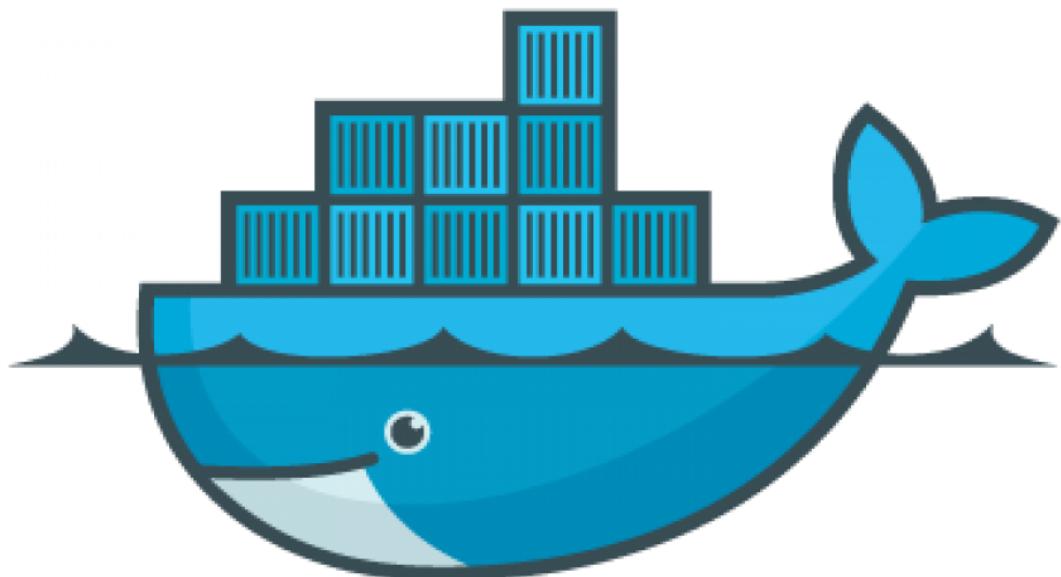


20/10/2025

Docker



docker

Aghilas Ould Brahim

Table des matières

Qu'est-ce que Docker ?	3
Qu'est-ce que l'isolation dans Docker ?	3
1. Les namespaces	3
2. Les cgroups (control groups)	3
Principes clés.....	4
Éléments principaux.....	4
Architecture de Docker	4
Les registres Docker	5
Avantages de Docker.....	5
Installation	6
Desktop	6
CLI (sous ubuntu)	6
Vérification.....	6
Premier conteneur	7
Observer les namespaces et cgroups d'un conteneur Docker.....	8
Commande docker	9
Debian sous docker.....	9
Suppression de la totalité des conteneurs.....	10
Les volumes pour persister les données.....	10
Création d'un simple site web sous nginx	13
Les différents types de volumes docker	14
Volume persistant (ou volume Docker)	14
Bind mount.....	14
tmpfs mount	15
Importance du UserID pour les volumes	15
Container Runtimes.....	17
Les réseaux docker	18
Mode bridge (par défaut).....	18
Visualiser docker0	18
Expose vs Publish.....	20
La CLI network	20
Ip d'un conteneur	20
CLI : docker network	20
Connect/disconnect.....	22
Docker0.....	22
Configurer le docker0.....	22

Création d'un nouveau bridge :.....	23
Création d'images	24
Commit (sans Dockerfile).....	24
Historique de modification d'une image.....	25
Dockerfile.....	25
Les instructions Dockerfile.....	26
Bonnes Pratiques pour Écrire un Dockerfile de Qualité.....	32

Qu'est-ce que Docker ?

Docker est une plateforme open source qui permet de créer, déployer et exécuter des applications dans des conteneurs un isolé appelé **conteneurs**.

L'isolation et la sécurité offertes permettent d'exécuter plusieurs conteneurs simultanément sur un même hôte.

Les conteneurs sont légers et contiennent tout ce dont une application a besoin pour fonctionner il n'est donc pas nécessaire d'installés des logiciels sur la machine hôte, autrement dit un **conteneur** est un environnement isolé et léger qui contient tout le nécessaire pour exécuter une application (code, dépendances, configuration...).

Qu'est-ce que l'isolation dans Docker ?

L'isolation signifie que chaque **conteneur Docker** fonctionne dans son propre environnement indépendant : il a son propre *système de fichiers*, ses processus, son réseau, et ses ressources.

Docker utilise des **mécanismes du noyau Linux** pour assurer cette isolation.

Les deux principaux sont :

1. Les namespaces

Les namespaces limitent ce qu'un conteneur peut voir. Chaque conteneur obtient son propre espace :

- **PID namespace** : isole les processus (un conteneur ne voit que ses propres processus)
- **NET namespace** : isole le réseau (chaque conteneur a ses propres interfaces, adresses IP, ports, etc.)
- **MNT namespace** : isole le système de fichiers (chaque conteneur a sa propre arborescence)
- **UTS namespace** : permet d'avoir un nom d'hôte distinct
- **IPC namespace** : isole la communication inter-processus (partage de mémoire, sémaphores...)

2. Les cgroups (control groups)

Les cgroups limitent ce qu'un conteneur peut utiliser. Ils permettent de :

- Limiter l'utilisation du **CPU**, de la **mémoire**, du **disque** et du **réseau**,
- Éviter qu'un conteneur ne consomme toutes les ressources du système.

```
docker run --memory="512m" --cpus="1" ubuntu
```

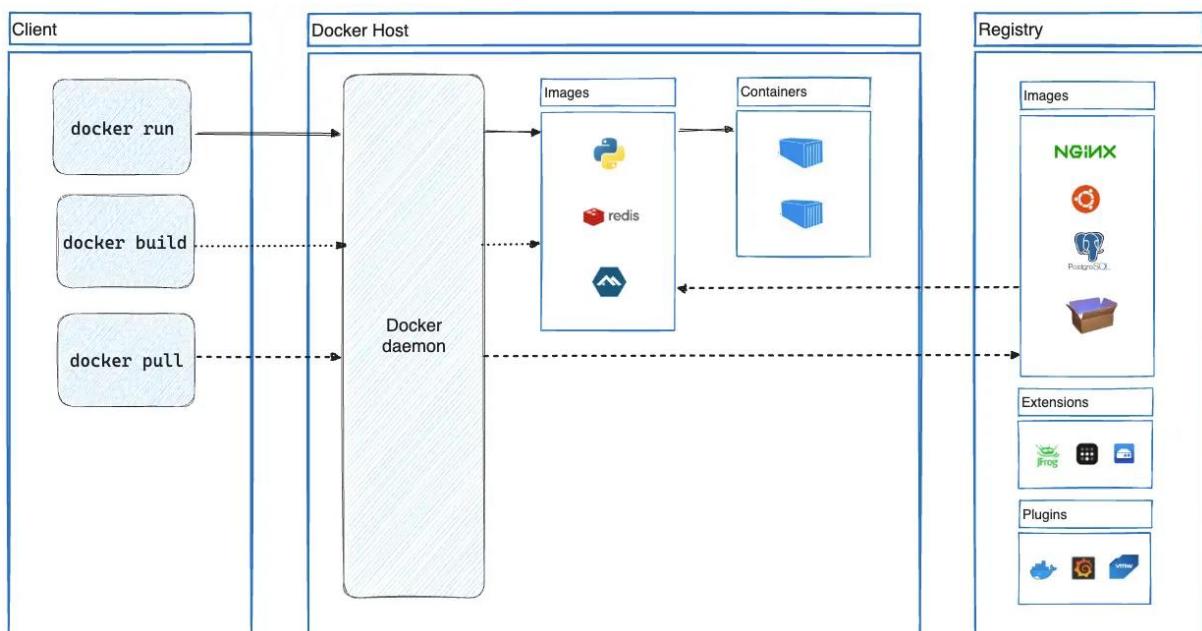
Principes clés

- **Portabilité** : un conteneur peut tourner sur n'importe quel système (ordinateur, serveur, cloud...).
- **Légereté** : les conteneurs partagent le noyau du système d'exploitation, contrairement aux machines virtuelles, ce qui les rend rapides et économies en ressources.
- **Reproductibilité** : le même conteneur fonctionne partout de la même manière (dev, test, prod).
 - **Cycle de vie** : décrit toutes les **étapes qu'un conteneur traverse**, depuis sa création jusqu'à sa suppression (création, exécution, en cours, arrêt, suppression).

