

TP1 Solutions de conteneurisation avec docker

Helec Bastien
29/04/2024

1. Docker installation :

```
sudo apt-get update

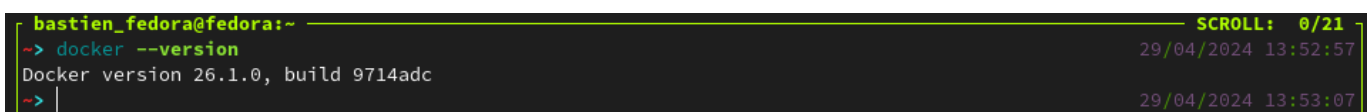
# Desinstaller les anciennes versions de docker
for pkg in docker.io docker-doc docker-compose podman-docker containerd
runc; do sudo apt-get remove $pkg; done

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

2. Docker sous linux :

1. Retrouvez la version de Docker installée ?

```
docker --version
```



```
bastien_fedora@fedora:~ SCROLL: 0/21
-> docker --version 29/04/2024 13:52:57
Docker version 26.1.0, build 9714adc
-> | 29/04/2024 13:53:07
```

2. Verifiez que votre installation fonctionne bien :

```
docker run hello-world
```

```
bastien_fedora@fedora:~ SCROLL: 9/37
-> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:a26bffa933ddc26d5cdf7faa98b4ae1e3ec20c4985e6f87ac0973052224d24302
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:
```

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


1. Le client docker a contacté le demon docker
2. Le docker demon tire l'image hello-world depuis le docker hub
3. Le demon docker créer un nouveau container à partir de l'image hello-world et le lance avec la sortie lu actuellement
4. Le demon docker affiche le message sur cette sortie du client docker sur le terminal actuelle

Pour essayer quelque chose de plus ambitieux , vous pouvez lancez un container ubuntu avec la commande suivante :


```
docker run -it ubuntu bash
```

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

[Explore](#) / [Official Images](#) / [hello-world](#)



hello-world

 Docker Official Image · 1B+ · 2.2K

Hello World! (an example of minimal Dockerization)

docker pull hello-world [Copy](#)

Overview

Tags

Quick reference

- Maintained by:
[the Docker Community](#)
- Where to get help:
[the Docker Community Slack](#), [Server Fault](#), [Unix & Linux](#), or [Stack Overflow](#)

Supported tags and respective Dockerfile links

(See "What's the difference between 'Shared' and 'Simple' tags?" in the FAQ.)

Simple Tags

- [linux](#)
- [nanoserver-ltsc2022](#)
- [nanoserver-1809](#)

Shared Tags

Recent Tags

[nanoserver-ltsc2022](#) [nanoserver-1809](#) [nanoserver-latest](#) [linux](#) [nanoserver-1803](#) [nanoserver-1709](#) [nanoserver-sac2016](#) [nanoserver1709](#)

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

/

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

Les mécanismes sont les suivants :

- La récupération de l'image
- La création du container
- Le lancement du container

Pour la création du container helloworld , le demon docker a besoin de l'image hello-world qui est stocké sur le docker hub. Le fichier qui permet de créer ce container est le fichier Dockerfile.

3. Recherchez les images officielles Debian à l'aide de docker search. Récupérez-les ainsi que les images officielles busybox.

```
docker search debian
docker search busybox
```

```
Search Docker Hub for images
-> docker search debian | docker search busybox 1 29/04/2024 14:13:11
NAME DESCRIPTION STARS OFFICIAL
busybox Busybox base image. 3243 [OK]
rancher/busybox 0
chainguard/busybox 0
openebs/busybox-client 1
antrea/busybox 1
airbyte/busybox 0
hugegraph/busybox test image 2
privatebin/chown Docker image providing busybox' chown, stat... 1
yauritux/busybox-curl Busybox with CURL 25
radial/busyboxplus Full-chain, Internet enabled, busybox made f... 56
vukomir/busybox busybox and curl 1
arm64v8/busybox Busybox base image. 8
odise/busybox-curl 4
busybox42/zimbra-docker-centos A Zimbra Docker image, based in ZCS 8.8.9 an... 2
amd64/busybox Busybox base image. 1
busybox42/alpine-pod 0
joeshaw/busybox-nonroot Busybox container with non-root user nobody 2
```

```
docker pull debian
docker pull busybox
```

```
-> docker pull debian 29/04/2024 14:14:37
Using default tag: latest
latest: Pulling from library/debian
1468e7ff95fc: Pull complete
Digest: sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd517a862cb9
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest

-> docker pull busybox 29/04/2024 14:15:38
Using default tag: latest
latest: Pulling from library/busybox
7b2699543f22: Pull complete
Digest: sha256:c3839dd800b9eb7603340509769c43e146a74c63dca3045a8e7dc8ee07e53966
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
```

4. Créez votre premier container a partir de l'image debian officielle et en utilisant la commande

```
docker run -d debian
```

```
-> docker run -d debian 29/04/2024 14:18:27
8b619538840fd52b831bd38aa75eb9da6294cf99650da0186e69a303853d63c8
```

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

