



# TÉLÉCOMMUNICATION ET RÉSEAUX

## LICENCE 3

ANNÉE-SCOLAIRE 2019 – 2020

### **RAPPORT: Memento DOCKER**

Étudiant:  
Innonce DISU – Hafid

Encadreur:  
Samuel Ouya

# Objectifs

## I – Installation de Docker

## II – Gestion d'image Dockers

Rechercher une image depuis le docker hub

Télécharger une image

Lister les images

Inspecter une image

Supprimer une image

Supprimer toutes les image

## III – Gestion de conteneurs Dockers

Créer un conteneur à partir d'une image

Liste les conteneurs

Processus

Inspecter un conteneur

Supprimer une image

Supprimer toutes les image

Exécuter une commande dans docker

La sauvegarde et restauration

## IV – Les drivers (réseaux) Dockers

Le driver Bridge

Le driver None

Le driver Host

Le driver Overlay

Le driver Macvlan

## V – API-Docker

### 1 – Images

Lister les images

Rechercher une image

créer une image

Inspecter une image

Obtenir l'historique d'une image

Supprimer une image

### 2 – Conteneurs

Lister les conteneurs

Inspecter un conteneur

Lister des processus en cours d'exécution à l'intérieur d'un conteneur

Démarrer un conteneur

Arrêter un conteneur

Supprimer un conteneur

## Docker FILE (prospective)

# I – INSTALLATION DE DOCKER

Ce tutoriel aidera à installer Docker sur Ubuntu 18.10, 18.04 LTS et 16.04 LTS.

La toute première étape consiste à supprimer tous les packages Docker par défaut du système avant d'installer Docker sur un système Linux. Exécutez des commandes pour supprimer les versions inutiles de Docker.

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO:~# apt-get purge docker lxc-docker docker-engine docker.io  
Lecture des listes de paquets... Fait  
Construction de l'arbre des dépendances  
Lecture des informations d'état... Fait  
E: Impossible de trouver le paquet lxc-docker  
E: Impossible de trouver le paquet docker-engine  
root@ubuntu-ESPRIMO:~#
```

Maintenant, installons quelques paquets requis sur notre système pour installer Docker sur un système Ubuntu. Exécutons les commandes ci-dessous pour ce faire:

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO:~# apt-get install curl apt-transport-https ca-certificates software-properties-common  
Lecture des listes de paquets... Fait  
Construction de l'arbre des dépendances  
Lecture des informations d'état... Fait  
apt-transport-https est déjà la version la plus récente (1.2.32).  
apt-transport-https passé en « installé manuellement ».  
ca-certificates est déjà la version la plus récente (20170717-16.04.2).  
ca-certificates passé en « installé manuellement ».  
curl est déjà la version la plus récente (7.47.0-1ubuntu2.14).  
software-properties-common est déjà la version la plus récente (0.96.20.9).  
software-properties-common passé en « installé manuellement ».  
0 mis à jour, 0 nouvellement installés, 0 à enlever et 4 non mis à jour.  
root@ubuntu-ESPRIMO:~#
```

Maintenant, importons la clé GPG officielle des dockers pour vérifier la signature des paquets avant de les installer avec **apt-get**.

Exécutons la commande ci-dessous

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add  
OK  
root@ubuntu-ESPRIMO:~#
```

Après cela, ajoutons le référentiel Docker sur notre système Ubuntu qui contient les packages Docker, y compris ses dépendances.

Nous devons activer ce référentiel pour installer Docker

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO:~# add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"  
root@ubuntu-ESPRIMO:~#
```

Notre système est maintenant prêt pour l'installation de Docker.

Exécutons les commandes suivantes pour mettre à niveau l'index APT puis installer Docker Community Edition sur Ubuntu.

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO: ~ x root@ubuntu-ESPRIMO: /var/www/html/rest-api... x root@ubuntu-ESPRIMO: ~ x  
root@ubuntu-ESPRIMO:~# apt-get update  
Atteint :1 http://ppa.launchpad.net/ansible/ansible/ubuntu xenial InRelease  
Réception de :2 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]  
Atteint :3 http://sn.archive.ubuntu.com/ubuntu xenial InRelease  
Réception de :4 http://sn.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]  
Réception de :5 https://download.docker.com/linux/ubuntu xenial InRelease [66,2 kB]  
Atteint :6 https://packages.microsoft.com/repos/vscode stable InRelease
```

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO: ~ x root@ubuntu-ESPRIMO: /var/www/html/rest-api... x root@ubuntu-ESPRIMO: ~ x  
root@ubuntu-ESPRIMO:~# apt-get install docker-ce  
Lecture des listes de paquets... Fait  
Construction de l'arbre des dépendances  
Lecture des informations d'état... Fait  
Les paquets supplémentaires suivants seront installés :  
  aufs-tools cgroupfs-mount containerd.io docker-ce-cli pigz  
Les NOUVEAUX paquets suivants seront installés :  
  aufs-tools cgroupfs-mount containerd.io docker-ce docker-ce-cli pigz  
0 mis à jour, 6 nouvellement installés, 0 à enlever et 4 non mis à jour.  
Il est nécessaire de prendre 85,3 Mo dans les archives.  
Après cette opération, 384 Mo d'espace disque supplémentaires seront utilisés.  
Souhaitez-vous continuer ? [O/n]  
Réception de :1 http://sn.archive.ubuntu.com/ubuntu xenial/universe amd64 pigz amd64 2.3.1-2 [61,1 kB]
```

Une fois l'installation réussie de Docker Community Edition terminée, le service démarre automatiquement. La commande ci-dessous permettra de vérifier l'état du service.

```
root@ubuntu-ESPRIMO: ~  
root@ubuntu-ESPRIMO: ~ x root@ubuntu-ESPRIMO: /var/www/html/rest-api... x root@ubuntu-ESPRIMO: ~ x  
root@ubuntu-ESPRIMO:~# systemctl status docker  
● docker.service - Docker Application Container Engine  
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)  
   Active: active (running) since alt 2019-11-04 15:08:45 GMT; 4min 38s ago  
     Docs: https://docs.docker.com  
  Main PID: 12843 (dockerd)  
    Tasks: 10  
   Memory: 42.9M  
      CPU: 457ms  
   CGroup: /system.slice/docker.service  
           └─12843 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock  
  
now 04 15:08:39 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:39.506357953Z" level=warning msg="Your  
now 04 15:08:39 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:39.506398510Z" level=warning msg="Your  
now 04 15:08:39 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:39.506412536Z" level=warning msg="Your  
now 04 15:08:39 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:39.506929653Z" level=info msg="Loading  
now 04 15:08:40 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:40.240035259Z" level=info msg="Default  
now 04 15:08:41 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:41.449971385Z" level=info msg="Loading  
now 04 15:08:44 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:44.433838836Z" level=info msg="Docker d  
now 04 15:08:44 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:44.433981360Z" level=info msg="Daemon h  
now 04 15:08:45 ubuntu-ESPRIMO systemd[1]: Started Docker Application Container Engine.  
now 04 15:08:45 ubuntu-ESPRIMO dockerd[12843]: time="2019-11-04T15:08:45.297482956Z" level=info msg="API list  
lines 1-21/21 (END)
```