Éléments principaux

- **Image** : modèle **figé** en lecture seule décrivant le contenu et le comportement du conteneur (ex os : Ubuntu, Alpine, etc.).
- **Conteneur** : instance exécutable d'une image.
- **Dockerfile** : fichier de configuration qui décrit comment construire une image.
- **Docker Daemon (dockerd)** : service qui gère les conteneurs. Écoute les requêtes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux et les volumes.
Un démon peut aussi communiquer avec d'autres démons pour gérer des services Docker distribués.
- **Docker Client (docker)** : outil en ligne de commande pour interagir avec Docker.
- **Registry (ex : Docker Hub)** : entrepôt où sont stockées et partagées les images

Architecture de Docker



- Docker repose sur une **architecture client-serveur**.
- Le **client Docker** communique avec le **démon Docker (daemon)**, qui effectue les opérations lourdes : création, exécution et distribution de conteneurs.
- Le client et le démon peuvent s'exécuter sur la même machine, ou à distance via une API REST (sur socket UNIX ou interface réseau).
- Un autre client, **Docker Compose**, permet de gérer des applications composées de plusieurs conteneurs.
- Le **client Docker (docker)** est l'outil principal d'interaction avec Docker.
- Lorsque vous exécutez une commande comme docker run, le client envoie l'instruction au démon dockerd, qui la réalise.
- Le client peut communiquer avec plusieurs démons à la fois.

Les registres Docker

Un **registre Docker** stocke des **images Docker**.

Le registre public par défaut est **Docker Hub**, mais il est possible d'héberger un registre privé.

- **docker pull** ou **docker run** : télécharge les images nécessaires.
- **docker push** : envoie vos images vers le registre configuré.

Avantages de Docker

- Facilite le **développement et le déploiement rapide** (CI/CD).
- Simplifie les **tests et la mise à l'échelle** d'applications.
- Permet de faire tourner plus d'applications sur le même matériel.
- Fonctionne de la même façon en **local, en cloud ou en environnement hybride**

Installation

Desktop

- <https://docs.docker.com/get-started/introduction/get-docker-desktop/>

CLI (sous ubuntu)

- Configurer le **Docker apt** repository

```
# Add Docker's official GPG key:  
sudo apt-get update  
sudo apt-get install ca-certificates curl  
sudo install -m 0755 -d /etc/apt/keyrings  
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc  
sudo chmod a+r /etc/apt/keyrings/docker.asc  
  
# Add the repository to Apt sources:  
echo \  
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.  
  ${. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}"}) stable" | \  
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update
```

- Installation du **package Docker**

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

Vérification

```
agh1@Exodus: ~ docker -v  
Docker version 20.5.1, build e180ab8  
agh1@Exodus: ~  
agh1@Exodus: ~$ sudo systemctl status docker  
● docker.service - Docker Application Container Engine  
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)  
  Active: active (running) since Mon 2025-10-20 00:57:31 CEST; 1min 8s ago  
TriggeredBy: ● docker.socket  
    Docs: https://docs.docker.com  
      Main PID: 4446 (dockerd)  
        Tasks: 9  
         Memory: 21.5M  
          CPU: 449ms  
          CGroup: /system.slice/docker.service  
             └─4446 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock  
  
oct. 20 00:57:30 Exodus dockerd[4446]: time="2025-10-20T00:57:30.588267331+02:00" level=info msg="Creating a containerd client" address=/run/containerd/containerd.sock timeout=1m0s  
oct. 20 00:57:30 Exodus dockerd[4446]: time="2025-10-20T00:57:30.624400210+02:00" level=info msg="[graphdriver] using prior storage driver: overlay2"  
oct. 20 00:57:30 Exodus dockerd[4446]: time="2025-10-20T00:57:30.624617171+02:00" level=info msg="Loading containers: start."  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.147661406+02:00" level=info msg="Loading containers: done."  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.356049451+02:00" level=info msg="Docker daemon" commit=f8215cc containerd-snapshotter=false storage-driver=overlay2  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.356274936+02:00" level=info msg="Initializing buildkit"  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.399972946+02:00" level=info msg="Completed buildkit initialization"  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.408720434+02:00" level=info msg="Daemon has completed initialization"  
oct. 20 00:57:31 Exodus dockerd[4446]: time="2025-10-20T00:57:31.409028576+02:00" level=info msg="API listen on /run/docker.sock"  
oct. 20 00:57:31 Exodus systemd[1]: Started Docker Application Container Engine.  
agh1@Exodus: ~  
agh1@Exodus: ~$ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
17ee7bbc9d7: Pull complete  
Digest: sha256:6dc565aa63992705211f823c303948cf83670a3903ffa3849f1488ab517f891  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.
```

Docker est installé et qu'il fonctionne correctement. La commande **docker -v** confirme l'installation, et **systemctl status docker** montre que le service Docker tourne bien. Ensuite, en exécutant **sudo docker run hello-world**, Docker a téléchargé l'image « **hello-world** » depuis **Docker Hub**, créé un conteneur à partir de cette image, et affiché le message "Hello

from Docker!”. Cela prouve que l’installation Docker est opérationnelle et prête à exécuter des conteneurs.

Premier conteneur

```
docker run -d -p 8080:80 docker/welcome-to-docker
```

docker run

Lance un **nouveau conteneur** à partir d’une image Docker.

-d

exécute le conteneur **en arrière-plan**. Tu ne verras pas les logs dans ton terminal, mais le conteneur continue de tourner.

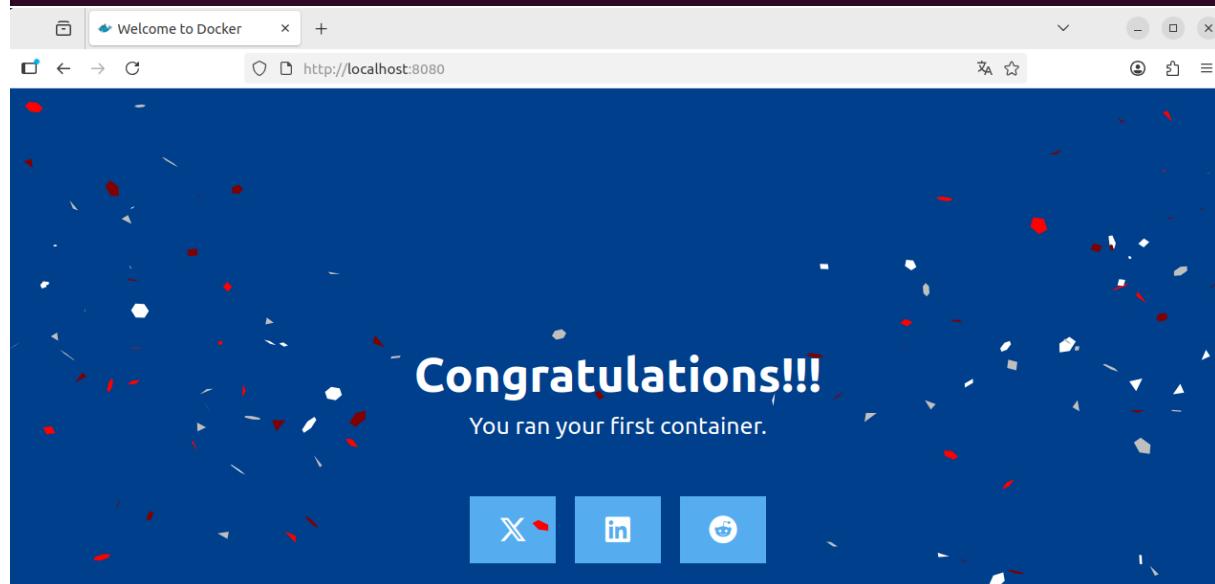
-p 8080:80

Fait un **mappage de ports** l’hôte et le conteneur : le port **8080 de ton PC** redirigé vers le port **80 du conteneur**, on peut y accéder via <http://localhost:8080>.

