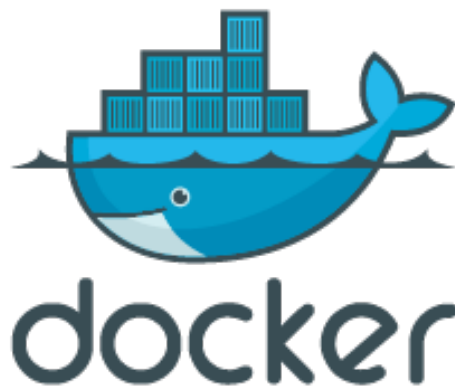


Rapport Docker

Université d'Avignon

Filière : Master 1 en intelligence artificielle



Auteur : KELI Kékéli Christ

Date : 26 septembre 2025

Table des matières

0.1	Partie 2	4
0.2	Utilisez une commande docker pour afficher la version (client et serveur!) .	4
0.3	Comment voir les composants du daemon Docker qui tournent?	4
0.4	Quels sont les services/socket utilisés par docker? Quel utilisateur a démarré ces services?	6
0.5	Que faire pour arrêter docker? Quel est le statut du socket?	7
0.6	Que faire pour désactiver/réactiver docker?	7
0.7	réessayez d'afficher la version sans passer en sudo	8
0.8	Partie 3	8
0.9	Quel est le répertoire dans lequel Docker stocke ses objets?	8
0.10	Quels sont les différentes catégories d'objets Docker qui peuvent être stockés?	9
0.11	Consultez le contenu du répertoire approprié qui contient les conteneurs : combien y en a-t-il pour l'instant?	9
0.12	Utilisez une commande pour rechercher des images, essayez avec l'image hello-world	9
0.13	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. . . .	10
0.14	Vérifiez maintenant le contenu du répertoire des conteneurs. Combien y a-t-il de conteneurs?	12
0.15	Vérifiez aussi le contenu du répertoire des images	12
0.16	Quel est le sha256 de l'image?	12
0.17	Utilisez la commande docker (sans option) qui permet de lister les conteneurs en exécution.	13
0.18	Utilisez la commande docker qui permet de lister les images. Quel est l'identifiant de l'image utilisée?	13
0.19	Quel est l'alias de la commande que vous venez d'utiliser?	13
0.20	Réexécutez le conteneur et comparez	13
0.21	Nombre de conteneurs actifs et stockés	14
0.22	Essayez de supprimer l'image et expliquez	14
0.23	Lister tous les conteneurs et observer leur statut	14
0.24	Nom des conteneurs	14
0.25	Affichage non tronqué	14
0.26	Exécuter l'image et supprimer le conteneur par son nom	15
0.27	Exécuter l'image avec son sha256 et nommer le conteneur	15
0.28	Supprimer l'image (sans forcer)	15
0.29	Infos globales sur le système	15
0.30	Afficher uniquement les IDs des conteneurs	15
0.31	Supprimer tous les conteneurs en une seule commande	16
0.32	Supprimer l'image	16

0.33	Exécuter un conteneur nommé hello1	16
0.34	Identifier l'image utilisée	16
0.35	Créer un conteneur nommé hello2 sans le démarrer	16
0.36	Démarrer hello2 et observer son statut	16
0.37	Démarrer hello2 avec attachement à l'entrée/sortie	16
0.38	Démarrer hello1 avec attachement à l'entrée/sortie	17
0.39	Exécuter hello3 en arrière-plan	17
0.40	Exécuter hello4 et le faire disparaître après exécution	17
0.41	Supprimer l'image hello-world	17
0.42	Partie 4	17
0.43	Types d'images sur Docker Hub	17
0.44	Image officielle Ubuntu	18
0.45	Versions proposées et distinction	18
0.46	Comparaison des versions	18
0.47	Vulnérabilités	18
0.48	Classification des vulnérabilités	18
0.49	Intérêt de la comparaison latest vs nooble	18
0.50	Différences dans les couches	19
0.51	Liste du répertoire Docker local	19
0.52	Téléchargement de l'image Ubuntu par défaut	19
0.53	Téléchargement de la version la plus récente	19
0.54	Filtrer les images Ubuntu	19
0.55	Commande exécutée par défaut dans le conteneur	20
0.56	Suppression de l'image ubuntu:rolling	20
0.57	Partie 5	20
0.58	Lancer un conteneur Ubuntu et observer son statut	20
0.59	Redémarrer le conteneur en interactif	21
0.60	Supprimer le conteneur	21
0.61	Créer un conteneur nommé os_ubuntu en interactif	21
0.62	Identifier le processus PID 1 dans le conteneur	21
0.63	Exécuter les commandes système dans le conteneur	22
0.64	Observer le statut du conteneur depuis un autre terminal	22
0.65	Se déplacer dans le répertoire home	22
0.66	Quitter le conteneur avec exit et observer son statut	22
0.67	Redémarrer le conteneur en interactif et observer le répertoire courant	22
0.68	Inspecter le conteneur et retrouver son PID	22
0.69	Utiliser jq pour extraire le PID	23
0.70	Observer le processus correspondant au PID	23
0.71	Retourner dans le répertoire home	23
0.72	Quitter le conteneur avec Ctrl-p Ctrl-q	23
0.73	Attacher le terminal et créer un fichier	23
0.74	Voir les différences dans le système de fichiers	24
0.75	Exécuter hostname sans foreground	24
0.76	Exécuter bash en interactif et vérifier les processus	24
0.77	Afficher les processus du conteneur	24
0.78	Arrêter le conteneur en le quittant	25
0.79	Lancer un conteneur nommé ll avec ls -l	25
0.80	Redémarrer le conteneur ll	25