## II – IMAGES DOCKER

Une image est un package qui inclut tout ce qui est nécessaire à l'exécution d'une application, à savoir:

- L'exécution
- Les variables d'environnement
- Les bibliothèques
- Les fichiers de configuration

### Rechercher une image depuis le docker hub

```
root@ubuntu-VirtualBox:~# docker search ubuntu
```

NAME	STARS	OFFICIAL	DESCRIPTION AUTOMATED
ubuntu			Ubuntu is a Debian-based Lin
ux operating sys...	10420	[OK]	
dorowu/ubuntu-desktop-lxde-vnc			Docker image to provide HTML
5 VNC interface ...	385		[OK]
rastasheep/ubuntu-sshd			Dockerized SSH service, buil
t on top of offi...	240		[OK]
consol/ubuntu-xfce-vnc			Ubuntu container with "headl
ess" VNC session...	208		[OK]
ubuntu-upstart			Upstart is an event-based re
placement for th...	103	[OK]	
ansible/ubuntu14.04-ansible			Ubuntu 14.04 LTS with ansibl
e	98		[OK]
neurodebian			NeuroDebian provides neurosc
ience research s...	63	[OK]	
land1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5			ubuntu-16-nginx-php-phpmyadm
in-mysql-5	50		[OK]
ubuntu-debootstrap			debootstrap --variant=minbas
e --components=m...	42	[OK]	
nuagebec/ubuntu			Simple always updated Ubuntu
docker images w...	24		[OK]

### Télécharger une image

```

root@ubuntu-VirtualBox:~# docker pull rastasheep/ubuntu-sshd
Using default tag: latest
latest: Pulling from rastasheep/ubuntu-sshd
a48c500ed24e: Pull complete
1e1de00ff7e1: Pull complete
0330ca45a200: Pull complete
471db38bcfbf: Pull complete
0b4aba487617: Pull complete
b42109ad2a3d: Pull complete
dde737735b18: Pull complete
d836c14266f7: Pull complete
5ed86b5d4a15: Pull complete
5273c120f396: Pull complete
b0299e0551df: Pull complete
0ae38e059780: Pull complete
ca79c723275f: Pull complete
Digest: sha256:1a4010f95f6b3292f95fb26e442f85885d523f9a0bb82027b718df62fdd0d9e9
Status: Downloaded newer image for rastasheep/ubuntu-sshd:latest
docker.io/rastasheep/ubuntu-sshd:latest
root@ubuntu-VirtualBox:~#

```

## Lister les images

```

root@ubuntu-VirtualBox:~# docker image ls

```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rastasheep/ubuntu-sshd	latest	49533628fb37	20 months ago	34MB

```

root@ubuntu-VirtualBox:~#

```

**NB:** On peut également lister avec la commande **docker image ls**

## Inspecter une image

```

root@ubuntu-VirtualBox:~# docker inspect rastasheep/ubuntu-sshd:latest
[
  {
    "Id": "sha256:49533628fb371c9f1952c06cedf912c78a81fbe3914901334673c369376e077e",
    "RepoTags": [
      "rastasheep/ubuntu-sshd:latest"
    ],
    "RepoDigests": [
      "rastasheep/ubuntu-sshd@sha256:1a4010f95f6b3292f95fb26e442f85885d523f9a0bb82027b718df62fdd0d9e9"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2018-05-20T11:41:21.71259791Z",
    "Container": "214b3850ca9ff6597b3e0018e498bc124bfde159badba33062567c0436e22a2c",
    "ContainerConfig": {
      "Hostname": "214b3850ca9f",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {

```

## Supprimer une image

```

root@ubuntu-VirtualBox:~# docker rmi rastasheep/ubuntu-sshd:latest
Untagged: rastasheep/ubuntu-sshd:latest
Untagged: rastasheep/ubuntu-sshd@sha256:1a4010f95f6b3292f95fb26e442f85885d523f9a0bb82027b718df62fdd0d9e9
Deleted: sha256:49533628fb371c9f1952c06cedf912c78a81fbe3914901334673c369376e077e
Deleted: sha256:960e66412cf588185f24f8fac03db74594a546607a933dcf592f1002ddb77c57
Deleted: sha256:eefb4d470ad31e2c91e69f207761ebc12ef1405e6f6737bf18c076430fda21dc
Deleted: sha256:76c35b3da31b0f597563a70d5301fc4677958c226ad02c810e290bba1197e01d
Deleted: sha256:f0065152c0699148241aa873b38ffa07726caae9d1a9f596fd2fe0379d47a157
Deleted: sha256:7fc4a9e903ef12585776f004287eeec3cabfb5b8e55c642d94970c529de08217
Deleted: sha256:974567066a60a8394e7ebd794e6858d3e1af15e56ea41778b070c5408e1015ca
Deleted: sha256:cee9caddbec862fac6fdc92a68d42bccb8403e45eb522262aaecacd539c93e26
Deleted: sha256:2d7fd33194935c70e602350debb24e030629ada77eda2c0a77ca88e19d80afb3
Deleted: sha256:96fccbf869d3c0ee0fb2e976fdf356dc5872f6410030fd094bbc5b34a7559cdb
Deleted: sha256:38ffa1479cb9fd81d0d4d057c282a155a4a83bff5d2b507ee9563f996d74272d
Deleted: sha256:cc6967c5525a55626688a773e4fe578321a2e126a3b1df1bc0763cfd1583c50c
Deleted: sha256:2a2d486f02032f5a6cc56290a244512daa07a8efe0124bccc5701f0a778aa947
Deleted: sha256:65bdd50ee76a485049a2d3c2e92438ac379348e7b576783669dac6f604f6241b
root@ubuntu-VirtualBox:~# █

```

## Supprimer toutes les images

```

root@innonce-Lenovo:~# docker rmi -f $(docker images -aq) █

```



### III – CONTENEURS DOCKER

#### Créer un conteneur à partir d'une image

```
root@ubuntu-VirtualBox:~# docker run -it --name africains rastasheep/ubuntu-sshd bash
root@f16578e90925:/#
```

#### Liste les conteneurs

```
root@ubuntu-VirtualBox:~# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
13b6c247b413       rastasheep/ubuntu-sshd "bash"             13 seconds ago
Up 5 seconds       22/tcp            africains
root@ubuntu-VirtualBox:~#
```

#### Processus

```
root@d46ba1553797:/ root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~#
root@ubuntu-VirtualBox:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
d46ba1553797       rastasheep/ubuntu-sshd "bash"             24 seconds ago
Up 16 seconds      22/tcp            africains
root@ubuntu-VirtualBox:~#
```

```
root@ubuntu-VirtualBox:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
ecc82dc6a988       rastasheep/ubuntu-sshd "bash"             22 seconds ago
Exited (0) 8 seconds ago                                machine2
0d7bcdb31bb8       phpmyadmin/phpmyadmin "/docker-entrypoint..." 2 minutes ago
Exited (0) About a minute ago                            machine1
d46ba1553797       rastasheep/ubuntu-sshd "bash"             12 minutes ago
Up 12 minutes      22/tcp            africains
root@ubuntu-VirtualBox:~#
```