docker/welcome-to-docker

C’est le **nom de l’image** utilisée. Elle contient une petite application web de démonstration fournie par Docker.

```
aghia@Exodus:~$ docker run -d -p 8080:80 docker/welcome-to-docker
Unable to find image 'docker/welcome-to-docker:latest' locally
latest: Pulling from docker/welcome-to-docker
9824c27679d3: Pull complete
a5585638209e: Pull complete
fd372c3c84a2: Pull complete
958a7dd6a238: Pull complete
c1d2dc189e38: Pull complete
828fa206d77b: Pull complete
bdaad27fd04a: Pull complete
9745203f5d34: Pull complete
Digest: sha256:c4d56c24daaf009ecf8352146b43497fe78953edb4c679b841732beb97e588b0
Status: Downloaded newer image for docker/welcome-to-docker:latest
593ea12e894dc6f3c1d2189b565f236ee083c1e23c868a5d8d257b6eb2f06b95
aghia@Exodus:~$
aghia@Exodus:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
hello-world         latest   1b44b5a3e06a  2 months ago  10.1kB
docker/welcome-to-docker  latest   6caf772f5178  2 months ago  14.1MB
aghia@Exodus:~$
aghia@Exodus:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS              PORTS                 NAMES
593ea12e894d        docker/welcome-to-docker   "/docker-entrypoint..."   26 seconds ago    Up 25 seconds    0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   keen_hypatia
aghia@Exodus:~$
```



Observer les namespaces et cgroups d'un conteneur Docker

```
aight@Exodus:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
627ddb227b05 docker/welcome-to-docker "/docker-entrypoint..." 18 minutes ago Up 18 minutes 0.0.0.0:8080->80/tcp, ::1:8080->80/tcp distracted_murdock

aight@Exodus:~$ 
aight@Exodus:~$ 
aight@Exodus:~$ sudo ls -l /proc/4186/ns
total 0
lrwxrwxrwx 1 root root 0 oct. 20 14:29 cgroup -> 'cgroup:[4026532692]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 ipc -> 'ipc:[4026532690]'
lrwxrwxrwx 1 root root 0 oct. 20 14:14 mnt -> 'mnt:[4026532687]'
lrwxrwxrwx 1 root root 0 oct. 20 14:14 net -> 'net:[4026532693]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 pid -> 'pid:[4026532691]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 pid_for_children -> 'pid:[4026532691]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 oct. 20 14:29 uts -> 'uts:[4026532689]'

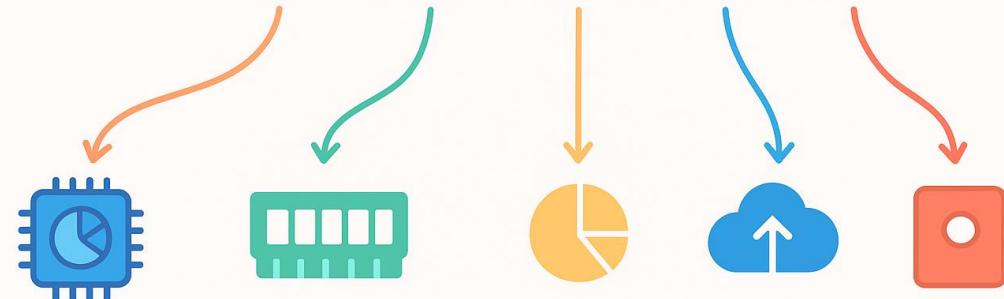
aight@Exodus:~$ 
aight@Exodus:~$ 
aight@Exodus:~$ sudo cat /proc/4186/cgroup
0:::/system.slice/docker-627ddb227b05af381edfc9a4f02b906371e85622f63edddcd04096dc8d516bb86.scope
aight@Exodus:~$ 
```

sudo docker stats

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
627ddb227b05	distracted_murdock	0.00%	10.59MiB / 5.602GiB	0.18%	17.5kB / 126B	7.82MB / 12.3kB	3

À quoi servent ces colonnes ?

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
627ddb227b05	distracted_murdock	0.00%	10.59MiB / 5.502GiB	0.18%	17.5kB / 126B	7.82MB



Utilisation
du CPU

Utilisation
de la mémoire

Pourcentage
de mémoire

Entrées / sorties
réseau

Entrées / sorties
disque

Commande docker

Cheatsheet for Docker CLI			
Run a new Container	Manage Containers	Manage Images	Info & Stats
<p>Start a new Container from an Image <code>docker run IMAGE</code> <code>docker run nginx</code></p> <p>...and assign it a name <code>docker run --name CONTAINER IMAGE</code> <code>docker run --name web nginx</code></p> <p>...and map a port <code>docker run -p HOSTPORT:CONTAINERPORT IMAGE</code> <code>docker run -p 8080:80 nginx</code></p> <p>...and map all ports <code>docker run -P IMAGE</code> <code>docker run -P nginx</code></p> <p>...and start container in background <code>docker run -d IMAGE</code> <code>docker run -d nginx</code></p> <p>...and assign it a hostname <code>docker run --hostname HOSTNAME IMAGE</code> <code>docker run --hostname srv nginx</code></p> <p>...and add a dns entry <code>docker run --add-host HOSTNAME:IP IMAGE</code></p> <p>...and map a local directory into the container <code>docker run -v HOSTDIR:TARGETDIR IMAGE</code> <code>docker run -v ~/usr/share/nginx/html nginx</code></p> <p>...but change the entrypoint <code>docker run -it --entrypoint EXECUTABLE IMAGE</code> <code>docker run -it --entrypoint bash nginx</code></p>	<p>Show a list of running containers <code>docker ps</code></p> <p>Show a list of all containers <code>docker ps -a</code></p> <p>Delete a container <code>docker rm CONTAINER</code> <code>docker rm web</code></p> <p>Delete a running container <code>docker rm -f CONTAINER</code> <code>docker rm -f web</code></p> <p>Delete stopped containers <code>docker container prune</code></p> <p>Stop a running container <code>docker stop CONTAINER</code> <code>docker stop web</code></p> <p>Start a stopped container <code>docker start CONTAINER</code> <code>docker start web</code></p> <p>Copy a file from a container to the host <code>docker cp CONTAINER:SOURCE TARGET</code> <code>docker cp web:/index.html index.html</code></p> <p>Copy a file from the host to a container <code>docker cp TARGET CONTAINER:SOURCE</code> <code>docker cp index.html web:/index.html</code></p> <p>Start a shell inside a running container <code>docker exec -it CONTAINER EXECUTABLE</code> <code>docker exec -it web bash</code></p> <p>Rename a container <code>docker rename OLD_NAME NEW_NAME</code> <code>docker rename 096 web</code></p> <p>Create an image out of container <code>docker commit CONTAINER</code> <code>docker commit web</code></p>	<p>Download an image <code>docker pull IMAGE[:TAG]</code> <code>docker pull nginx</code></p> <p>Upload an image to a repository <code>docker push IMAGE</code> <code>docker push myimage:1.0</code></p> <p>Delete an image <code>docker rmi IMAGE</code></p> <p>Show a list of all Images <code>docker images</code></p> <p>Delete dangling images <code>docker image prune</code></p> <p>Delete all unused images <code>docker image prune -a</code></p> <p>Build an image from a Dockerfile <code>docker build DIRECTORY</code> <code>docker build .</code></p> <p>Tag an image <code>docker tag IMAGE NEWIMAGE</code> <code>docker tag ubuntu ubuntu:18.04</code></p> <p>Build and tag an image from a Dockerfile <code>docker build -t IMAGE DIRECTORY</code> <code>docker build -t myimage .</code></p> <p>Save an image to tar file <code>docker save IMAGE > FILE</code> <code>docker save nginx > nginx.tar</code></p> <p>Load an image from a tar file <code>docker load -i TARFILE</code> <code>docker load -i nginx.tar</code></p>	<p>Show the logs of a container <code>docker logs CONTAINER</code> <code>docker logs web</code></p> <p>Show stats of running containers <code>docker stats</code></p> <p>Show processes of container <code>docker top CONTAINER</code> <code>docker top web</code></p> <p>Show installed docker version <code>docker version</code></p> <p>Get detailed info about an object <code>docker inspect NAME</code> <code>docker inspect nginx</code></p> <p>Show all modified files in container <code>docker diff CONTAINER</code> <code>docker diff web</code></p> <p>Show mapped ports of a container <code>docker port CONTAINER</code> <code>docker port web</code></p>

```
aghi@Exodus:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
hello-world         latest   1b44b5a3e06a  2 months ago  10.1kB
docker/welcome-to-docker  latest  6caf772f5178  2 months ago  14.1MB
aghi@Exodus:~$ 
aghi@Exodus:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
c613780f0ecb      docker/welcome-to-docker "/docker-entrypoint..." 7 minutes ago   Up 7 minutes    0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   youthful_moser
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo docker stop c613780f0ecb
c613780f0ecb
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo docker rm -f youthful_moser
youthful_moser
aghi@Exodus:~$ 
```