0.81	Lancer un conteneur ps avec ps aux et suppression automatique	25
0.82	Lancer un conteneur salut avec echo Bonjour	25
0.83	Lancer une commande infinie et supprimer le conteneur	25
0.84	Afficher salut toutes les 3 secondes dans os_ubuntu	25
0.85	Sattacher au conteneur depuis un autre terminal	26
0.86	Interrompre le processus et exécuter une commande	26
0.87	Quitter avec un code retour non nul	26
0.88	Voir les logs du conteneur os_ubuntu	26
0.89	Inspecter le fichier de log avec jq	26
0.90	Supprimer tous les conteneurs arrêtés	27
0.91	Arrêter et supprimer tous les conteneurs actifs	27
0.92	Partie 6	27
0.93	Téléchargez l'image python:3.9	27
0.94	Combien de couches ont été téléchargées ?	27
0.95	Observez les couches de cette image (option non tronquée)	28
0.96	Quelle est la dernière couche ?	28
0.97	Téléchargez l'image python:3.14.0a1	28
0.98	Combien de couches ont été téléchargées ? Expliquez.	29
0.99	Afficher les couches des deux images avec jq et observer les couches communes	29
0.100	Créer un conteneur os_ubuntu et ajouter un fichier dans /home	29
0.101	Exporter le système de fichiers dans une archive tar	29
0.102	Créer une image image_ubuntu_with_file à partir de l'archive	29
0.103	Comment est-elle taggée ? Observer ses couches	30
0.104	Créer un conteneur à partir de cette image et vérifier le fichier	30
0.105	Commiter le conteneur os_ubuntu en une image image2 et consulter ses couches	30
0.106	Créer un conteneur à partir de l'image image2	30
0.107	Nettoyage final : supprimer les images créées	30
0.108	Partie 7 Quelques informations générales	31
0.108.1	Consulter la consommation des conteneurs en exécution	31
0.108.2	Voir la consommation disque des objets Docker	31
0.108.3	Version détaillée de la consommation disque	31

0.1 Partie 2

0.2 Utilisez une commande docker pour afficher la version (client et serveur !)

Commande

```
sudo docker version
```

```
dockeruser@vmdocker:~$ sudo docker version
[sudo] password for dockeruser:
Client: Docker Engine - Community
 Version: 28.4.0
 API version: 1.49 (downgraded from 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:
 Engine:
  Version: 28.1.1+1
  API version: 1.49 (minimum version 1.24)
  Go version: go1.23.8
  Git commit: 01f442b
  Built: Fri Jun 13 16:12:14 2025
  OS/Arch: linux/amd64
  Experimental: false
 containerd:
  Version: v1.7.27
  GitCommit: 05044ec0a9a75232cad458027ca83437aae3f4da
 runc:
  Version: 1.2.6
  GitCommit:
 docker-init:
  Version: 0.19.0
  GitCommit: de40ad0
```

La version du client docker est 28.4.0

La version du serveur docker est 28.1.1

0.3 Comment voir les composants du daemon Docker qui tournent ?

Commande

Pour voir les composants docker qui tourne il faut utiliser la commande
`sudo docker info`

```

dockeruser@vmdocker:~$ sudo docker info
[sudo] password for dockeruser:
Client: Docker Engine - Community
Version: 28.4.0
Context: default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version: v0.27.0
    Path: /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version: v2.39.2
    Path: /usr/libexec/docker/cli-plugins/docker-compose

Server:
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 28.1.1+1
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 journald json-file local splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 05044ec0a9a75232cad458027ca83437aee3f4da
runc version:
init version: de40ad0
Security Options:
  apparmor
  seccomp
   Profile: builtin
  cgroupns
Kernel Version: 5.15.0-118-generic
Operating System: Ubuntu Core 22
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.918GiB
Name: vmdocker
ID: 5a8bc8cf-dfe0-433c-811e-254437e343e9
Docker Root Dir: /var/snap/docker/common/var-lib-docker
Debug Mode: false
Experimental: false
Insecure Registries:
  ::1/128
  127.0.0.0/8
Live Restore Enabled: false

```

Les composants du daemon docker sont :

- Image
- Volume
- Container
- Réseau

0.4 Quels sont les services/socket utilisés par docker ? Quel utilisateur a démarré ces services ?

Commande

Pour voir le service docker `systemctl status docker` et
`ps -aux | grep dockerd` pour voir l'utilisateur qui a démarré ces services .

```
dockeruser@vmdocker:~$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-09-19 11:57:25 UTC; 1h 9min ago
     TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
      Main PID: 787 (dockerd)
         Tasks: 9
        Memory: 39.7M
           CPU: 1.086s
       CGroup: /system.slice/docker.service
               └─787 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

sept. 19 11:57:24 vmdocker dockerd[787]: time="2025-09-19T11:57:24.514153344Z" level=info msg=
sept. 19 11:57:24 vmdocker dockerd[787]: time="2025-09-19T11:57:24.523556567Z" level=info msg=
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.210721696Z" level=warning m
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.336576531Z" level=info msg=
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.483255601Z" level=info msg=
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.484930168Z" level=info msg=
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.702229030Z" level=info msg=
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.711054611Z" level=info msg=
sept. 19 11:57:25 vmdocker systemd[1]: Started Docker Application Container Engine.
sept. 19 11:57:25 vmdocker dockerd[787]: time="2025-09-19T11:57:25.716388482Z" level=info msg=

dockeruser@vmdocker:~$ ps -aux | grep dockerd
root      787   0.0  1.9 1899308 38872 ?        Ssl  11:57   0:00 /usr/bin/dockerd -H fd:// -
root      990   0.0  1.5 1972212 30828 ?        Ssl  11:57   0:02 dockerd --group docker --ex
mon.json
dockeru+  40160 0.0  0.1  6612  2288 pts/0    S+   13:07   0:00 grep --color=auto dockerd
dockeruser@vmdocker:~$ sudo docker inspect
[sudo] password for dockeruser:
docker: 'docker inspect' requires at least 1 argument

Usage:  docker inspect [OPTIONS] NAME|ID [NAME|ID...]

See 'docker inspect --help' for more information
```

