

Julien Jamme M1 Ilsen

TP2 uce principes et outils pour le devops

Le fichier d'origine est le markdown si le pdf a des problèmes suite à sa génération, mais contient quelques capture d'écran.

Partie 2 – Installation et démarrage de Docker

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

```
sudo docker version
```

Comment voir les composants du daemon Docker qui tournent ?

```
sudo docker info
```

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

```
systemctl list-units --type=service | grep docker systemctl
status docker.service
```

Il n'y a que docker.service d'actif, et c'est l'utilisateur root qui l'a démarré.

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

```
sudo systemctl stop docker.service sudo
systemctl stop docker.socket sudo
systemctl status docker.service
sudo systemctl status docker.socket
```

Le status est inactive (dead) pour les deux

Que faire pour désactiver/réactiver docker ?

Pour activer : `bash sudo systemctl start docker.service`

Pour désactiver : `bash sudo systemctl stop docker.service`

Le docker.socket se lancera automatiquement si on start docker.service mais pas pour la désactivation.

- Vous constatez que les commandes docker s'utilisent en mode sudo. Pour permettre à l'utilisateur dockeruser d'exécuter les commandes docker sans passer par sudo, effectuez les étapes suivantes :
 - créez un groupe nommé docker, passez en root dans son répertoire home, ajoutez (append) l'utilisateur dockeruser au groupe docker, puis redémarrer la VM,
 - réessayez d'afficher la version sans passer en sudo.

```
grep docker
/etc/group sudo
usermod -aG docker julien newgrp
docker docker version
```

Partie 3 – Premières manipulations de conteneurs et d'images

Quel est le répertoire dans lequel Docker stocke ses objets.

Le répertoire dans lequel docker stocke ses objets est : `/var/lib/docker`

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

Il existe plusieurs catégories d'objets que docker peut stocker

```
- Les images Docker : qui sont des modèles en lecture seule utilisés pour créer des conteneurs. Elles contiennent tout le nécessaire
- Les conteneurs Docker : qui sont des instances exécutables d'image Docker. Un conteneur utilise une image comme base et exécute
```

une- Volumes Docker : Les volumes sont des espaces de stockage persistant utilisés par les conteneurs pour stocker des données.

- Réseaux Docker : permet au coteneurs de communiquer entre eux et avec le monde extérieur

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

```
sudo ls /var/lib/docker/containers/ | wc -l
```

Pour le moment il y a 0 conteneur.

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

```
docker search hello-world
```

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.

```
docker pull hello-world docker  
run -it hello-world
```

```
julien@LAPTOP-3L50EL9F:/mnt/c/Users/julie$ docker pull hello-world  
Using default tag: latest  
latest: Pulling from library/hello-world  
c1ec31eb5944: Pull complete  
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fcb966  
Status: Downloaded newer image for hello-world:latest  
docker.io/library/hello-world:latest  
julien@LAPTOP-3L50EL9F:/mnt/c/Users/julie$ docker run -it 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/
```

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

```
ls /var/lib/docker/containers/ | wc -l
```

Il y a maintenant 1 conteneur.

Vérifiez aussi le contenu du répertoire des images

- `sudo ls -l /var/lib/docker/image/overlay2/`

Le dossier overlay2 est le système de fichier que docker utilise pour stocker les images et leurs couches. c'est un drivers de stockage.

Quel est le sha256 de l'image ?

```
sudo docker images --no-trunc
```

Il y a la colonne image ID avec le sha256 sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a

Utilisez la commande docker (sans option) qui permet de lister les conteneurs en exécution. Combien de conteneurs qui s'exécutent ?

```
docker ps
```

Il y a 0 conteneur qui s'exécutent car hello-world est conçu pour se lancer et s'arrêter automatiquement.

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

```
docker ps -a
```

Son alias est cranky_lederberg

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

```
docker images
```

d2c94e258dcb

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

A chaque exécution de docker run hello-world, Docker crée un nouveau conteneur à partir de l'image. Le conteneur a un nouvel ID et un nouveau nom aléatoire.

L'id de l'image reste inchangé si on ne met pas à jour l'image.

Combien de conteneurs s'exécutent et combien sont stockés localement.

```
docker ps -q | wc -l
```

Il y a 0 conteneurs lancé actuellement

```
docker ps -a -q | wc -l
```

Il y a 3 conteneurs qui sont stockés localement (en cours d'exécution et arrêtés)

Essayez de supprimer l'image et expliquez. Ne la supprimez pas finalement.

```
docker rmi hello-world
```

Il y a une erreur car docker ne permet de supprimer une image qui est liée à un conteneur en cours d'exécution ou arrêté. Réponse du terminal : Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container 94538e17d99b is using its referenced image d2c94e258dcb

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

```
docker ps -a
```

On voit que tous les conteneurs ont un status Exited (0) x minutes ago :

Exited(0) pour dire que le conteneur s'est terminé avec succès il y a x minutes ago. Dernier

conteneur lancé : Exited (0) 9 minutes ago

Quel est le nom de ces conteneurs ?