Debian sous docker

Objectif :

- Création d'un conteneur qui tourne sur un os debian, avec possibilité d'utiliser le terminal.

```
docker run -it --rm --name deb_os
```

--rm : dès que le conteneur s'arrête, il s'efface

```
aghi@Exodus:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
debian              latest   61d0976acea   3 weeks ago  120MB
```

```

root@08e232449f30 ~# ls
bin@ boot/ dev/ etc/ home/ lib@ lib64@ media/ mnt/ opt/ proc/ root/ run/ sbin@ srv/ sys/ tmp/ usr/ var/
root@08e232449f30 ~#
root@08e232449f30 ~# cd _/
root@08e232449f30 ~# mkdir test
root@08e232449f30 ~# cd test
root@08e232449f30 ~/test# touch file
root@08e232449f30 ~/test# echo "file teste" > file && cat file
file teste
root@08e232449f30 ~/test# 

```

- Ajout de **DNS** et modification du **Hostname** :

```

aghi@Exodus:~$ docker run -it --rm --name deb_os --dns 8.8.8.8 --hostname titi debian:latest
root@titi:~/#

```

Suppression de la totalité des conteneurs

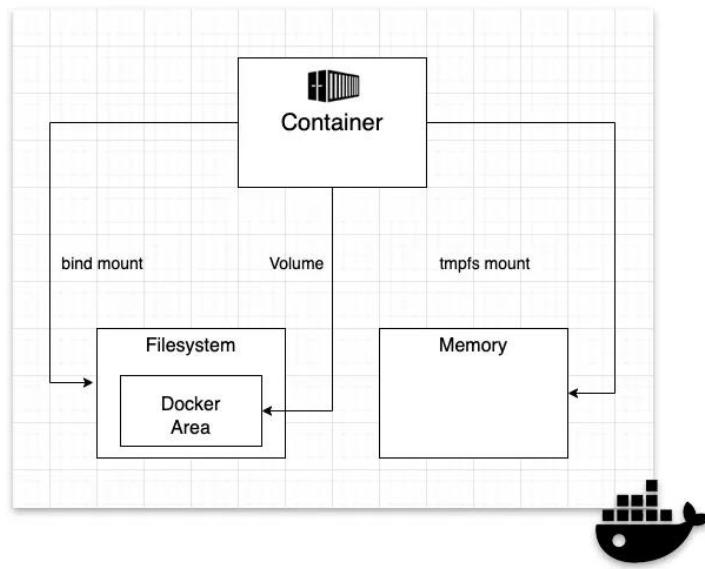
```

aghi@Exodus:~$ docker ps -q
aghi@Exodus:~$ docker rm -f $(docker ps -qa)
751e430be638
aghi@Exodus:~$ 
aghi@Exodus:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
aghi@Exodus:~$ 

```

Les volumes pour persister les données

- Facile pour persister de la donnée
- Pratique pour faire des backups
- Partager entre de multiple conteneur
- Multi-conteneur et permissions
- Locale ou distant



The Images shown are for illustration purposes only

```
aghi@Exodus:~$ docker volume ls
DRIVER      VOLUME NAME
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker volume create my_nginx
my_nginx
aghi@Exodus:~$ 
aghi@Exodus:~$ docker volume ls
DRIVER      VOLUME NAME
local      my_nginx
aghi@Exodus:~$ 
aghi@Exodus:~$ docker run -d --name web -v my_nginx:/usr/share/nginx/html/ nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8c7716127147: Pull complete
250b90fb2b9a: Pull complete
5d8ea9f4c626: Pull complete
58d144c4badd: Pull complete
b459da543435: Pull complete
8da8ed3552af: Pull complete
54e822d8ee0c: Pull complete
Digest: sha256:3b7732505933ca591ce4a6d860cb713ad96a3176b82f7979a8dfa9973486a0d6
Status: Downloaded newer image for nginx:latest
3b8fdbd5509d9bee1cb1959749d8df74147df4221d9c56ebefc369a139aaeee723
aghi@Exodus:~$ 
aghi@Exodus:~$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED        STATUS        PORTS
NAMES
3b8fdbd5509d9   nginx:latest   "/docker-entrypoint...."   17 seconds ago   Up 16 seconds   80/tcp
web
aghi@Exodus:~$ docker exec -it web bash
root@3b8fdbd5509d9:/# ls -l /usr/share/nginx/html/
total 8
-rw-r--r-- 1 root root 497 Oct  7 17:04 50x.html
-rw-r--r-- 1 root root 615 Oct  7 17:04 index.html
root@3b8fdbd5509d9:/# █
```

```
aghi@Exodus:~$ docker volume inspect my_nginx
[
  {
    "CreatedAt": "2025-10-20T15:55:53+02:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/my_nginx/_data",
    "Name": "my_nginx",
    "Options": null,
    "Scope": "local"
  }
]
```

```

[✓] aghi@Exodus: ~ 80x46
aghi@Exodus:~$ sudo ls -l /var/lib/docker/volumes/
total 28
brw----- 1 root root 8, 3 oct. 20 14:11 backingFsBlockDev
-rw----- 1 root root 32768 oct. 20 15:55 metadata.db
drwx-----x 3 root root 4096 oct. 20 15:55 my_nginx
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo ls -l /var/lib/docker/volumes/my_nginx
total 4
drwxr-xr-x 2 root root 4096 oct. 20 15:59 _data
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo cat /var/lib/docker/volumes/my_nginx/_data/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
aghi@Exodus:~$ 