```
docker ps
```

```
-> docker ps 29/04/2024 14:18:32
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
```

Aucun container n'est actuellement pas vivant car le logiciel docker stop automatiquement les containers qui n'ont pas de processus en cours d'execution dans le container il est egalement inexistant.

6. Relancez le "docker run" en lui donnant comme argument

```
bash -c "while ;; do echo "coucou"; sleep 1;done".
```

```
docker run -d debian bash -c "while ;; do echo "coucou"; sleep 1;done"
```

```
bastien_fedora@fedora:~ SCROLL: 0/395
-> docker run -d debian bash -c "while ;; do echo "coucou"; sleep 1;done" 29/04/2024 14:24:28
7bae8ba4cb7ff2aa1bb12f3efc1bc97552e3a66fd16ff3adedeb479f9b22ac2e
-> docker ps 29/04/2024 14:25:16
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
7bae8ba4cb7f    debian    "bash -c 'while ;; d..." 4 seconds ago    Up 3 seconds    lucid_shannon
-> | 29/04/2024 14:25:20
```

Ici on peut voir que le processus est en cours d'execution dans le container.

7. Stoppez et redémarrez le container.

```
docker stop <container_id>
docker start <container_id>
```

```
-> docker stop affectionate_lichterman 1 29/04/2024 14:38:27
affectionate_lichterman
-> docker start affectionate_lichterman 29/04/2024 14:38:43
affectionate_lichterman
-> docker ps 29/04/2024 14:38:49
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
fdac682b6b47    debian    "bash -c 'while ;; d..." 49 seconds ago    Up 3 seconds    affectionate_lichterman
-> | 29/04/2024 14:38:53
```

8. Supprimez le container.

```
docker rm <container_id>
```

```
bastien_fedora@fedora:~ — SCROLL: 0/848
-> docker container rm affectionate_lichterman 29/04/2024 14:39:49
Error response from daemon: cannot remove container "/affectionate_lichterman": container is running: stop the container before removing or force remove
-> docker container rm -f affectionate_lichterman 1 29/04/2024 14:40:00
affectionate_lichterman
-> | 29/04/2024 14:40:07
```

9. Utilisez les options -it afin d'intégrer RUN pip3 install flaskle container a son lancement

```
docker run -it debian bash
```

```
bastien_fedora@fedora:~ — SCROLL: 0/868
-> docker run -it debian bash 29/04/2024 14:44:21
root@e305f77b5ac1:/# |
```

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

```
docker run -it --name DebianOne --hostname DebianOne debian bash
```

```
bastien_fedora@fedora:~ — SCROLL: 0/888
-> docker run -it --name DebianOne --hostname DebianOne debian bash 29/04/2024 14:47:05
root@DebianOne:/# |
```

11. Détachez vous du container debianone puis rattachez vous a lui de nouveau.

```
exit
```

```
docker attach DebianOne
```

```
bastien_fedora@fedora:~ — SCROLL: 0/943
-> docker attach DebianOne 29/04/2024 14:52:02
You cannot attach to a stopped container, start it first
-> docker start DebianOne 1 29/04/2024 14:52:04
DebianOne
-> docker attach DebianOne 29/04/2024 14:52:09
root@DebianOne:/# exit
exit
-> | 29/04/2024 14:52:13
```

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

```
docker exec -it DebianOne bash -c "echo coucou"
```

```
bastien_fedora@fedora:~ SCROLL: 0/1026
-> docker exec -it DebianOne bash -c "echo coucou" 29/04/2024 14:57:52
coucou
-> | 29/04/2024 14:58:05
```

13. Listez le container restant. Ne listez ensuite que le dernier ContainerID.

```
docker ps -a
docker ps -l
```

```
bastien_fedora@fedora:~ SCROLL: 0/1086
-> docker ps -a 29/04/2024 15:01:25
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS          NAMES
7e737a373dce   debian    "bash"                  14 minutes ago   Up 6 minutes                               DebianOne
e305f77b5ac1   debian    "bash"                  15 minutes ago   Exited (0) 14 minutes ago                 modest_jennings
6cd5e31f1d87   hello-world "/hello"                About an hour ago Exited (0) About an hour ago                 serene_bassi
-> docker ps -l 29/04/2024 15:01:26
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS          NAMES
7e737a373dce   debian    "bash"                  14 minutes ago   Up 6 minutes                               DebianOne
```

14. Utilisez un volume pour donner à votre container l'accès à un répertoire de l'hôte. Quels sont les avantages de l'utilisation d'un volume ? un inconvénient ? A l'aide de la commande docker volume affichez les volumes présents sur votre hôte.

```
docker run -it --name DebianOne --hostname DebianOne -v
VolDebone:/Documents debian bash
```

```
bastien_fedora@fedora:~ SCROLL: 0/44
-> docker run -it --name DebianOne --hostname DebianOne -v VolDebone:/Documents debian bash 29/04/2024 15:26:25
root@DebianOne:/# ls
Documents bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@DebianOne:/# |
```

Les avantages de l'utilisation d'un volume sont les suivants :

- Les données sont persistantes
- Les données sont partagées entre les containers
- Les données sont partagées entre l'hôte et le container

L'inconvénient de l'utilisation d'un volume est la sécurité des données un container non protégé peut ainsi permettre la vole de donnée.