Les différents noms sont eloquent_mccarthy puis quizzical_leakey et cranky_lederberg

```
julien@LAPTOP-3L50EL9F:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b6a224ed8dde	hello-world	"/hello"	13 minutes ago	Exited (0) 13 minutes ago		eloquent_mccarthy
425e206596a9	hello-world	"/hello"	21 minutes ago	Exited (0) 21 minutes ago		quizzical_leakey
94538e17d99b	hello-world	"/hello"	36 minutes ago	Exited (0) 36 minutes ago		cranky_lederberg

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

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

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

```
docker run -it hello-world
docker ps -a
docker rm
elegant_engelbart
```

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

```
docker run --name hello sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
```

Ça exécute le conteneur en utilisant son identifiant SHA256 puis on définit nous même le nom du conteneur au lieu de lui donner un nom aléatoire géré par docker

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

```
docker rm hello
docker rm
eloquent_mccarthy
quizzical_leakey
cranky_lederberg
```

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

```
docker info
```

Il y a désormais 0 conteneur et 0 image dans le système de docker.

Quelle commande (et options) docker pouvez-vous utiliser pour n'afficher que les IDs des conteneurs ?

```
docker ps -a --format '{{.ID}}'
```

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

```
docker stop $(docker ps -a --format '{{.ID}}') && docker rm $(docker ps -a --format '{{.ID}}')
```

Supprimez maintenant l'image et constatez.

```
docker rmi d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
```

Comment tout les conteneurs ont été supprimé avant. Docker ne s'oppose pas qu'on supprime l'image car elle n'est plus lié à aucun conteneur.

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

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

L'image n'existe plus dans n'autre docker local, Il va tenter de télécharger depuis Docker HUB Une fois l'image téléchargée, Docker crée un conteneur à partir de cette image et exécute le script qui produit le message de bienvenue

```
julien@LAPTOP-3L50EL9F:~$ docker run --name hello1 hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Quel est l'identifiant de l'image ? Constatez.

L'id unique de l'image reste la même que les précédentes fois car l'image n'a pas changé dans le DOCKER HUB id = d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a

Créez un conteneur, que vous nommerez hello1, mais sans le démarrer, puis observez son statut.

```
docker create --name hello1 hello-world
```

Le status du conteneur est en Created comme on ne l'a pas démarré

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

```
docker start hello1 docker
ps -a
```

On voit que le status est passé à "Exited (0) 2 seconds ago". Car on a démarré le conteneur il a perdu son status de created.

Utilisez maintenant l'option qui permettra d'attacher l'entrée et la sortie standard pour démarrer ce conteneur hello2.

```
docker run -it --name hello2 hello-world
```

Utilisez la même option pour démarrer le conteneur hello1.

```
docker start -ai hello1
```

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.

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

Le hello3 a bien été exécuté en arrière-plan et ne génère aucune sortie dans le terminal car on l'a démarré avec l'option détaché (-d)

