BALBASRY Reda

TP/TD Docker et sécurité Docker

12/12/19

Jean-Marc Pouchoulon
Février 2019

1 Objectifs du TP.

— Comprendre l'architecture d'une solution de containérisation.
— Construire une image Docker.
— Manipuler images et containers Docker.
— Mettre en réseau un container Docker de différentes façons.
— Sécuriser un container.
— Clusteriser des containers.

2 Pré-requis, recommandations et notation du TP.

Les pré-requis sont les suivants :
— Avoir un PC sous Linux.
— Avoir installé Docker ou utilisé une OVA prête à l'emploi. Merci de ne pas utiliser les packages fournis par les distributions qui sont souvent moins récents que les packages fournis par Docker.
— Avoir Installé docker-compose, il est présent sur les ova de l'IUT.
— Vous devrez avoir un compte sur le site Docker https://hub.docker.com/.

Vous travaillerez individuellement. Il vous explicitement demandé de faire valider votre travail par l'enseignant. Ces "checks" permettront de vous noter. Un compte rendu succinct (fichiers de configuration , copie d'écran montrant la réussite de la construction ...) est demandé et à rendre sur Moodle Didex.

2.1 Installation de Docker et obtenir de l'aide.

2.1.1 Rappel : Installation de Docker sous Linux.
Vous travaillerez avec une VM en utilisant l'OVA Ubuntu en version LTS 18.04 présente sur
`http://store.iutbeziers.fr/`

2.1.2 Aide sur Docker.
man docker-run
man docker-create

Accéder à la Documentation Docker :
`https://docs.docker.com/`

Documentation sur les commandes Docker :
`https://docs.docker.com/engine/reference/commandline/`

La complétion avec la touche tab fonctionne aussi.

3 Docker sous Linux.

3.1 Installation de Docker sous Linux

L'installation est faite sur la machine virtuelle mais vous pouvez retrouver la procédure d'installation sous `https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/`
1. Quelle la version de Docker installée ? Retrouvez des informations sur le daemon docker.

```
root@ubuntu:~# docker --version
Docker version 18.06.0-ce, build 0ffa825
```

2. Vérifiez que votre installation fonctionne bien avec la commande : docker run hello-world

```
root@ubuntu:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:4fe721ccc2e8dc7362278a29dc660d833570ec2682f4e4194f4ee23e415e1064
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.

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/
```

a) Que vous explique le retour de cette commande (au delà de "tout s'est bien passé" reformulez en Français..) ?

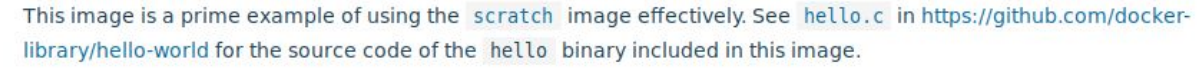
Cette commande retourne le processus détaillé de la commande : docker run hello-world

b) Retrouvez sur https://hub.docker.com/ l'image hello-world.

c) Expliquez les mécanismes en jeux pour la création du container helloworld. Quel est le fichier sur DockerHub qui permet de créer le container ?
Alors :

1 - le client docker à contacter le démon docker
2 - le démon docker a pull l'image hello world du docker hub
3 - le démon docker a créé un nouveau conteneur via l'image qui fait tourner l'exécutable qui permet la sortie du fichier que l'on lit actuellement.
4 -le démon docker a envoyer le flux de la sortie au client docker qui nous l'envoie sur notre terminal.

le fichier est hello.c :



3. Recherchez les images officielles Debian à l'aide de docker search. Récupérez-les ainsi que les images officielles busybox (une distribution légère).

```
root@ubuntu:~# docker search debian
NAME          DESCRIPTION                 STARS               OFFICIAL
ubuntuUbuntu is a Debian-based Linux      10255                [OK]
debian        Debian is a Linux            3297               [OK]
```

```
NAME                      DESCRIPTION            STARS             OFFICIAL
busybox                   Busybox base image.     1742             [OK]
```

4. Créez votre premier container à partir de l'image Debian officielle et en utilisant la commande "docker run -d" sans argument.

```
root@ubuntu:~# docker run -d debian
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
16ea0e8c8879: Already exists
Digest: sha256:79f0b1682af1a6a29ff63182c8103027f4de98b22d8fb50040e9c4bb13e3de78
Status: Downloaded newer image for debian:latest
fa4a155c44e0a3232d7ab8c5d82cffeaf95b88ac21cc87b02a06ba68cbe9d954
```