Le service utilisé est : `docvker.service`
et l'utilisateur qui l'a démarré est le `root`

0.5 Que faire pour arrêter docker ? Quel est le statut du socket ?

Commande

Pour arrêter docker il faut utiliser la commande `sudo systemctl stop docker` et pour vérifier le statut du socket il faut utiliser `sudo systemctl status docker.socket`

```
dockeruser@vmdocker:~$ sudo systemctl status docker.socket
o 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 13:50:56 UTC; 5s ago
     Triggers: ● docker.service
    Listen: /run/docker.sock (Stream)
     CPU: 482us

sept. 19 11:57:17 vmdocker systemd[1]: Starting Docker Socket for the API...
sept. 19 11:57:17 vmdocker systemd[1]: Listening on Docker Socket for the API.
sept. 19 13:50:56 vmdocker systemd[1]: docker.socket: Deactivated successfully.
sept. 19 13:50:56 vmdocker systemd[1]: Closed Docker Socket for the API.
```

Le statut du socket est inactif

0.6 Que faire pour désactiver/réactiver docker ?

Commande

Pour désactiver docker il faut utiliser `sudo systemctl disable docker` et pour réactiver docker il faut utiliser `sudo systemctl enable docker`

```
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.
```


0.7 réessayez d'afficher la version sans passer en sudo

```
dockeruser@vmdocker:~$ docker version
Client: Docker Engine - Community
Version: 28.4.0
API version: 1.49 (downgraded from 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:
Engine:
Version: 28.1.1+1
API version: 1.49 (minimum version 1.24)
Go version: go1.23.8
Git commit: 01f442b
Built: Fri Jun 13 16:12:14 2025
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: v1.7.27
GitCommit: 05044ec0a9a75232cad458027ca83437aae3f4da
runc:
Version: 1.2.6
GitCommit:
docker-init:
Version: 0.19.0
GitCommit: de40ad0
```

0.8 Partie 3

0.9 Quel est le répertoire dans lequel Docker stocke ses objets ?

Commande

Docker stocke ses objets ici : `/var/snap/docker/common/var-lib-docker`

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

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

Commande

Les différentes catégories d'objets Docker qui peuvent être stockés sont :

- Images
- Containers
- Volumes
- Cache

```
dockeruser@vmdocker:~$ docker system df
TYPE                TOTAL        ACTIVE        SIZE        RECLAIMABLE
Images              0            0            0B          0B
Containers          0            0            0B          0B
Local Volumes       0            0            0B          0B
Build Cache         0            0            0B          0B
```

0.11 Consultez le contenu du répertoire approprié qui contient les conteneurs : combien y en a-t-il pour l'instant ?

Commande

Il y'a zéro conteneur pour l'instant .

```
dockeruser@vmdocker:~$ sudo ls /var/snap/docker/common/var-lib-docker/
buildkit containerd containers engine-id image network overlay2 plugins runtimes swarm tmp volumes
dockeruser@vmdocker:~$ sudo ls /var/snap/docker/common/var-lib-docker/containers
```

0.12 Utilisez une commande pour rechercher des images, essayez avec l'image hello-world

Commande

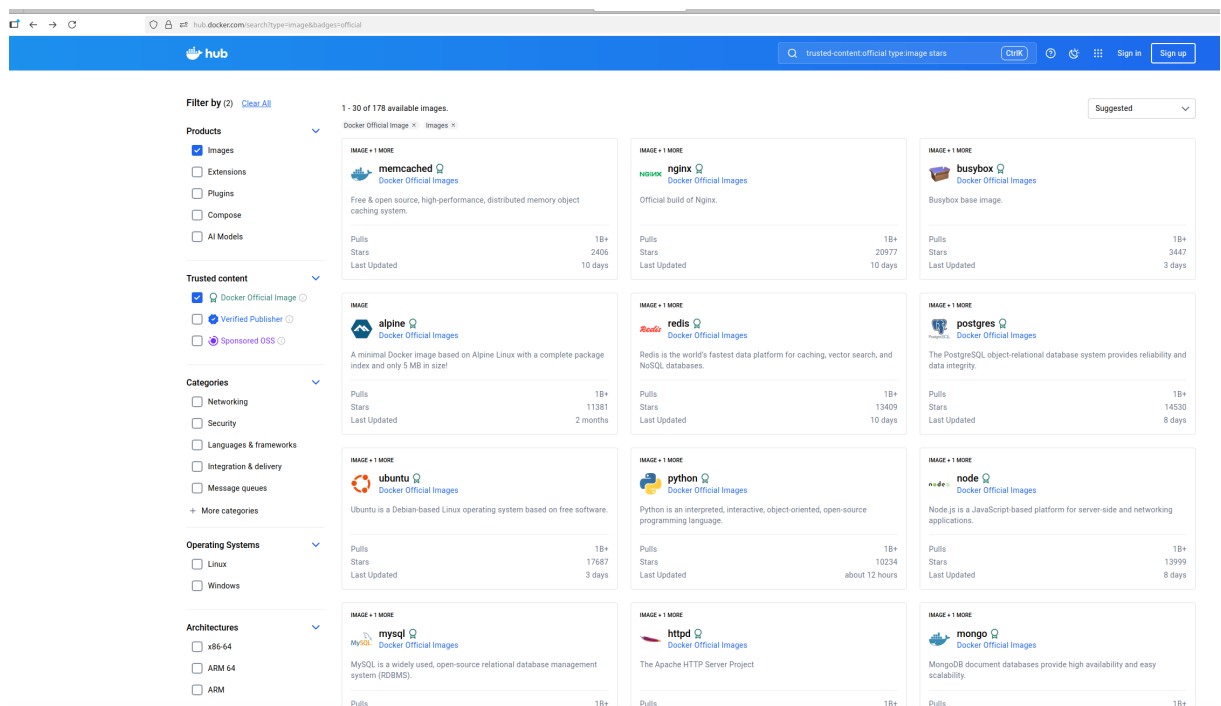
Il faut utiliser la commande `docker search hello-world`

```

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                   91
dockercloud/hello-world             Hello World!                                20
crccheck/hello-world                Hello World web server in under 2.5 MB        26
koudaiii/hello-world                0
ppc64le/hello-world                 2
tsepotesting123/hello-world          0
prajwalendra/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                 3
twistlocktest/hello-world            0
datawire/hello-world                Hello World! Simple Hello World implementati... 1
uniplaces/hello-world                0
wjimenez5271/hello-world             0
arm64v8/hello-world                 3
danfengliu/hello-world              0
lbadger/hello-world                 0
ansibleplaybookbundle/hello-world   Simple containerized application that tests ... 0
swarna3005/hello-world               0

