

CERI Avignon

Le DevOps (S-E06-3010) – Semestre 0

TP2 – Docker : Conteneurs et Images

ALDENAWI Hamad

Étudiant(e) : uapv2603127

Fichier de projet

Le Date : /09/2025

Partie 2 – Installation et démarrage de Docker

- 1- Pour afficher la version de Docker, y compris le client et le serveur (daemon), utilise la commande suivante :

```
sudo docker version
```

```
dockeruser@vmdocker:~$ sudo docker version
Client: Docker Engine - Community
  Version:          28.4.0
  API version:     1.51
  Go version:      go1.24.7
  Git commit:      d8eb465
  Built:           Wed Sep  3 20:57:05 2025
  OS/Arch:         linux/amd64
  Context:         default

Server: Docker Engine - Community
  Engine:
    Version:          28.4.0
    API version:     1.51 (minimum version 1.24)
    Go version:      go1.24.7
    Git commit:      249d679
    Built:           Wed Sep  3 20:57:05 2025
    OS/Arch:         linux/amd64
    Experimental:   false
  containerd:
    Version:          1.7.27
    GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
  runc:
    Version:          1.2.5
    GitCommit:        v1.2.5-0-g59923ef
  docker-init:
    Version:          0.19.0
    GitCommit:        de40ad0
dockeruser@vmdocker:~$
```

- 2-Pour voir les composants du daemon Docker qui tournent :

```
ps aux | grep dockerd
```

```
dockeruser@vmdocker:~$ ps aux | grep dockerd
root      56124  0.0  3.6 1899308 72720 ?        Ssl  14:31   0:00 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
dockeruser+ 56690  0.0  0.1   6480  2248 pts/0    S+   14:42   0:00 grep --color=auto dockerd
dockeruser@vmdocker:~$
```

- 3-Pour voir les services et sockets utilisés par Docker ainsi que l'utilisateur qui les a démarrés

```
sudo systemctl list-units --type=service | grep docker
```

```
dockeruser@vmdocker:~$ sudo systemctl list-sockets | grep docker
dockeruser@vmdocker:~$ sudo systemctl list-units --type=service | grep docker
● docker.service                                     loaded active running Docker Application Co
ntainer Engine
dockeruser@vmdocker:~$ sudo systemctl list-sockets | grep docker
dockeruser@vmdocker:~$
```

4- Pour arrêter Docker et vérifier le statut du socket :

■ Arrêter Docker

```
sudo systemctl stop docker
```

■ Vérifier le statut du socket

```
sudo systemctl status docker.socket
```

```
dockeruser@vmdocker:~$ sudo systemctl stop docker
dockeruser@vmdocker:~$ sudo systemctl status docker.socket
● docker.socket - Docker Socket for the API
   Loaded: loaded (/lib/systemd/system/docker.socket; enabled; vendor preset: enabled)
     Active: inactive (dead) since Fri 2025-09-19 14:31:49 UTC; 19min ago
    Triggers: ● docker.service
   Listen: /run/docker.sock (Stream)
      CPU: 933us

sept. 19 14:31:49 vmdocker systemd[1]: Starting Docker Socket for the API...
sept. 19 14:31:49 vmdocker systemd[1]: Listening on Docker Socket for the API.
sept. 19 14:31:49 vmdocker systemd[1]: docker.socket: Deactivated successfully.
sept. 19 14:31:49 vmdocker systemd[1]: Closed Docker Socket for the API.
sept. 19 14:31:49 vmdocker systemd[1]: Stopping Docker Socket for the API...
sept. 19 14:31:49 vmdocker systemd[1]: docker.socket: Socket service docker.service already active, refusing.
sept. 19 14:31:49 vmdocker systemd[1]: Failed to listen on Docker Socket for the API.
sept. 19 14:33:15 vmdocker systemd[1]: docker.socket: Socket service docker.service already active, refusing.
sept. 19 14:33:15 vmdocker systemd[1]: Failed to listen on Docker Socket for the API.
dockeruser@vmdocker:~$
```

5-Désactiver Docker au démarrage

```
sudo systemctl disable docker
```

- Empêche Docker de démarrer automatiquement au boot.
-

Réactiver Docker au démarrage

```
sudo systemctl enable docker
```

- Permet à Docker de démarrer automatiquement au boot.

```
dockeruser@vmdocker:~$ sudo systemctl disable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable docker
Removed /etc/systemd/system/multi-user.target.wants/docker.service.
dockeruser@vmdocker:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
dockeruser@vmdocker:~$
```

6-1-Créer un groupe docker

```
sudo groupadd docker
```

2-Ajouter dockeruser au groupe docker

```
sudo usermod -aG docker dockeruser
```

```
utilise sudo reebot pour déconnecter/reconnecter dockeruser
```

```
dockeruser@vmdocker:~$ sudo reboot
Connection to 192.168.57.101 closed by remote host.
Connection to 192.168.57.101 closed.

C:\Users\User>ssh dockeruser@192.168.57.101

Last login: Fri Sep 19 13:43:00 2025 from 192.168.57.1
dockeruser@vmdocker:~$ docker version
Client: Docker Engine - Community
  Version:          28.4.0
  API version:     1.51
  Go version:      go1.24.7
  Git commit:      d8eb465
  Built:           Wed Sep  3 20:57:05 2025
  OS/Arch:         linux/amd64
  Context:         default

Server: Docker Engine - Community
  Engine:
    Version:          28.4.0
    API version:     1.51 (minimum version 1.24)
    Go version:      go1.24.7
    Git commit:      249d679
    Built:           Wed Sep  3 20:57:05 2025
    OS/Arch:         linux/amd64
    Experimental:   false
  containerd:
    Version:          1.7.27
    GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
  runc:
    Version:          1.2.5
    GitCommit:        v1.2.5-0-g59923ef
  docker-init:
    Version:          0.19.0
    GitCommit:        de40ad0
```

```
dockeruser@vmdocker:~$ sudo groupadd docker
groupadd: group 'docker' already exists
dockeruser@vmdocker:~$ sudo usermod -aG docker dockeruser
dockeruser@vmdocker:~$
```

Partie 3

1-dans quel répertoire Docker stocke-t-il ses objets (images, conteneurs, volumes...) ?

- docker info | grep "Docker Root Dir"

Contenu de ce répertoire

- containers/ → tous les conteneurs existants et arrêtés
- image/ → toutes les images téléchargées
- volumes/ → les volumes persistants
- overlay2/ ou aufs/ → système de fichiers des conteneurs (selon le driver)
- network/ → configuration des réseaux Docker

```
dockeruser@vmdocker:~$ docker info | grep "Docker Root Dir"
Docker Root Dir: /var/lib/docker
dockeruser@vmdocker:~$
```

2-Quels sont les différentes catégories d'objets Docker qui peuvent être stockés ?

Objets Docker :

Les objets Docker sont les éléments gérés par Docker pour créer et exécuter des applications : **images** (templates des conteneurs), **conteneurs** (instances actives ou arrêtées), **volumes** (stockage persistant) et **réseaux** (communication entre conteneurs).

```
docker system df
```

```
dockeruser@vmdocker:~$ docker system df
TYPE      TOTAL      ACTIVE      SIZE      RECLAMABLE
Images      1          1      10.07kB      0B (0%)
Containers    1          0          0B          0B
Local Volumes  0          0          0B          0B
Build Cache    0          0          0B          0B
dockeruser@vmdocker:~$
```

3-Pour consulter le contenu du répertoire où Docker stocke les conteneurs :

- cd /var/lib/docker/containers
- ls -l
- ls -1 /var/lib/docker/containers | wc -l

```

dockeruser@vmdocker:~$ cd /var/lib/docker/containers
-bash: cd: /var/lib/docker/containers: Permission denied
dockeruser@vmdocker:~$ ls -l
total 0
dockeruser@vmdocker:~$ ls -l /var/lib/docker/containers | wc -l
ls: cannot access '/var/lib/docker/containers': Permission denied
0
dockeruser@vmdocker:~$
```

4-Pour rechercher une image sur Docker Hub, tu peux utiliser la commande :

- docker search hello-world

Explication :

- docker search permet de **chercher des images disponibles sur Docker Hub.**
- hello-world est le nom de l'image que tu veux trouver.

```

dockeruser@vmdocker:~$ docker search hello-world
NAME                  DESCRIPTION                                     STARS      OFFICIAL
hello-world           Hello World! (an example of minimal Dockeriz... 2489      [OK]
rancher/hello-world   This container image is no longer maintained... 6
okteto/hello-world    0
atlassian/hello-world 1
goharbor/hello-world   0
tutum/hello-world     Image to test docker deployments. Has Apache... 91
dockercloud/hello-world  Hello World!
crccheck/hello-world   Hello World web server in under 2.5 MB       26
koudaiii/hello-world   0
ppc64le/hello-world    Hello World! (an example of minimal Dockeriz... 2
tsepotesting123/hello-world  0
brajwalendra/hello-world  0
kevindockercompany/hello-world  0
infrastructureascode/hello-world A tiny "Hello World" web server with a healt... 1
cloudflare/hello-world  A simple example application which can be ru... 0
arm32v7/hello-world    Hello World! (an example of minimal Dockeriz... 3
datawire/hello-world   Hello World! Simple Hello World implementati... 1
twistlocktest/hello-world  0
uniplaces/hello-world   0
wjimenez5271/hello-world  0
arm64v8/hello-world    Hello World! (an example of minimal Dockeriz... 3
danfengliu/hello-world   0
lbadger/hello-world    0
ansibleplaybookbundle/hello-world Simple containerized application that tests ... 0
swarna3005/hello-world   0
dockeruser@vmdocker:~$
```

5-- Faites exécuter un conteneur qui correspond à l'image ayant le plus d'étoiles. Faites une copie d'écran qui enregistre les étapes réalisées par docker.

- Télécharger l'image: Télécharger l'image : **docker pull hello-world**

- Lancer un conteneur à partir de cette image : `docker run hello-world`

Explication

- `docker run` : crée et exécute un conteneur à partir de l'image spécifiée.
- Docker affichera dans le terminal un message de test confirmant que le conteneur fonctionne

Résultat:

```
dockeruser@vmdocker:~$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
Digest: sha256:54e66cc1dd1fcbb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
Status: Image is up to date for hello-world:latest
docker.io/library/hello-world:latest
dockeruser@vmdocker:~$ docker run hello-world
```

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/>

6-1-Vérifiez maintenant le contenu du répertoire des conteneurs.