5. En utilisant la commande "docker ps" vérifiez que le container est "vivant" ? expliquez ?

```
root@ubuntu:~# docker ps
CONTAINER ID          IMAGE                COMMAND              CREATED
STATUS                PORTS                NAMES
```

```
root@ubuntu:~# docker ps -l
CONTAINER ID    IMAGE    COMMAND     CREATED     STATUS         PORTS          NAMES
2f057b2664d7          debian          "bash"       21 seconds ago  Exited (0) 20 seconds
ago                          nervous_minsky
```

le conteneur n'est pas vivant, etat = EXITED

6. Relancez le "docker run" en lui donnant comme argument "trap : TERM INT ; sleep infinity & wait".
root@ubuntu:~# docker run -d debian trap : TERM INT ; sleep infinity & wait
c36263af0af93d2161adb50ee81821f7d33d6375f117a67aaa3239f0d4b7ed7e

```
root@ubuntu:~# docker ps -l
CONTAINER ID    IMAGE     COMMAND             CREATED        STATUS     PORTS     NAMES
6175cec27b57 debian "sh -c 'trap : TERM …" 7 seconds ago  Up 5 seconds
romantic_bohr
```

7. Stoppez et redémarrez le container.

```
CONTAINER ID   IMAGE    COMMAND                      CREATED     STATUS  PORTS    NAMES
6175cec27b57          debian"sh -c 'trap : TERM ..."   3 minutes ago  Up 3 minutes
romantic_bohr
```

```
root@ubuntu:~# docker restart 6175cec27b57
6175cec27b57
```

```
root@ubuntu:~# docker ps
CONTAINER ID   IMAGE    COMMAND                  CREATED            STATUS
NAMES
6175cec27b57          debian  "sh -c 'trap : TERM ..."   3 minutes ago    Up 9 seconds
romantic_bohr
```

8. Supprimez le container.

```
root@ubuntu:~# docker stop 6175cec27b57
```

```
6175cec27b57
root@ubuntu:~# docker rm 6175cec27b57
6175cec27b57
```

9. Utilisez les options "-it" afin d'être dans le container après son lancement.

```
root@ubuntu:~# docker run -it
root@8d2516952329:/#
```

10. Même opération mais nommant le container et son hostname DebianOne.

```
root@ubuntu:~# docker run -it --hostname DebianOne --name DebianOne debian
root@DebianOne:/#
```

11. Attachez-vous et détachez-vous du container DebianOne.

```
root@ubuntu:~# docker attach 16d34b77219e
You cannot attach to a stopped container, start it first
root@ubuntu:~# docker start 16d34b77219e
16d34b77219e
root@ubuntu:~# docker attach 16d34b77219e
root@DebianOne:/# read escape sequence
```

12. Lancez un processus bash supplémentaire dans le container DebianOne. Pour celà utilisez la commande docker exec.

```
root@ubuntu:~# docker start 16d34b77219e
16d34b77219e
root@ubuntu:~# docker exec 16d34b77219e ls
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

13. Listez le container nouvellement créé. Ne listez ensuite que le dernier ContainerId.

```
root@ubuntu:~# docker ps --help
```

```
Usage:    docker ps [OPTIONS]
List containers
Options:
  -a, --all           Show all containers (default shows just running)
  -f, --filter filter   Filter output based on conditions provided
  --format string   Pretty-print containers using a Go template
  -n, --last int      Show n last created containers (includes all states) (default -1)
  -l, --latest        Show the latest created container (includes all states)
  --no-trunc          Don't truncate output
   -q, --quiet        Only display numeric IDs
  -s, --size          Display total file sizes
root@ubuntu:~# docker ps -l
CONTAINER ID   IMAGE    COMMAND     CREATED                  STATUS
NAMES
16d34b77219e          debian         "bash"        8 minutes ago         Up About a
minute           DebianOne
```

```
root@ubuntu:~# docker ps -n 1
CONTAINER ID          IMAGE                COMMAND              CREATED
STATUS                PORTS                NAMES
16d34b77219e          debian               "bash"               11 minutes ago
Up 3 minutes                          DebianOne
```

14. Supprimez le container et son image.