#### Inspecter un conteneur



```
root@ubuntu-VirtualBox:~# docker inspect africains
[
  {
    "Id": "a3d93f975eb6190f15c4c46d8941984466d481b13ad0c8becfa586055cf0dc68",
    "Created": "2020-01-27T15:12:45.471653928Z",
    "Path": "bash",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 20779,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2020-01-27T15:12:56.272210016Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:49533628fb371c9f1952c06cedf912c78a81fbe3914901334673c369376e077e",
    "ResolvConfPath": "/var/lib/docker/containers/a3d93f975eb6190f15c4c46d8941984466d481b13ad0c8becfa586055cf0dc68/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/a3d93f975eb6190f15c4c46d89419844
```

### Supprimer un conteneur

```
root@ubuntu-VirtualBox:~# docker rm africains
africains
root@ubuntu-VirtualBox:~#
```

### Supprimer tous les conteneurs

```
root@ubuntu-VirtualBox:~# docker rm -f $(docker ps -aq)
8b470c4e45ff
e0ff9801c48a
a3d93f975eb6
root@ubuntu-VirtualBox:~#
```

### Exécuter une commande dans un conteneur

```

root@ubuntu-VirtualBox:~# docker exec africains ls -l
total 64
drwxr-xr-x 1 root root 4096 May 20 2018 bin
drwxr-xr-x 2 root root 4096 Apr 24 2018 boot
drwxr-xr-x 5 root root 360 Jan 27 16:02 dev
drwxr-xr-x 1 root root 4096 Jan 27 16:02 etc
drwxr-xr-x 2 root root 4096 Apr 24 2018 home
drwxr-xr-x 1 root root 4096 May 20 2018 lib
drwxr-xr-x 2 root root 4096 Apr 26 2018 lib64
drwxr-xr-x 2 root root 4096 Apr 26 2018 media
drwxr-xr-x 2 root root 4096 Apr 26 2018 mnt
drwxr-xr-x 2 root root 4096 Apr 26 2018 opt
dr-xr-xr-x 250 root root 0 Jan 27 16:02 proc
drwx----- 1 root root 4096 May 20 2018 root
drwxr-xr-x 1 root root 4096 May 20 2018 run
drwxr-xr-x 1 root root 4096 May 20 2018 sbin
drwxr-xr-x 2 root root 4096 Apr 26 2018 srv
dr-xr-xr-x 13 root root 0 Jan 27 16:02 sys
drwxrwxrwt 1 root root 4096 May 20 2018 tmp
drwxr-xr-x 1 root root 4096 May 20 2018 usr
drwxr-xr-x 1 root root 4096 May 20 2018 var
root@ubuntu-VirtualBox:~#

```

## Quelques manipulations...

### La sauvegarde

#### – Locale

```

root@innonce-Lenovo:~# docker commit 8d1b710d2743 image_sauv
sha256:2256197dca685d03e9ec9c81a508d6dfb4e1bfdf2ff97a88d9a3095691799e2a
root@innonce-Lenovo:~#

```

A noter que le commit peut-être fait à partir de l'id ou du nom donné au conteneur.

#### – exportable(format tar)

```

root@innonce-Lenovo:~# docker save image_sauv > image_sauv.tar
root@innonce-Lenovo:~# ls image_sauv.tar
image_sauv.tar
root@innonce-Lenovo:~#

```

### La restauration

#### – Locale

```
root@innonce-Lenovo:~# docker run -it --name pc image_sauv bash
root@9ce4b3f46ea2:/#
```

– exportable

```
root@innonce-Lenovo:~# docker load < image_sauv.tar
Loaded image: image_sauv:latest
root@innonce-Lenovo:~#
```

## IV – RÉSEAUX DOCKER

### Présentation

#### 1 – Le driver bridge

Le driver bridge est le réseau par défaut suite à l'installation de Docker. Il crée automatiquement un réseau nommé **bridge** connecté à l'interface réseau

Nous pouvons comme suite:

```
root@innonce-Lenovo:~# ifconfig docker0
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:a1ff:fe17:c175 prefixlen 64 scopeid 0x20<link>
    ether 02:42:a1:17:c1:75 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 115 bytes 14413 (14.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Chaque nouveau conteneur est automatiquement connecté à ce réseau sauf si un réseau personnalisé est spécifié.

Il est le réseau couramment utilisé et les conteneurs utilisant ce driver peuvent communiquer entre eux, cependant ils ne sont pas accessibles depuis l'extérieur.

L'alternative permettant aux conteneurs du réseau bridge de communiquer ou d'être accessibles au monde extérieur consiste à configurer le mappage des ports.

## Test

```
root@innonce-Lenovo:~# docker run -it --name machine1 conteneurserver bash
root@9f29eea7f414:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.4 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:ac:11:00:04 txqueuelen 0 (Ethernet)
    RX packets 22 bytes 3225 (3.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
root@innonce-Lenovo:~# docker run -it --name machine_2 conteneurserver bash
root@b966ccb15b3c:/# ping 172.17.0.4
PING 172.17.0.4 (172.17.0.4): 56 data bytes
64 bytes from 172.17.0.4: icmp_seq=0 ttl=64 time=0.358 ms
64 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.174 ms
64 bytes from 172.17.0.4: icmp_seq=2 ttl=64 time=0.258 ms
^C--- 172.17.0.4 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.174/0.263/0.358/0.075 ms
root@b966ccb15b3c:/#
```

## 2 – Le driver none

Ce type de réseau permet d'interdire toute communication interne et externe avec le conteneur car le conteneur sera dépourvu de toute interface réseau (sauf l'interface de **loopback**).

```
root@innonce-Lenovo:~# docker run -it --name pc --network none conteneurserver bash
root@b391ecd7d09f:/# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@b391ecd7d09f:/#
```

## 3 – Le driver host

Le type de driver **host** permet aux conteneurs d'utiliser la même interface que l'hôte. Il supprime l'isolation réseau entre les conteneurs et seront par défaut accessibles de l'extérieur et prendra de ce fait la même adresse IP que la machine hôte.

## test

```
root@innonce-Lenovo:~# docker run --rm --network host -it --name machine_pc conteneurserver bash
root@innonce-Lenovo:/# ifconfig enp1s0
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.243 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::7c8b:174c:4b92:5ebd prefixlen 64 scopeid 0x20<link>
    ether b8:88:e3:92:f9:78 txqueuelen 1000 (Ethernet)
    RX packets 220472 bytes 113418943 (113.4 MB)
    RX errors 2 dropped 0 overruns 0 frame 2
    TX packets 58265 bytes 7865926 (7.8 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 16
```

Nous pouvons remarquer que le conteneur se présente comme notre machine physique

Nous constatons la même sortie de la commande «**ifconfig enp1s0**» sur la machine hôte mais il faut bien prendre en compte que la seconde commande «**hostnamectl**» va générer une erreur si on l'exécute sur le conteneur.