```
docker volume ls
```

```

bastien_fedora@fedora:~
-> docker volume ls
DRIVER      VOLUME NAME
local       9600228e35b9150d6c60e454c234074c3882e9b006290f4b185d2ce4f0f7808e
local       VolDebone
29/04/2024 15:27:16
-> |
29/04/2024 15:28:06

```

15. Supprimez le container et son image.

```
docker rm DebianOne
docker rmi -f debian
```

```

bastien_fedora@fedora:~
-> docker rmi -f debian
Untagged: debian:latest
Untagged: debian@sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd517a862cb9
Deleted: sha256:d2a2c1ada45a61ee343effe4bda1f987c453840593c9622bd08919f06b508d0d
-> |
29/04/2024 15:31:51
29/04/2024 15:31:56

```

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

```
docker rm $(docker ps -a -q)
docker rmi $(docker images -q)
```

```

bastien_fedora@fedora:~
-> exit
root@debian:~# docker rm $(docker ps -a -q)
e305f77b5ac1
6cd5e31f1d87
root@debian:~# docker rmi $(docker images -q)
Untagged: registry.iutbeziers.fr/debian12:ssh
Untagged: registry.iutbeziers.fr/debian12@sha256:efec558354de995bf6739b6b71af4643444bdb5f64121cd2ad0c09477acf20d8
Deleted: sha256:6872a86771a16a5d729b2892099e008fce72028cf5cbcd4706bd3c6d64f92553
Deleted: sha256:64b5c12fcc0b5869fa3bd6f3819d18f4e2c043ac2cea73aad13d3b8671cdd403
Deleted: sha256:1029816959f3f391c383be1cd7181c031115d904e3f7fa4a8e4b89ef306bf799
Deleted: sha256:e0e6be9440c8c54d7f93219a9806f8587632542d19c7f8b6f0fbb8f6c3dd67b2
Deleted: sha256:20f78b5d219a425270d3073bcac0c45ea51d4016ecd35f11dd912a085425fe13
Deleted: sha256:e85247ac0cdad047b433f979ad4836104cc983dcac8d2b67af48181fa7499a84
Deleted: sha256:fe02fc00cc260011f45acc363604e4c5059f51dbf27024eb6bed1fa9f7924187
Deleted: sha256:c904f884b574edd9f517d9ccf428641da6e43720a0b58566deb050bf8b64131e
Deleted: sha256:b1a8788e001a9ea3651855459681077a1c60919f6932b8c9e62d7eee5da6dcdf
Deleted: sha256:42c20d45fdb3fe6b22ba7bbfd8919496b06af4f7f1d9a0b5c3f7660f64f1583f
Untagged: registry.iutbeziers.fr/debianiut:latest
Untagged: registry.iutbeziers.fr/debianiut@sha256:c161071aa4fee057d117fe565608c776868c1d7846f98a4f3c7102f8bb1f5f73
Deleted: sha256:970e58cf357d011a484d3401242bc507835567f13273b7ade894432cd9c05b18

```

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

```
docker system prune
```

```

bastien_fedora@fedora:~
-> docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- unused build cache

Are you sure you want to continue? [y/N] y

```

3 Création d'images Docker :

Récupérez les fichiers pour cet exercice via git :

```
git clone https://github.com/pushou/tpdocker.git
```

!d

3.1 Build d'une image docker debian:

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

```
cd tpdocker/
docker build -t debian:HB -f Dockerfile.bookworm .
```

```

-/tpdocker> docker build -t debian:HB -f Dockerfile.bookworm .
[+] Building 5.4s (4/9)
=> [internal] load build definition from Dockerfile.bookworm
=> => transferring dockerfile: 1.64kB
=> [internal] load metadata for docker.io/library/debian:bookworm
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/debian:bookworm@sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd
=> => resolve docker.io/library/debian:bookworm@sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd
=> => sha256:1aadfee8d292f64b045adb830f8a58bfacc15789ae5f489a0fedcd517a862cb9 1.85kB / 1.85kB
=> => sha256:40f71cd223a60afc1bac2adflb204bfabef29cdef725e74993f86098ff87f92f 529B / 529B
=> => sha256:d2a2clada45a6lee343effe4bda1f987c453840593c9622bd08919f06b508d0d 1.46kB / 1.46kB
=> => sha256:1468e7ff95fcb865fbc4dee7094f8b99c4dcddd6eb2180cf044c7396baf6fc2f 40.89MB / 49.58MB
=> [internal] load build context
=> => transferring context: 859B

```

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

- RUN : execute les nouvelles maj et installe les paquets comme cmatrix , wget ...
- ENV : Defini les nouvelles variables
- FROM : permet de définir l'image de base

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).

Cela evite de creer plusieurs couches dans l'image docker et donc de reduire la taille de l'image.

1. 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 défaut. Vous utiliserez les commandes "ENTRYPOINT" et "CMD" dans le Dockerfile.