```
root@ubuntu:~# docker stop 653d2f2276fe
653d2f2276fe
root@ubuntu:~# docker rm 653d2f2276fe
653d2f2276fe
```

```
root@ubuntu:~# docker rmi -f 67e34c1c9477
Untagged: debian:latest
Untagged: debian@sha256:79f0b1682af1a6a29ff63182c8103027f4de98b22d8fb50040e9c4bb13e3de78
Deleted: sha256:67e34c1c9477023c0ce84c20ae5af961a6509f9952c2ebbf834c5ea0a286f2b8
```

15. Supprimez tous les containers avec un oneliner sous bash. Idem pour les images.

```
root@ubuntu:~# docker rm $(docker ps -a -q)
"docker rm" requires at least 1 argument.
See 'docker rm --help'.
Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]
Remove one or more containers
```

```
root@ubuntu:~# docker rmi $(docker ps -a -q)
"docker rmi" requires at least 1 argument.
See 'docker rmi --help'.
```

```
Usage:  docker rmi [OPTIONS] IMAGE [IMAGE...]
Remove one or more images
root@ubuntu:~# docker rmi $(docker images -a -q)
Untagged: registry.iutbeziers.fr/debianiut:latest
Untagged:registry.iutbeziers.fr/debianiut@sha256:c80179bbfac915215648a2a58a6ee7f76d8c281
fbf1d850b20e218949d96fb88
Deleted: sha256:e3794e4b525deb426bf17bf6c3f689ced36e5758acf1e159b1db4669fff19833
Deleted: sha256:4ba4cc786e63fbb5936524344721598fb5cb58d0c03e0125fce218dc7741604f
Deleted: sha256:04ece69e091082217c52b434f4aae40554535ae30dc46dbcd2ada49927a8753d
Deleted: sha256:b7dd831717f9c1cdc4ae1d2d1fa345d7f8ceac0ab9a589d0a0660fc9c83a5f86
Deleted: sha256:0c0ce25a6587b6800271bea2915394b9d85a6deb0a4774f9a3a1298f2c052203
Deleted: sha256:dbfe21d30f4145470ddbac0baf77fbce674f2d2843f9cc2d418cc716429460e0
Deleted: sha256:f2b4f0674ba3e6119088fe8a98c7921ed850c48d4d76e8caecd7f3d57721b4cb
Untagged: hello-world:latest
Untagged:
hello-world@sha256:4fe721ccc2e8dc7362278a29dc660d833570ec2682f4e4194f4ee23e415e1064
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
Deleted: sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3
Untagged:registry.iutbeziers.fr/debianiut@sha256:4aa64142212d9e6fd1c0478cd32e94e28c4b320
f329531e205c89686ca0da9e7
Deleted: sha256:820ba6f8ba287c0ec222f056d2a6600ad3362644151ef46dd1135dee609f610a
Deleted: sha256:0baf8357f568cca9acf05c28052a51f33acf3793cc3b1a59e35ec52f716cc280
Deleted: sha256:c84c9159d1952cf98c8d56e243e6fa3ebf680b0d5a87f7e4d66e96756d28959d
Deleted: sha256:7454f13225732436c9099fc7463eda7b75dc6322f53eb0c333231d0891529ebf
Deleted: sha256:fa13e0b28aaecd7abb06f490fe28eb412ae0939205c2f27b63ce65564f157b5b
Deleted: sha256:3b10514a95bec77489a57d6e2fbfddb7ddfdb643907470ce5de0f1b05c603706
```

16. Supprimez les images et les containers non utilisés avec la commande "docker system prune".

```
root@ubuntu:~# docker system prune
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
```

4 Création d'images Docker

Dans cette partie nous allons apprendre à créer une image Docker.

Récupérez les fichiers pour cet exercice via git :
git clone https://registry.iutbeziers.fr:5443/pouchou/tpdocker.git

4.1 Build d'une image Docker Debian

1. Construisez l'image "debian :vosinitiales" à partir du Dockerfile du repository et de la commande "docker build..."

Pour le Dockerfile:
vim tpdocker/Dockerfile
commenter ligne 12 et 13

docker build -f Dockerfile
-t debian:RB
exemple :