Avec la commande Docker : `docker ps -a`

Explication :

- `docker ps` → liste seulement les conteneurs actifs.
- `docker ps -a` → liste **tous** les conteneurs (actifs et stoppés).

Résultat:

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
707e6d7ff906        hello-world        "/hello"          26 hours ago       Exited (0) 26 hours ago
4a76373af545        hello-world        "/hello"          29 hours ago       Exited (0) 29 hours ago
dockeruser@vmdocker:~$
```

Combien y a-t-il de conteneurs ?

- Les conteneurs sont stockés dans : **sudo ls /var/lib/docker/containers**
- Chaque sous-répertoire correspond à un conteneur : **sudo ls /var/lib/docker/containers | wc -l**

Résultat:

```
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/containers
[sudo] password for dockeruser:
4a76373af54598f2ef13ad063323f691d204c113067eac45c33d80e798d2eb5d 707e6d7ff906f4d3bac86e3a2e2c5b5b2403e240abe22dfa82636028e3e19c6f
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/containers | wc -l
2
dockeruser@vmdocker:~$
```

7 - Vérifiez aussi le contenu du répertoire des images.

- Pour compter le nombre d'images : **sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256 | wc -l**
sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256

Résultat :

```
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/image
overlay2
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256
1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256 | wc -l
1
dockeruser@vmdocker:~$
```

8- Utilisez la commande docker (sans option) qui permet de lister les conteneurs en exécution.

docker ps
docker ps -a

Explication :

- docker ps affiche uniquement les conteneurs en cours d'exécution.
- Dans cet exemple le conteneur hello-world s'est déjà terminé automatiquement donc il peut apparaître vide.

Résultat :

```
dockeruser@vmdocker:~$ docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS     NAMES
dockeruser@vmdocker:~$ docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS     NAMES
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS     NAMES
707e6d7ff906   hello-world "/hello"  27 hours ago Exited (0) 27 hours ago
4a76373af545   hello-world "/hello"  30 hours ago Exited (0) 30 hours ago
dockeruser@vmdocker:~$
```

9- Combien de conteneurs qui s'exécutent ?

docker ps | wc -l

Explication :

- docker ps = liste les conteneurs actifs.
- wc -l = compte le nombre de lignes affichées. Donc il y a 0 conteneur actif .
- Comme la première ligne est l'en-tête, on soustrait 1 pour avoir le nombre réel.

10- Quel est l'alias de la commande que vous venez d'utiliser ? Remarque : par défaut, si on ne précise pas d'objet, docker considère que l'opération demandée s'applique aux conteneurs.

l'alias de la commande docker ps : docker container ls -a

Explication :

- docker ps est un alias pour docker container ls.
- Par défaut, si on ne précise pas d'objet (container, image, etc.), Docker considère que l'opération s'applique aux conteneurs.

Donc docker ps et docker container ls sont équivalentes.

Résultat :

```
dockeruser@vmdocker:~$ docker container ls -a
CONTAINER ID   IMAGE      COMMAND   CREATED     STATUS      PORTS     NAMES
707e6d7ff906   hello-world "/hello"  27 hours ago Exited (0) 27 hours ago
4a76373af545   hello-world "/hello"  30 hours ago Exited (0) 30 hours ago
dockeruser@vmdocker:~$
```

11- Utilisez la commande docker qui permet de lister les images et identifiez l'image utilisée .

docker images

Explication :

- docker images liste toutes les images présentes localement.
- Chaque image a un **IMAGE ID** unique (identifiant court ou long) qui sert à référencer l'image pour créer des conteneurs.
- Ici, hello-world est l'image que nous avons utilisée pour lancer le conteneur.

12- Réexécutez le conteneur et comparez à la précédente exécution : expliquez.

Explication :

- Lors de la **première exécution**, Docker crée un conteneur temporaire à partir de l'image hello-world, exécute le programme et termine automatiquement le conteneur.

- Lors de la **réexécution**, Docker **réutilise la même image locale** pour créer un **nouveau conteneur temporaire**.
- Chaque exécution produit un conteneur **different**, mais le message affiché reste **identique**, car l'image hello-world fait toujours la même opération (tester l'installation de Docker).
- Conclusion : **le comportement est identique**, mais un nouveau conteneur est créé à chaque fois.

Résultat :

```
dockeruser@vmdocker:~$ docker run hello-world

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/

dockeruser@vmdocker:~$
```

13- Combien de conteneurs s'exécutent et combien sont stockés localement

- **Pour les conteneurs en cours d'exécution :** docker ps -q | wc -l
- **Pour tous les conteneurs stockés localement (actifs + stoppés):** docker ps -aq | wc -l

Explication :

- docker ps -q → liste uniquement les **IDs des conteneurs actifs**, donc ici **0** car hello-world se termine immédiatement.
- docker ps -aq → liste tous les conteneurs créés sur la machine (actifs et arrêtés). Ici **1** conteneur stocké localement après avoir exécuté hello-world.

Chaque exécution de docker run hello-world crée un nouveau conteneur, même si l'image est déjà présente.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -q | wc -l
0
dockeruser@vmdocker:~$ docker ps -aq | wc -l
3
dockeruser@vmdocker:~$
```

14- Utilisez une commande qui permet de lister tous les conteneurs et pas uniquement ceux en exécution et observez leur statut.

- Lister tous les conteneurs (actifs et stoppés) et observer leur statut : `docker ps -a`

Explication :

- docker ps -a affiche **tous les conteneurs**, pas seulement ceux en cours d'exécution.
- **STATUS** indique l'état de chaque conteneur :
 - Exited (0) → conteneur terminé normalement
 - Up X minutes → conteneur en cours d'exécution
- Ici, le conteneur hello-world s'est exécuté puis **s'est terminé automatiquement**, d'où Exited (0).

Cela permet de suivre l'historique de tous les conteneurs créés à partir d'images Docker.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
dd2036fb761d hello-world "/hello" 2 hours ago Exited (0) 2 hours ago
707e6d7ff906 hello-world "/hello" 29 hours ago Exited (0) 29 hours ago
4a76373af545 hello-world "/hello" 32 hours ago Exited (0) 32 hours ago
dockeruser@vmdocker:~$
```

15- Quel est le nom de ces conteneurs ?

`docker ps -a --format "{{.Names}}"`

Explication :

- Chaque conteneur Docker reçoit un **nom unique** automatiquement si vous n'en spécifiez pas lors de la création.
- Le nom peut être utilisé pour **référencer le conteneur** dans d'autres commandes .

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a --format "{{.Names}}"
adoring_khorana
jovial_carson
flamboyant_elgamal
dockeruser@vmdocker:~$
```

17- Utilisez une option pour afficher l'information de façon non tronquée.

`docker ps -a --no-trunc`

Explication :

- L'option --no-trunc **affiche toutes les informations complètes** des conteneurs sans tronquer les ID, noms ou commandes.
- Utile pour voir le **CONTAINER ID complet**, le **nom complet**, et la **commande exacte** utilisée. Par défaut, Docker tronque certaines colonnes pour faciliter la lecture.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a --no-trunc
CONTAINER ID        NAMES              IMAGE       COMMAND    CREATED          STATUS      PORTS
dd2036fb761de2dbb4efcf3d09f02259604a8430dcead11bf73c3eeee9303f84   hello-world   "/hello"    3 hours ago   Exited (0) 3 hours ago
707e6d7ff906f4d3bac86e3a2e2c5b5b2403e240abe22dfa82636028e3e19c6f   hello-world   "/hello"    30 hours ago  Exited (0) 30 hours ago
4a76373af54598f2ef13ad063323f691d204c113067eac45c33d80e798d2eb5d   hello-world   "/hello"    33 hours ago  Exited (0) 33 hours ago
dockeruser@vmdocker:~$
```

18- Exécutez une nouvelle fois l'image, puis supprimez ce dernier conteneur en utilisant son nom.

- **docker run hello-world**
- Supprimer le conteneur en utilisant son nom : **docker rm adoring_khorana**

Résultat :

```
dockeruser@vmdocker:~$ docker run hello-world
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/

dockeruser@vmdocker:~$ docker rm adoring_khorana
adoring_khorana
dockeruser@vmdocker:~$
```

19- Exédez une nouvelle fois l'image en utilisant son sha256 en donnant un nom au nouveau conteneur, puis constatez.

- Identifier le SHA256 de l'image hello-world: **docker images --no-trunc**
- Exécuter un conteneur avec le SHA256 et donner un nom : **docker run --name mon_hello sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634**
- Vérifier le conteneur : **docker ps -a --format "{{.Names}}: {{.Status}}"**

Résultat :

```
dockeruser@vmdocker:~$ docker images --no-trunc
REPOSITORY      TAG      IMAGE ID
hello-world    latest   sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634      CREATED      SIZE
6 weeks ago    10.1kB
dockeruser@vmdocker:~$ docker run --name mon_hello sha256:1b44b5a3e06a1234567890abcdef1234567890abcdef1234567890abcdef
Unable to find image 'sha256:1b44b5a3e06a1234567890abcdef1234567890abcdef' locally
docker: Error response from daemon: pull access denied for sha256, repository does not exist or may require 'docker login': denied: requested access to the resource is denied

Run 'docker run --help' for more information
dockeruser@vmdocker:~$ docker run --name mon_hello sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634