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
```

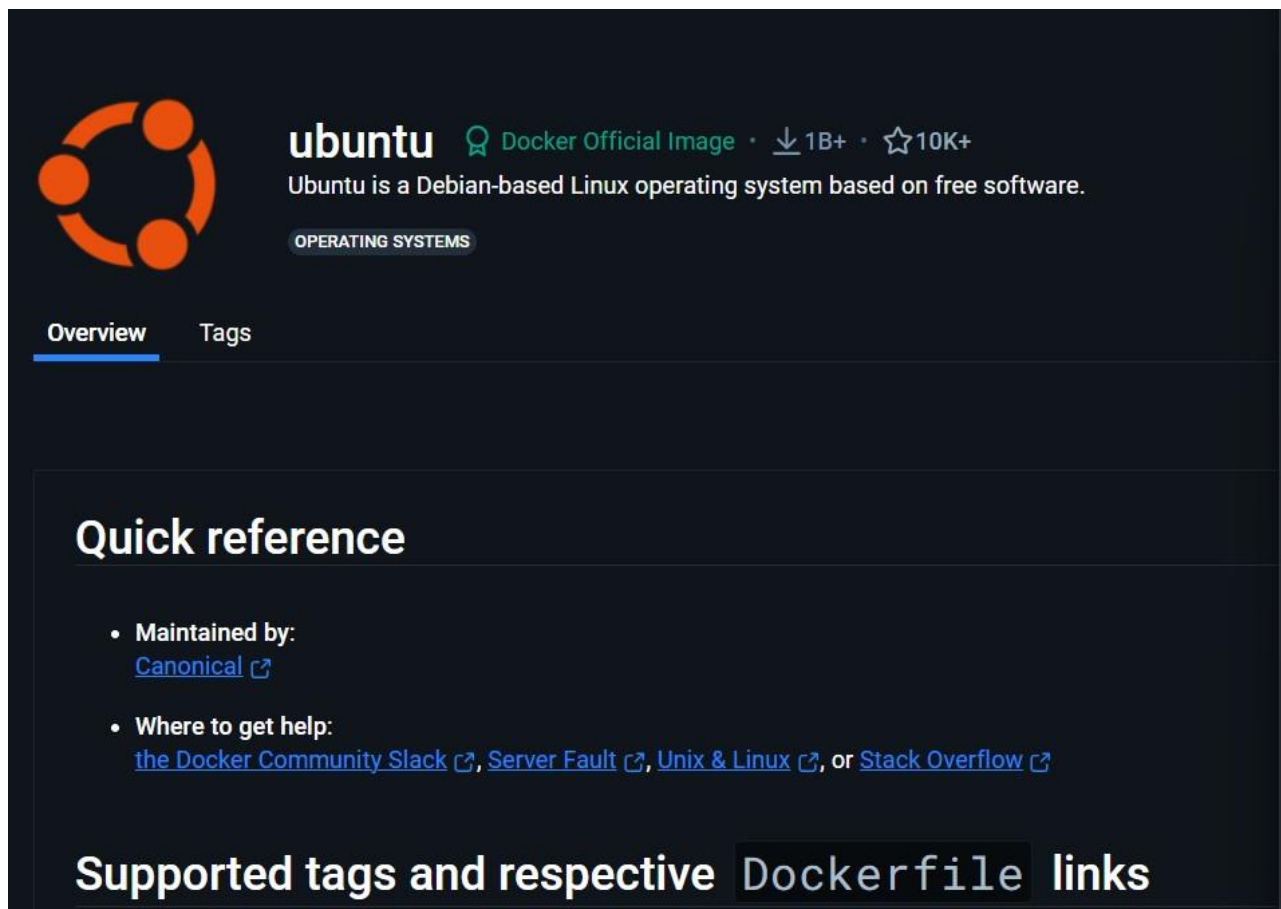
On constate qu'il n'y a aucune trace du conteneur hello4. Cela est dû au --rm qui supprime le conteneur une fois qu'il a fini de s'exécuter

Partie 4 : DockerHub

Combien de différents types d'images trouve-t-on sur ce registre public, comment se fait la classification ?

On y retrouve environ 10 000, il y a divers type de classification on y retrouve notamment Les docker official image, verified publisher, Sponsored OSS, Data Science, API management et pour chaque image on constate leur nombre de stars, de téléchargement et de pulls par les utilisateurs.

Retrouvez l'image officielle de l'OS Ubuntu.



ubuntu Docker Official Image · 1B+ · 10K+

Ubuntu is a Debian-based Linux operating system based on free software.

OPERATING SYSTEMS

Overview Tags

Quick reference

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

Supported tags and respective Dockerfile links

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

Les versions sont repérées par des tags correspondant aux versions d'Ubuntu (20.04, 22.04, 24.04). Les tags incluent des nom de code (focal,jammy), des dates de compilation, et des mentions comme latest pour la version la plus récente ou rolling pour les versions en dev continu

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 d'ubuntu sur docker est la 24.10 avec le tag oracular car elle est marqué en rolling ce qui veut dire qu'elle reçoit des mises à jours en continu pour les versions de dev

La version noble latest (24.04) quand à elle est la dernière version stable d'ubuntu(Long Term Support). Elle ne reçoit plus de mises à jours en continue elle possèdent toujours un support étendu mais moins de mises à jours que la version rolling.

Quelles sont les vulnérabilités de la version la plus récente ? Et celles de la version latest ?

La version la plus récente n'a pas de vulnérabilité (rolling) La version noble (latest) possèdent 1 vulnérabilité medium et 4 petite vulnérabilité

COMPRESSED SIZE28.37 MB

LAST PUSHED25 days ago by [dsjjacky](#)

TYPEImage

VULNERABILITIES00140

MANIFEST DIGESTsha256:5d070ad5...

Image hierarchy

ALLubuntu:noble-20241011

Layers (6)

0	ARG RELEASE	0 B	✓
1	ARG LAUNCHPAD_BUILD_ARCH	0 B	✓
2	LABEL org.opencontainers.image.ref.name=ubuntu	0 B	✓
3	LABEL org.opencontainers.image.version=24.04	0 B	✓
4	ADD file:34dc4f3ab7a694ecde47ff7a610be18591834c45f1d7251813...	29.75 MB	⚠
5	CMD ["/bin/bash"]	0 B	✓

Images (1)Vulnerabilities (5)Packages (130)

Give feedback

Package or CVE name

FixableShow exceptedReset filters

Package	Vulnerabilities
ubuntu/libcrypt20 1.10.3-2build1	00100
CVE-2024-2236	N/A

A timing-based side-channel flaw was found in libcrypt's RSA implementation. This issue may allow a remote attacker to initiate a Bleichenbacher-style attack, which can lead to the decryption of RSA ciphertexts.

EPSS Score0.00043 (0.101)

Affected range>=0

Fix versionNot yet available

Publish date2024-03-07

View package locations

La vulnérabilité medium ici est causé par Une faille de canal auxiliaire basée sur le timing dans l'implémentation RSA de libcrypt.

Observez comment ces vulnérabilités sont classées.

Les vulnérabilités dans l'image Docker ubuntu sont classées par type et selon leur impact potentiel sur le système. Chaque vulnérabilité reçoit un identifiant CVE, Criticité et impact potentiel, Plage affectée et correctif, Packages ou bibliothèques concernés

Ces classifications permettent de prioriser les actions de sécurité en fonction de la criticité, de l'exploitabilité, et de la disponibilité de correctifs.

Comparez la version latest avec la version noble : quel pourrait être l'intérêt de cette situation ? le tag latest permet aux utilisateurs de rester sur un environnement LTS sécurisé et stable tandis que le tag noble peut offrir des versions plus dynamiques et des possibilités de mises à jour rapides, tout en conservant un niveau de stabilité élevé.

En cliquant sur le tag de chacune des 2, consultez les couches de ces images, et observez à partir de quand elles diffèrent. elles commencent à différer après des mises à jour mineures appliquées dans les couches. On constate que dans les layers la version noble est à 29.75 MB avec des vulnérabilités tandis que la rolling à 30.6 sans vulnérabilités La version noble utilise la version 24.04 et la rolling la version 24.10

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

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

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

