For execute something in an executive container or enter in, we use : exec

Ex: docker exec -it NameOfContainer /bin/bash

We can stop by docker stop NameContainer/ID

# Lecture: Image and Container Management

We know how to pull/install images and instantiate containers. Removing them is equally important but there are some caveats, let's look at what we need to know!

# Redirection – Ports and Volumes

Pour supprimer une image on tape : docker rm NomImage

We have the ability to redirect container ports and volumes to our host operating system for ease of use and flexibility.

Direct the port that is listening for http on our container to a port on our host

docker run –d --name=Webserver nginx

docker inspect Webserver |grep IPAddr

docker inspect Webserver | grep IPAddr

"SecondaryIPAddresses": null,

"IPAddress": "172.17.0.2",

"IPAddress": "172.17.0.2"

docker run –d --name=Webserver -P nginx

-P make any ports exposed to my container available through the host OS

## Published ports

By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container’s network, use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host. Here are some examples.

|  |  |
| --- | --- |
| -p 8080:80 | Map TCP port 80 in the container to port 8080 on the Docker host. |

Avec un petit p tu peux spécifier le numéro de port que tu desires, ex:

docker run –d –p 8080 :80 --name=Web nginx

### Les ports

#### Entre l'host et les containers

De la même façon que le reste, les ports réseaux à l'intérieur d'un container sont isolés du reste du système. C'est une notion importante car, on va devoir faire de la redirection de port. Lorsqu'un service expose un port à l'intérieur du container, il n'est, par conséquent, pas accessible depuis l'extérieur par défaut.

Nous avons besoin d'exposer un port du host sur un port interne à un container. Pour être plus clair, je vais prendre un exemple.

Le port par défaut d'un serveur Rails est 3000. Pour pouvoir accéder à ce port depuis mon host, je dois rediriger le port 3000 interne du container vers un port de mon choix sur l'host. Il n'y a aucune restriction à ce niveau-là.

**Mappage de ports (NAT)**

Comment accéder de l’extérieur de l’hôte au serveur web du conteneur ?

Il faut faire un mappage c’est-à-dire associer le port 8080 de l’hôte au port 80 du conteneur lors de la création :

docker run -dp 8080 :80 --name web nginx

Cette commande créé le conteneur en lui attribuant le nom web avec l’image nginx tout en faisant un nat entre le port 8080 de l’hôte et le port 80 du conteneur.

Si ensuite tu vas sur IpHôte :8080 tu devrais accéder au serveur web de ton conteneur.

La page index par défaut dans nginx se trouve dans : usr/share/nginx/html/

Si tu veux accéder à plusieurs conteneurs il faudra bien choisir un port différent pour chaque conteneur :

docker run -dp 8000 :80 --name web2 nginx

docker run -dp 8001 :80 --name web3 nginx

**Gestion des volumes de données**

On va créer un répertoire sur l’hôte dans lequel il y aura des données, puis on va faire une liaison de ce répertoire au répertoire du conteneur.

Si ensuite tu recréé ton conteneur, tu auras toujours accès à tes données, c’est du mappage de répertoire.

docker run –d –name web –v /web:/usr/share/nginx/html nginx

docker run –d –p 8080:80 –name=Webserver2 –v /mnt/data nginx ( à mieux comprendre)

Pour supprimer l’ensemble des conteneurs crées :

docker rm `docker ps -a -q`

# The Dockerfile

Le Dockerfile est un fichier qui contient toutes les instructions pour créer une image, comme des métadonnées (Mainteneur, label, etc.), ou même les commandes à exécuter pour installer un logiciel.

Tu vas donc créer ce fichier avec vim par exemple vim Dockerfile

Syntaxe :

FROM debian:stable

MAINTAINER qeyss [<qeyss@gmail.com](mailto:%3cqeyss@gmail.com)>

RUN apt-get update

RUN apt-get upgrade

Ensuite pour construire l’image :

docker build –t qeyss/myapache .

Le point final indique que c’est le répertoire courant.

Il est préférable de réduire les layers, car à chaque ligne de RUN un container sera construit

FROM debian:stable

MAINTAINER qeyss [<qeyss@gmail.com](mailto:%3cqeyss@gmail.com)>

RUN apt-get update && apt-get upgrade –y && apt-get install –y apache2 telnet elinks openssh-server

(ne fonctionne pas chez moi à cause du –y)

Dans le dockerfile tu peux avoir des RUN, mais aussi des CMD. ET aussi des EXPOSE, ENV, etc..

Les RUN sont effectués lors de la création de l’image de base, tandis que les CMD sont exécutés dans le conteneur qui sera créé à partir de l’image.

FROM centos

MAINTAINER qeyss [qeyss@gmail.com](mailto:qeyss@gmail.com)

RUN yum install -y httpd net-tools

RUN echo ''<h1> Welcome to my website </h1>" > /usr/share/httpd/noindex/index.html

EXPOSE 80

CMD ['' -D'', ''FOREGROUND'']

ENTRYPOINT [''/usr/sbin/httpd'']

Ensuite pour construire l’image :

docker build –tag=NomAdonner . 🡪 le . indique que c’est le répertoire courant.

Lors de la construction de l’image, tous les STEP sont des conteneurs

A chaque fois qu’un conteneur est créé une IP lui est attribuée et d NAT est mis en place, ce qui permet du conteneur de pouvoir pinguer l’extérieur.

Pour se connecter au conteneur on va sur l’ip de l’hôte :NuméroDePort.

Par défaut c’est en bridge.

La commande docker network + ses options permet d’avoir des infos sur le réseau et y faire des choses comme connecter le conteneur à un réseau, etc.

Docker machine

C’est un outil qui permet d’installer Docker Engine sur des hôtes virtuels ou distants

Installation de docker machine

base=https://github.com/docker/machine/releases/download/v0.16.0 &&

curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&

install /tmp/docker-machine /usr/local/bin/docker-machine

Check the installation by displaying the Machine version:

docker-machine version