```

0.13 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.



Commande

Il faut utiliser la commande `docker search hello-world` pour rechercher l'image `hello-world` et s'assurer qu'il existe. Une fois terminer télécharger nginx avec `docker pull hello-world` et ensuite démarrer avec `docker run hello-world`

```
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!                20
cnccheck/hello-world  Hello World web server in under 2.5 MB         26
koudaiiii/hello-world 0
ppc64le/hello-world   Hello World! (an example of minimal Dockeriz... 2
tsepotesting123/hello-world 0
prajwalendra/hello-world 0
kevinodockercompany/hello-world 0
infrastructureascode/hello-world A tiny "Hello World" web server with a healt... 1
arm32v7/hello-world   Hello World! (an example of minimal Dockeriz... 3
cloudflare/hello-world A simple example application which can be ru... 0
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:~$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
17eec7bbc9d7: Pull complete
Digest: sha256:54e66c1dd1fcb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31
Status: Downloaded newer image 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/
```

0.14 Vérifiez maintenant le contenu du répertoire des conteneurs. Combien y a-t-il de conteneurs ?

Commande

Il faut utiliser la commande

```
sudo ls /var/snap/docker/common/var-lib-docker/containers
```

```
dockeruser@vmdocker:~$ sudo ls /var/snap/docker/common/var-lib-docker/containers
[sudo] password for dockeruser:
893dfc262d89e36f29fd0e7b17306b216e30738d9181fea661e51a7c2f3e5350
```

On remarque qu'il y'a un container.

0.15 Vérifiez aussi le contenu du répertoire des images

Commande

Il faut utiliser la commande

```
sudo ls /var/snap/docker/common/var-lib-docker/image
```

```
dockeruser@vmdocker:~$ sudo ls /var/snap/docker/common/var-lib-docker/
buildkit containerd containers engine-id image network overlay2 plugins runtimes swarm tmp volumes
dockeruser@vmdocker:~$ sudo ls /var/snap/docker/common/var-lib-docker/image
overlay2
```

On remarque qu'il y'a une image.

0.16 Quel est le sha256 de l'image ?

Commande

Il faut utiliser la commande

```
docker images --digests
```

```
dockeruser@vmdocker:~$ docker images --digests
REPOSITORY TAG DIGEST IMAGE ID CREATED SIZE
hello-world latest sha256:54e66cc1dd1fcb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31 1b44b5a3e06a 5 weeks ago 10.1kB
```

On remarque que le sha256 de l'image est
sha256:54e66cc1dd1fcb1c3c58bd8017914dbed8701e2d8c74d9262e26bd9cc1642d31.

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

Commande

Il faut utiliser la commande
`docker ps`

On remarque qu'il y'a aucun container en cours d'exécution.

0.18 Utilisez la commande docker qui permet de lister les images. Quel est l'identifiant de l'image utilisée ?

Commande

Il faut utiliser la commande
`docker images`

On remarque que l'identifiant de l'image utilisé est **1b44b5a3e06a**.

0.19 Quel est l'alias de la commande que vous venez d'utiliser ?

Commande

Docker container ls -a

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

0.20 Réexécuter le conteneur et comparer

Commande

`docker run hello-world`

Lors de la première exécution, Docker télécharge l'image si elle n'est pas présente. Lors des exécutions suivantes, l'image est réutilisée localement, ce qui accélère le processus. Chaque exécution crée un nouveau conteneur.

0.21 Nombre de conteneurs actifs et stockés

Commandes

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

```
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
4957dde4d5fb   hello-world  "/hello"   6 days ago   Exited (0) 6 days ago   beautiful_liskov
dockeruser@vmdocker:~$
```

`docker ps` affiche les conteneurs en cours d'exécution. `docker ps -a` affiche tous les conteneurs, y compris ceux arrêtés.

0.22 Essayer de supprimer l'image et expliquer

Commande

```
docker rmi hello-world
```

Si des conteneurs utilisent encore l'image, Docker empêche sa suppression. Il faut d'abord supprimer les conteneurs associés.

0.23 Lister tous les conteneurs et observer leur statut

Commande

```
docker ps -a
```

Cette commande affiche tous les conteneurs avec leur statut : Up, Exited, Created, etc.