```
Step 14/14 : RUN mkdir -p /home/git/.ssh
 ---> Running in b99dbf6182ef
Removing intermediate container b99dbf6182ef
 ---> f2bc35bb915f
Successfully built f2bc35bb915f
Successfully tagged debian:RB
root@ubuntu:~#
```

2. Expliquez ce que font les différentes commandes "RUN, ENV, FROM" de ce Dockerfile

Les commandes RUN ENV ET FROM sont des instructions :
FROM : il spécifie l'image parent avec laquelle on construit.
ENV : elle spécifie la variable d'environnement
RUN : elle permet d'exécuter n'importe quelle commande sur l'image et afficher les résultats.

3. Quel est l'intérêt de faire tous les apt-get en une seule fois pour la taille de l'image Docker. (indice : voir AUFS et Docker).
L'intérêt que je vois ce serait d'avoir tout a jour sur une même version compatible et de co

4. A partir de l'image "debian :vosinitiales" générez une image "pingfour" qui permettra de lancer un container de type ping se limitant à 4 envois ICMP vers www.iutbeziers.fr par defaut. Vous utiliserez les commandes "ENTRYPOINT" et "CMD" dans le Dockerfile.

exemple de commande docker build -t debian:pingfour /root/tpdocker/

```
Step 15/16 : ENTRYPOINT ping -c 4 www.iutbeziers.fr
 ---> Running in 2d2462cbbf25
Removing intermediate container 2d2462cbbf25
 ---> f211ea8fcc11
Step 16/16 : CMD ping -c 4 www.iutbeziers.fr
 ---> Running in 1c51597748fb
Removing intermediate container 1c51597748fb
 ---> 482ee979db03
Successfully built 482ee979db03
Successfully tagged debian:pingfour
```

```
root@ubuntu:~/tpdocker# docker images
REPOSITORY     TAG       IMAGE ID           CREATED              SIZE
debian             pingfour       482ee979db03   35 seconds ago   288MB
```

```
root@ubuntu:~/tpdocker# docker run -it debian:pingfour
PING www.iutbeziers.fr (194.199.227.80) 56(84) bytes of data.
64 bytes from www.iutbeziers.fr (194.199.227.80): icmp_seq=1 ttl=62 time=1.06 ms
64 bytes from www.iutbeziers.fr (194.199.227.80): icmp_seq=2 ttl=62 time=0.938 ms
```

```
64 bytes from www.iutbeziers.fr (194.199.227.80): icmp_seq=3 ttl=62 time=0.956 ms
64 bytes from www.iutbeziers.fr (194.199.227.80): icmp_seq=4 ttl=62 time=1.27 ms

--- www.iutbeziers.fr ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.938/1.059/1.275/0.136 ms
root@ubuntu:~/tpdocker#
```

5. Lancez un container issu de cette image au travers de la commande "docker run –rm -it ...". A quoi sert le –rm ?
Le --rm sert a supprimer le conteneur quand on en a plus besoin.

6. Peux-t-on changer la destination du ping ? Le nombre de ping ?
Alors, j'ai du changer le contenu de dockerfile parce que ma commande initiale n'est pas adaptative :

```
Step 15/16 : ENTRYPOINT ["/bin/ping","-c","4"]
 ---> Running in ecc5fa87517c
Removing intermediate container ecc5fa87517c
 ---> b98e4dfb3a61
Step 16/16 : CMD ["www.iutbeziers.fr"]
 ---> Running in 86a1cba0f799
Removing intermediate container 86a1cba0f799
 ---> 366c8f903ecd
Successfully built 366c8f903ecd
Successfully tagged debian:pingfour2
```

le test :

```
root@ubuntu:~# docker run -it debian:pingfour2 -c8 google.fr
PING google.fr (172.217.171.195) 56(84) bytes of data.
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=1 ttl=53 time=6.00 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=2 ttl=53 time=6.03 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=3 ttl=53 time=6.01 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=4 ttl=53 time=5.53 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=5 ttl=53 time=6.01 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=6 ttl=53 time=6.02 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=7 ttl=53 time=5.85 ms
64 bytes from mrs09s06-in-f3.1e100.net (172.217.171.195): icmp_seq=8 ttl=53 time=6.06 ms