```
docker pull ubuntu
```

c'est la latest qui est téléchargée

Téléchargez maintenant l'image ubuntu la plus récente.

```
docker pull ubuntu:rolling
```

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

```
docker images --filter "reference=ubuntu*"
```

la commande qui est lancé est CMD ["/bin/bash"] Le conteneur démarre un shell Bash (/bin/bash) et permet d'interagir avec le système Ubuntu de manière interactive.

```
julien@LAPTOP-3L50EL9F:~$ docker images --filter "reference=ubuntu*"
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    59ab366372d5   4 weeks ago    78.1MB
ubuntu        oracular  bf21d82156ce   4 weeks ago    80.1MB
```

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

```
docker rmi ubuntu:rolling
```

Partie 5 – Interagir avec un conteneur

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

```
docker run --name ubuntu Ubuntu docker
ps -a    docker start -i Ubuntu docker
rm Ubuntu
```

Son status est Exited, car il se termine immédiatement après son lancement si on ne définit pas de commande ou d'argument supplémentaire

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

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

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

```
docker top os_ubuntu
```

Le processus /bin/bash est le processus principal du conteneur, avec un PID 1.

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

hostname: f06f8e16d4c2

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) ?

Je constate que l'on retrouve l'ID que l'on avait dans hostname car c'est également le CONTAINER ID f06f8e16d4c2

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

```
cd /home
```

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

Je constate que le conteneur c'est automatiquement arrêté après avoir quitter le conteneur avec la commande unix Exited(0)

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

```
docker start -i os_ubuntu
```

Pour l'image Ubuntu, le répertoire de travail par défaut est défini comme / (la racine), de ce fait on sera là à chaque lancement du conteneur

Dans le second terminal, utilisez une commande qui permet d'inspecter le conteneur. Constatez qu'il est en cours d'exécution. Retrouvez son Pid.

```
docker ps -a
```

On le trouve facilement car il a le status UP qui indique qu'il est en cours d'exécution

```
docker inspect --format '{{ .State.Pid }}' f06f8e16d4c2
```

Son PID est 2283

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

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

On retombe sur 2283

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

```
ps -p 2283 -f
```

Il est directement lié à Docker. C'est le processus /bin/bash qui agit en tant que PID 1 à l'intérieur du conteneur.

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

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

Le conteneur est resté ouvert son statut est UP

Attachez le terminal au conteneur. Quel est le répertoire courant ? Ajoutez-y un fichier avec un contenu quelconque.