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/
```

```
dockeruser@vmdocker:~$ docker ps -a --format "{{.Names}}: {{.Status}}"
mon_hello: Exited (0) 2 minutes ago
awesome_bell: Exited (0) 15 minutes ago
jovial_carson: Exited (0) 30 hours ago
flamboyant_elgamal: Exited (0) 33 hours ago
dockeruser@vmdocker:~$
```

20- Faites ce qu'il faut pour supprimer l'image (sans forcer).

- Vérifier les conteneurs qui utilisent l'image : `docker ps -a --format "{{.ID}}: {{.Image}}`

Explication :

On voit que plusieurs conteneurs (hello-world ou le sha256) utilisent encore cette image.

Donc **Docker refusera la suppression sans d'abord supprimer ces conteneurs.**

- Supprimer les conteneurs liés à l'image : `docker rm mon_hello adoring_khorana jovial_carson flamboyant_elgamal`
 - On supprime les conteneurs un par un (ou en liste).
 - Une fois supprimés, l'image n'a plus de conteneurs qui en dépendent.
- Supprimer l'image sans forcer : `docker rmi hello-world`
 - docker rmi supprime l'image si aucun conteneur ne l'utilise.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a --format "{{.ID}}: {{.Image}}"
a82b2e7d49da: 1b44b5a3e06a
e2b4c7ab4fa2: hello-world
707e6d7ff906: hello-world
4a76373af545: hello-world
dockeruser@vmdocker:~$ docker rm mon_hello adoring_khorana jovial_carson flamboyant_elgamal
mon_hello
jovial_carson
flamboyant_elgamal
Error response from daemon: No such container: adoring_khorana
dockeruser@vmdocker:~$ docker rmi hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container e2b4c7ab4fa2 is using its referenced image 1b44b5a3e06a
dockeruser@vmdocker:~$ sudo docker rmi hello-world
[sudo] password for dockeruser:
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container e2b4c7ab4fa2 is using its referenced image 1b44b5a3e06a
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND     CREATED      STATUS          PORTS     NAMES
e2b4c7ab4fa2   hello-world "/hello"   42 minutes ago   Exited (0) 42 minutes ago   awesome_bell
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND     CREATED      STATUS          PORTS     NAMES
e2b4c7ab4fa2   hello-world "/hello"   42 minutes ago   Exited (0) 42 minutes ago   awesome_bell
dockeruser@vmdocker:~$ docker rm e2b4c7ab4fa2
e2b4c7ab4fa2
dockeruser@vmdocker:~$ docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:54e66cc1dd1fcbb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
Deleted: sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634
Deleted: sha256:53d204b3dc5ddbc129df4ce71996b8168711e211274c785de5e0d4eb68ec3851
```

21- Utilisez une commande qui donne des infos globales sur le système et observez le nombre de conteneurs et d'images.

docker info

Explication :

- **Containers** → indique le nombre total de conteneurs présents localement, avec leur état (en cours d'exécution, en pause ou arrêtés).
- **Images** → indique le nombre total d'images stockées localement.

Résultat :

```
dockeruser@vmdocker:~$ docker info
Client: Docker Engine - Community
  Version: 28.4.0
  Context: default
  Debug Mode: false
  Plugins:
    buildx: Docker Buildx (Docker Inc.)
      Version: v0.28.0
      Path: /usr/libexec/docker/cli-plugins/docker-buildx
    compose: Docker Compose (Docker Inc.)
      Version: v2.39.4
      Path: /usr/libexec/docker/cli-plugins/docker-compose

Server:
  Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
  Images: 0
  Server Version: 28.4.0
  Storage Driver: overlay2
    Backing Filesystem: extfs
    Supports d_type: true
    Using metacopy: false
    Native Overlay Diff: true
    userxattr: false
  Logging Driver: json-file
  Cgroup Driver: systemd
  Cgroup Version: 2
```

```
Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journalctl json-file local splunk syslog
  CDI spec directories:
    /etc/cdi
    /var/run/cdi
  Swarm: inactive
  Runtimes: runc io.containerd.runc.v2
  Default Runtime: runc
  Init Binary: docker-init
  containerd version: 05044ec0a9a75232cad458027ca83437aae3f4da
  runc version: v1.2.5-0-g59923ef
  init version: de40ad0
  Security Options:
    apparmor
    seccomp
      Profile: builtin
    cgroupns
  Kernel Version: 5.15.0-153-generic
  Operating System: Ubuntu 22.04.5 LTS
  OSType: linux
  Architecture: x86_64
  CPUs: 2
  Total Memory: 1.918GiB
  Name: vmdocker
  ID: 10fb5f2e-d220-4990-a5fc-73d414225c16
  Docker Root Dir: /var/lib/docker
  Debug Mode: false
```

22- Quelle commande docker pouvez-vous utiliser pour n'afficher que les IDs des conteneurs ?

docker ps -aq -----→ [aucune sortie]

Explication :

L'option -a permet d'afficher tous les conteneurs et -q affiche uniquement leurs identifiants. Ici, la sortie vide signifie qu'il n'existe aucun conteneur localement.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -aq
dockeruser@vmdocker:~$ 
dockeruser@vmdocker:~$
```

23- Utilisez une possibilité du bash pour exploiter les résultats de cette dernière commande afin de supprimer tous les conteneurs en une seule commande.

docker rm \$(docker ps -aq)

Explication :

- docker ps -aq → récupère tous les IDs des conteneurs.
- \$(...) → substitution de commande Bash : insère les résultats de docker ps -aq comme arguments.
- docker rm → supprime les conteneurs listés.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -aq
dockeruser@vmdocker:~$ 
dockeruser@vmdocker:~$ docker rm $(docker ps -aq)
docker: 'docker rm' requires at least 1 argument

Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]

See 'docker rm --help' for more information
dockeruser@vmdocker:~$
```

24- - Supprimez maintenant l'image et constatez.

```
docker rmi hello-world
```

- **Explication :**

L'image hello-world a déjà été supprimée précédemment, donc il n'y a plus rien à supprimer. La commande renvoie une erreur indiquant que l'image n'existe pas .

Résultat :

```
dockeruser@vmdocker:~$ docker rm $(docker ps -aq)
docker: 'docker rm' requires at least 1 argument

Usage: docker rm [OPTIONS] CONTAINER [CONTAINER...]

See 'docker rm --help' for more information
dockeruser@vmdocker:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
dockeruser@vmdocker:~$ docker rmi IMAGE_ID
Error response from daemon: invalid reference format: repository name (library/IMAGE_ID) must be lowercase
dockeruser@vmdocker:~$ REPOSITORY TAG IMAGE ID SIZE
REPOSITORY: command not found
dockeruser@vmdocker:~$ hello-world latest 1b44b5a3e06a 13.3kB
Command 'hello-world' not found, but can be installed with:
sudo snap install hello-world
[sudo] password for dockeruser:
```

25- Faites à nouveau exécuter un conteneur, que vous nommerez hello1, pour la même image hello-world, que se passe-t-il ?

il va créer le conteneur nommé hello1 et l'exécuter car hello-world n'est pas présente localement

```
docker run --name hello1 hello-world
```

Explication :

- docker run → crée et exécute un conteneur.
- --name hello1 → donne un nom explicite au conteneur (hello1).
- hello-world → l'image à utiliser.

Résultat :

```
dockeruser@vmdocker:~$ docker run --name hello1 hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:54e66cc1dd1fcbb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
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.
```

26- Quel est l'identifiant de l'image ? Constatez

- Pour connaître l'ID de l'image : docker images

Résultat :

```
dockeruser@vmdocker:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
hello-world    latest    1b44b5a3e06a   6 weeks ago   10.1kB
dockeruser@vmdocker:~$
```

27- Créez un conteneur, que vous nommerez hello2, mais sans le démarrer, puis observez son statut.

- Créer un conteneur nommé hello2 sans le démarrer et observer son statut : docker create --name hello2 hello-world
- docker ps -a

Explication :

- docker create → crée un conteneur à partir d'une image **sans l'exécuter**.
- name hello2 → nom donné au conteneur.
- hello-world → image utilisée.
- Le conteneur est créé mais **reste à l'état Created**.

Résultat :

```
dockeruser@vmdocker:~$ docker create --name hello2 hello-world
886dc8487927b370eaa59a4f7f62d4761ccca787f4f6a58e772d22b373fbe2a1
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS          PORTS      NAMES
886dc8487927    hello-world    "/hello"    26 seconds ago  Created
71aea5e80256    hello-world    "/hello"    10 minutes ago  Exited (0) 10 minutes ago
dockeruser@vmdocker:~$
```

28- - Démarrez-le ensuite en utilisant son nom et constatez son statut ensuite. Quel est l'affichage ?

- Démarrer le conteneur hello2 et observer son statut et affichage : docker start -a hello2

Explication :

- docker start → démarre un conteneur existant.
- -a (attach) → affiche directement dans le terminal la sortie du conteneur.
- hello2 → nom du conteneur à démarrer.
- **Ce qui se passe :**
- Le conteneur s'exécute.
- Comme hello-world est une image très simple, elle **affiche son message puis se termine immédiatement**.
- Le conteneur passe ensuite à l'état Exited (0).

- docker ps -a

Résultat :

```
dockeruser@vmdocker:~$ docker start -a hello2

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/
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
886dc8487927	hello-world	"/hello"	16 minutes ago	Exited (0) 28 seconds ago		hello2

29- Utilisez la même option pour démarrer le conteneur hello1.

- **docker start -ai hello1**

Explication :

- Comme pour **hello2**, l'option **-ai** attache l'entrée/sortie au terminal.
- Le conteneur **hello1** exécute l'image **hello-world**, affiche son message, puis passe en état **Exited (0)**.

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai hello1

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/
```

30- Exécutez maintenant un nouveau conteneur nommé hello3, mais en faisant en sorte qu'il n'affiche pas d'information sur la sortie standard (en background donc). Constatez dans la liste (totale) des conteneurs .

- Un identifiant de conteneur long s'affiche : `docker run -d --name hello3 hello-world`

Résultat :

```
dockeruser@vmdocker:~$ docker run -d --name hello3 hello-world
e21a7cba987bb3720b44798ea707fbe564fcfa4f7cedb546d4a1279cf6d6c6ef
```

- Vérification : `docker ps -a`

Explication :

- L'option `-d` signifie *detached mode* → le conteneur s'exécute en arrière-plan.
- L'image **hello-world** s'arrête immédiatement après avoir affiché son message, donc son statut sera **Exited (0)**.
- On le retrouve dans la liste des conteneurs avec le nom **hello3**.

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED      STATUS          PORTS   NAMES
e21a7cba987b   hello-world   "/hello"  12 seconds ago   Exited (0)  11 seconds ago
386dc8487927   hello-world   "/hello"  12 hours ago   Exited (0)  15 minutes ago
71aea5e80256   hello-world   "/hello"  12 hours ago   Exited (0)  5 minutes ago
dockeruser@vmdocker:~$
```

31- Exécutez maintenant un nouveau conteneur nommé hello4, mais en faisant en sorte que ce conteneur ait totalement disparu après son exécution. Constatez dans la liste (totale) des conteneurs.

- docker run --rm --name hello4 hello-world
- docker ps -a

Explication :

- L'option **--rm** supprime automatiquement le conteneur dès qu'il s'arrête.
- Contrairement à hello1, hello2 et hello3 qui restent visibles avec le statut **Exited**, le conteneur **hello4** n'apparaît plus du tout dans la liste.

Résultat :

```
dockeruser@vmdocker:~$ docker run --rm --name hello4 hello-world

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/
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e21a7cba987b	hello-world	"/hello"	12 minutes ago	Exited (0) 12 minutes ago		hello3
886dc8487927	hello-world	"/hello"	12 hours ago	Exited (0) 27 minutes ago		hello2
71aea5e80256	hello-world	"/hello"	12 hours ago	Exited (0) 17 minutes ago		hello1

32- Vous pouvez supprimer l'image hello-world avant de passer à la partie suivante.