0.24 Nom des conteneurs

Les noms des conteneurs sont affichés dans la colonne `NAMES` de la commande `docker ps -a`.

0.25 Affichage non tronqué

Commande

```
docker ps -a --no-trunc
```

Cette option permet d'afficher les identifiants et commandes complets, sans coupure.

0.26 Exécuter l'image et supprimer le conteneur par son nom

Commandes

```
docker run -name test1 hello-world  
docker rm test1
```

0.27 Exécuter l'image avec son sha256 et nommer le conteneur

Commande

```
docker run -name test2 hello-world@sha256:<digest>
```

Le conteneur est créé avec le nom `test2` et l'image est identifiée par son digest SHA256.

0.28 Supprimer l'image (sans forcer)

Commande

```
docker rmi hello-world
```

La suppression est possible uniquement si aucun conteneur n'utilise l'image.

0.29 Infos globales sur le système

Commande

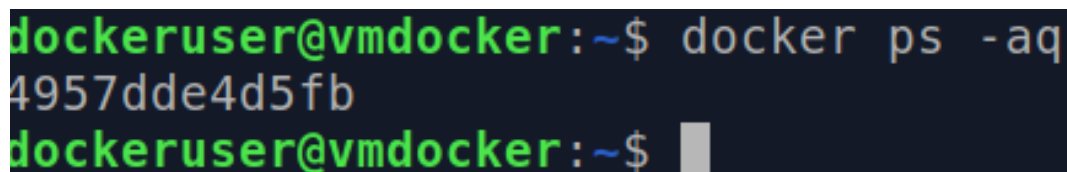
```
docker system info
```

Cette commande affiche des statistiques sur les conteneurs, images, volume [utf8]inputencs, et autres ressources Docker.

0.30 Afficher uniquement les IDs des conteneurs

Commande

```
docker ps -aq
```



```
dockeruser@vmdocker:~$ docker ps -aq  
4957dde4d5fb  
dockeruser@vmdocker:~$
```


0.31 Supprimer tous les conteneurs en une seule commande

Commande

```
docker rm $(docker ps -aq)
```

0.32 Supprimer l'image

Commande

```
docker rmi hello-world
```

0.33 Exécuter un conteneur nommé hello1

Commande

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

Le conteneur s'exécute et affiche le message de bienvenue.

0.34 Identifier l'image utilisée

Commande

```
docker images
```

0.35 Créer un conteneur nommé hello2 sans le démarrer

Commande

```
docker create -name hello2 hello-world
```

0.36 Démarrer hello2 et observer son statut

Commande

```
docker start hello2
```

Le conteneur s'exécute brièvement puis passe à l'état Exited.

0.37 Démarrer hello2 avec attachement à l'entrée/sortie

Commande

```
docker start -a hello2
```

0.38 Démarrer hello1 avec attachement à l'entrée/sortie

Commande

```
docker start -a hello1
```

0.39 Exécuter hello3 en arrière-plan

Commande

```
docker run -d -name hello3 hello-world
```

0.40 Exécuter hello4 et le faire disparaître après exécution

Commande

```
docker run -rm -name hello4 hello-world
```

Le conteneur est supprimé automatiquement après son exécution.

0.41 Supprimer l'image hello-world

Commande

```
docker rmi hello-world
```

0.42 Partie 4

0.43 Types d'images sur Docker Hub

Sur Docker Hub, on retrouve principalement trois types d'images :

- **Images officielles** : Ces images sont maintenues soit par Docker lui-même, soit par les éditeurs des logiciels (comme Ubuntu ou MySQL). Elles sont généralement bien documentées, régulièrement mises à jour et testées.
- **Images Verified Publishers** : Elles sont publiées par des éditeurs reconnus, comme Microsoft, Red Hat ou Bitnami. Ces images sont identifiées par un badge de vérification, ce qui garantit leur authenticité et leur fiabilité.
- **Images communautaires** : Ces images sont créées et partagées par des membres de la communauté Docker. Elles ne sont pas vérifiées officiellement, et leur qualité peut varier en fonction de leur auteur.

Cette classification repose essentiellement sur la provenance de l'image, sa fiabilité, et sa fonction (système d'exploitation, base de données, etc.).

0.44 Image officielle Ubuntu

L'image officielle d'Ubuntu est disponible sur Docker Hub à l'adresse suivante : https://hub.docker.com/_/ubuntu

Elle est maintenue par Canonical et validée par Docker. C'est une image de base très utilisée pour créer des conteneurs Linux.

0.45 Versions proposées et distinction

Les différentes versions d'Ubuntu disponibles sur Docker Hub sont identifiées par des **tags**. Les plus courants sont :

- **latest** : représente la version stable actuelle, utilisée par défaut si aucun tag n'est précisé.
- **noble**, **jammy**, **22.04**, etc. : correspondent à des versions précises d'Ubuntu, soit par leur nom de code, soit par leur numéro de version.

Chaque tag est associé à un **digest SHA256** unique qui permet d'identifier précisément l'image, quelle que soit sa provenance.

0.46 Comparaison des versions

Voici un tableau comparatif entre deux tags couramment utilisés :

Tag	Nom de version	Remarque
latest	Ubuntu 22.04 LTS (Jammy)	Version stable par défaut
noble	Ubuntu 24.04	Version la plus récente

0.47 Vulnérabilités

Les vulnérabilités présentes dans les images Docker peuvent être analysées avec des outils comme Docker Scout. On remarque que :

- L'image **latest** contient parfois des vulnérabilités connues, mais elles sont généralement corrigées rapidement par les mainteneurs.
- L'image **noble**, étant plus récente, peut inclure des failles encore non identifiées ou non corrigées.