```

```

[✓] aghi@Exodus: ~ 89x46
aghi@Exodus:~$ docker run -it --name deb -v my_nginx:/data debian:latest
root@160e26624ab4:/#
root@160e26624ab4:/# ls
bin  data  etc  lib  media  opt  root  sbin  sys  usr
boot  dev  home  lib64  mnt  proc  run  srv  tmp  var
root@160e26624ab4:/# ls /data
50x.html index.html
root@160e26624ab4:/# cat /data/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

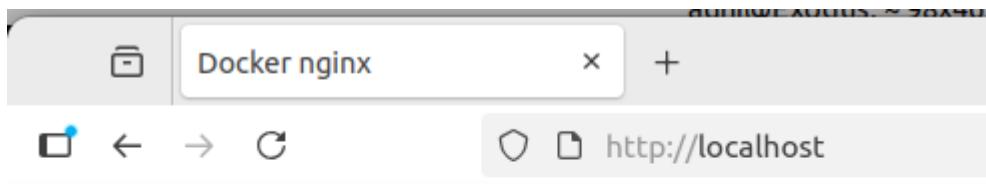
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@160e26624ab4:/# echo "hellllo" > /data/index.html
root@160e26624ab4:/#
root@160e26624ab4:/# cat /data/index.html
hellllo
root@160e26624ab4:/# 
aghi@Exodus:~$ docker volume rm my_nginx
Error from daemon: remove my_nginx: volume is in use - [3b8fdb5509d9bee1cb1959749d8df74147df4221d9c56ebefc369a139aaee72
3, 160e26624ab455ec7621a9f507362a8b4fa0561c70603ba8d1c063936b78ef72]
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker rm -f $(docker ps -qa)
160e26624ab4
3b8fdb5509d9
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker volume rm my_nginx
my_nginx
aghi@Exodus:~$ docker volume ls
DRIVER      VOLUME NAME
aghi@Exodus:~$ 

```

Création d'un simple site web sous nginx

```
docker run -d --name web -p 80:80 -v web:/usr/share/nginx/html/ nginx:latest
```



Welcome to Docker!

Votre conteneur fonctionne bien !!

```
aghi@Exodus:~$ docker volume ls
DRIVER      VOLUME NAME
aghi@Exodus:~$
aghi@Exodus:~$ docker volume create web
web
aghi@Exodus:~$
aghi@Exodus:~$ docker volume inspect web
[
    {
        "CreatedAt": "2025-10-20T16:36:13+02:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/web/_data",
        "Name": "web",
        "Options": null,
        "Scope": "local"
    }
]
[  aghi@Exodus: ~ 80x46
aghi@Exodus:~$ sudo nano /var/lib/docker/volumes/web/_data/index.html
aghi@Exodus:~$ sudo nano /var/lib/docker/volumes/web/_data/index.html
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo cat /var/lib/docker/volumes/web/_data/index.html
<!DOCTYPE html>
<html>
<head>
<title>Docker nginx</title>

</head>
<body>
<h1>Welcome to Docker!</h1>
<p>Votre conteneur fonctionne bien !!</p>
</body>
</html>
aghi@Exodus:~$
```

On a créé un **volume Docker** nommé **web** avec la commande `docker volume create web`, puis on l'a utilisé pour lancer un conteneur **Nginx**.

Ainsi, on a mis en place un **serveur web** fonctionnant dans un conteneur isolé, relié à un **volume persistant** pour stocker les fichiers du site.

Grâce à cette configuration, le contenu du site reste sauvegardé même si le conteneur est supprimé ou redémarré.

En résumé, on a configuré un environnement Docker capable d'héberger un site web accessible sur le port 80 de la machine, tout en conservant les données de manière durable.

Les différents types de volumes docker

Volume persistant (ou volume Docker)

C'est un volume géré directement par Docker.

Il est stocké dans un emplacement interne de Docker (`/var/lib/docker/volumes/...`) et reste présent même si le conteneur est supprimé.

On l'utilise pour **conserver les données durablement** (ex. : base de données, fichiers web).

```
aghi@Exodus:~$ docker volume create test
|test
aghi@Exodus:~$ docker run -d --name c1 --mount type=volume,src=test,destination=/usr/share/nginx/html/ nginx:latest
b41f91a439ad58d724e8fea885114fd0513ed3b1375791f2909b2af6a9958aa1
aghi@Exodus:~$ docker inspect --format {{.Mounts}} c1
[{"volume": "test", "source": "/var/lib/docker/volumes/test/_data", "target": "/usr/share/nginx/html", "type": "local", "z": true}]
aghi@Exodus:~$
```

Bind mount

Ce type de volume relie un **dossier du système hôte** à un dossier du conteneur.

Les modifications faites dans l'un sont immédiatement visibles dans l'autre.

C'est pratique pour le **développement**, car on peut modifier les fichiers sur l'hôte et voir les changements en temps réel dans le conteneur.

```
aghi@Exodus:~$ mkdir data
aghi@Exodus:~$ touch data/toto.html
aghi@Exodus:~$ echo "<p>ligne = toto</p>
<p>ligne = toto</p>"
```

```
aghi@Exodus:~$ docker run -d --name c2 --mount type=bind,source=~/data,target=/usr/share/nginx/html/ nginx:latest
360a7337e299d5d543de6bc91443b9e333c09f37239ffdcc27b58cbe92bcd3
aghi@Exodus:~$ docker inspect --format {{.Mounts}} c2
[{"bind": "/home/aghi/data", "source": "/usr/share/nginx/html", "target": "/usr/share/nginx/html", "type": "bind", "z": true, "rprivate": true}
aghi@Exodus:~$ docker exec -it c2 bash
root@360a7337e299:/#
root@360a7337e299:/# ls
bin  docker-entrypoint.d  home  media  proc  sbin  tmp
boot docker-entrypoint.sh  lib   mnt   root  srv  usr
dev   etc                lib64  opt   run   sys  var
root@360a7337e299:/# ls /usr/share/nginx/html/toto.html
/usr/share/nginx/html/toto.html
root@360a7337e299:/#
```

tmpfs mount

Ce volume est stocké uniquement en **mémoire RAM**, donc il est **temporaire**.

Il disparaît dès que le conteneur s'arrête.

On l'utilise pour des **données sensibles ou temporaires** qui ne doivent pas être écrites sur le disque.

```
aghi@Exodus:~$ docker run -d --name c3 --mount type=tmpfs,destination=/usr/share/nginx/html/ nginx:latest
51599fe04e370280de6bb2c71bd4f1a9929660d4538f60aba93c7d32206e54b8
aghi@Exodus:~$ docker inspect --format="{{.Mounts}} c3
[{"tmpfs": "/usr/share/nginx/html", "true"}]
aghi@Exodus:~$ aghi@Exodus:~$ docker exec -it c3 bash
root@51599fe04e37:/#
root@51599fe04e37:/# ls /usr/share/nginx/html/
root@51599fe04e37:/# ls -la /usr/share/nginx/html/
total 4
drwxr-xr-x 2 root root 40 Oct 20 22:31 .
drwxr-xr-x 3 root root 4096 Oct 8 00:14 ..
root@51599fe04e37:/# ← vide
```

Importance du UserID pour les volumes

- **Éviter l'exécution en root :**

Même si exécuter le conteneur en root semble plus simple (car il contourne les problèmes de permissions), c'est une mauvaise pratique de sécurité.

Il vaut mieux :

- Créer un utilisateur non-root dans le **Dockerfile**
- Adapter les permissions du volume à cet utilisateur.