```
root@innonce-Lenovo:~# ifconfig enp1s0
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.243 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::7c8b:174c:4b92:5ebd prefixlen 64 scopeid 0x20<link>
    ether b8:88:e3:92:f9:78 txqueuelen 1000 (Ethernet)
    RX packets 227876 bytes 114588180 (114.5 MB)
    RX errors 2 dropped 0 overruns 0 frame 2
    TX packets 58668 bytes 7950581 (7.9 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 16

root@innonce-Lenovo:~# hostnamectl
Static hostname: innonce-Lenovo
Icon name: computer-laptop
Chassis: laptop
Machine ID: 01ec1b4836e7444086bb525082a5d238
Boot ID: 3e20b4a1bec44689b04cc0922111e098
Operating System: Ubuntu 18.04.3 LTS
Kernel: Linux 4.15.0-74-generic
Architecture: x86_64
```

## 4 – Le driver overlay

Pour la mise en place d'un driver de type overlay nous utiliserons deux hôtes Docker. La première sera une machine physique et la seconde une machine virtuelle également dont les conteneurs pourrons communiquer.

### Machine1

### – Générer un jeton (token)

```
root@ubuntu-VirtualBox:~# docker swarm init
Swarm initialized: current node (rk1r5byf0ng10hty5vx2vu3ko) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3d8sy3287sgt6ss4r2tagmaqz7egr51yetlwab1k3nzgdnmrv
c-0pajaovdafzymv864k3k8v5nu 192.168.1.135:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

L'information colorée contient le jeton qui sera envoyé au niveau des différentes machines afin que les conteneurs soient intégrés dans le réseau.

A noter que l'option `--advertise-addr` permet de spécifier l'adresse IP de l'interface qui communique avec les autres.

### – Créer un réseau

```
root@ubuntu-VirtualBox:~# docker network create --driver=overlay --attachable
reseau-overlay
m5rh7v7ol8q3c37j95wn9lyuf
root@ubuntu-VirtualBox:~#
```

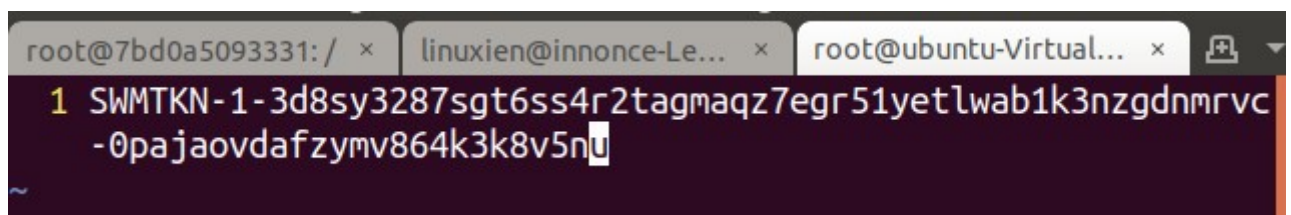
L'Id retourné sera reçu lorsque nous nous connecterons.

### – Lancer un conteneur dans le réseau

```
root@ubuntu-VirtualBox:~# docker run -it --name machine1 --network reseau-overlay
ubuntu bash
root@d4732bc25320:/#
```

### – Envoie du Token

Nous allons créer un fichier contenant le token généré et qui sera envoyé aux différentes machines



The screenshot shows a terminal window with three tabs: 'root@7bd0a5093331: /', 'linuxien@innonce-Le...', and 'root@ubuntu-Virtual...'. The active tab is the third one. It shows the token 'SWMTKN-1-3d8sy3287sgt6ss4r2tagmaqz7egr51yetlwab1k3nzgdnmrv c-0pajaovdafzymv864k3k8v5nu' being copied from a previous command output. The token is highlighted in yellow.

Envoie du token par SSH au niveau de la machine2



```

root@ubuntu-VirtualBox:~# scp jeton linuxien@192.168.1.243:.
linuxien@192.168.1.243's password:
jeton                                100% 86   76.4KB/s   00:00
root@ubuntu-VirtualBox:~# █

```

## Machine 2

Au niveau de la machine2 nous allons rejoindre le réseau grâce au jeton reçu

```

root@innonce-Lenovo:~# docker swarm join --token SWMTKN-1-3d8sy3287sgt6ss4r2tagmaqz7egr51yetlwab1k3nzgdnmrv
c-0pajaovdafzymv864k3k8v5nu 192.168.1.135:2377
This node joined a swarm as a worker.
root@innonce-Lenovo:~# █

```

En listant les différents réseaux on peut remarquer le driver overlay

```

root@innonce-Lenovo:~# docker network ls

```

NETWORK ID	NAME	DRIVER	SCOPE
752d04b4d813	bridge	bridge	local
b0dba9e9cb56	docker_gwbridge	bridge	local
196b2783fed1	host	host	local
wbgwbtmajqru	ingress	overlay	swarm
8d7a566271d9	none	null	local

```

root@innonce-Lenovo:~# █

```

### – Connecter un conteneur de la machine2 au réseau

```

root@innonce-Lenovo:~# docker run -it --name Machine2 --network reseau-overlay
conteneurserver bash
root@7bd0a5093331:/# █

```

### – Test de connectivité

```

root@7bd0a5093331:/# ping machine1
PING machine1 (10.0.1.2): 56 data bytes
64 bytes from 10.0.1.2: icmp_seq=0 ttl=64 time=1.245 ms
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=1.221 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.838 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=1.220 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=0.793 ms
^C--- machine1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.793/1.063/1.245/0.203 ms
root@7bd0a5093331:/# █

```



– Pour quitter le réseau overlay

```
root@innonce-Lenovo:~# docker swarm leave
Node left the swarm.
root@innonce-Lenovo:~#
```

– Supprimer le réseau

```
root@innonce-Lenovo:~# docker network rm reseau-overlay
reseau-overlay
root@innonce-Lenovo:~#
```

## 5 – Le driver macvlan

Le driver Macvlan permet d'attribuer une adresse MAC à un conteneur le faisant apparaître comme un périphérique physique sur le réseau.

En effet le moteur Docker route le trafic vers les conteneurs en fonction de leurs adresses MAC. Les conteneurs dans un même réseau de type Macvlan peuvent communiquer et ceux dans des réseaux différents devront faire appel aux notions de routage inter-vlan pour un éventuel dialogue.