0.48 Classification des vulnérabilités

Les vulnérabilités sont identifiées sous forme de **CVE** (Common Vulnerabilities and Exposures) et classées selon :

- **Leur niveau de sévérité** : Low, Medium, High ou Critical.
- **Le composant concerné** : par exemple, un paquet système, une bibliothèque, ou une dépendance logicielle.

0.49 Intérêt de la comparaison latest vs noble

Comparer les deux tags permet de mieux choisir l'image selon les besoins du projet :

- **latest** : idéal pour les environnements stables, avec un support à long terme.
- **noble** : utile pour tester les dernières nouveautés, ou bénéficier des dernières améliorations.

Il sagit donc de trouver un équilibre entre fiabilité et modernité.

0.50 Différences dans les couches

En observant les différentes couches des images (layers) disponibles sur Docker Hub, on constate que les différences apparaissent principalement lors de l'installation des paquets via **apt**. Cela permet d'identifier les changements entre deux versions précises.

0.51 Liste du répertoire Docker local

Commande

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

Cette commande permet de lister les couches d'images Docker stockées localement, identifiées par leur empreinte SHA256.

0.52 Téléchargement de l'image Ubuntu par défaut

Commande

```
docker pull ubuntu
```

Cette commande télécharge l'image Ubuntu par défaut, qui correspond au tag **latest**.

0.53 Téléchargement de la version la plus récente

Commande

```
docker pull ubuntu:noble
```

Télécharge explicitement la version Ubuntu 24.04 (tag **noble**).

0.54 Filtrer les images Ubuntu

Commande

```
docker images | grep ubuntu
```

Cette commande affiche uniquement les images Docker locales dont le nom contient le mot ubuntu.

0.55 Commande exécutée par défaut dans le conteneur

Commande

```
docker inspect ubuntu --format='{{.Config.Cmd}}'
```

Affiche la commande qui est exécutée par défaut à l'intérieur du conteneur basé sur l'image Ubuntu.
Par exemple : ["bash"]

0.56 Suppression de l'image ubuntu:rolling

Commande

```
docker rmi ubuntu:rolling
```

Supprime l'image locale ubuntu:rolling du système Docker.

0.57 Partie 5

0.58 Lancer un conteneur Ubuntu et observer son statut

Commande

```
docker run -it --name test_ubuntu ubuntu  
docker ps -a
```

```
Last login: Thu Sep 23 10:14:58 2023 from 192.168.57.1  
dockeruser@vmdocker:~$ docker run -it --name test_ubuntu ubuntu  
docker ps -a  
Unable to find image 'ubuntu:latest' locally  
latest: Pulling from library/ubuntu  
953cdd413371: Pull complete  
Digest: sha256:353675e2a41babd526e2b837d7ec780c2a05bca0164f7ea5dbbd433d21d166fc  
Status: Downloaded newer image for ubuntu:latest  
root@c9f04b7f95f7:/#
```

0.59 Redémarrer le conteneur en interactif

Commande

```
docker start -ai test_ubuntu
```

```
dockeruser@vmdocker:~$ docker start -ai test_ubuntu
root@c9f04b7f95f7:/#
```

0.60 Supprimer le conteneur

Commande

```
docker rm test_ubuntu
```

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

0.61 Créer un conteneur nommé os_ubuntu en interactif

Commande

```
docker run -it --name os_ubuntu ubuntu
```

```
dockeruser@vmdocker:~$ docker run -it --name os_ubuntu ubuntu
root@6e3461400762:/#
```

0.62 Identifier le processus PID 1 dans le conteneur

Commande

Dans le conteneur : `ps -aux`

```
root@6e3461400762:/# ps -aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1  4588  3824 pts/0    Ss   12:26   0:00 /bin/bash
root          10  0.0  0.2  7888  4036 pts/0    R+   12:27   0:00 ps -aux
root@6e3461400762:/# exit
```

Il s'agit du bash

0.63 Exécuter les commandes système dans le conteneur

Commande

```
whoami, pwd, ls, hostname
```

0.64 Observer le statut du conteneur depuis un autre terminal

Commande

```
docker ps
```

0.65 Se déplacer dans le répertoire home

Commande

```
cd /home
```

0.66 Quitter le conteneur avec exit et observer son statut

Commande

```
exit puis docker ps -a
```

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

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6e3461400762	ubuntu	"/bin/bash"	7 minutes ago	Exited (0) 5 minutes ago		os_ubuntu
4957dde4d5fb	hello-world	"/hello"	6 days ago	Exited (0) 6 days ago		beautiful_liskov

0.67 Redémarrer le conteneur en interactif et observer le répertoire courant

Commande

```
docker start -ai os_ubuntu puis pwd
```

on se trouve dans le repertoire racine (/)

0.68 Inspecter le conteneur et retrouver son PID

Commande

```
docker inspect -format '{{.State.Pid}}' os_ubuntu
```

0.69 Utiliser jq pour extraire le PID

Commande

```
docker inspect os_ubuntu | jq '.[0].State.Pid'
```

```
dockeruser@vmdocker:/home$ docker inspect --format '{{.State.Pid}}' os_ubuntu
2639
dockeruser@vmdocker:/home$ docker inspect os_ubuntu | jq '.[0].State.Pid'
2639
dockeruser@vmdocker:/home$
```

0.70 Observer le processus correspondant au PID

Commande

```
ps -fp 2639
```

```
dockeruser@vmdocker:/home$ ps -fp 2639
UID          PID    PPID  C STIME TTY          TIME CMD
root         2639    2616  0 12:51 pts/0      00:00:00 /bin/bash
dockeruser@vmdocker:/home$
```

0.71 Retourner dans le répertoire home

Commande

```
cd /home
```

0.72 Quitter le conteneur avec Ctrl-p Ctrl-q

Commande

Taper Ctrl-p Ctrl-q pour détacher sans arrêter.