```
docker attach os_ubuntu echo "Ceci est un
fichier test." > test_file.txt
```

Le répertoire courant est /home car le conteneur ne s'est pas arrêté.

Quittez à nouveau ce conteneur sans l'arrêter, puis utilisez une commande docker pour voir les différences dans le système de fichiers du conteneur.

```
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker diff os_ubuntu
C /home A /home/test_file.txt
C /root
A /root/.bash_history
```

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

```
docker exec os_ubuntu hostname
```

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.

```
docker exec -it os_ubuntu bash

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker exec -it os_ubuntu bash root@f06f8e16d4c2:/# ps aux | grep bash root
1 0.0 0.1 4588 3960 pts/0 Ss+ 22:47 0:00 /bin/bash root 18 0.2 0.0 4588 3872 pts/1 Ss 23:21
0:00 bash root 27 33.3 0.0 3528 1624 pts/1 S+ 23:21 0:00 grep --color=auto bash
```

Il y a trois processus en cours

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

```
docker top os_ubuntu

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker top os_ubuntu
UID PID PPID C STIME TTY TIME CMD
root 2283 2263 0 Dec03 pts/0 00:00:00 /bin/bash
root 3012 2263 0 Dec03 pts/1 00:00:00 bash
```

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

```
docker attach os_ubuntu
exit
docker ps -a
```

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

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ 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 Dec 4 09:15 dev drwxr-xr-x 1 root root 4096 Dec 4 09:15 etc drwxr-xr-x
3 root root 4096 Oct 11 02:09 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 Oct 11 02:03 media drwxr-xr-x 2 root root 4096 Oct 11 02:03 mnt drwxr-xr-x 2 root root 4096 Oct 11 02:03 opt dr-xr-xr-x 225 root root 0 Dec 4 09:15 proc drwx----- 2 root root 4096 Oct 11 02:09 root drwxr-xr-x 4 root root 4096 Oct 11 02:09 run lrwxrwxrwx 1 root root 8 Apr 22 2024 sbin -> usr/sbin drwxr-xr-x 2 root root 4096 Oct 11 02:03 srv dr-xr-xr-x 11 root root 0 Dec 4 09:15 sys drwxrwxrwt 2 root root 4096 Oct 11 02:09 tmp drwxr-xr-x 12 root root 4096 Oct 11 02:03 usr drwxr-xr-x 11 root root 4096 Oct 11 02:09 var
```

Redémarrez ce conteneur pour obtenir le même affichage.

```
docker start -a ll
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker start -a ll 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 Dec 4 09:16 dev drwxr-xr-x 1 root root 4096 Dec 4 09:15 etc drwxr-xr-x 3 root root 4096 Oct 11 02:09 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 Oct 11 02:03 media drwxr-xr-x 2 root root 4096 Oct 11 02:03 mnt drwxr-xr-x 2 root root 4096 Oct 11 02:03 opt dr-xr-xr-x 229 root root 0 Dec 4 09:16 proc drwx----- 2 root root 4096 Oct 11 02:09 root drwxr-xr-x 4 root root 4096 Oct 11 02:09 run lrwxrwxrwx 1 root root 8 Apr 22 2024 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 Oct 11 02:03 srv dr-xr-xr-x 11 root root 0 Dec 4 09:15 sys drwxrwxrwt 2 root root 4096 Oct 11 02:09 tmp drwxr-xr-x 12 root root 4096 Oct 11 02:03 usr drwxr-xr-x 11 root root 4096 Oct 11 02:09 var
```

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

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

```
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker run --rm --name ps ubuntu ps aux USER PID %CPU %MEM
VSZ RSS TTY STAT START TIME COMMAND root 1 17.8 0.0 7888 3604 ? Rs 09:18
```

```
0:00 ps aux
```

Le PID est 1 donc le processus principal est ps aux du conteneur Le processus est en étant Running (R) et en processus Leader de session (s)

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

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

```
docker ps -a
```

```
docker start -a salut
```

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

```
docker run --name infinie ubuntu sh -c "while true; do sleep 3600; done"
docker stop infinie docker rm infinie
```

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

```
docker start -i os_ubuntu while true;
do echo salut; sleep 3; done
```

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. Interrompez le processus qui effectue le salut, lancez une commande basique (par exemple ls) et constatez dans l'autre terminal.

```
docker attach os_ubuntu
ls
```

On constate que les deux terminal sont lié si j'effectue une commande dans une l'autre elle sera affiché dans la seconde et inversement

Quittez ce conteneur en utilisant la commande bash exit avec un code retour non nul. Constatez le statut de ce conteneur.

```
exit 42
```

```
docker ps -a
```

Son status de retour est 42 Exited (42) 13 seconds ago

Utilisez une commande docker pour voir le log du conteneur os_ubuntu.

```
docker logs os_ubuntu
```