Cela limite les risques d'accès non autorisé ou de corruption de données sur l'hôte.

```
aghi@Exodus ~> cd docker/img1/
aghi@Exodus ~/d/img1>
aghi@Exodus ~/d/img1> pwd
/home/aghi/docker/img1
aghi@Exodus ~/d/img1>
aghi@Exodus ~/d/img1> ls -l
total 4
-rw-rw-r-- 1 aghi aghi 71 oct. 21 14:22 Dockerfile
aghi@Exodus ~/d/img1>
aghi@Exodus ~/d/img1> cat Dockerfile
FROM debian:latest
RUN useradd -u 1111 user1
RUN useradd -u 2222 user2
```

```
docker build -t img_deb .
```

```

aghi@Exodus:~/docker/img1$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
nginx          latest   07ccdb783875  13 days ago   160MB
debian          latest   61d0976aceca  3 weeks ago   120MB
hello-world    latest   1b44b5a3e06a  2 months ago  10.1kB
docker/welcome-to-docker latest   6caf772f5178  2 months ago  14.1MB
[+] Building 0.1s (7/7) FINISHED
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 108B
--> [internal] load metadata for docker.io/library/debian:latest
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [1/3] FROM docker.io/library/debian:latest
--> CACHED [2/3] RUN useradd -u 1111 user1
--> CACHED [3/3] RUN useradd -u 2222 user2
--> exporting to image
--> => exporting layers
--> => writing image sha256:d60d09ffcf02d1ef6b7e18ed55a32aceb9977970bc53d16e2c4dcba850bb941
--> => naming to docker.io/library/img1:v1.0
[+] Building 0.1s (7/7) FINISHED
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
img1           v1.0    d60d09ffcf0   27 minutes ago  120MB
nginx          latest   07ccdb783875  13 days ago   160MB
debian          latest   61d0976aceca  3 weeks ago   120MB
hello-world    latest   1b44b5a3e06a  2 months ago  10.1kB
docker/welcome-to-docker latest   6caf772f5178  2 months ago  14.1MB
[+] Building 0.1s (7/7) FINISHED
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
img1           v1.0    d60d09ffcf0   27 minutes ago  120MB
nginx          latest   07ccdb783875  13 days ago   160MB
debian          latest   61d0976aceca  3 weeks ago   120MB
hello-world    latest   1b44b5a3e06a  2 months ago  10.1kB
docker/welcome-to-docker latest   6caf772f5178  2 months ago  14.1MB

```

```
sudo chown -R 1111:1111 /home/aghi/docker/img1/data/
```

```

aghi@Exodus:~/docker/img1$ ls -lh
total 8,0K
drwxrwxr-x 2 1111 1111 4,0K oct. 21 14:27 data
-rw-rw-r-- 1 aghi aghi 71 oct. 21 14:22 Dockerfile
aghi@Exodus:~/docker/img1$ 

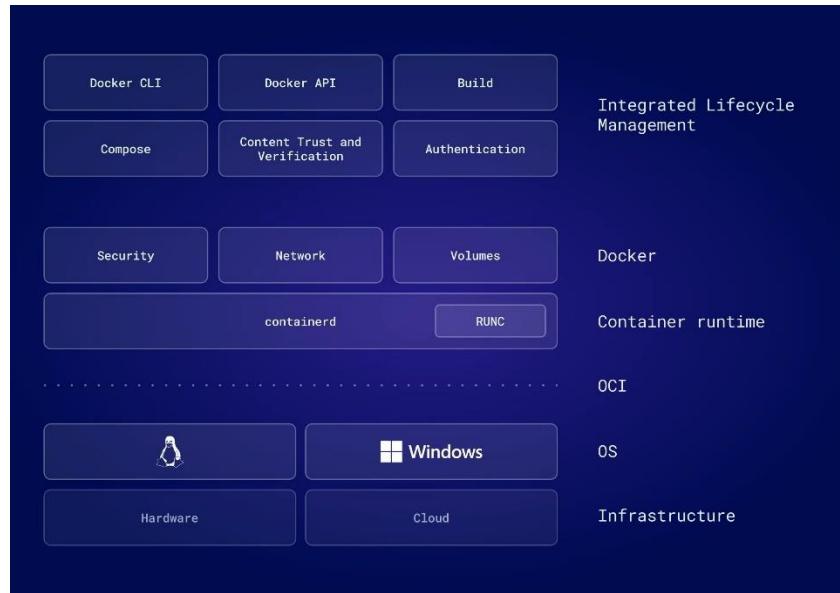
```

```

aghi@Exodus:~/docker/img1$ docker run -d --name mydeb -v /home/aghi/docker/img1/data/:/data/ -u 1111 img1:v1.0 sleep infinity
8c1c8aa9035a4664ec278f569fd8f83414d93c8d33aaaf4f17d08c00f5a25124a
aghi@Exodus:~/docker/img1$ docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS     NAMES
8c1c8aa9035a        img1:v1.0   "sleep infinity"    7 seconds ago      Up 7 seconds          mydeb
aghi@Exodus:~/docker/img1$ docker ps -aux | grep "sleep"
1111      3422  0.1  0.0   2588 1152 ?          Ss   14:58  0:00 sleep infinity
aghi      3472  0.0  0.0   11648 2560 pts/0  S+   14:59  0:00 grep --color=auto sleep
aghi@Exodus:~/docker/img1$ docker exec -it -u 1111 mydeb bash
user1@8c1c8aa9035a:/S
user1@8c1c8aa9035a:/S
user1@8c1c8aa9035a:/S cd /data/
user1@8c1c8aa9035a:/data$ ls -l
total 0
-rw-r--r-- 1 user1 user1 0 Oct 21 12:27 toto
user1@8c1c8aa9035a:/data$ 
user1@8c1c8aa9035a:/data$ touch titi
user1@8c1c8aa9035a:/data$ echo "modification avec user1" > titi
modification avec user1
user1@8c1c8aa9035a:/data$ 
user1@8c1c8aa9035a:/data$ exit
exit
aghi@Exodus:~/docker/img1$ 
aghi@Exodus:~/docker/img1$ 
aghi@Exodus:~/docker/img1$ ls data/
titi toto
aghi@Exodus:~/docker/img1$ ls -lh data/
total 0
-rw-r--r-- 1 1111 1111 0 oct. 21 14:59 titi
-rw-r--r-- 1 1111 1111 0 oct. 21 14:27 toto
aghi@Exodus:~/docker/img1$ cat toto
cat: toto: Aucun fichier ou dossier de ce nom
aghi@Exodus:~/docker/img1$ 
aghi@Exodus:~/docker/img1$ cat data/toto
aghi@Exodus:~/docker/img1$ echo "modification avec user1" > toto
aghi@Exodus:~/docker/img1$ docker exec -it -u 1111 mydeb bash
user1@8c1c8aa9035a:/S cat /data/toto
user1@8c1c8aa9035a:/S
user1@8c1c8aa9035a:/S echo "modification avec user1" > toto
bash: toto: Permission denied
user1@8c1c8aa9035a:/S echo "modification avec user1" > data/toto
user1@8c1c8aa9035a:/S exit
exit
aghi@Exodus:~/docker/img1$ cat data/toto
modification avec user1
←
aghi@Exodus:~/docker/img1$ 

```

Container Runtimes



Un **container runtime** est le moteur qui exécute réellement les conteneurs. Il gère toutes les opérations de bas niveau nécessaires pour :

- Démarrer et arrêter les conteneurs
- Gérer leur isolation (namespaces, cgroups)
- Monter les volumes et réseaux
- Exécuter les processus à l'intérieur du conteneur.