0.73 Attacher le terminal et créer un fichier

Commande

```
docker attach os_ubuntu puis echo "contenu" > fichier.txt
```

```
dockeruser@vmdocker:/home$ docker attach os_ubuntu
echo "contenu" > fichier.txt
```


0.74 Voir les différences dans le système de fichiers

Commande

```
docker diff os_ubuntu
```

```
dockeruser@vmdocker:~$ docker diff os_ubuntu
C /root
A /root/.bash_history
dockeruser@vmdocker:~$
```

0.75 Exécuter hostname sans foreground

Commande

```
docker exec os_ubuntu hostname
```

```
dockeruser@vmdocker:~$ docker exec os_ubuntu hostname
6e3461400762
dockeruser@vmdocker:~$
```

0.76 Exécuter bash en interactif et vérifier les processus

Commande

```
docker exec -it os_ubuntu bash puis ps aux | grep bash
```

```
dockeruser@vmdocker:~$ docker exec -it os_ubuntu bash
ps aux | grep bash
root@6e3461400762:/# ps aux | grep bash
root      1  0.0  0.1  4588  3868 pts/0    Ss+  12:57   0:00 /bin/bash
root     16  0.0  0.1  4588  3896 pts/1    Ss   12:59   0:00 bash
root     25  0.0  0.0   3528  1624 pts/1    S+   12:59   0:00 grep --color=auto bash
root@6e3461400762:/#
```

0.77 Afficher les processus du conteneur

Commande

```
docker top os_ubuntu
```

```
dockeruser@vmdocker:~$ docker top os_ubuntu
      PID             PPID             C      STIME      TTY      TIME      CMD
root    2786             2764             0      12:57      ?         00:00:00 /bin/bash
dockeruser@vmdocker:~$
```

0.78 Arrêter le conteneur en le quittant

Commande

```
exit
```

0.79 Lancer un conteneur nommé ll avec ls -l

Commande

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

0.80 Redémarrer le conteneur ll

Commande

```
docker start -ai ll
```

0.81 Lancer un conteneur ps avec ps aux et suppression automatique

Commande

```
docker run -name ps -rm ubuntu ps aux
```

0.82 Lancer un conteneur salut avec echo Bonjour

Commande

```
docker run -name salut ubuntu echo Bonjour  
docker start -ai salut
```

0.83 Lancer une commande infinie et supprimer le conteneur

Commande

```
docker run -name infini -d ubuntu sh -c "while true; do sleep 3600;  
done"  
docker rm -f infini
```

0.84 Afficher salut toutes les 3 secondes dans os_ubuntu

Commande

```
while true; do echo salut; sleep 3; done
```

0.85 Sattacher au conteneur depuis un autre terminal

Commande

```
docker attach os_ubuntu
```

0.86 Interrompre le processus et exécuter une commande

Commande

```
Ctrl-C puis ls
```

0.87 Quitter avec un code retour non nul

Commande

```
exit 1 puis docker ps -a
```

0.88 Voir les logs du conteneur os_ubuntu

Commande

```
docker logs os_ubuntu
```

```
dockeruser@vmdocker:~$ docker logs os_ubuntu
root@6e3461400762:/# ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  4588  3824 pts/0    Ss   12:26   0:00 /bin/bash
root        10  0.0  0.2   788   4036 pts/0    R+   12:27   0:00 ps -aux
root@6e3461400762:/# exit
exit
root@6e3461400762:/#
root@6e3461400762:/# pwd
/
root@6e3461400762:/# docker ps
bash: docker: command not found
root@6e3461400762:/# docker inspect --format '{{.State.Pid}}' os_ubuntu
bash: docker: command not found
root@6e3461400762:/# exit
exit
root@6e3461400762:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@6e3461400762:/# cd home
root@6e3461400762:/home# ls
ubuntu
root@6e3461400762:/home# exit
exit
dockeruser@vmdocker:~$
```

0.89 Inspecter le fichier de log avec jq

Commande

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

```
Fichier Edition Affichage Terminal Onglets Aide
dockeruser@vmdocker:~$ docker inspect os_ubuntu | jq '[0].LogPath'
"/var/lib/docker/containers/0e3461400762f6323952c0de978a29ee0532e19f0580bda8ac48287f5ce28e5/0e3461400762f6323952c0de978a29ee0532e19f0580bda8ac48287f5ce28e5.json.log"
dockeruser@vmdocker:~$
```

0.90 Supprimer tous les conteneurs arrêtés

Commande

```
docker container prune
```

0.91 Arrêter et supprimer tous les conteneurs actifs

Commande

```
docker rm -f $(docker ps -q)
```

0.92 Partie 6

0.93 Téléchargez l'image python:3.9

Commande

```
docker pull python:3.9
```

```
dockeruser@vmdocker:~$ docker pull python:3.9
3.9: Pulling from library/python
15b1d8a5ff03: Pull complete
22718812f617: Pull complete
401a98f7495b: Pull complete
ad446e7df19a: Pull complete
93e46ac5a04b: Pull complete
2dfb635db596: Pull complete
01cb890d7a54: Pull complete
Digest: sha256:16a475293c7b2a3abc191278120d93dd11d47c7e8d3ca216b0d146e30d273689
Status: Downloaded newer image for python:3.9
docker.io/library/python:3.9
dockeruser@vmdocker:~$
```

0.94 Combien de couches ont été téléchargées ?

Commande

```
docker image inspect python:3.9 | jq '[0].RootFS.Layers | length'
```

```
dockeruser@vmdocker:~$ docker image inspect python:3.9 | jq '[0].RootFS.Layers | length'
7
dockeruser@vmdocker:~$
```