- docker rm hello1 hello2 hello3
- docker ps -a

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e21a7cba987b hello-world "/hello" 21 minutes ago Exited (0) 21 minutes ago
886dc8487927 hello-world "/hello" 12 hours ago Exited (0) 36 minutes ago
71aee5e80256 hello-world "/hello" 12 hours ago Exited (0) 26 minutes ago
dockeruser@vmdocker:~$ docker rm 1b44b5a3e06a
Error response from daemon: No such container: 1b44b5a3e06a
dockeruser@vmdocker:~$ docker rm hello1 hello2 hello3
hello1
hello2
hello3
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
dockeruser@vmdocker:~$
```

Partie 4 – DockerHub

1-Types d'images sur DockerHub et classification

Type d'image	Description
Official Images	Maintenues par docker et éditeur officiel(nginx ,ubuntu.....)
Verified Publisher Images	Créées par des éditeurs ou organisations de confiance, avec badge de vérification.
Community Images	Créées par la communauté Docker, moins garanties.
Private Images	Créées par un utilisateur et visibles uniquement pour lui .

- Retrouver l'image officielle Ubuntu → Commande Docker pour rechercher :
`docker search ubuntu`

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
dockeruser@vmdocker:~$ docker search ubuntu
NAME                           DESCRIPTION                                         STARS   OFFICIAL
ubuntu                         Ubuntu is a Debian-based Linux operating sys... 17691   [OK]
ubuntu/squid                   Squid is a caching proxy for the Web. Long-t... 119
ubuntu/nginx                   Nginx, a high-performance reverse proxy & we... 133
ubuntu/cortex                  Cortex provides storage for Prometheus. Long... 4
ubuntu/bind9                   BIND 9 is a very flexible, full-featured DNS... 113
ubuntu/kafka                   Apache Kafka, a distributed event streaming ... 55
ubuntu/apache2                 Apache, a secure & extensible open-source HT... 97
ubuntu/prometheus              Prometheus is a systems and service monitori... 73
ubuntu/zookeeper              ZooKeeper maintains configuration informatio... 13
ubuntu/mysql                   MySQL open source fast, stable, multi-thread... 70
ubuntu/postgres                PostgreSQL is an open source object-relation... 41
ubuntu/jre                      Distroless Java runtime based on Ubuntu. Lon... 21
ubuntu/dotnet-aspnet            Chiselled Ubuntu runtime image for ASP.NET a... 25
ubuntu/redis                    Redis, an open source key-value store. Long-... 23
ubuntu/python                  A chiselled Ubuntu rock with the Python runt... 27
ubuntu/dotnet-deps              Chiselled Ubuntu for self-contained .NET & A... 16
ubuntu/grafana                 Grafana, a feature rich metrics dashboard & ... 12
ubuntu/dotnet-runtime            Chiselled Ubuntu runtime image for .NET apps... 21

```

2-Téléchargement de l'image officielle : docker pull ubuntu

Résultat :

```
dockeruser@vmdocker:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
953cdd413371: Downloading [==>] 1.536MB/29.72MB
```

3- Consultez les différentes versions proposées, comment les distingue-t-on ?

- Pour voir les versions disponibles d'une image (par exemple Ubuntu) :
docker image inspect ubuntu

Explication :

- Le **tag** latest indique que c'est la **dernière version stable** téléchargée.
- L'attribut "org.opencontainers.image.version": "24.04" précise la **version exacte de l'image**, ici Ubuntu 24.04.
- "Architecture": "amd64" → l'architecture CPU de l'image.
- "Os": "linux" → le système d'exploitation utilisé.
- "Size": 78123424 → taille de l'image en octets.
- "GraphDriver" et "RootFS" → localisation et couches de fichiers de Docker pour cette image.

Résultat :

```
dockeruser@vmdocker:~$ docker image inspect ubuntu
[{"Id": "sha256:6d79abd4c96299aa91f5a4a46551042407568a3858b00ab460f4ba430984f62c",
 "RepoTags": [
     "ubuntu:latest"
 ],
 "RepoDigests": [
     "ubuntu@sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc"
 ],
 "Parent": "",
 "Comment": "",
 "Created": "2025-09-10T05:42:34.634203762Z",
 "DockerVersion": "24.0.7",
 "Author": "",
 "Architecture": "amd64",
 "Os": "linux",
 "Size": 78123424,
 "GraphDriver": {
     "Data": {
         "MergedDir": "/var/lib/docker/overlay2/87f952d039badc6ff5e884150c515d24e7bd3509de9de1ffdeb55d4c5a123b3b/merged",
         "UpperDir": "/var/lib/docker/overlay2/87f952d039badc6ff5e884150c515d24e7bd3509de9de1ffdeb55d4c5a123b3b/diff",
         "WorkDir": "/var/lib/docker/overlay2/87f952d039badc6ff5e884150c515d24e7bd3509de9de1ffdeb55d4c5a123b3b"
     }
 }
```

4- Quelle est la version la plus récente ? Quelle différence a-t-elle avec la version latest ? Quels sont leurs identifiants respectifs ?

- Docker images

Résultat :

```
dockeruser@vmdocker:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ubuntu          latest   6d79abd4c962   11 days ago  78.1MB
hello-world     latest   1b44b5a3e06a   6 weeks ago  10.1kB
```

5- Quelle est la version la plus récente ? Quelle différence a-t-elle avec la version latest ? Quels sont leurs identifiants respectifs ?

- La version la plus récente est : Ubuntu 24.04
- Différence avec la version latest :
 - **24.04** = une version précise (tag numérique fixé).
 - **latest** = un tag générique qui pointe en général vers la version stable par défaut, mais qui ne correspond pas toujours à la version numérique la plus récente.

- Les identifiants respectifs :

1-Ubuntu 24.04 → docker image inspect ubuntu:24.04

```
dockeruser@vmdocker:~$ docker image inspect ubuntu:24.04
[{"Id": "sha256:6d79abd4c96299aa91f5a4a46551042407568a3858b00ab460f4ba430984f62c",
 "RepoTags": [
     "ubuntu:24.04",
     "ubuntu:latest"
 ],
 "RepoDigests": [
     "ubuntu@sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc"
 ]}
```

2- Ubuntu:latest → docker image inspect ubuntu:latest

```
dockeruser@vmdocker:~$ docker image inspect ubuntu:latest
[{"Id": "sha256:6d79abd4c96299aa91f5a4a46551042407568a3858b00ab460f4ba430984f62c",
 "RepoTags": [
     "ubuntu:24.04",
     "ubuntu:latest"
 ],
 "RepoDigests": [
     "ubuntu@sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc"
 ]}
```

6- Observez comment ces vulnérabilités sont classées.

- **test de sécurité ubuntu :24.04 → trivy image ubuntu:24.04**

Résultat :

Library Title	Vulnerability	Severity	Status	Installed Version	Fixed Version
coreutils -privileged session can escape to the parent root	CVE-2016-2781	LOW	affected	9.4-3ubuntu6.1	coreutils: Non
					session in chr
					https://avd.aq
gpgv of service issue (resource consumption) using keys	CVE-2022-3219	using	2.4.4-2ubuntu17.3	gnupg: denial	
					compressed pac
					https://avd.aq
libc-bin free in glibc	CVE-2025-8058	MEDIUM	2.39-0ubuntu8.5	glibc: Double	
					https://avd.aq
libc6					
libgcrypt20 vulnerable to Marvin Attack	CVE-2024-2236	LOW	1.10.3-2build1	libgcrypt: vul	
					https://avd.aq

- **ubuntu :latest → trivy image ubuntu:latest**

Résultat :

ubuntu:latest (ubuntu 24.04)					
Total: 17 (UNKNOWN: 0, LOW: 6, MEDIUM: 11, HIGH: 0, CRITICAL: 0)					
Library Title	Vulnerability	Severity	Status	Installed Version	Fixed Version
coreutils	CVE-2016-2781 privileged session can escape to the parent	LOW	affected	9.4-3ubuntu6.1	coreutils: Non
oot					session in chr
uasec.com/nvd/cve-2016-2781					https://avd.aq
gpgv	CVE-2022-3219 of service issue (resource consumption) using			2.4.4-2ubuntu17.3	gnupg: denial
kets					compressed pac
uasec.com/nvd/cve-2022-3219					https://avd.aq
libc-bin	CVE-2025-8058 free in glibc	MEDIUM		2.39-0ubuntu8.5	glibc: Double
uasec.com/nvd/cve-2025-8058					https://avd.aq

7- Comparez la version latest avec la version noble : quel pourrait être l'intérêt de cette situation ?

- **latest** : pointe vers la dernière version publiée.
- **24.04** : est une version fixe et spécifique.

Explication :

- **Latest** : est pratique pour récupérer automatiquement les dernières mises à jour.
- **24.04** : permet de garantir la stabilité et la reproductibilité d'un environnement, utile pour la production ou un projet où l'on ne veut pas que l'image change.

8- En cliquant sur le tag de chacune des 2, consultez les couches de ces images, et observez à partir de quand elles diffèrent.

- docker history ubuntu:latest
- docker history ubuntu:24.04

Explication :

- Cela permet de voir où latest a été modifiée par rapport à la version stable 24.04.
- Comprendre les couches est utile pour identifier les changements dans l'image sans avoir à reconstruire entièrement la base.