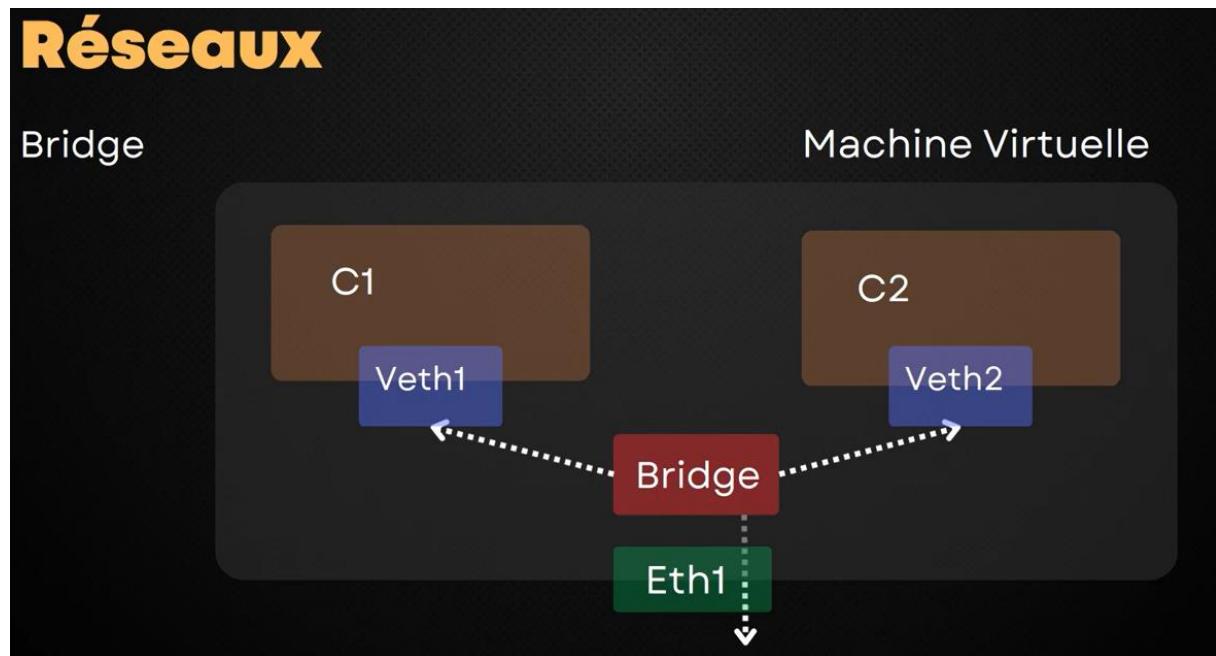
docker run nginx

1. **Docker CLI** : C'est l'outil que tu utilises dans le terminal.
Il envoie la commande au **Docker Daemon (dockerd)**.
2. **Docker Daemon (dockerd)** : C'est le service principal ou container manager.
Il gère les images (build, inspect, login, exec...), **Il ne lance pas directement les conteneurs**, mais délègue cette tâche à **un runtime**.
3. **Containerd** : C'est le **container runtime de haut niveau** intégré à Docker.
Il reçoit les instructions du **démon Docker** et s'occupe :
 - Démarrer **runc**
 - Il gère la partie réseau/passerelles.
4. **runc** : C'est le **runtime de bas niveau**.
containerd l'utilise pour créer les processus isolés (via namespaces, cgroups).
Il interagit directement au **noyau Linux** pour lancer le conteneur.

Les réseaux docker

- Communication entre les conteneurs ou vers l'extérieur
- Types : **host, bridge, none, overlay...**
 - Host : utiliser directement le réseau de l'hôte sur la quel tourne le conteneur.
 - None : ne pas autoriser la communication entre l'intérieur et l'extérieur.
 - Overlay : communiquer entre différents conteneurs situer sur différent machines en un réseau unique.
- **Un conteneur n'a pas d'ip fixe.**

Mode bridge (par défaut)



C'est une couche logique connecter à l'interface réseau de la machine, permet de créer un réseau virtuel interne à notre hôte.

Crée une interface Virtual pour chaque conteneur avec le format «vethX », fait office de passerelle (**docker0**)

Visualiser docker0

```
aghi@Exodus:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 10
00
    link/loopback 00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:5b:82:f4 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.232.131/24 brd 192.168.232.255 scope global dynamic noprefixroute ens33
        valid_lft 1006sec preferred_lft 1006sec
    inet6 fe80::41c5:86c3:abda:1f67/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 42:36:7a:5b:cd:23 brd ff:ff:ff:ff:ff:ff
4: veth6527523@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether f6:8d:31:49:6e:38 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::f48d:31ff:fe49:6e38/64 scope link
        valid_lft forever preferred_lft forever
aghi@Exodus:~$ docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
5921512f7ccc   bridge    bridge      local
0a055346858f   host      host      local
3705128926c0   none      null      local
aghi@Exodus:~$ docker inspect --format {{.Driver}} 5921512f7ccc
bridge
aghi@Exodus:~$ docker inspect --format {{.IPAM}} 5921512f7ccc
"default" map[] [{"Subnet": "172.17.0.0/16", "Gateway": "172.17.0.1"}]
aghi@Exodus:~$
```

docker0 est la **passerelle réseau virtuel par défaut** que Docker crée sur la machine hôte lors de l'installation et du démarrage du service Docker. On peut le voir comme un **commutateur virtuel** qui permet de connecter nos conteneurs Docker entre eux et avec l'hôte. Pour chaque conteneur créé, Docker crée une **paire Ethernet virtuelle (veth)** :

- Une extrémité va dans le conteneur.
- L'autre extrémité se connecte au pont docker0.

```
aghi@Exodus:~$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED      STATUS      PORTS     NAMES
f4099027f2aa        img1:v1.0   "sleep infinity"   6 minutes ago   Up 6 minutes   deb

aghi@Exodus:~$ docker network ls
NETWORK ID          NAME        DRIVER      SCOPE
77b2c609a343        bridge      bridge      local
0a055346858f        host        host        local
3705128926c0        none       null       local

aghi@Exodus:~$
aghi@Exodus:~$
aghi@Exodus:~$ docker exec -ti deb bash
root@f4099027f2aa:#
root@f4099027f2aa: # ip a
bash: ip: command not found
root@f4099027f2aa: # ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:d8:f5:36:50:c4  txqueuelen 0  (Ethernet)
          RX packets 852  bytes 13656765 (13.0 MiB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 745  bytes 43200 (42.1 KiB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
```

```
root@f4099027f2aa: # ping -c 2 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.036 ms

--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1019ms
rtt min/avg/max/mdev = 0.036/0.071/0.106/0.035 ms
root@f4099027f2aa: #
root@f4099027f2aa: # ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=4.36 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=12.4 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 4.357/8.354/12.351/3.997 ms
root@f4099027f2aa: #
```

Expose vs Publish

- Expose : Metadata d'une image prise en compte au lancement du conteneur
 - **EXPOSE** ou **-expose**
- Publish : Mapping de ports soit :
 - Aléatoire : **-P** (numéro de port > 32000)
 - Défini : **-p**

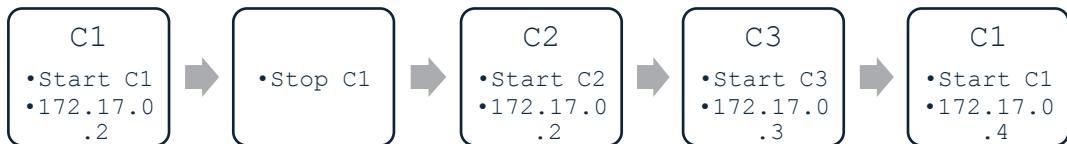
```
docker run -d --name web --expose 8080 nginx
```

The terminal session shows the following sequence:

- Initial state: `9ac3c1d742f7a5a0734c3decf5ad431af17268654a7f0ff3390a1e5d5b01d49a`
- `docker ps`: Shows a single container named `web` running an `nginx` image.
- `curl 172.17.0.2:8080`: Returns the default Nginx welcome page.
- `ping 172.17.0.2`: Shows a ping to the container's IP.
- `curl 172.17.0.2:8080`: Returns a connection refused error.

La CLI network

Ip d'un conteneur



CLI : docker network

- Problématique de l'*IP non statique* :
 - 1^{ère} solution : Le **DNS**
 - **BUT** : Création de notre propre réseau personnaliser, afin d'éviter d'utiliser le **dockero**

The terminal session shows the following steps:

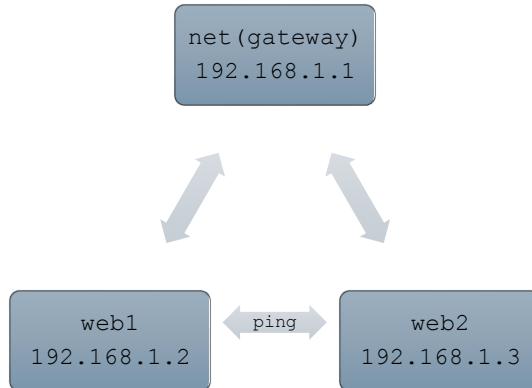
- `docker network ls`: Lists existing networks.
- `docker network create --driver=bridge --subnet=192.168.1.0/24 net`: Creates a new bridge network named `net`.
- `docker network ls`: Lists the newly created network `net`.
- `curl 172.17.0.2:8080`: Returns the default Nginx welcome page.
- `curl 172.17.0.3:8080`: Returns the default Nginx welcome page.
- `curl 172.17.0.4:8080`: Returns the default Nginx welcome page.