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

```
cd /var/lib/docker/containers/f06f8e16d4c2000e219a90a91e92161ba0c7177ab3f4fac5a214150cd8aace32
cat f06f8e16d4c2000e219a90a91e92161ba0c7177ab3f4fac5a214150cd8aace32-json.log
```

jq ne marche pas

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).

```
docker stop $(docker ps -q) && docker rm $(docker ps -aq)
```

Partie 6 – Inspection et manipulation d’images

Téléchargez l'image python:3.9. Combien de couches ont été téléchargées ?

```
docker pull python:3.9 docker
history python:3.9
```

Il y a 11 couche qui ont été téléchargées. Observez les couches de cette image (il existe aussi une option non tronquée)

```
docker history --no-trunc python:3.9
```

Quelle est la dernière couche ?

La dernière couche est

<missing>	5 days ago	RUN /bin/sh -c set -eux; for src in idle3 p...	36B	buildkit.dockerfile.v0
-----------	------------	--	-----	------------------------

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

```
docker pull python:3.14.0a1 docker
history --no-trunc python:3.14.0a1
```

Les différentes couches correspondent à des instructions dans le Dockerfile qui a été utilisé pour construire l'image. avec diffénret type d'instruction qu'on peut constater.

"RUN" qui exécute des commandes à l'intérieur de l'image. "CMD" qui définit la commande à exécuter quand le conteneur démarre. "ENV" qui définit des variables d'environnement. "ADD" qui ajoute des fichiers dans l'image.

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


```
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ jq '.[0].RootFS.Layers' python_3.9_inspect.json
[
  "sha256:301c1bb42cc0bc6618fcaf036e8711f2aad66f76697f541e2014a69e1f456aa4",
  "sha256:0e82d78b3ealb1db9fa3elf18d6745e0c2380c25f2c7cec420257084e9cc44fe",
  "sha256:c81d4fdb67fcd8ffbf9f93f440264d36a2d9e7c4e79b9ae5152c5ed2e3fd36",
  "sha256:0aeceb7c293df4fb677b2771713e9c6abeabf8b7f06bfb071310e6cc1a3aa084",
  "sha256:8f9a13bfb118975875edd547c5c0762eed442b686d86fa46832bf04337f75316",
  "sha256:24f0c2413cd7a5e1e06bbb497657405c0d81b86142567b8425dea83b3ald635d",
  "sha256:fe5bbd4f8a4224acb21f695f361216e88e2db8bc531064ae2d482635d5f357ae"
]
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ jq '.[0].RootFS.Layers'
python_3.14.0a1_inspect.json [
  "sha256:24b5ce0f1e07d37a35460f50d058afcf738619e431013d2e1727609bdf2d7fc",
  "sha256:b6ca42156b9f492afa27c366f20e4e864cef8dd8d0e0a100497764b05b39e6fc",
  "sha256:00547dd240c419fa2e1b33e66aba302e8dfa4bfe6401a972d94a03b1355cbc6c",
  "sha256:96d99c63b722657062d3f33cc230e33b191ea9855c050f44871e173709597e35",
  "sha256:9744b636d758d56bfeceb5e712ddfecbe662951562155cc3f93af8cfd538422c",
  "sha256:4068925b787de0e570d68e70bc04de2380276817a36a343c08aad3435c221113",
  "sha256:da64f9c6a005a83af29c8389262e6de4bb9b1b5ae96ceb6c567e01e7c7525cde"
]

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker inspect python:3.9 > python_3.9_inspect.json
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker inspect python:3.9
[
  {
    "Id":
      "sha256:f327fe247a0655427e3cfd615405bfab360e70acf34bf0385e7f1b0a07e4f8d8",
    "RepoTags": [
      "python:3.9"
    ],
    "RepoDigests": [
      "python@sha256:dd8b65c39a729f946398d2e03a3e6defc8c0cfec409b9f536200634ad6408b54"
    ],
    "Parent": "",
    "Comment": "buildkit.dockerfile.v0",
    "Created": "2024-12-04T04:30:01Z",
    "DockerVersion": "",
    "Author": "",
    "Config": {
      "Hostname": "",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "Tty": false,
      "OpenStdin": false,
      "StdinOnce": false,
      "Env": [
        "PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "LANG=C.UTF-8",
        "GPG_KEY=E3FF2839C048B25C084DEBE9B26995E310250568",
        "PYTHON_VERSION=3.9.21",
        "PYTHON_SHA256=3126f59592c9b0d798584755f2bf7b081falca35ce7a6fea980108d752a05bb1"
      ],
      "Cmd": [
        "python3"
      ],
      "Image": "",
      "Volumes": null,
      "WorkingDir": "",
      "Entrypoint": null,
      "OnBuild": null,
      "Labels": null
    },
    "Architecture": "amd64",
```

```
"Os": "linux",  
"Size": 998561943,  
"GraphDriver": {  
  "Data": {
```