Pour la mise en place, nous allons créer deux réseaux contenant chacun deux conteneurs

– Création des réseaux

réseau vlan\_net1

```
root@innonce-Lenovo:~# docker network create -d macvlan --subnet=172.16.10.0/24
--gateway=172.16.10.1 -o parent=enp1s0.10 vlan_net1
6ab2758b57ee4525173e498fdefdfa469d5f2a3b349ded962a2d90aad1307bba
root@innonce-Lenovo:~#
```

réseau vlan\_net2

```
root@innonce-Lenovo:~# docker network create -d macvlan --subnet=172.16.20.0/24
--gateway=172.16.20.1 -o parent=enp1s0.20 vlan_net2
3e802d95f1a85bcb43a426478fafb2259384f852bf9234259032b4bc8d3f8a04
root@innonce-Lenovo:~#
```

## – Intégration des conteneurs dans le réseau «vlan\_net1» et test de connectivité

```
root@innonce-Lenovo:~# docker run -it --name pc1 --network vlan_net1 conteneurserver
bash
root@977dd5cf8060:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.10.2 netmask 255.255.255.0 broadcast 172.16.10.255
    ether 02:42:ac:10:0a:02 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
root@innonce-Lenovo:~# docker run -it --name pc2 --network vlan_net1 conteneurserver
bash
root@b4ac0cd921e6:/# ping 172.16.10.2
PING 172.16.10.2 (172.16.10.2): 56 data bytes
64 bytes from 172.16.10.2: icmp_seq=0 ttl=64 time=0.162 ms
64 bytes from 172.16.10.2: icmp_seq=1 ttl=64 time=0.183 ms
64 bytes from 172.16.10.2: icmp_seq=2 ttl=64 time=0.332 ms
^C--- 172.16.10.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.162/0.226/0.332/0.076 ms
root@b4ac0cd921e6:/#
```

En ajoutant le second conteneur dans le réseau, nous avons pu joindre le premier qui a l'adresse IP 176.16.10.2

Nous allons à présent intégrer un conteneur dans le réseau «vlan\_net2» et voir que le test de connectivité vers un conteneur du réseau vlan\_net1 en occurrence de PC1 ayant l'adresse IP 176.16.10.2 sera négatif.

```
root@innonce-Lenovo:~# docker run -it --name pc3 --network vlan_net2 conteneurserver
bash
root@92fbb408b9a1:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.20.2 netmask 255.255.255.0 broadcast 172.16.20.255
    ether 02:42:ac:10:14:02 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
root@92fbb408b9a1:/# ping 176.16.10.2
PING 176.16.10.2 (176.16.10.2): 56 data bytes
92 bytes from 92fbb408b9a1 (172.16.20.2): Destination Host Unreachable
92 bytes from 92fbb408b9a1 (172.16.20.2): Destination Host Unreachable
92 bytes from 92fbb408b9a1 (172.16.20.2): Destination Host Unreachable
92 bytes from 92fbb408b9a1 (172.16.20.2): Destination Host Unreachable
^C--- 176.16.10.2 ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
root@92fbb408b9a1:/#
```

## V – API DOCKER

L'API Engine est une API HTTP issu de Docker Engine. Il s'agit de l'API que le client Docker utilise pour communiquer avec le moteur, donc tout ce que le client Docker fait peut-être fait également avec l'API.

La plupart des commandes du client sont mappées directement aux points de terminaison API à l'exception notable de l'exécution de conteneurs qui se composent de plusieurs appels d'API. L'API utilise des codes d'état HTTP standard pour indiquer la réussite ou l'échec de l'appel d'API et le corps de réponse sera en format JSON.

Pour nos différents tests nous pouvons utiliser l'utilitaire **CURL** qui est une bibliothèque de requêtes aux URL pour client. Le logiciel **Postman** est un autre moyen permettant d'effectuer des tests.

### Prise en compte des API avec Docker

Éditer le fichier `/lib/systemd/system/docker.service` et ajouter option `-H tcp://0.0.0.0:4243`. Cela permet d'activer les API en utilisant le protocole tcp sur n'importe quelle adresse IP avec le port 4243.

```
1 [Unit]
2 Description=Docker Application Container Engine
3 Documentation=https://docs.docker.com
4 BindsTo=containerd.service
5 After=network-online.target firewalld.service containerd.service
6 Wants=network-online.target
7 Requires=docker.socket
8
9 [Service]
10 Type=notify
11 # the default is not to use systemd for cgroups because the delegate issues still
12 # exists and systemd currently does not support the cgroup feature set required
13 # for containers run by docker
14 ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:4243 --containerd=/run/containerd/containerd.sock
15 ExecReload=/bin/kill -s HUP $MAINPID
16 TimeoutSec=0
17 RestartSec=2
18 Restart=always
```

Redémarrer le service et vérifier que le port 4243 est bien en écoute

```
root@innonce-Lenovo:~# systemctl restart docker.service
root@innonce-Lenovo:~# netstat -anp | grep -w 4243
tcp6      0      0 :::4243          :::*              LISTEN      5767/dockerd
root@innonce-Lenovo:~#
```

# Images

## – Lister les images

```
root@innonce-Lenovo:~# curl -X GET http://localhost:4243/images/json
[{"Containers": -1, "Created": 1580318040, "Id": "sha256:ae59e44ffa2d71e73664fc47e68f0959055f564bbd902df43c8e2eecbafcc86e", "Labels": {}, "ParentId": "sha256:549b9b86cb8d75a2b668c21c50ee092716d070f129fd1493f95ab7e43767eab8", "RepoDigests": null, "RepoTags": ["serveurweb:latest"], "SharedSize": -1, "Size": 256790072, "VirtualSize": 256790072}, {"Containers": -1, "Created": 1580312387, "Id": "sha256:8a0109fd9c5d8ff86aab0c287a991ed7b354ee63d7d27f5975ce8e967b8fcde8", "Labels": {}, "ParentId": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb", "RepoDigests": null, "RepoTags": ["serveurdb:latest"], "SharedSize": -1, "Size": 465244880, "VirtualSize": 465244880}, {"Containers": -1, "Created": 1579193123, "Id": "sha256:ca9ab479b8ac356dcadb0331d8eef7e69e57bce9d05c3e3a0c2864fba70a10c", "Labels": {}, "ParentId": "sha256:549b9b86cb8d75a2b668c21c50ee092716d070f129fd1493f95ab7e43767eab8", "RepoDigests": null, "RepoTags": ["conteneurserver:latest"], "SharedSize": -1, "Size": 810677329, "VirtualSize": 810677329}, {"Containers": -1, "Created": 1579044058, "Id": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb", "Labels": null, "ParentId": "", "RepoDigests": ["mysql@sha256:f1df505c4c6e8eae599a0482e3bde3e761cd700c00cbc371a8161648a26817c0"], "RepoTags": ["mysql:latest"], "SharedSize": -
```

L'utilitaire **CURL** nous donne un résultat brute contrairement à **POSTMAN** comme le montre la capture ci dessous