```
nano Dockerfile.pingfour
```

```
FROM debian:HB
```

```
docker build -t pingfour -f Dockerfile.ping4 .
```

```

- /tpdocker> docker build -t pingfour -f Dockerfile.ping4 .
[+] Building 2.0s (6/6) FINISHED
=> [internal] load build definition from Dockerfile.ping4
=> => transferring dockerfile: 205B
=> [internal] load metadata for docker.io/library/debian:HB
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/2] FROM docker.io/library/debian:HB
=> [2/2] RUN apt-get update && apt-get upgrade -y && apt-get install -y iputils-ping
=> exporting to image
=> => exporting layers
=> => writing image sha256:6e7bde7ddc56851e04e00a07c4223b41a40a62d61c587b365ca3345f4d1a329a
=> => naming to docker.io/library/pingfour
- /tpdocker>

```

Entrypoint et CMD sont les memes choses , elles permettent de lancer une commande lors du lancement du container. La difference entre les deux est que CMD permet de surcharger la commande lors du lancement du container contrairement à l'entrypoint.

5. Lancez un container issu de cette image au travers de la commande `docker run --rm -it`.
A quoi sert la commande -rm

```
docker run --rm -it pingfour
```

La commande -rm permet de supprimer le container lorsqu'il est arreté.

```

~/tpdocker> docker run --rm -it pingfour
PING www.iutbeziers.fr (146.59.209.152) 56(84) bytes of data.
64 bytes from cluster031.hosting.ovh.net (146.59.209.152): icmp_seq=1 ttl=44 time=19.3 ms
64 bytes from cluster031.hosting.ovh.net (146.59.209.152): icmp_seq=2 ttl=44 time=19.7 ms
64 bytes from cluster031.hosting.ovh.net (146.59.209.152): icmp_seq=3 ttl=44 time=19.6 ms
64 bytes from cluster031.hosting.ovh.net (146.59.209.152): icmp_seq=4 ttl=44 time=19.6 ms

--- www.iutbeziers.fr ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 19.300/19.547/19.707/0.153 ms
~/tpdocker>

```

6. Peut-t-on changer la destination du ping ? le nombre de ping ?

Oui dans le dockerfile on peut changer la destination du ping et le nombre de ping. au niveau du CMD final dans le dockerfile.

7. Utilisez l'option entrypoint de "docker run" pour changer la commande ping par traceroute.

```
docker run -it --entrypoint /bin/traceroute pingfour www.iutbeziers.fr
```

```

bastien_fedora@fedora:~
~/tpdocker> docker run --rm -it --entrypoint traceroute pingfour www.iutbeziers.fr
traceroute to www.iutbeziers.fr (146.59.209.152), 64 hops max
 1  172.17.0.1  0,002ms  0,001ms  0,001ms
 2  10.202.255.254  1,599ms  1,741ms  2,218ms
 3  194.199.227.254  0,404ms  0,455ms  0,477ms
 4  100.77.22.254  1,489ms  1,555ms  1,436ms
 5  100.74.101.65  2,508ms  2,190ms  2,206ms
 6  100.77.255.243  1,740ms  1,729ms  1,822ms
 7  100.77.255.241  1,827ms  1,714ms  1,788ms
 8  100.77.255.113  1,718ms  1,736ms  1,743ms
 9  100.77.255.114  1,788ms  1,750ms  1,757ms
10  193.55.200.140  2,458ms  2,508ms  2,454ms
11  193.51.180.100  15,815ms  16,370ms  15,862ms
12  193.51.177.127  16,745ms  17,889ms  15,381ms
13  193.51.177.255  15,377ms  15,643ms  15,448ms
14  193.51.177.238  15,613ms  15,482ms  15,468ms
15  57.128.121.26  17,498ms  17,768ms  18,419ms
16  * * *
17  * * *
18  * * |

```

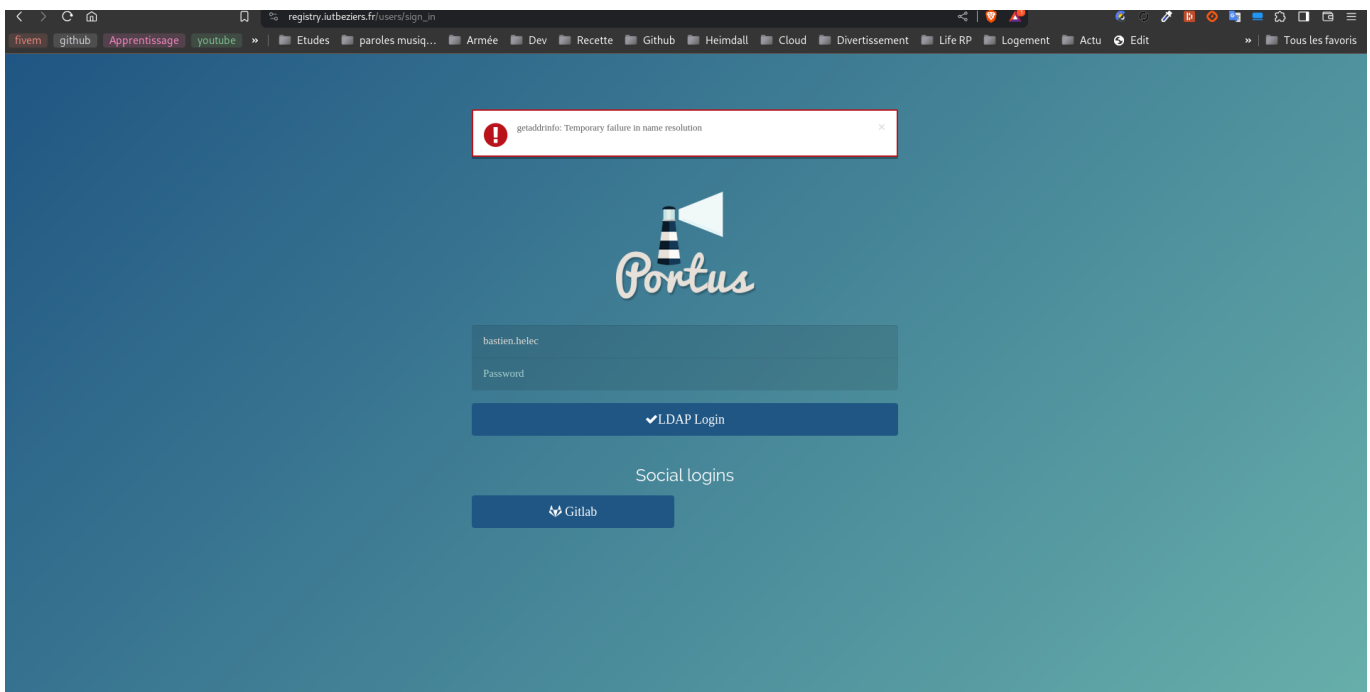
8. Transformez votre container en image en le "commitant" via la commande "docker commit..."