Il y'a donc 7 couches

0.95 Observez les couches de cette image (option non tronquée)

Commande

```
docker image inspect python:3.9 | jq '.[0].RootFS.Layers'
```

```
dockeruser@vmdocker:~$ docker image inspect python:3.9 | jq '.[0].RootFS.Layers'
[
  "sha256:185e04da9d947141fd703dbf36361bdc2ff77cc27cbf500fb9f4881cb5ddbe95",
  "sha256:607ddfe5f3c3f9e9df2b45f6275ad18bc76e49fdebcf0777c1c02c66f5012956",
  "sha256:0dd5860cbc60e77cc364ce36be1a9055d4139f2123324e14f756af1af719ffb0",
  "sha256:08e14ec5b7497da231d70d47d1d80440ba7d9997d43c0796a8394923bbc98183",
  "sha256:17ba588b71727e8f8f119a800d7c17fc880acea30183ed9d824a89cf63ac031c",
  "sha256:ea18e4e51fc0ae5ad6b58321aa1d0b4cf0d98b565c4864459333399f93cad9ad",
  "sha256:2db39521cdc2287700b4b6bdca5e48da5afad7a38c79b37d04c7498994a2d925"
]
dockeruser@vmdocker:~$
```

0.96 Quelle est la dernière couche ?

Commande

```
docker image inspect python:3.9 | jq '.[0].RootFS.Layers[-1]'
```

la dernière couche est "sha256:2db39521cdc2287700b4b6bdca5e48da5afad7a38c79b37d04c7498994a2d925"

0.97 Téléchargez l'image python:3.14.0a1

Commande

```
docker pull python:3.14.0a1
```

```
dockeruser@vmdocker:~$ docker pull python:3.14.0a1
3.14.0a1: Pulling from library/python
b2b31b28ee3c: Pulling fs layer
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
```

0.98 Combien de couches ont été téléchargées ? Expliquez.

Commande

```
docker image inspect python:3.14.0a1 | jq '.[0].RootFS.Layers | length'
```

Les couches communes avec `python:3.9` ne sont pas re-téléchargées grâce au système de cache de Docker.

0.99 Afficher les couches des deux images avec jq et observer les couches communes

Commande

```
docker image inspect python:3.9 | jq '.[0].RootFS.Layers'
docker image inspect python:3.14.0a1 | jq '.[0].RootFS.Layers'
```

0.100 Créer un conteneur `os_ubuntu` et ajouter un fichier dans `/home`

Commande

```
docker run -it -name os_ubuntu ubuntu
cd /home
echo "contenu" > fichier.txt
Ctrl-p Ctrl-q
```

0.101 Exporter le système de fichiers dans une archive tar

Commande

```
docker export os_ubuntu > ubuntu_fs.tar
```

0.102 Créer une image `image_ubuntu_with_file` à partir de l'archive

Commande

```
cat ubuntu_fs.tar | docker import - image_ubuntu_with_file
```

cat ubuntu_{fs}.tar|dockerimport - image_ubuntu_with_file

0.103 Comment est-elle taggée ? Observer ses couches

Commande

```
docker images
docker history image_ubuntu_with_file
```

0.104 Créer un conteneur à partir de cette image et vérifier le fichier

Commande

```
docker run -it image_ubuntu_with_file bash
ls /home
cat /home/fichier.txt
```

0.105 Commiter le conteneur os_ubuntu en une image image2 et consulter ses couches

Commande

```
docker commit os_ubuntu image2
docker history image2
```

```
dockeruser@vmdocker:/home$ docker commit os_ubuntu image2
docker history image2
sha256:e419175ef4f7d4b16166eb65a8476361dcc0dcd1609fea65c8285ae20882171c
IMAGE          CREATED          CREATED BY          SIZE      COMMENT
e419175ef4f7    1 second ago    /bin/bash           147B
6d79abd4c962    2 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]   0B
<missing>       2 weeks ago     /bin/sh -c #(nop)  ADD file:dafefa97de6dc66a6... 78.1MB
<missing>       2 weeks ago     /bin/sh -c #(nop)  LABEL org.opencontainers... 0B
<missing>       2 weeks ago     /bin/sh -c #(nop)  LABEL org.opencontainers... 0B
<missing>       2 weeks ago     /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH    0B
<missing>       2 weeks ago     /bin/sh -c #(nop)  ARG RELEASE                0B
dockeruser@vmdocker:/home$
```

0.106 Créer un conteneur à partir de l'image image2

Commande

```
docker run -it image2 bash
```

0.107 Nettoyage final : supprimer les images créées

Commande

```
docker image rm image2 image_ubuntu_with_file
```

0.108 Partie 7 Quelques informations générales

0.108.1 Consulter la consommation des conteneurs en exécution

Commande

```
docker stats
```

Cette commande affiche en temps réel la consommation des conteneurs actifs : utilisation CPU, mémoire, réseau et disque. Chaque ligne correspond à un conteneur en cours d'exécution. C'est équivalent d'un `top` pour Docker.

0.108.2 Voir la consommation disque des objets Docker

Commande simple

```
docker system df
```

Cette commande donne un aperçu global de l'espace disque utilisé par les images, conteneurs, volumes et caches. Elle permet d'identifier les objets Docker qui occupent le plus d'espace.

0.108.3 Version détaillée de la consommation disque

Commande

```
docker system df -v
```

La version avec l'option `-v` (verbose) affiche des informations détaillées pour chaque image, conteneur et volume : taille exacte, nombre d'objets, et leur état (utilisé ou non). C'est utile pour faire le ménage intelligemment.