"LowerDir":

"/var/lib/docker/overlay2/48c676e1fa0f594b8dd3ec3eb5f74d48e7bcc4510895f2faf0807f7b4388c07/diff:/var/lib/

```

        "MergedDir": "/var/lib/docker/overlay2/c230dalb8a1b909d123785dd5994aa79f792b48bfd482cfee5ec3569aa10a134/merged",
        "UpperDir": "/var/lib/docker/overlay2/c230dalb8a1b909d123785dd5994aa79f792b48bfd482cfee5ec3569aa10a134/diff",
        "WorkDir": "/var/lib/docker/overlay2/c230dalb8a1b909d123785dd5994aa79f792b48bfd482cfee5ec3569aa10a134/work"
    },
    "Name": "overlay2"
},
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:301c1bb42cc0bc6618fcaf036e8711f2aad66f76697f541e2014a69e1f456aa4",
        "sha256:0e82d78b3ealb1db9fa3e1f18d6745e0c2380c25f2c7cec420257084e9cc44fe",
        "sha256:c81d4fdb67fcfd8ffbf93f440264d36a2d9e7c4e79b9ae5152c5ed2e3fd36",
        "sha256:0aeeeb7c293df4fb677b2771713e9c6abeabf8b7f06bfb071310e6cc1a3aa084",
        "sha256:8f9a13bfb118975875edd547c5c0762eed442b686d86fa46832bf04337f75316",
        "sha256:24f0c2413cd7a5e1e06bbb497657405c0d81b86142567b8425dea83b3a1d635d",
        "sha256:fe5bbd4f8a4224acb21f695f361216e88e2db8bc531064ae2d482635d5f357ae"
    ]
},
"Metadata": {
    "LastTagTime":
"0001-01-01T00:00:00Z"
}
]

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker inspect python:3.14.0a1 > python_3.14.0a1_inspect.json
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker inspect python:3.14.0a1
[
    {
        "Id":
"sha256:80ad471000e75ca8cbd36355ed7e632f9fe83a0c3fa859a704a5af204bcd8853",
        "RepoTags": [
            "python:3.14.0a1"
        ],
        "RepoDigests": [
            "python@sha256:3a852c145c357b55d0fdbf624207bb81c5a546f17f622e5c208cc0b36edaaa0e"
        ],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2024-10-18T23:23:40Z",
        "DockerVersion": "",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "PYTHON_VERSION=3.14.0a1",
                "PYTHON_SHA256=3e464b0ccb7535e2db34262fd19a0a393d0e62be0f43b1513ed98379b054ead4"
            ],
            "Cmd": [
                "python3"
            ],
            "ArgsEscaped": true,
            "Image": "",
            "Volumes": null,

```

```

        "WorkingDir": "/",
        "Entrypoint": null,
        "OnBuild": null,
        "Labels": null
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 1019807870,
    "GraphDriver": {
        "Data": {
            "LowerDir":
"/var/lib/docker/overlay2/978760a5ca01201daf02c2d7d468959a2438f7e3e2b02bbdaafaf7680bc42dd4/diff:/var/lib/

            "MergedDir": "/var/lib/docker/overlay2/3020c9294225699ae4acb9ac8d159870aef1626cf43848ecb78a28d4005866e5/merged",
            "UpperDir": "/var/lib/docker/overlay2/3020c9294225699ae4acb9ac8d159870aef1626cf43848ecb78a28d4005866e5/diff",
            "WorkDir": "/var/lib/docker/overlay2/3020c9294225699ae4acb9ac8d159870aef1626cf43848ecb78a28d4005866e5/work"

        },
        "Name": "overlay2"
    },
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:24b5ce0f1e07d37a35460f50d058afcf738619e431013d2e1727609bdf2d7fc",
            "sha256:b6ca42156b9f492afa27c366f20e4e864cef8dd8d0e0a100497764b05b39e6fc",
            "sha256:00547dd240c419fa2e1b33e66aba302e8dfa4bfe6401a972d94a03b1355cbc6c",
            "sha256:96d99c63b722657062d3f33cc230e33b191ea9855c050f44871e173709597e35",
            "sha256:9744b636d758d56bfeceb5e712ddfecbe662951562155cc3f93af8cfd538422c",
            "sha256:4068925b787de0e570d68e70bc04de2380276817a36a343c08aad3435c221113",

"sha256:da64f9c6a005a83af29c8389262e6de4bb9b1b5ae96ceb6c567e01e7c7525cde"

        ]
    },
    "Metadata": {
        "LastTagTime":

"0001-01-01T00:00:00Z"

    }
}
]

```

Les deux images Python (python:3.9 et python:3.14.0a1) on aucune couches communes.