```
docker commit xenodochial_diffie pingfour:traceroute
```

```
bastien_fedora@fedora:~ SCROLL: 0/990
22 * * *
23 * * *
24 * * *
25 * * *
26 * * *
27 * * *
28 * * *
29 * * *
30 * * *
31 * * *
32 * * *
33 * * *
34 * * *
35 * * *
36 * * *
37 * * *
38 * * *
39 * * *
40 * * *

bastien_fedora@fedora:~ SCROLL: 0/91
-> docker commit xenodochial_diffie pingfour:traceroute
sha256:8334ecdc323895fc814316c2fe2865a7bc2c0f3c5bd6adcc9e
3c99aaa2e8e585
-> |
29/04/2024 16:45:20
```

9. Créez un projet dont le nom sera de la forme "prenom.nom" sur registry.iutbeziers.fr
 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 précédemment. Poussez cette image sur votre namespace généré précédemment vers le registry de l'IUT de Béziers.
 11. Récupérez l'image de votre voisin via un docker pull sur le registry mis en place par votre enseignant. Instanciez-la afin de vérifier qu'elle fonctionne.
- le 9 10 et 11 a besoin de l'accès au registry de l'IUT de Béziers pour pouvoir les réaliser. Mais il y a un problème de connexion :



3.2 Installation d'un "insecure registry" sur votre poste de travail

En suivant <https://docs.docker.com/registry/insecure> installez un registry sur votre VM et testez-le. L'installation d'un certificat n'est pas demandée.

```
bastien_redorag@redora:~$ docker pull localhost:5000/myfirstimage
Using default tag: latest
latest: Pulling from myfirstimage
8f5159575f7b: Pull complete
Digest: sha256:18e322a5001f278b9a61c56883205a2aa6feaab02ee657023357b2987f7b41cc
Status: Downloaded newer image for localhost:5000/myfirstimage:latest
localhost:5000/myfirstimage:latest
-> docker rmi localhost:5000/myfirstimage
Untagged: localhost:5000/myfirstimage:latest
Untagged: localhost:5000/myfirstimage@sha256:18e322a5001f278b9a61c56883205a2aa6feaab02ee657023357b2987f7b41cc
Deleted: sha256:de52d803b2245ea6d6b4235e43533dc5a70ea3837c0db840603d0b828d42266c
Deleted: sha256:3e1ed584ae0e22f951b55e86be65fe081c95af0b5810f67ba0be7681c6b7bf0f
-> docker pull localhost:5000/myfirstimage
```

3.3 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 comme image de base.

Vousinstancierez 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.

```
nano Dockerfile.ssh
```

```
FROM registry.iutbeziers.fr/debianiut
```

```
RUN apt-get update && apt-get install -y \
openssh-server
```

```
RUN echo 'root:root' | chpasswd
```

```
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/'
/etc/ssh/sshd_config
```

```
RUN mkdir /var/run/sshd
```

```
EXPOSE 2222
```

```
CMD ["/usr/sbin/sshd", "-D"]
```

```
docker build -t debian:ssh -f Dockerfile.ssh .
```

```

~/tpdocker> nano Dockerfile.ssh
~/tpdocker> docker build -t debian:ssh -f Dockerfile.ssh .
[+] Building 6.2s (9/9) FINISHED
=> [internal] load build definition from Dockerfile.ssh
=> => transferring dockerfile: 337B
=> [internal] load metadata for registry.iutbeziers.fr/debianiut:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [1/5] FROM registry.iutbeziers.fr/debianiut:latest@sha256:c161071aa4fee057d117fe565608c776868c1d784
=> [2/5] RUN apt-get update && apt-get install -y openssh-server
=> [3/5] RUN echo 'root:root' | chpasswd
=> [4/5] RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd_config
=> [5/5] RUN mkdir /var/run/ssh
=> exporting to image
=> => exporting layers
=> => writing image sha256:44a39cbfc499555128dfeafeaa142f3285e30422371edc248df7b7b022eae2d5
=> => naming to docker.io/library/debian:ssh
~/tpdocker>

```

3.4 Dockérisation d'une application python :

Sans le container lancez l'appliquette suivante fonctionnant avec python3