--- google.fr ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7010ms
rtt min/avg/max/mdev = 5.531/5.942/6.060/0.169 ms
root@ubuntu:~#
```

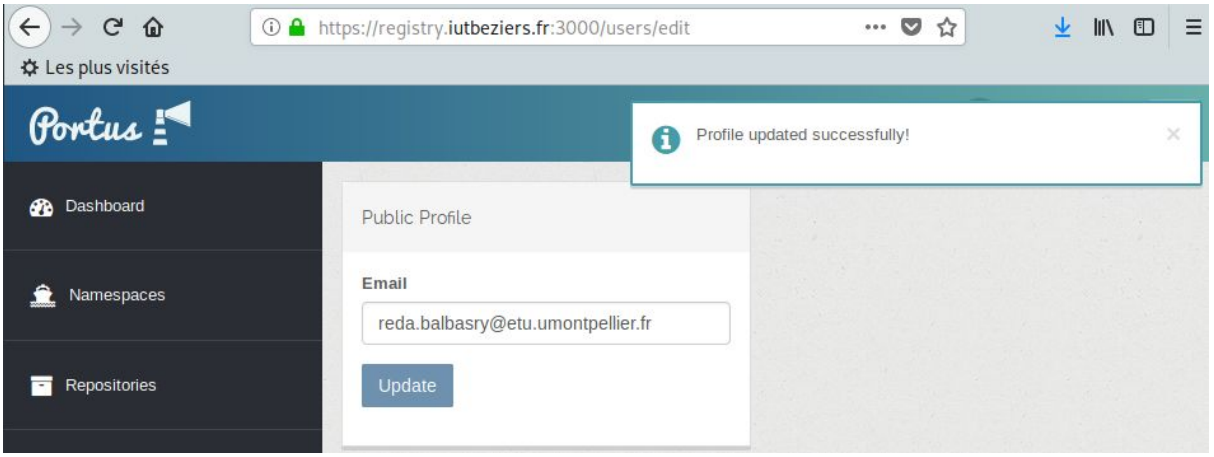
7. Utilisez l'option entrypoint de "docker run" pour changer la commande ping par ping6.Pingez : :1 pour vérifier le bon fonctionnement.

```
root@ubuntu:~# docker run -it --entrypoint "/bin/ping6" debian:pingfour2 ::1
connect: Ne peut attribuer l'adresse demandée
```

8. Transformez votre container en image en le "comittant" via la commande "docker commit...".

```
root@ubuntu:~# docker commit c6481ee652be
sha256:55350f3f5139f3bd34a2073e89b5d6b3badd395bdee7fff076002550a1719189
```

9. Authentifiez-vous à l'aide de votre Compte LDAP de l'IUT sur `https://registry.iutbeziers.fr:3000`) et renseigner votre adresse de mel de l'IUT. Portus créé ainsi un namespace "votre-prenom.votre-nom" dans lequel vous pouvez stocker vos images.



10. Utilisez la commande "docker tag" pour générer une image registry.iutbeziers.fr/votre-prenom.votre-nom/adet-ping à partir de l'image comittée précedemment. Poussez cette image sur votre namespace généré précédemment vers le registry de l'IUT de Béziers.

```
root@ubuntu:~# docker commit c6481ee652be test:cool
ensuite
root@ubuntu:~# docker tag test:cool registry.iutbeziers.fr/reda.balbasry/adet-ping
enfin
root@ubuntu:~# docker push registry.iutbeziers.fr/reda.balbasry/adet-ping
```

11. Récupérez l'image de votre voisin via un docker pull sur le registry de l'IUT de béziers. Instanciez-la afin de vérifier qu'elle fonctionne.

On ne peut pas vu que le ldap ne marche pas.

4.2 Création d'un Dockerfile afin de générer une image debian ssh

Créez un Dockerfile afin de générer un container fournissant un serveur SSH. Vous utiliserez l'image registry.iutbeziers.fr/debianiut :latest.
Vous instanciez cette image sous forme d'un container accessible en ssh sur le port 2222. Le container permettra l'authentification sur le compte root.

Indication : Utilisez chpasswd pour saisir le mot de passe root du container lors de son build.

Il faut configurer le dockerfile avec les bons arguments :

```
FROM debian:stretch
MAINTAINER jean-marc Pouchoulon

EXPOSE 22
RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo "root:root" | chpasswd
RUN sed -i 's/#PermitRootLogin .*/PermitRootLogin yes/' /etc/ssh/sshd_config
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g' -i
/etc/pam.d/sshd
ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile
CMD ["/usr/sbin/sshd", "-D"]
```