The screenshot shows the Postman interface. At the top, the method is set to 'GET' and the URL is 'http://localhost:4243/images/json'. The 'Send' button is visible. Below the URL bar, the 'Body' tab is selected, showing a JSON response. The response is formatted as 'Pretty' and shows a list of three container objects. The status is '200 OK', time is '103ms', and size is '3.79 KB'. The JSON response is as follows:

```
{
  "Containers": -1,
  "Created": 1580318040,
  "Id": "sha256:ae59e44ffa2d71e73664fc47e68f0959055f564bbd902df43c8e2eecbafcc86e",
  "Labels": {},
  "ParentId": "sha256:549b9b86cb8d75a2b668c21c50ee092716d070f129fd1493f95ab7e43767eab8",
  "RepoDigests": null,
  "RepoTags": [
    "serveurweb:latest"
  ],
  "SharedSize": -1,
  "Size": 256790072,
  "VirtualSize": 256790072
}, {
  "Containers": -1,
  "Created": 1580312387,
  "Id": "sha256:8a0109fd9c5d8ff86aab0c287a991ed7b354ee63d7d27f5975ce8e967b8fcde8",
  "Labels": {},
  "ParentId": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb",
  "RepoDigests": null,
  "RepoTags": [
    "serveurdb:latest"
  ],
  "SharedSize": -1,
  "Size": 465244880,
  "VirtualSize": 465244880
}, {
  "Containers": -1,
  "Created": 1579193123,
  "Id": "sha256:ca9ab479b8ac356dcadb0331d8eef7e69e57bce9d05c3e3a0c2864fba70a10c",
  "Labels": {},
  "ParentId": "sha256:549b9b86cb8d75a2b668c21c50ee092716d070f129fd1493f95ab7e43767eab8",
  "RepoDigests": null,
  "RepoTags": [
    "conteneurserver:latest"
  ],
  "SharedSize": -1,
  "Size": 810677329,
  "VirtualSize": 810677329
}, {
  "Containers": -1,
  "Created": 1579044058,
  "Id": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb",
  "Labels": null,
  "ParentId": "",
  "RepoDigests": [
    "mysql@sha256:f1df505c4c6e8eae599a0482e3bde3e761cd700c00cbc371a8161648a26817c0"
  ],
  "RepoTags": [
    "mysql:latest"
  ],
  "SharedSize": -1,
  "Size": 810677329,
  "VirtualSize": 810677329
}]
```

## – Rechercher une image

```
GET http://localhost:4243/images/search?term=sshd Send Save

Pretty Raw Preview Visualize BETA JSON

1 {
2   {
3     "star_count": 32,
4     "is_official": false,
5     "name": "panubo/sshd",
6     "is_automated": true,
7     "description": "Minimal Alpine Linux Docker container with sshd exposed and rsync installed."
8   },
9   {
10    "star_count": 3,
11    "is_official": false,
12    "name": "kubernetesio/sshd-jumpserver",
13    "is_automated": true,
14    "description": "sshd-jumpserver"
15  },
16  {
17    "star_count": 6,
18    "is_official": false,
19    "name": "macropin/sshd",
20    "is_automated": true,
21    "description": "(moved) Use docker.io/panubo/sshd"
22  },
23  {
24    "star_count": 2,
25    "is_official": false,
26    "name": "linuxkit/sshd",
27    "is_automated": false
```

## Paramètres de requêtes

- **term** terme à rechercher
- **limit** résultat de la recherche
- **filters** une valeur codée JSON des filtres à traiter dans la liste des conteneurs.

## – Créer une image

```
POST http://localhost:4243/images/create?fromImage=busybox&tag=latest Send Save

Body Cookies Headers (7) Test Results Status: 200 OK Time: 8.87s Size: 2.77 KB Save Response

Pretty Raw Preview Visualize BETA JSON

1 {
2   "status": "Pulling from library/busybox",
3   "id": "latest"
4 }
5 {
6   "status": "Pulling fs layer",
7   "progressDetail": {},
8   "id": "bdbbaa22dec6"
9 }
10 {
11   "status": "Downloading",
12   "progressDetail": {
13     "current": 8214,
14     "total": 760984
15   },
16   "progress": "[> 8.214kB/761kB]",
17   "id": "bdbbaa22dec6"
18 }
19 {
20   "status": "Downloading",
21   "progressDetail": {
22     "current": 49196,
23     "total": 760984
24   },
```

## Paramètres de requêtes

– **fromImage:** Nom de l'image à extraire. Le nom peut inclure une étiquette ou un résumé. Ce paramètre ne peut-être utilisé que lors de l'extraction d'une image.

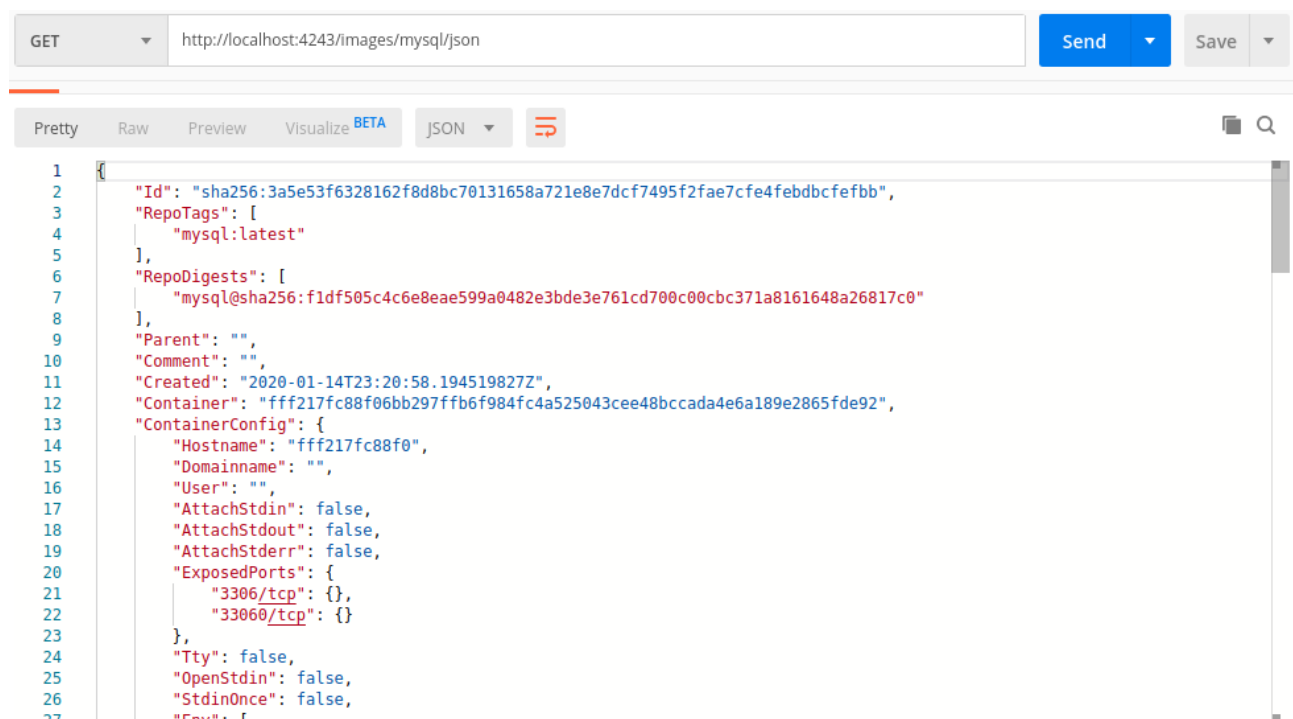
La traction est annulée si la connexion HTTP est fermée.