```

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')

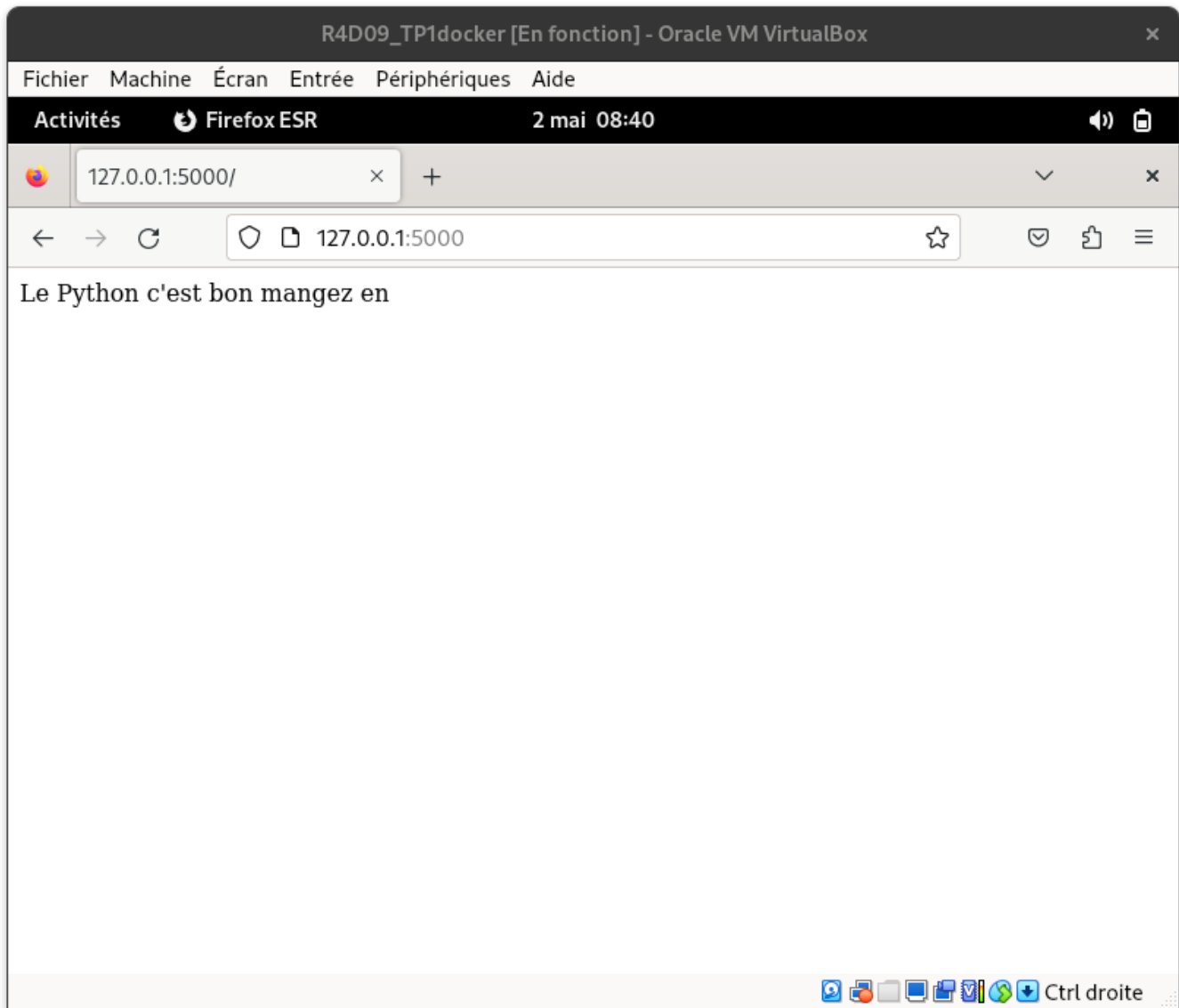
```

puis on lance l'application via :

```

apt install python3-flask
export ENV FLASK_APP=app.py
export ENV FLASK_APP=development
FLASK_APP=app.py flask run -p 9999

```



Pour la prochaine étapes Dockériser cette application en créant un Dockerfile et en utilisant une image python 3 (base Debian) 1. L'application doit être accessible sur le port 9999 de l'hôte.