```
root@debian:~/tpdocker# docker build -t ssh .
Sending build context to Docker daemon  60.93kB
Step 1/11 : FROM debian:stretch
 ---> ef74351b551c
Step 2/11 : MAINTAINER jean-marc Pouchoulon
 ---> Using cache
 ---> babf74c77f67
Step 3/11 : EXPOSE 22
 ---> Using cache
 ---> 09fd03898f36
Step 4/11 : RUN apt-get update && apt-get install -y openssh-server
 ---> Using cache
 ---> 301eaf7cac1c
Step 5/11 : RUN mkdir /var/run/sshd
 ---> Using cache
 ---> 3a35f75b2342
Step 6/11 : RUN echo "root:root" | chpasswd
 ---> Using cache
 ---> e27721bf0f6b
Step 7/11 : RUN sed -i 's/#PermitRootLogin .*/PermitRootLogin yes/' /etc/ssh/sshd_config
 ---> Using cache
 ---> 0615555d68d5
Step 8/11 : RUN sed 's@session\s*required\s*pam_loginuid.so@session optional
pam_loginuid.so@g' -i /etc/pam.d/sshd
 ---> Using cache
 ---> b70b80d9f8aa
Step 9/11 : ENV NOTVISIBLE "in users profile"
 ---> Using cache
 ---> 75d31f515632
Step 10/11 : RUN echo "export VISIBLE=now" >> /etc/profile
 ---> Using cache
 ---> dbfe5b99465c
Step 11/11 : CMD ["/usr/sbin/sshd", "-D"]
 ---> Using cache
 ---> e5c5c5714935
```

```
Successfully built e5c5c5714935
Successfully tagged ssh:latest
```

```
root@debian:~/tpdocker# docker run -p 2222:22 -d ssh
9c0592020332e2b84e2e5623aa1edf8d99f75d35f5a46e174c162787ac5647ed
```

```
root@debian:~/tpdocker# docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS
PORTS                    NAMES
9c0592020332        ssh                 "/usr/sbin/sshd -D"   25 seconds ago      Up 25
seconds       0.0.0.0:2222->22/tcp   practical_galois
```

```
root@debian:~# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED
SIZE
ssh                                 latest              e5c5c5714935        About a
minute ago      206MB
testssh                             latest              e5c5c5714935        2 days ago
206MB
<none>                              <none>              88b73f2779af        2 days ago
101MB
<none>                              <none>              b51ac0295e22        2 days ago
101MB
<none>                              <none>              aa31136ab61f        2 days ago
359MB
registry.iutbeziers.fr/debianiut    latest              cf25ca6ca17a        3 days ago
363MB
debian                              stretch             ef74351b551c        7 days ago
101MB
```

4.3 Dockérisation d'une application Python

4.3.1 Installation de Python3 sur votre machine

root@debian:~/tpdocker# python3 --version
Python 3.7.3

4.3.2 Sans la container lancez l'appliquette suivante fonctionnant avec Python3

fichier python app.py :

```
from flask
import Flask app = Flask(name)
@app.route('/')
def hello_world(): return "Le Python c'est bon mangez en"
 if name == 'main':
app.run(debug=True,host='0.0.0.0')
```

on installe flask avec "pip3 install flask"

pyenv : curl https://pyenv.run | bash

Puis on installe python 3.7.2 avec “pyenv install”

4.3.3 "Dockérisez" cette application
Mon fichier dockerfile :

```
FROM python:3-alpine
WORKDIR /usr/src/app
COPY app.py ./
RUN pip install --no-cache-dir flask
ENV FLASK_APP=app.py
ENV FLASK_ENV=development
EXPOSE 1999

ENTRYPOINT ["python3"]
CMD ["./app.py" ]
```

```
root@debian:~/tpdocker# cat /usr/src/app/appl.py
from flask import Flask
app = Flask(name)
@app.route('/')
def hello_world():
return "Le Python c'est bon mangez en"
if name == 'main':
app.run(debug=True,host='0.0.0.0')
root@debian:~/tpdocker# docker build --tag python-tp .
Sending build context to Docker daemon  63.49kB
Step 1/8 : FROM python:3-alpine
 ---> d5e5ad4a4fc0
Step 2/8 : WORKDIR /usr/src/app
 ---> Using cache
 ---> a6b90a17af70
Step 3/8 : COPY app.py ./
 ---> f576c6a08a9b
Step 4/8 : RUN pip install --no-cache-dir flask
 ---> Running in a3c0bae9029a
Collecting flask
  Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
Collecting itsdangerous>=0.24
  Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting Werkzeug>=0.15
  Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
Collecting click>=5.1
  Downloading click-7.1.1-py2.py3-none-any.whl (82 kB)
Collecting Jinja2>=2.10.1
```