Résultat :

```
dockeruser@vmdocker:~$ docker history ubuntu:24.04
IMAGE      CREATED      CREATED BY
6d79abd4c962  12 days ago  /bin/sh -c #(nop)  CMD ["/bin/bash"]          SIZE      COMMENT
<missing>    12 days ago  /bin/sh -c #(nop) ADD file:dafefafa97de6dc66a6...  0B
<missing>    12 days ago  /bin/sh -c #(nop) LABEL org.opencontainers...  78.1MB
<missing>    12 days ago  /bin/sh -c #(nop) LABEL org.opencontainers...  0B
<missing>    12 days ago  /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH  0B
<missing>    12 days ago  /bin/sh -c #(nop) ARG RELEASE               0B
dockeruser@vmdocker:~$ docker history ubuntu:latest
IMAGE      CREATED      CREATED BY
6d79abd4c962  12 days ago  /bin/sh -c #(nop)  CMD ["/bin/bash"]          SIZE      COMMENT
<missing>    12 days ago  /bin/sh -c #(nop) ADD file:dafefafa97de6dc66a6...  0B
<missing>    12 days ago  /bin/sh -c #(nop) LABEL org.opencontainers...  78.1MB
<missing>    12 days ago  /bin/sh -c #(nop) LABEL org.opencontainers...  0B
<missing>    12 days ago  /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH  0B
<missing>    12 days ago  /bin/sh -c #(nop) ARG RELEASE               0B
dockeruser@vmdocker:~$
```

9- Placez-vous dans le répertoire /var/lib/docker/image/overlay2/layerdb/sha256 et listez son contenu.

- `sudo ls /var/lib/docker/image/overlay2/layerdb/sha256`

Explication :

- utilise ces couches pour **réutiliser des parties d'images entre conteneurs**, et accélérant la création de nouveaux conteneurs.
- Ces fichiers ne sont pas les images elles-mêmes

Résultat :

```
dockeruser@vmdocker:~$ sudo ls /var/lib/docker/image/overlay2/layerdb/sha256
53d204b3dc5ddbc129df4ce71996b8168711e211274c785de5e0d4eb68ec3851
f9f52dc133e2af9188960e5a5165cafaa51657ef740ff20219e45a561d78c591
dockeruser@vmdocker:~$
```

10- Téléchargez l'image ubuntu : par défaut, quelle est celle qui est téléchargée.

- Docker pull ubuntu
- Docker images

Résultat :

```
dockeruser@vmdocker:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
dockeruser@vmdocker:~$ docker images
REPOSITORY      TAG          IMAGE ID          CREATED        SIZE
ubuntu          24.04        6d79abd4c962    12 days ago   78.1MB
ubuntu          latest        6d79abd4c962    12 days ago   78.1MB
hello-world     latest        1b44b5a3e06a    6 weeks ago   10.1kB
dockeruser@vmdocker:~$
```

11 - Téléchargez maintenant l'image ubuntu la plus récente.

- Docker pull ubutnu :24.04
- Docker images

Résultat :

```
dockeruser@vmdocker:~$ docker pull ubuntu:24.04
24.04: Pulling from library/ubuntu
Digest: sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc
Status: Image is up to date for ubuntu:24.04
docker.io/library/ubuntu:24.04
dockeruser@vmdocker:~$ docker images
REPOSITORY      TAG          IMAGE ID          CREATED        SIZE
ubuntu          24.04        6d79abd4c962    12 days ago   78.1MB
ubuntu          latest        6d79abd4c962    12 days ago   78.1MB
hello-world     latest        1b44b5a3e06a    6 weeks ago   10.1kB
dockeruser@vmdocker:~$
```

12- Utilisez un filtre pour n'afficher que les images référençant ubuntu.

- docker images --filter(reference='ubuntu*)'

Résultat :

```
dockeruser@vmdocker:~$ docker images --filter=reference='ubuntu*'
REPOSITORY      TAG          IMAGE ID          CREATED        SIZE
ubuntu          24.04        6d79abd4c962    12 days ago   78.1MB
ubuntu          latest        6d79abd4c962    12 days ago   78.1MB
dockeruser@vmdocker:~$
```

13 - Identifiez la commande qui est lancée lorsqu'on exécute un conteneur à partir de l'image ubuntu.

- Docker run -it ubuntu
- L'option -it permet de lancer le conteneur en mode interactif avec un terminal attaché.

Résultat :

```
dockeruser@vmdocker:~$ docker run -it ubuntu
root@470bb05b2e68:/#
```

14- Pour la suite, vous pouvez supprimer l'image ubuntu:rolling.

- Docker rmi ubuntu :24.04

Résultat :

```
dockeruser@vmdocker:~$ docker rmi ubuntu:24.04
Untagged: ubuntu:24.04
```

```
dockeruser@vmdocker:~$ docker rmi ubuntu:rolling
Error response from daemon: No such image: ubuntu:rolling
```

Partie 5 – Interagir avec un conteneur

1- Lancez un conteneur à partir de l'image ubuntu et constatez son statut. Essayez de redémarrer ce même conteneur en interactif. Enfin supprimez le.

- un nouveau conteneur nommé testubuntu → `docker run --name testubuntu -it ubuntu`
- liste tous les conteneur → `docker ps -a`
- redémarre le même conteneur et ouvre une session interactive à l'intérieur → `docker start -ai testubuntu`
- supprime le conteneur → `docker rm testubuntu`

Résultat :

```
dockeruser@vmdocker:~$ docker run --name testubuntu -it ubuntu
root@8acbee46117e:/# ^C
root@8acbee46117e:/# exit
exit
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND     CREATED        STATUS          PORTS   NAMES
8acbee46117e   ubuntu     "/bin/bash"  18 seconds ago  Exited (130) 12 seconds ago
470bb05b2e68   ubuntu     "/bin/bash"  About an hour ago   Exited (0)  About an hour ago
dockeruser@vmdocker:~$ docker start -ai testubuntu
root@8acbee46117e:/#
exit
dockeruser@vmdocker:~$ docker rm testubuntu
testubuntu
dockeruser@vmdocker:~$
```

- 2- Lancez, à partir de l'image ubuntu, un conteneur nommé os_ubuntu en interactif et attaché à un terminal.

- nouveau conteneur à partir de l'image Ubuntu avec le nom os_ubuntu, en mode interactif (-it) et attaché au terminal : `docker run --name os_ubuntu -it ubuntu`

Résultat :

```
dockeruser@vmdocker:~$ docker run --name os_ubuntu -it ubuntu
root@b2a4b38cd6d2:/#
```

- 3- Quelle est la commande qui correspond au processus de PID 1 de ce conteneur ?

- dans le conteneur affichez le processus PID 1 → `docker start -ai os_ubuntu` → `(root) ps -p 1 -o pid,comm,args`

Résultat :

```
root@b2a4b38cd6d2:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:/# ps -p 1 -o pid,comm,args
  PID COMMAND          COMMAND
    1 bash            /bin/bash
root@b2a4b38cd6d2:/#
```

- 4- Dans cet OS, exécutez les commandes whoami, pwd, ls et hostname. Notez son ID.

- Affiche root, donc vous êtes connecté en tant que super utilisateur → `whoami`
- Le répertoire est /, c'est la racine du système de fichiers → `pwd`
- liste le contenu du répertoire racine → `ls`
- affiche l'ID du conteneur → `hostname`

Résultat :

```
root@b2a4b38cd6d2:/# whoami
root
root@b2a4b38cd6d2:/# pwd
/
root@b2a4b38cd6d2:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@b2a4b38cd6d2:/# hostname
b2a4b38cd6d2
root@b2a4b38cd6d2:/#
```

5- Ouvrez un deuxième terminal, connectez-vous en ssh sur la VM et observez le statut du conteneur en cours d'exécution. Que constatez-vous (à part le statut) ?

- docker ps

Résultat :

```
dockeruser@vmdocker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS     NAMES
dockeruser@vmdocker:~$
```

5- Revenez dans le conteneur sur le premier terminal. Déplacez-vous dans le répertoire home.

- docker start -ai os_ubuntu
 - cd /root
 - pwd

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:/# cd /root
root@b2a4b38cd6d2:~/# pwd
/root
root@b2a4b38cd6d2:~#
```

6- Quittez ce conteneur avec la commande unix classique puis observez le statut du conteneur.

- Exit
- Docker ps -a

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:#
exit
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS     NAMES
b2a4b38cd6d2        ubuntu              "/bin/bash"   17 hours ago    Exited (0) 18 seconds ago
470bb05b2e68        ubuntu              "/bin/bash"   19 hours ago    Exited (0) 18 hours ago
dockeruser@vmdocker:~$
```

7- Démarrez ce conteneur en interactif. Dans quel répertoire vous trouvez-vous ?

Pourquoi ?

- docker start -ai os_ubuntu
 - cd /root
 - pwd
 - exit
- docker ps -a

Pourquoi ?

Parce que par défaut, l'image Ubuntu n'a pas de commande personnalisée ni de répertoire de travail configuré.

- Le conteneur exécute /bin/bash comme processus PID 1.
- Le répertoire courant est donc défini par défaut sur /.

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:/# cd /root
root@b2a4b38cd6d2:~# pwd
/root
root@b2a4b38cd6d2:~# exit
exit
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS           PORTS     NAMES
b2a4b38cd6d2        ubuntu      "/bin/bash"   17 hours ago    Exited (0) 28 seconds ago
470bb05b2e68        ubuntu      "/bin/bash"   19 hours ago    Exited (0) 19 hours ago
dockeruser@vmdocker:~$
```

8- Dans le second terminal, utilisez une commande qui permet d'inspecter le conteneur.

Constatez qu'il est en cours d'exécution. Retrouvez son Pid.

- inspecter conteneur → docker inspect os_ubuntu
- uniquement le PID → docker inspect -f '{{.State.Pid}}' os_ubuntu

Résultat :

```
dockeruser@vmdocker:~$ docker inspect os_ubuntu
[{"Id": "b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952",
 "Created": "2025-09-22T20:53:26.520685462Z",
 "Path": "/bin/bash",
 "Args": [],
 "State": {
     "Status": "exited",
     "Running": false,
     "Paused": false,
     "Restarting": false,
     "OOMKilled": false,
     "Dead": false,
     "Pid": 0,
     "ExitCode": 0,
     "Error": "",
     "StartedAt": "2025-09-23T14:01:01.758814336Z",
     "FinishedAt": "2025-09-23T14:01:30.69479439Z"
 },
 "Image": "sha256:6d79abd4c96299aa91f5a4a46551042407568a3858b00ab460f4ba430984f62c",
 "ResolvConfPath": "/var/lib/docker/containers/b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952/resolv.conf",
 "HostnamePath": "/var/lib/docker/containers/b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952/hostname",
 "HostsPath": "/var/lib/docker/containers/b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952/hosts"}
```