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`.

```

docker run -it --name os_ubuntu ubuntu /bin/bash echo
"Ceci est un fichier de test" > /home/example.txt
docker export os_ubuntu > os_ubuntu.tar

```

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

```

docker import os_ubuntu.tar image_ubuntu_with_file
docker images docker history
image_ubuntu_with_file

julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker history image_ubuntu_with_file
IMAGE          CREATED          CREATED BY          SIZE              COMMENT           da5099556649    12
seconds ago    78.1MB          Imported from -

```

```

docker inspect image_ubuntu_with_file julien@LAPTOP-
3L5OEL9F:/mnt/c/Users/julie$ docker inspect image_ubuntu_with_file [
  {
    "Id":
"sha256:da50995566495ce0ee58809c7bcdaf2c3a6531a262853bd0dbd0510cab72ab44",

    "RepoTags": [
      "image_ubuntu_with_file:latest"
    ],
    "RepoDigests": [],
    "Parent": "",
    "Comment": "Imported from -",

    "Created": "2024-12-09T14:10:14.628925073Z",
    "DockerVersion": "27.3.1",
    "Author": "",
    "Config": {
      "Hostname": "",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "Tty": false,
      "OpenStdin": false,
      "StdinOnce": false,
      "Env": null,
      "Cmd": null,
      "Image": "",
      "Volumes": null,
      "WorkingDir": "",
      "Entrypoint": null,
      "OnBuild": null,
      "Labels": null
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Size": 78120848,
    "GraphDriver": {
      "Data": {
        "MergedDir": "/var/lib/docker/overlay2/c4f10c519d0133d64352418bd202b3bbc6e64a8bbb028facbe92afccc0b1e694/merged",
        "UpperDir": "/var/lib/docker/overlay2/c4f10c519d0133d64352418bd202b3bbc6e64a8bbb028facbe92afccc0b1e694/diff",
        "WorkDir": "/var/lib/docker/overlay2/c4f10c519d0133d64352418bd202b3bbc6e64a8bbb028facbe92afccc0b1e694/work"
      },
      "Name": "overlay2"
    },
    "RootFS": {
      "Type": "layers",
      "Layers": [
        "sha256:66aed64eecf7975d9c7bcd2f7740cdd3d18287da26d42eb1f92361274f742cbb"
      ]
    },
    "Metadata": {
      "LastTagTime": "2024-12-09T15:10:14.639064206+01:00"
    }
  }
]

```

L'importation du fichier tar avec la commande `docker import` génère une seule couche qui contient tout le système de fichiers extrait de l'archive.

Partie 7 – Quelques informations générales

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

```
docker stats julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
```

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

```
julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$ docker system df TYPE

TOTAL        ACTIVE      SIZE        RECLAIMABLE
Images        6           2           2.253GB     2.175GB (96%)
Containers    2           0           28B         28B (100%)
Local Volumes 0           0           0B          0B Build Cache  0

0            0B          0B julien@LAPTOP-3L5OEL9F:/mnt/c/Users/julie$
docker system df -v Images space usage:
REPOSITORY    TAG          IMAGE ID      CREATED      SIZE        SHARED SIZE  UNIQUE SIZE
CONTAINERS image_ubuntu_with_file latest       da5099556649 9 minutes ago 78.1MB       0B           78.12MB
0 python      3.9         f327fe247a06 5 days ago   999MB        0B           998.6MB     0 ubuntu
latest      b1d9df8ab815 2 weeks ago 78.1MB      0B           78.12MB      1 python
3.14.0a1    80ad471000e7 7 weeks ago 1.02GB      0B           1.02GB       0 <none>    <none>
59ab366372d5 8 weeks ago 78.1MB     0B          78.11MB     0 hello-world latest      d2c94e258dcb
19 months ago 13.3kB     0B         13.26kB     1 Containers space usage:
CONTAINER ID   IMAGE      COMMAND          LOCAL VOLUMES   SIZE      CREATED        STATUS          NAMES
0c6257a3e5e7   ubuntu    "/bin/bash"      0               28B       11 minutes ago Exited (137)    About a minute ago os_ubuntu
cd0c2ce9380f   hello-world "/hello"         0               0B        2 days ago     Exited (0) 2 days ago
intelligent_j Local Volumes space usage:
VOLUME NAME    LINKS      SIZE

Build cache usage: 0B

CACHE ID   CACHE TYPE  SIZE      CREATED   LAST USED  USAGE      SHARED
```