```
  Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)
Collecting MarkupSafe>=0.23
  Downloading MarkupSafe-1.1.1.tar.gz (19 kB)
Building wheels for collected packages: MarkupSafe
  Building wheel for MarkupSafe (setup.py): started
  Building wheel for MarkupSafe (setup.py): finished with status 'done'
  Created wheel for MarkupSafe:
filename=MarkupSafe-1.1.1-py3-none-any.whl size=12629
sha256=3f0920c434cc1af2347522ee84d8255d3585cbe13816c452eafce297cb0ab613
  Stored in directory:
/tmp/pip-ephem-wheel-cache-1977mvsu/wheels/0c/61/d6/4db4f4c28254856e8230
5fdb1f752ed7f8482e54c384d8cb0e

curl localhost:1999
Le Python c'est bon mangez en
```

4.3.4 Création d'un docker-compose pour une application en "micro-services"
Mon fichier docker.compose.yml

```
version: '3'
services:
  flask:
    build: .
    ports:
      - "1999:5000"
```

```
docker-compose up
Starting python-tp-app ... done
Attaching to python-tp-app
flask_1  |  * Serving Flask app "app" (lazy loading)
flask_1  |  * Environment: development
```

Je démarre le container avec un port spécifique :

```
root@debian:~/tpdocker# docker run --name python-tp -p 1999:5000
python-tp-app
   •  Serving Flask app “app” (lazy loading)
```

6 Réseaux Docker

lister les réseaux :

```
root@debian:~/tpdocker# docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
7027088f5121        bridge              bridge              local
f9ef2cdb88fe        host                host                local
78c8c3f73f28        none                null                local
```

créer un réseau nat :

root@debian:~/tpdocker# docker network create -d bridge nat

attacher un conteneur à un réseau :
docker run -itd --network=nat python:3-alpine
docker run -itd --network=nat python:3-alpine

attacher un conteneur au réseau macvlan :

```
root@debian:~/tpdocker# docker network create -d macvlan
--subnet=192.168.1.0/24 --gateway=192.168.1.254
--ip-range=192.168.1.64/26  -o parent=eth0 net-macvlan
d1fdb37d3a5da2b09dcc8d1587e60678613882cce1fc80d8f3c278886be9777d
```

attacher un conteneur au réseau ipvlan :

```
docker network create -d ipvlan --subnet=192.168.1.0/24
--ip-range=192.168.2.0/24 -o ipvlan_mode=l2 -o parent=eth0 ipvlan1
```

6. les conteneurs sont connectés au Bridge docker0.En utilisant le type de réseau macvlan le réseau du container va avoir un subnet identique a l'host docker.

7.Ça va utiliser le DNS du démon docker et ensuite le DNS de l'host.

7 Utilisation des namespaces par Docker

7.1 Accéder au namespace de l'hôte depuis un container Docker
Création d'un container avec docker run

```
root@debian:~/tpdocker# docker run -it --pid=host --net=host
corfr/tcpdump
```

7.2 Utilisation des usernamespaces par Docker.

L'option --user-remap=default permet d'activer les usernamespaces pour l'ensemble de containers :

le fichier /lib/systemd/system/docker.service :

```
# kill only the docker process, not all processes in the cgroup
KillMode=process

[Install]
WantedBy=multi-user.target

ExecStart=/usr/bin/docker daemon -H fd:// --userns-remap=default
```

2. Lancer un container avec un process bash et vérifier l'appartenance du process sur le docker :

root@debian:~/tpdocker# docker run -it registry.fr/debian:latest bash

7.3 Contrôle des ressources des containers au travers des cgroups

Après avoir généré une image du container a partir du dockerfile :

```
FROM debian:latest
RUN apt-get update && \
apt-get install stress
```

```
cd ../buildstress
docker rmi jmp/stress
docker build -t jmp/stress .
```

L'utilisation du cpu sur htop est de 100 %

```
root@debian:~/tpdocker# docker run --cpus="1" jmp/stress --cpu 10
--timeout 50s
```

8 Sécurisation des containers docker.