```
dockeruser@vmdocker:~$ docker inspect -f '{{.State.Pid}}' os_ubuntu
0
dockeruser@vmdocker:~$ docker inspect -f '{{.State.Pid}}' os_ubuntu
0
dockeruser@vmdocker:~$
```

- 9- Vous pouvez aussi essayer d'utiliser jq pour obtenir cette information, par exemple voir : <https://blog.madrzejewski.com/jq-traiter-parser-json-shell-cli/> (ou tout lien de votre choix).**

- Pour obtenir le PID du conteneur → `docker inspect os_ubuntu | jq '.[].State.Pid'`

Résultat :

```
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '.[].State.Pid'
0
dockeruser@vmdocker:~$ docker start os_ubuntu
os_ubuntu
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '.[].State.Pid'
3991
dockeruser@vmdocker:~$
```

- 10-Dans ce même terminal, retrouvez le processus dont le PID est celui que vous venez d'identifier et constatez.**

- voir le processus sur le système → `ps -p 3991 -f`

Résultat :

```
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '.[].State.Pid'  
0  
dockeruser@vmdocker:~$ docker start os_ubuntu  
os_ubuntu  
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '.[].State.Pid'  
3991  
dockeruser@vmdocker:~$
```

10-Revenez dans le conteneur (dans le premier terminal) et déplacez-vous à nouveau dans home.

- Démarrer le conteneur → `docker start os_ubuntu`
- Revenir dans le conteneur → `docker exec -it os_ubuntu bash`
 - se déplacer dans le répertoire /home : `cd /home`
 - vérifie dans /home : `pwd`

Résultat :

```
dockeruser@vmdocker:~$ docker exec -it os_ubuntu bash  
root@b2a4b38cd6d2:/# cd /home  
root@b2a4b38cd6d2:/home# pwd  
/home  
root@b2a4b38cd6d2:/home#
```

11-Quittez maintenant ce conteneur en utilisant la combinaison de touches Ctrl-p Ctrl-q. Quel est maintenant le statut du conteneur ?

- `docker ps -a`

Résultat :

```
root@b2a4b38cd6d2:/home# read escape sequence  
dockeruser@vmdocker:~$ docker ps -a  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
b2a4b38cd6d2 ubuntu "/bin/bash" 20 hours ago Up 47 minutes  
470bb05b2e68 ubuntu "/bin/bash" 22 hours ago Exited (0) 22 hours ago  
dockeruser@vmdocker:~$
```

12-Utilisez une commande docker pour exécuter la commande Unix hostname dans ce conteneur (sans le passer en foreground)

- `docker exec os_ubuntu hostname`

Résultat :

```
dockeruser@vmdocker:~$ docker exec os_ubuntu hostname  
b2a4b38cd6d2  
dockeruser@vmdocker:~$
```

13-Utilisez une commande docker pour exécuter la commande Unix bash en interactif dans ce conteneur. Vérifiez le nombre de processus bash qui tournent dans le conteneur.

- Lancer un shell Bash interactif dans le conteneur : **docker exec -it os_ubuntu bash**

○ **ps -ef | grep bash**

Résultat :

```
dockeruser@vmdocker:~$ docker exec -it os_ubuntu bash
root@b2a4b38cd6d2:/# ps -ef | grep bash
root          1      0 16:25 pts/0    00:00:00 /bin/bash
root          19      0 16:47 pts/2    00:00:00 bash
root          33      0 17:20 pts/1    00:00:00 bash
root          42      33 17:20 pts/1    00:00:00 grep --color=auto bash
root@b2a4b38cd6d2:/#
```

14-Quittez à nouveau ce conteneur sans l'arrêter, puis utilisez une commande docker qui affiche les processus du conteneur.

- voir tous les processus qui tournent dans le conteneur → **docker top os_ubuntu**

Explication :

- docker top <nom_conteneur> → affiche tous les processus en cours dans ce conteneur.
- Tu verras les PID, l'utilisateur, la commande exécutée et d'autres infos, similaire à la commande ps dans Linux.

Résultat :

```
root@b2a4b38cd6d2:#
exit
dockeruser@vmdocker:~$ docker top os_ubuntu
UID      PID  PPID  C      STIME   TTY      TIME
root     3991  3967  0   16:25   pts/0    00:00:
00      /bin/bash
root     4227  3967  0   16:47   ?        00:00:
00      bash
dockeruser@vmdocker:~$
```

15-Revenez dans le conteneur et arrêtez-le en le quittant.

- **docker stop os_ubuntu**
- **docker ps -a**

Résultat :

```
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE    COMMAND      CREATED      STATUS           PORTS      NAMES
b2a4b38cd6d2   ubuntu   "/bin/bash"  21 hours ago  Exited (137) 27 seconds ago
470bb05b2e68   ubuntu   "/bin/bash"  23 hours ago  Exited (0) 23 hours ago
dockeruser@vmdocker:~$
```

16-Lancez maintenant un nouveau conteneur nommé ll à partir de l'image ubuntu dont la commande est maintenant ls -l.

- **docker run --name ll ubuntu ls -l**

Résultat :

```
dockeruser@vmdocker:~$ docker run --name ll ubuntu ls -l
total 48
lrwxrwxrwx  1 root root   7 Apr 22  2024 bin -> usr/bin
drwxr-xr-x  2 root root 4096 Apr 22  2024 boot
drwxr-xr-x  5 root root  340 Sep 23 18:27 dev
drwxr-xr-x  1 root root 4096 Sep 23 18:27 etc
drwxr-xr-x  3 root root 4096 Sep 10 02:18 home
lrwxrwxrwx  1 root root   7 Apr 22  2024 lib -> usr/lib
lrwxrwxrwx  1 root root   9 Apr 22  2024 lib64 -> usr/lib64
drwxr-xr-x  2 root root 4096 Sep 10 02:11 media
drwxr-xr-x  2 root root 4096 Sep 10 02:11 mnt
drwxr-xr-x  2 root root 4096 Sep 10 02:11 opt
dr-xr-xr-x 185 root root    0 Sep 23 18:27 proc
drwxr----- 2 root root 4096 Sep 10 02:17 root
drwxr-xr-x  4 root root 4096 Sep 10 02:18 run
lrwxrwxrwx  1 root root   8 Apr 22  2024 sbin -> usr/sbin
drwxr-xr-x  2 root root 4096 Sep 10 02:11 srv
dr-xr-xr-x 13 root root    0 Sep 23 18:27 sys
drwxrwxrwt  2 root root 4096 Sep 10 02:17 tmp
drwxr-xr-x 12 root root 4096 Sep 10 02:11 usr
drwxr-xr-x 11 root root 4096 Sep 10 02:17 var

```

17-Redémarrez ce conteneur pour obtenir le même affichage.

- docker start -ai ll

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai ll
total 48
lwxrwxrwx 1 root root 7 Apr 22 2024 bin -> usr/bin
drwxr-xr-x 2 root root 4096 Apr 22 2024 boot
drwxr-xr-x 5 root root 340 Sep 23 18:42 dev
drwxr-xr-x 1 root root 4096 Sep 23 18:27 etc
drwxr-xr-x 3 root root 4096 Sep 10 02:18 home
lwxrwxrwx 1 root root 7 Apr 22 2024 lib -> usr/lib
lwxrwxrwx 1 root root 9 Apr 22 2024 lib64 -> usr/lib64
drwxr-xr-x 2 root root 4096 Sep 10 02:11 media
drwxr-xr-x 2 root root 4096 Sep 10 02:11 mnt
drwxr-xr-x 2 root root 4096 Sep 10 02:11 opt
dr-xr-xr-x 186 root root 0 Sep 23 18:42 proc
drwx----- 2 root root 4096 Sep 10 02:17 root
drwxr-xr-x 4 root root 4096 Sep 10 02:18 run
lwxrwxrwx 1 root root 8 Apr 22 2024 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 Sep 10 02:11 srv
dr-xr-xr-x 13 root root 0 Sep 23 18:27 sys
drwxrwxrwt 2 root root 4096 Sep 10 02:17 tmp
drwxr-xr-x 12 root root 4096 Sep 10 02:11 usr
drwxr-xr-x 11 root root 4096 Sep 10 02:17 var
```

18-Lancez un nouveau conteneur nommé ps avec la commande ps aux, mais en faisant en sorte que ce conteneur disparaîsse après son exécution. Constatez le PID et constatez le statut.

- Lancer un conteneur éphémère pour exécuter ps aux → **docker run --rm --name ps ubuntu ps aux**
- **docker ps -a**
- **Explication :**
 - docker run → crée et lance un conteneur.
 - --rm → **supprime automatiquement** le conteneur après l'exécution de la commande.
 - --name ps → nom du conteneur pour le repérer pendant l'exécution.
 - ubuntu → image utilisée.
 - ps aux → commande Unix pour lister tous les processus du conteneur.

Résultat :

```
dockeruser@vmdocker:~$ docker run --rm --name ps ubuntu ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        1  41.1  0.1    7888  3564 ?        Rs   18:48   0:00 ps aux
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED      STATUS      PORTS     NAMES
9e26fb03bce6   ubuntu    "ls -l"      21 minutes ago   Exited (0) 6 minutes ago
b2a4b38cd6d2   ubuntu    "/bin/bash"  22 hours ago   Exited (137) 33 minutes ago
470bb05b2e68   ubuntu    "/bin/bash"  24 hours ago   Exited (0) 23 hours ago
dockeruser@vmdocker:~$
```

19-Lancez un nouveau conteneur nommé salut avec la commande echo Bonjour, puis relancez ce conteneur.

- **un conteneur nommé salut avec la commande echo Bonjour → docker run --name salut ubuntu echo Bonjour**
- **Relancer le conteneur pour voir la même sortie → docker start -ai salut**

Résultat :

```
dockeruser@vmdocker:~$ docker run --name salut ubuntu echo Bonjour
Bonjour
dockeruser@vmdocker:~$ docker start -ai salut
Bonjour
dockeruser@vmdocker:~$
```

20-Lancez un nouveau conteneur avec une commande infinie (sh -c "while true ; sleep 3600 ; done"), puis faites en sorte de le supprimer.