– **fromSrc:** Source à importer. La valeur peut être récupérée ou pour lire l'image à partir du corps de la demande. Ce paramètre ne peut-être utilisé que lors de l'importation d'une image.

– **repo:** Nom de référentiel attribué à une image lors de son importation. Le dépôt peut inclure une étiquette. Ce paramètre ne peut-être utilisé que lors de l'importation d'une image.

– **tag:** Tag ou digest. S'il est vide lors de l'extraction d'une image, cela entraîne l'extraction de toutes les balises de l'image donnée.

## – Inspecter une image



```
1  {
2    "Id": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb",
3    "RepoTags": [
4      "mysql:latest"
5    ],
6    "RepoDigests": [
7      "mysql@sha256:f1df505c4c6e8eae599a0482e3bde3e761cd700c00cbc371a8161648a26817c0"
8    ],
9    "Parent": "",
10   "Comment": "",
11   "Created": "2020-01-14T23:20:58.194519827Z",
12   "Container": "fff217fc88f06bb297ffb6f984fc4a525043cee48bccada4e6a189e2865fde92",
13   "ContainerConfig": {
14     "Hostname": "fff217fc88f0",
15     "Domainname": "",
16     "User": "",
17     "AttachStdin": false,
18     "AttachStdout": false,
19     "AttachStderr": false,
20     "ExposedPorts": {
21       "3306/tcp": {},
22       "33060/tcp": {}
23     },
24     "Tty": false,
25     "OpenStdin": false,
26     "StdinOnce": false,
27     "Env": [
```

## – Obtenir l'historique d'une image

GET http://localhost:4243/images/mysql/history Send

Pretty Raw Preview Visualize BETA JSON

```

1 [
2   {
3     "Comment": "",
4     "Created": 1579044058,
5     "CreatedBy": "/bin/sh -c #(nop) CMD [\"mysql\"]",
6     "Id": "sha256:3a5e53f6328162f8d8bc70131658a721e8e7dcf7495f2fae7cfe4febdbcfefbb",
7     "Size": 0,
8     "Tags": [
9       "mysql:latest"
10    ]
11  },
12  {
13    "Comment": "",
14    "Created": 1579044058,
15    "CreatedBy": "/bin/sh -c #(nop) EXPOSE 3306 33060",
16    "Id": "<missing>",
17    "Size": 0,
18    "Tags": null
19  },
20  {
21    "Comment": "",
22    "Created": 1579044057,
23    "CreatedBy": "/bin/sh -c #(nop) ENTRYPOINT [\"docker-entrypoint.sh\"]",
24    "Id": "<missing>",
25    "Size": 0,
26    "Tags": null
27  }
28 ]

```

## – Supprimer une image

DELETE http://localhost:4243/images/mysql/mysql-router Send Save

Body Cookies Headers (7) Test Results Status: 200 OK Time: 543ms Size: 696 B Save Response

Pretty Raw Preview Visualize BETA JSON

```

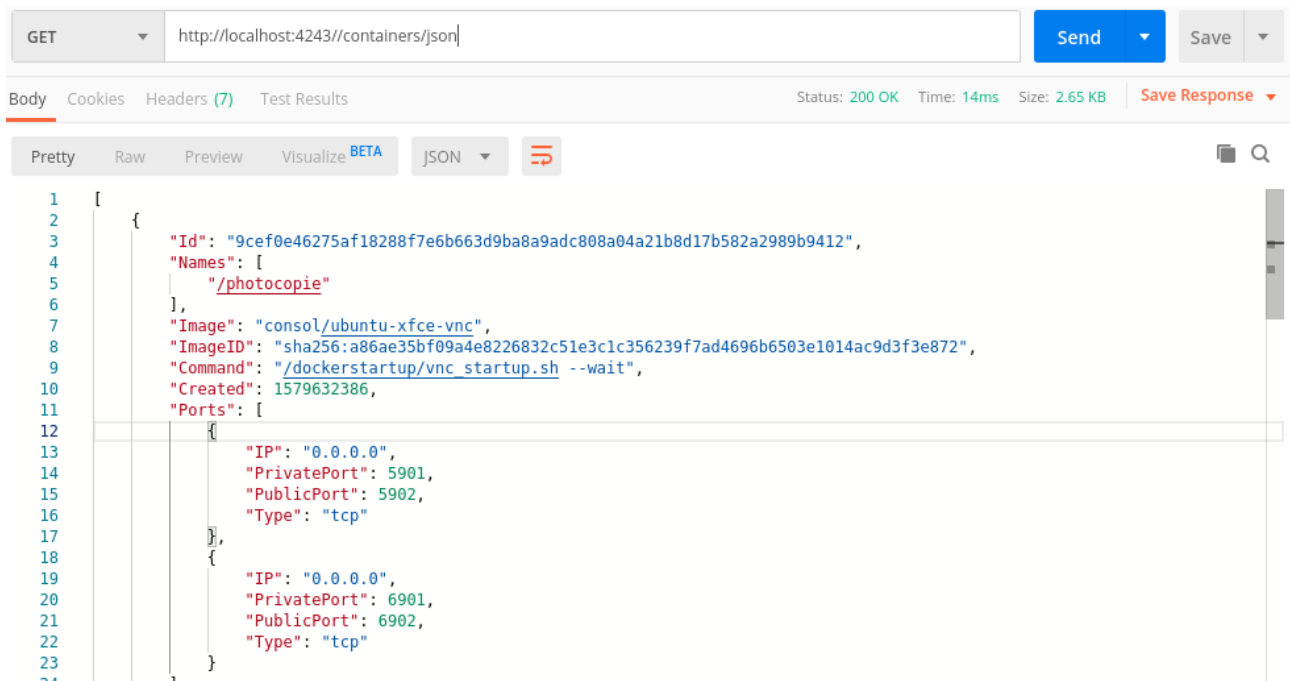
1 [
2   {
3     "Untagged": "mysql/mysql-router:latest"
4   },
5   {
6     "Untagged": "mysql/mysql-router@sha256:29e7160c51114e76dadbfd6fbaa9c01d2b3b3dd2397ddf5a6240ceced6f3e8c7"
7   },
8   {
9     "Deleted": "sha256:0102160128d79cce6ea11caec0fdc9c2e1b24cc275e68d9576bc129a2178a88b"
10  },
11  {
12    "Deleted": "sha256:bcbd35df660c55edaa4056eb6cd0cc861ecb8099e0b4d630a50aba689270896e"
13  },
14  {
15    "Deleted": "sha256:0796ac2ca1f862423f9fb0586fd00ae7cd50163b80508ad3dcc23597fe5f2a2b"
16  },
17  {
18    "Deleted": "sha256:5102fc2ee26e308a905f5f8cb41cf434cd31db05a6a1971afa2bc91d8040018d"
19  }
20 ]