```
nano Dockerfile.flask

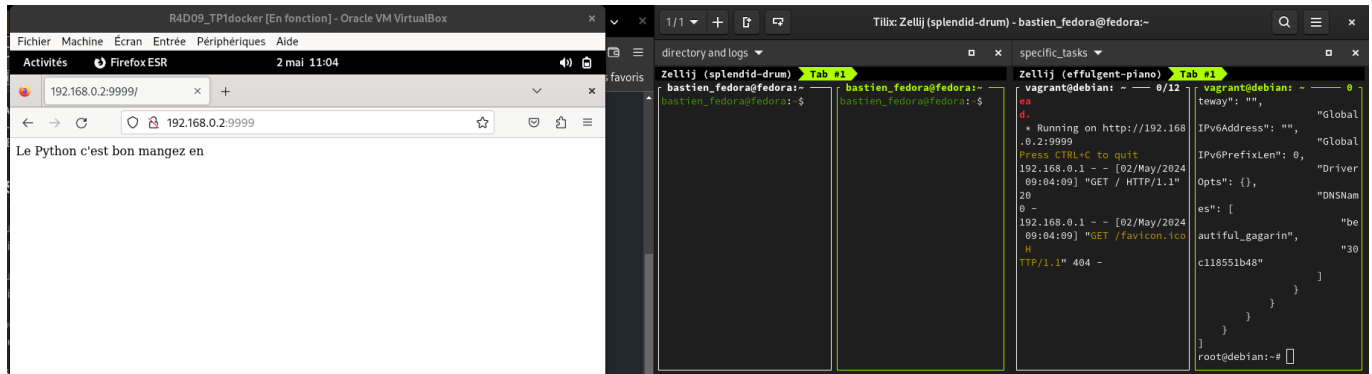
FROM debian:bookworm

RUN apt-get update && apt-get install -y \
python3 \
python3-pip \
python3-flask \
nano

EXPOSE 9999
```

```
docker build -t flask -f Dockerfile.flask .
```

```
docker network disconnect flask [container_id]
docker network create --subnet=192.168.0.0/24 flask
docker network connect flask [container_id]
docker run -d -p -h 192.168.0.2 9999:9999 flask
```



Création d'un docker-compose pour une application en microservices :

Il est possible de démarrer plusieurs containers afin de former un ensemble applicatif cohérent. Pour cela il faut utiliser un fichier docker-compose.yml qui sera utilisé par la commande docker-compose afin de démarrer cet ensemble cohérent de containers.

On utilise la distribution Debian pour les containers.

```
version: '3'
services:
  web:
    build: .
    ports:
      - "9999:9999"
  redis:
    image: "redis:alpine"
```

et le Dockerfile :

```
nano Dockerfile
```

CMD ["flask", "run", "--debug", "--host=172.18.0.2", "--port=9999"]

1. Modifiez l'application flask précédente de façon à afficher l'IP du container sur l'uri /whoami. Recréez une image.

```
from flask import Flask, jsonify, request
import os
app.run(debug=True, host='0.0.0.0', port=9999)
```

```

app = Flask(__name__)
@app.route('/')
def hello_world():
    return "Le Python c'est bon mangez en \n"

@app.route('/whoami')
def get_tasks():
    ipv4=os.popen('ip addr show eth0').read().split("inet ")[1].split("/")
    [0]
    return jsonify({'ip': ipv4}), 200

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=9999)

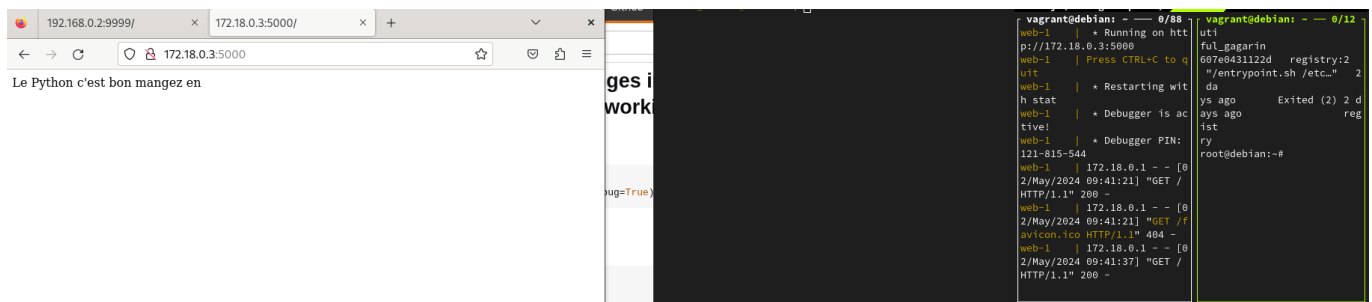
```

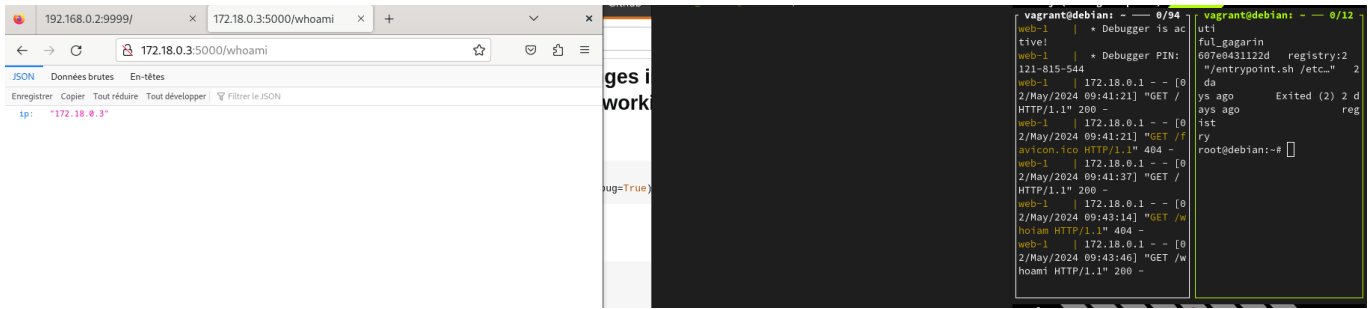
Je ne sais pas pourquoi malgres toutes les configurations l'application web ce lance sur le port 5000 et non sur le port 9999.