- **Infinite → docker run -d --name infini ubuntu sh -c "while true; do sleep 3600; done"**
- **docker ps**
- **arrête-le → docker stop infini**
- **supprime-le → docker rm infini**

Explication :

- **-d → lance le conteneur en arrière-plan (détaché).**
- **--name infini → donne le nom infini au conteneur.**
- **ubuntu → image de base.**
- **sh -c "while true; do sleep 3600; done" → boucle infinie qui dort 1h à chaque tour, donc le conteneur reste actif indéfiniment.**

Résultat :

```
dockeruser@vmdocker:~$ docker run -d --name infini ubuntu sh -c "while true; do sleep 3600; done"
474a599bc62f8cfa628dbe42daf273b811e4ebf5beb5a18c1ddde1bf2c151965
dockeruser@vmdocker:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED             STATUS              PORTS     NAMES
474a599bc62f   ubuntu     "sh -c 'while true; ..."   19 seconds ago   Up 18 seconds
dockeruser@vmdocker:~$ docker stop infini
infini
dockeruser@vmdocker:~$ docker rm infini
infini
dockeruser@vmdocker:~$
```

21-Démarrez le conteneur os_ubuntu en interactif. Dans ce shell, lancez une commande qui affiche salut toutes les 3 secondes.

- Démarrer le conteneur → docker start -ai os_ubuntu
 - LOOP→ while true; do echo salut; sleep 3; done

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:/# while true; do echo salut; sleep 3; done
salut
salut
salut
salut
salut
salut
salut
```

22-Dans le deuxième terminal connecté en ssh à la VM, utilisez une commande docker pour vous attacher au conteneur qui tourne dans le premier terminal puis constatez. Interrrompez le processus qui effectue le salut, lancez une commande basique (par exemple ls) et constatez dans l'autre terminal.

- Docker start os_ubuntu
- Docker attach os_ubuntu
- Ls
- Loop : while true ; do echo salut; sleep 3; done
- Ctrl c stop loop
- Ctrl d exit

Conclusion:

- Les deux terminaux voient la même sortie puisque les deux sont attachés au même conteneur.
- L'interruption d'un processus dans un terminal se reflète dans l'autre.

- Toute commande lancée est visible en simultané sur les deux attaches.

Résultat :

23-Quittez ce conteneur en utilisant la commande bash exit avec un code retour non nul.

Constatez le statut de ce conteneur.

- code retour non nul → **Exit 1**
 - **Docker ps -a**

Résultat :

```
dockeruser@vmdocker:~$ docker start -ai os_ubuntu
root@b2a4b38cd6d2:/# exit 1
exit
dockeruser@vmdocker:~$ docker ps -a
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS
              PORTS      NAMES
067b60736221   ubuntu     "echo Bonjour"  15 hours ago  Exited (0) 15 hours ago
                  salut
9e26fb03bce6   ubuntu     "ls -l"        15 hours ago  Exited (0) 15 hours ago
                  ll
b2a4b38cd6d2   ubuntu     "/bin/bash"    37 hours ago  Exited (1) About a minute ago
                  os_ubuntu
470bb05b2e68   ubuntu     "/bin/bash"    39 hours ago  Exited (0) 39 hours ago
                  busy_swartz
dockeruser@vmdocker:~$
```

24-Quittez ce conteneur en utilisant la commande bash exit avec un code retour non nul.

Constatez le statut de ce conteneur.

- Le conteneur os_ubuntu est déjà arrêté (Exited) → docker logs os_ubuntu
- Docker affiche tout le contenu du log qui a été généré avant que le conteneur se termine.

25-Inspectez le conteneur à la recherche du fichier contenant son journal (log). Vous pouvez le consulter avec jq .

- docker inspect os_ubuntu | jq '.[0].LogPath'

Résultat :

```
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '.[0].LogPath'  
"/var/lib/docker/containers/b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b9594  
2ef952/b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952-json.log"  
dockeruser@vmdocker:~$
```

26-Utilisez une commande docker pour supprimer tous les conteneurs arrêtés. A la fin de cette partie, arrêter et supprimer tous les conteneurs actifs (une seule commande).

- Pour supprimer tous les conteneurs arrêtés → docker container prune -f
- Pour supprimer tous les conteneurs actifs → docker rm -f \$(docker ps -aq)

Résultat :

```
dockeruser@vmdocker:~$ docker container prune -f  
Deleted Containers:  
067b60736221932f4250af166bd8434ab3b70c82497219c7b7c3443571bd4508  
9e26fb03bce688dc2c33667b14a5126237c040b37e87cd6aebede6fc7c73050e  
b2a4b38cd6d21c14cfb874dc13b0fe8df587ed0d8cbf6bf1c3fa5b95942ef952  
470bb05b2e68b20793b0986c9c94a56fabf07cc8c9a9a34107258bba43d58d56  
  
Total reclaimed space: 228B  
dockeruser@vmdocker:~$ docker rm -f $(docker ps -aq)  
docker: 'docker rm' requires at least 1 argument  
  
Usage: docker rm [OPTIONS] CONTAINER [CONTAINER...]  
  
See 'docker rm --help' for more information  
dockeruser@vmdocker:~$
```

Partie 6 – Inspection et manipulation d’images

- 1- Téléchargez l’image python:3.9. - Combien de couches ont été téléchargées ?

- docker pull python:3.9
- docker history python:3.9

Résultat :

```
dockeruser@vmdocker:~$ docker history python:3.9
IMAGE          CREATED      CREATED BY
239b4550132d  6 weeks ago   CMD ["python3"]
<missing>      6 weeks ago   RUN /bin/sh -c set -eux; for src in idle3 p...
<missing>      6 weeks ago   RUN /bin/sh -c set -eux; wget -O python.ta...
<missing>      6 weeks ago   ENV PYTHON_SHA256=61a42919e13d539f7673cf11d1...
<missing>      6 weeks ago   ENV PYTHON_VERSION=3.9.23
<missing>      6 weeks ago   ENV GPG_KEY=E3FF2839C048B25C084DEBE9B26995E3...
<missing>      6 weeks ago   RUN /bin/sh -c set -eux; apt-get update; a...
<missing>      6 weeks ago   ENV LANG=C.UTF-8
<missing>      6 weeks ago   ENV PATH=/usr/local/bin:/usr/local/sbin:/usr...
<missing>      20 months ago  RUN /bin/sh -c set -ex; apt-get update; ap...
<missing>      20 months ago  RUN /bin/sh -c set -eux; apt-get update; a...
<missing>      20 months ago  RUN /bin/sh -c set -eux; apt-get update; a...
<missing>      20 months ago  # debian.sh --arch 'amd64' out/ 'trixie' '@1...
dockeruser@vmdocker:~$
```

- 2- Observez les couches de cette image (il existe aussi une option non tronquée)

- docker history --no-trunc python:3.9

Explication:

- no-trunc montre tous les détails sans couper les colonnes.
- Chaque ligne représente un layer de l’image, avec sa taille, la commande qui l’a créé, et un éventuel commentaire.

Résultat :

```
dockeruser@vmdocker:~$ docker history --no-trunc python:3.9
IMAGE          CREATED      CREATED BY
SIZE          COMMENT
sha256:239b4550132def691be5e1bc8908f5c291d3581543c1ecc... 6 weeks ago    CMD ["python3"]
```

- 3- Quelle est la dernière couche ?

- # debian.sh --arch 'amd64' out/ 'trixie' '@1757289600'

4- Puis téléchargez python : 3.14.0a1. Combien de couches ont été téléchargées ? Expliquez .

- télécharger l'image → docker pull python :3.14.0a1
- couches ont été téléchargées → docker history python:3.14.0a1

Résultat :

```
dockeruser@vmdocker:~$ docker pull python:3.14.0a1
3.14.0a1: Pulling from library/python
b2b31b28ee3c: Pull complete
c3cc7b6f0473: Pull complete
2112e5e7c3ff: Pull complete
af247aac0764: Pull complete
46bd469f680f: Pull complete
db2f885547b6: Pull complete
33fcfb54d100: Pull complete
Digest: sha256:3a852c145c357b55d0fdbf624207bb81c5a546f17f622e5c208cc0b36edaaa0e
Status: Downloaded newer image for python:3.14.0a1
docker.io/library/python:3.14.0a1
dockeruser@vmdocker:~$ docker history python:3.14.0a1
IMAGE      CREATED      CREATED BY          SIZE      COMMENT
80ad471000e7  11 months ago  CMD ["python3"]    0B        buildkit.dockerfile.v0
<missing>   11 months ago  RUN /bin/sh -c set -eux; for src in idle3 p...  36B      buildkit.dockerfile.v0
<missing>   11 months ago  RUN /bin/sh -c set -eux; wget -O python.ta...  72.4MB  buildkit.dockerfile.v0
<missing>   11 months ago  ENV PYTHON_SHA256=3e464b0cbb7535e2db34262fd1...  0B      buildkit.dockerfile.v0
<missing>   11 months ago  ENV PYTHON_VERSION=3.14.0a1           0B      buildkit.dockerfile.v0
<missing>   11 months ago  RUN /bin/sh -c set -eux; apt-get update; a...  17.8MB  buildkit.dockerfile.v0
<missing>   11 months ago  ENV PATH=/usr/local/bin:/usr/local/sbin:/usr...  0B      buildkit.dockerfile.v0
<missing>   20 months ago  RUN /bin/sh -c set -ex; apt-get update; ap...  588MB  buildkit.dockerfile.v0
<missing>   20 months ago  RUN /bin/sh -c set -eux; apt-get update; a...  177MB  buildkit.dockerfile.v0
<missing>   2 years ago   RUN /bin/sh -c set -eux; apt-get update; a...  48.5MB  buildkit.dockerfile.v0
<missing>   2 years ago   CMD ["bash"]                         0B      buildkit.dockerfile.v0
<missing>   2 years ago   ADD rootfs.tar.xz / # buildkit          117MB  buildkit.dockerfile.v0
```

5- Utilisez l'inspection d'une image et l'outil jq pour afficher les couches des 2 images python, observez les couches communes.

- Inspecter l'image et afficher les layers avec jq : docker inspect python:3.9 | jq '.[0].RootFS.Layers' → docker inspect python:3.14.0a1 | jq '.[0].RootFS.Layers'
- Pas des couches communes

Résultat :