```



# Conteneurs

## – Lister les conteneurs



GET http://localhost:4243/containers/json

Status: 200 OK Time: 14ms Size: 2.65 KB Save Response

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize BETA JSON

```
[
  {
    "Id": "9cef0e46275af18288f7e6b663d9ba8a9adc808a04a21b8d17b582a2989b9412",
    "Names": [
      "/photocopie"
    ],
    "Image": "consol/ubuntu-xfce-vnc",
    "ImageID": "sha256:a86ae35bf09a4e8226832c51e3c1c356239f7ad4696b6503e1014ac9d3f3e872",
    "Command": "/dockerstartup/vnc_startup.sh --wait",
    "Created": 1579632386,
    "Ports": [
      {
        "IP": "0.0.0.0",
        "PrivatePort": 5901,
        "PublicPort": 5902,
        "Type": "tcp"
      },
      {
        "IP": "0.0.0.0",
        "PrivatePort": 6901,
        "PublicPort": 6902,
        "Type": "tcp"
      }
    ]
  }
]
```

## Paramètres de requêtes

- **all**: Afficher tous les conteneurs. Seuls les conteneurs en cours d'exécution sont affichés par défaut.
- **limit**: Afficher les limites conteneurs créés, y compris ceux qui ne sont pas en d'exécution.
- **since**: Afficher uniquement les conteneurs créés depuis l'ID, y compris ceux qui ne sont pas en cours d'exécution.
- **before**: Afficher uniquement les conteneurs créés avant l'ID, y compris ceux qui ne sont pas en cours d'exécution.
- **size**: Afficher la taille des conteneurs.
- **filters**: Une valeur codée en JSON des filtres à traiter dans la liste des conteneurs.

## – Inspecter un conteneur

GET http://localhost:4243/containers/test/json Send Save

Body Cookies Headers (7) Test Results Status: 200 OK Time: 16ms Size: 4.99 KB Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "Id": "731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2",
3   "Created": "2020-02-05T21:52:26.202711301Z",
4   "Path": "bash",
5   "Args": [],
6   "State": {
7     "Status": "running",
8     "Running": true,
9     "Paused": false,
10    "Restarting": false,
11    "OOMKilled": false,
12    "Dead": false,
13    "Pid": 9059,
14    "ExitCode": 0,
15    "Error": "",
16    "StartedAt": "2020-02-05T21:52:31.713847937Z",
17    "FinishedAt": "0001-01-01T00:00:00Z"
18  },
19  "Image": "sha256:ca9ab479b8ac356dcadcb0331d8eef7e69e57bce9d05c3e3a0c2864fba70a10c",
20  "ResolvConfPath": "/var/lib/docker/containers/731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2/resolv.conf",
21  "HostnamePath": "/var/lib/docker/containers/731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2/hostname",
22  "HostsPath": "/var/lib/docker/containers/731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2/hosts",
23  "LogPath": "/var/lib/docker/containers/731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2/731fb4e2f54048cb6fdc0c2a6fe7ab4d7fc059359586e2117674b8a1391d5cd2-json.log",
24 }
```

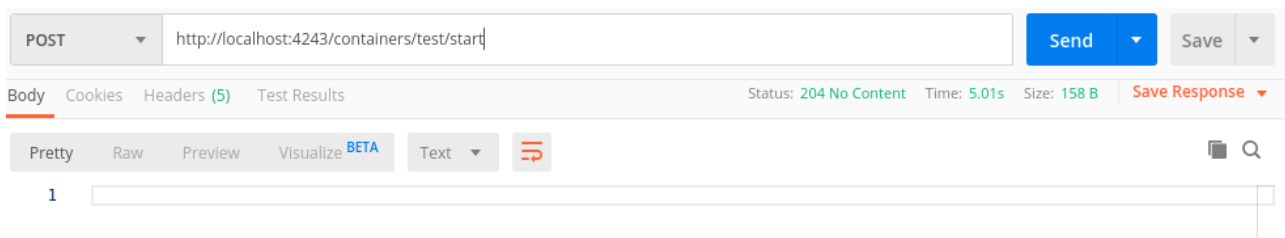
## – Lister des processus en cours d'exécution à l'intérieur d'un conteneur

GET http://localhost:4243/containers/test/top Send Save

Pretty Raw Preview Visualize BETA JSON

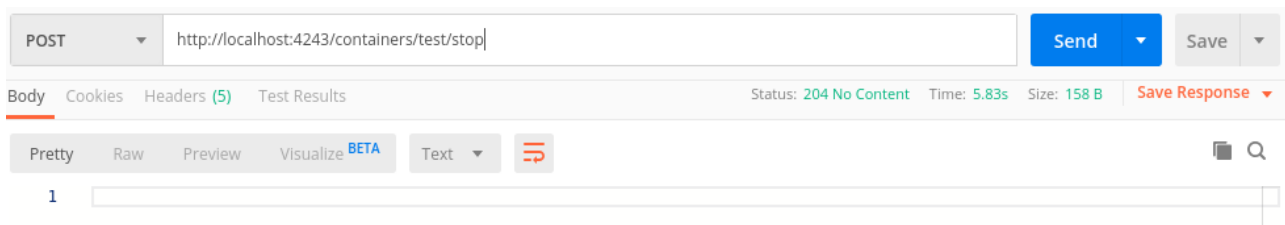
```
1 {
2   "Processes": [
3     [
4       "root",
5       "9059",
6       "9031",
7       "0",
8       "21:52",
9       "pts/0",
10      "00:00:00",
11      "bash"
12    ],
13    [
14      "root",
15      "9847",
16      "9059",
17      "0",
18      "22:01",
19      "pts/0",
20      "00:00:00",
21      "top"
22    ]
23  ],
24  "Titles": [
25    "UID",
26    "PID",
27    "PPID"
28  ]
29 }
```

## – Démarrer un conteneur



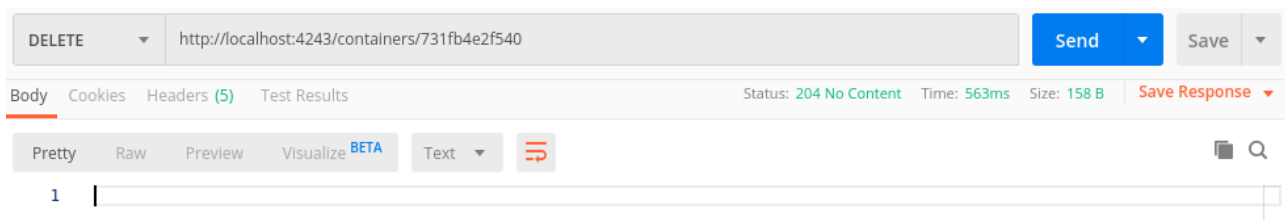
Le code de retour 204 nous indique que tout s'est bien passé.

## – Arrêter un conteneur



Code de retour 204

## – Supprimer un conteneur



Code de retour 204

# DOCKER FILE (**prospective**)