```

Zellij (effulgent-piano) Tab #1
vagrant@debian: ~
root@debian:~/docker_compose_flask# docker compose up
WARN[0000] /root/docker_compose_flask/compose.yaml: `version` is obsolete
[+] Running 7/9
  :: redis [██████████] Pulling                                6.2s
  ✓ 4abcf2066143 Pull complete                                0.8s
  ✓ 5c3180d10209 Pull complete                                1.5s
  ✓ f76326fd8e6b Pull complete                                1.5s
  ✓ 034c076ba1e7 Pull complete                                1.4s
  ✓ dffcad17539b Pull complete                                3.2s
  ✓ 5913474e0f39 Pull complete                                2.0s
  ✓ 4f4fb700ef54 Pull complete                                2.7s
  :: cc6fccbbefa3 Waiting                                     3.7s

```





2. Créez un fichier docker-compose.yml qui générera un ensemble contenant un container traefik (version 2+) et un container issus de notre application. Aidez-vous de <https://docs.traefik.io/>. Vous pouvez utiliser le domaine ni.io qui renvoie une ip privée lorsqu'on fait une requête (mon ip privée est ici 192.168.1.95) :

```
host 192.168.1.95.nip.io
192.168.1.95.nip.io has address 192.168.1.95
host whoami.192.168.1.95.nip.io
whoami.192.168.1.95.nip.io has address 192.168.1.95
```

4. Réseaux Docker :

1. Listez les réseaux présent sur votre hôte.

```
docker network ls
```

2. Créez un réseau bridge supplémentaire.

```
docker network create --driver bridge mybridge
```

3. Créez eux nouveaux containers en les rattachant à ce nouveau réseau NAT. Expliquez comment le container accède au réseau de la salle

Le container accède au réseau de la salle en utilisant le réseau NAT qui a pour gateway la machine hôte et qui permet de faire la translation d'adresse entre le réseau local et le réseau public via des commandes iptables ou nftables integrer.

```
docker run -d --network mybridge --name container1 debian
docker run -d --network mybridge --name container2 debian
```

4. Créez un nouveau container en le rattachant a un réseau macvlan.

```
docker network create -d macvlan --subnet=10.202.0.0/16
```

```
docker run -d --network macvlan --name container3 debian
```

5. Criez un container en le rattachant a un réseau ipvlan.

```
docker network create -d ipvlan --subnet=10.202.0.0/16  
docker run -d --network ipvlan --name container4 debian
```

6. Expliques l'utilité des différents types de réseaux :

- Bridge : permet de connecter les containers à un réseau local
- Host : permet de connecter les containers à l'hôte
- Overlay : permet de connecter les containers entre eux
- Macvlan : permet de connecter les containers à un réseau local
- Ipvlan : permet de connecter les containers à un réseau local
- None : permet de connecter les containers à aucun réseau

L'utilité pour chacun d'entre eux et de pouvoir avoir une connectivité interne sans passé par le réseau public et de pouvoir isoler les containers entre eux. Ou bien de pouvoir connecter les containers à un réseau local.

7. Quelle est la chaine de résolution DNS utilisée par le container ? Comment changer le DNS au run ?

La chaine de résolution DNS utilisée est celle de l'hôte. Pour changer le DNS au run il faut utiliser l'option --dns.

```
docker run --dns
```

5: Tips and Tricks :

Docker est batie sur une architecture modulaire ui lui permet de se connecter à un daemon Docker sur une machine distante en TLS ou SSH.

1. Créez un contexte pour vous connectez à distance au daemon Docker de votre VM (Vous pouvez utiliser une autre VM Linux ou installer Docker)

```
docker context create R4D09_TP1 --docker "host=ssh://vagrant@10.202.0.176"
```

2. Verifiez le bon fonctionneme du contexte en créant un container sur la VM

```
bastien_fedora@fedora:~$ docker context create R4D09_TP1 --docker "host=ssh://vagrant@10.202.0.176"
R4D09_TP1
Successfully created context "R4D09_TP1"
bastien_fedora@fedora:~$ ip -br a
lo                UNKNOWN    127.0.0.1/8  ::1/128
enp2s0            UP          10.202.0.151/16 fe80::59d2:9c0f:8df4:aa78/64
wlp0s20f3         DOWN
lxcbr0            DOWN        10.0.3.1/24 fc11:4514:1919:810::1/64
docker0           DOWN        172.17.0.1/16
vboxnet0          DOWN
bastien_fedora@fedora:~$
```

```
docker --context R4D09_TP1 run -d debian
```