```
dockeruser@vmdocker:~$ docker inspect python:3.9 | jq '.[0].RootFS.Layers'  
[  
    "sha256:185e04da9d947141fd703dbf36361bdc2ff77cc27cbf500fb9f4881cb5ddbe95",  
    "sha256:607ddfe5f3c3f9e9df2b45f6275ad18bc76e49fdebcf0777c1c02c66f5012956",  
    "sha256:0dd5860cbc60e77cc364ce36be1a9055d4139f2123324e14f756af1af719ffb0",  
    "sha256:08e14ec5b7497da231d70d47d1d80440ba7d9997d43c0796a8394923bbc98183",  
    "sha256:17ba588b71727e8f8f119a800d7c17fc880acea30183ed9d824a89cf63ac031c",  
    "sha256:ea18e4e51fc0ae5ad6b58321aa1d0b4cf0d98b565c486445933399f93cad9ad",  
    "sha256:2db39521cdc2287700b4b6bdca5e48da5afad7a38c79b37d04c7498994a2d925"  
]  
dockeruser@vmdocker:~$ docker inspect python:3.14.0a1 | jq '.[0].RootFS.Layers'  
[  
    "sha256:24b5ce0f1e07d37a35460f50d058acf738619e431013d2e1727609bdff2d7fc",  
    "sha256:b6ca42156b9f492afa27c366f20e4e864cef8dd8d0e0a100497764b05b39e6fc",  
    "sha256:00547dd240c419fa2e1b33e66aba302e8dfa4bfe6401a972d94a03b1355cbc6c",  
    "sha256:96d99c63b722657062d3f33cc230e33b191ea9855c050f44871e173709597e35",  
    "sha256:9744b636d758d56bfeceb5e712ddfecbe662951562155cc3f93af8cf538422c",  
    "sha256:4068925b787de0e570d68e70bc04de2380276817a36a343c08aad3435c221113",  
    "sha256:da64f9c6a005a83af29c8389262e6de4bb9b1b5ae96ceb6c567e01e7c7525cde"  
]  
dockeruser@vmdocker:~$
```

- 6- Recréez un nouveau conteneur os_ubuntu à partir de l'image ubuntu et ajoutez dans le répertoire home un nouveau fichier, avec un contenu quelconque. Quittez-le sans l'arrêter. Utilisez une commande docker pour exporter le système de fichiers dans une archive tar.

- nouveau conteneur et ajouter un fichier → `docker run -it --name os_ubuntu_new ubuntu /bin/bash`
- crée un fichier `mon_fichier.txt` dans `/home` → `echo "Contenu quelconque" > /home/mon_fichier.txt`
- Exporter le système de fichiers en archive tar → `docker export os_ubuntu_new -o os_ubuntu_new.tar`

Explication :

- Le fichier `os_ubuntu_new.tar` contient maintenant tout le système de fichiers du conteneur, y compris `/home/mon_fichier.txt`.

Résultat :

```
dockeruser@vmdocker:~$ docker run -it --name os_ubuntu_new ubuntu /bin/bash  
root@515f48291f9c:/# echo "Contenu quelconque" > /home/mon_fichier.txt  
root@515f48291f9c:/#  
exit  
dockeruser@vmdocker:~$ docker ps -a  
CONTAINER ID   IMAGE      COMMAND      CREATED     STATUS          PORTS      NAMES  
515f48291f9c   ubuntu     "/bin/bash"   About a minute ago   Exited (0) About a minute ago  
dockeruser@vmdocker:~$ docker start -ai os_ubuntu_new  
root@515f48291f9c:/# docker export os_ubuntu_new -o os_ubuntu_new.tar  
bash: docker: command not found  
root@515f48291f9c:/# dockeruser@vmdocker:~$  
dockeruser@vmdocker:~$ docker export os_ubuntu_new -o os_ubuntu_new.tar  
dockeruser@vmdocker:~$
```

- 7- Créez une image `image_ubuntu_with_file` en important l'archive. Comment est-elle taggée ? Observez ses couches (son historique).

- importer l'archive pour créer une nouvelle image → docker import os_ubuntu_new.tar image_ubuntu_with_file
- Vérifier l'image et son tag → docker images
- Observer les couches → docker history image_ubuntu_with_file

Résultat :

```
dockeruser@vmdocker:~$ docker import os_ubuntu_new.tar image_ubuntu_with_file
sha256:cb9b513941add3a0edd32d4f6ab7d065325523a10bcf82396f7fe8f8300e24ee
dockeruser@vmdocker:~$ docker images
REPOSITORY           TAG      IMAGE ID      CREATED       SIZE
image_ubuntu_with_file    latest   cb9b513941ad  20 seconds ago  78.1MB
ubuntu                latest   6d79abd4c962  2 weeks ago   78.1MB
hello-world            latest   1b44b5a3e06a  6 weeks ago   10.1kB
python                 3.9     239b4550132d  6 weeks ago   1.09GB
python                 3.14.0a1  80ad471000e7  11 months ago  1.02GB
dockeruser@vmdocker:~$ docker history image_ubuntu_with_file
IMAGE      CREATED      CREATED BY      SIZE      COMMENT
cb9b513941ad  4 minutes ago               78.1MB      Imported from -
dockeruser@vmdocker:~$
```

8- Créez un conteneur à partir de cette image. Si vous avez un bash, vous pouvez vérifier que le fichier créé précédemment existe.

- Créer un conteneur à partir de l'image → docker run -it --name mon_conteneur_file image_ubuntu_with_file /bin/bash
- Vérifier que le fichier existe → ls /home → cat /home/mon_fichier.txt

Résultat :

```
dockeruser@vmdocker:~$ docker run -it --name mon_conteneur_file image_ubuntu_with_file /bin/bash
root@94bbb0a5a68f:/# ls /home
mon_fichier.txt  ubuntu
root@94bbb0a5a68f:/# cat /home/mon_fichier.txt
Contenu quelconque
root@94bbb0a5a68f:/#
```

9- A partir du conteneur os_ubuntu, committez maintenant une image image2 et consultez ses couches. Utilisez maintenant cette image pour créer un conteneur.

- docker commit mon_conteneur_file image2 → docker commit mon_conteneur_file image2
- Vérifier les couches → docker history image2
- Créer un conteneur à partir de l'image commitée → docker run -it --name conteneur2 image2 /bin/bash

Résultat :

```
dockeruser@vmdocker:~$ docker commit mon_conteneur_file image2
sha256:fc92ac6b2539118db8eeeea0bcacf06dceeb45d9029297231537338ebb67cae9f
dockeruser@vmdocker:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
image2              latest   fc92ac6b2539  12 seconds ago  78.1M
image_ubuntu_with_file  latest   cb9b513941ad  43 minutes ago  78.1M
ubuntu              latest   6d79abd4c962  2 weeks ago   78.1M
hello-world         latest   1b44b5a3e06a  6 weeks ago   10.1M
python               3.9     239b4550132d  6 weeks ago   1.09G
python               3.14.0a1  80ad471000e7  11 months ago  1.02G
dockeruser@vmdocker:~$ docker history image2
IMAGE      CREATED      CREATED BY      SIZE      COMMENT
fc92ac6b2539  12 seconds ago  /bin/bash    0B
cb9b513941ad  43 minutes ago                  78.1MB  Imported from -
dockeruser@vmdocker:~$ docker run -it --name conteneur2 image2 /bin/bash
root@848ffff8e231b:/#
```

10- A la fin de cette partie, il n'est plus nécessaire de conserver les images créées.

- Supprimer tous les conteneurs → `docker rm -f $(docker ps -aq)`
 - Supprimer toutes les images → `docker rmi -f image2 image_ubuntu_with_file`

Résultat :

```
dockeruser@vmdocker:~$ # ፩፻፭ ፩፻ ፩፻፭፻፭፻፭፻፭
dockeruser@vmdocker:~$ docker rm -f $(docker ps -aq)
፩፻፭ ፩፻፭ ፩፻፭፻፭
docker rmi -f image2 image_ubuntu_with_file
848ffff8e231b
94bbbb0a5a68f
515f48291f9c
dockeruser@vmdocker:~$
dockeruser@vmdocker:~$ # ፩፻፭ ፩፻፭ ፩፻፭ ፩፻፭
dockeruser@vmdocker:~$ docker rmi -f image2 image_ubuntu_with_file
Untagged: image2:latest
Deleted: sha256:fc92ac6b2539118db8eeea0bcfa06dceeb45d90292972315373388ebb67c
Untagged: image_ubuntu_with_file:latest
Deleted: sha256:cb9b513941add3a0edd32d4f6ab7d065325523a10bcf82396f7fe8f8300e
Deleted: sha256:555ef82f394c85b6274cf9e3544620812a2a7791cd8a4b27f39acefd6780
```

Partie 7 – Quelques informations générales

1- Utilisez une commande docker pour consulter la consommation des conteneurs en exécution.

- docker run -d --name test_ubuntu ubuntu sleep 300
- docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS								
CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS	
d635e8dad42b	test_ubuntu	0.00%	644KiB / 1.918GiB	0.03%	876B / 126B	614kB / 0B	1	

2- Utilisez une commande docker pour voir la consommation disque des différents objets docker. Essayez aussi la version détaillée.

- commande de base → docker system df
- Version détaillée → docker system df -v

Affiche tous les objets en détail, y compris :

- chaque image avec ses couches et taille
- chaque conteneur et son état
- chaque volume et son utilisation exacte

```
dockeruser@vmdocker:~$ docker system df
TYPE      TOTAL     ACTIVE      SIZE      RECLAMABLE
Images      4          0      2.188GB  2.188GB (100%)
Containers    0          0        0B        0B
Local Volumes 0          0        0B        0B
Build Cache   0          0        0B        0B
dockeruser@vmdocker:~$ docker system df -v
Images space usage:
REPOSITORY    TAG      IMAGE ID      CREATED      SIZE      SHARED SIZE  UNIQUE SIZE  CONTAINERS
ubuntu        latest    6d79abd4c962  2 weeks ago  78.1MB    0B        78.12MB    0
hello-world   latest    1b44b5a3e06a  6 weeks ago  10.1kB    0B        10.07kB    0
python         3.9      239b4550132d  6 weeks ago  1.09GB    0B        1.09GB     0
python         3.14.0a1  80ad471000e7  11 months ago 1.02GB    0B        1.02GB     0

Containers space usage:
CONTAINER ID    IMAGE      COMMAND      LOCAL VOLUMES      SIZE      CREATED      STATUS      NAMES

Local Volumes space usage:
VOLUME NAME    LINKS      SIZE

Build cache usage: 0B
CACHE ID    CACHE TYPE      SIZE      CREATED      LAST USED      USAGE      SHARED
dockeruser@vmdocker:~$ ■
```