On the right, the output of `docker network inspect net` is shown, detailing the network configuration including its ID, driver (bridge), subnet (192.168.1.0/24), and other properties.

```

aghi@Exodus:~$ docker run -d --name web1 --network net nginx:latest
6a660098d9f88dbaaf8099fe52680a8c9388660e564e29912003f35d0be15d79
aghi@Exodus:~$ 
aghi@Exodus:~$ docker run -d --name web2 --network net nginx:latest
503897e4d0577a08b94e37f3c105fb35cdbb7170ca9ce35fe2f66c4d1c1c79f6
aghi@Exodus:~$ 
aghi@Exodus:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
503897e4d057 nginx:latest "/docker-entrypoint..." 5 seconds ago Up 4 seconds 80/tcp web2
6a660098d9f8 nginx:latest "/docker-entrypoint..." 11 seconds ago Up 10 seconds 80/tcp web1
aghi@Exodus:~$ 

```



```

aghi@Exodus:~ 84x46
root@6a660098d9f8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.2 netmask 255.255.255.0 broadcast 192.168.1.255
        ether f6:c2:e8:c7:fd:2b txqueuelen 0 (Ethernet)
        RX packets 3634 bytes 10473881 (9.9 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3035 bytes 166214 (162.3 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 10 bytes 924 (924.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 10 bytes 924 (924.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@6a660098d9f8:/# ping -c 2 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.139 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.106 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.106/0.122/0.139/0.016 ms
root@6a660098d9f8:/# 

root@6a660098d9f8:/# ping -c 2 web1
PING web1 (192.168.1.2) 56(84) bytes of data.
64 bytes from 6a660098d9f8 (192.168.1.2): icmp_seq=1 ttl=64 time=0.246 ms
64 bytes from 6a660098d9f8 (192.168.1.2): icmp_seq=2 ttl=64 time=0.077 ms

--- web1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.077/0.161/0.246/0.084 ms
root@6a660098d9f8:/#
root@6a660098d9f8:/# ping -c 2 web2
PING web2 (192.168.1.3) 56(84) bytes of data.
64 bytes from web2.net (192.168.1.3): icmp_seq=1 ttl=64 time=0.253 ms
64 bytes from web2.net (192.168.1.3): icmp_seq=2 ttl=64 time=0.105 ms

--- web2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1058ms
rtt min/avg/max/mdev = 0.105/0.179/0.253/0.074 ms
root@6a660098d9f8:/# 

```

Connect/disconnect

```
aghi@Exodus:~$ docker run -d --name web3 nginx:latest
9e73f4d01e015ceb532bdf54f790abef9df55fc0c56fa6c657ee32db528d37f3
aghi@Exodus:~$ docker inspect 9e73f4d01e01 | grep "IPAddress"
    "SecondaryIPAddresses": null,
    "IPAddress": "172.17.0.2",
    "IPAddress": "172.17.0.2",
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker network connect net web3
aghi@Exodus:~$ 
aghi@Exodus:~$ docker inspect 9e73f4d01e01 | grep "IPAddress"
    "SecondaryIPAddresses": null,
    "IPAddress": "172.17.0.2",
    "IPAddress": "172.17.0.2",
    "IPAddress": "192.168.1.4",
aghi@Exodus:~$ 
aghi@Exodus:~$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
b57d292ae180   bridge    bridge      local
0a055346858f   host      host       local
1d12f6da7abf   net       bridge      local
3705128926c0   none      null       local
aghi@Exodus:~$ 
aghi@Exodus:~$ docker network disconnect web3 bridge
Error response from daemon: No such container: bridge
aghi@Exodus:~$ 
aghi@Exodus:~$ docker network disconnect bridge web3
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker inspect 9e73f4d01e01 | grep "IPAddress"
    "SecondaryIPAddresses": null,
    "IPAddress": "",
    "IPAddress": "192.168.1.4",
aghi@Exodus:~$ 
```

Docker0

Configurer le docker0

- Fichier : `/etc/docker/daemon.json`
 - Bip
 - Mtu
 - Default-gateway
 - Dns...

```

aghi@Exodus:~$ sudo ls -la /etc/docker/
[sudo] Mot de passe de aghi :
total 16
drwxr-xr-x  2 root root  4096 mai   30 14:07 .
drwxr-xr-x 136 root root 12288 oct.  20 15:53 ..
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo nano /etc/docker/daemon.json
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo cat /etc/docker/daemon.json
[
    "bip" : "10.10.0.1/24"
]
aghi@Exodus:~$ 
aghi@Exodus:~$ sudo systemctl restart docker
aghi@Exodus:~$ 
aghi@Exodus:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:5b:82:f4 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.232.131/24 brd 192.168.232.255 scope global dynamic noprefixroute ens3
        valid_lft 1122sec preferred_lft 1122sec
    inet6 fe80::41c5:86c3:abda:1f67/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 5a:de:36:c9:a8:09 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.1/24 brd 10.10.0.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::58de:36ff:fec9:a809/64 scope link
        valid_lft forever preferred_lft forever

```

Création d'un nouveau bridge :

```

aghi@Exodus:~$ docker network create -o com.docker.network.bridge.name="net_dock0" aghi_net
5f41e10621d9add5f2d641ee004ac865100b0d253cfef6204b6ce68ec5fa11d07
aghi@Exodus:~$ 
aghi@Exodus:~$ 
aghi@Exodus:~$ docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
5f41e10621d9   aghi_net   bridge      local
9fdada1579513  bridge     bridge      local
0a055346858f   host      host      local
1d12f6da7abf   net       bridge      local
3705128926c0   none      null      local
aghi@Exodus:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:5b:82:f4 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.232.131/24 brd 192.168.232.255 scope global dynamic noprefixroute ens3
        valid_lft 1548sec preferred_lft 1548sec
    inet6 fe80::41c5:86c3:abda:1f67/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 5a:de:36:c9:a8:09 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.1/24 brd 10.10.0.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::58de:36ff:fec9:a809/64 scope link
        valid_lft forever preferred_lft forever
9: br-1d12f6da7abf: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 0a:ca:a7:20:e2:d5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global br-1d12f6da7abf
        valid_lft forever preferred_lft forever
    inet6 fe80::8ca:a7ff:fe20:e2d5/64 scope link
        valid_lft forever preferred_lft forever
18: net_dock0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether ce:c7:f1:01:f9:aa brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global net_dock0
        valid_lft forever preferred_lft forever
aghi@Exodus:~$ 

```

Création d'images

Commit (sans Dockerfile)

docker commit est une commande Docker qui permet de créer une nouvelle image à partir d'un conteneur en cours d'exécution ou arrêté, sans utiliser de Dockerfile.

Cas d'usage typiques

- Faire un **snapshot** rapide avant une modification risquée.
- Sauvegarder un **état expérimental** d'un conteneur.
- Créer une **base de travail** avant d'écrire un vrai Dockerfile.
- Utile pour du **prototypage rapide** ou du **debug**, pas pour la production.

```
aghi@Exodus:~$ docker run -d --name ubuntu00 ubuntu:latest sleep infinity

aghi@Exodus:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b6ca72a185bc ubuntu:latest "sleep infinity" 8 minutes ago Up 8 minutes
ubuntu00

aghi@Exodus:~$ docker commit ubuntu00 my_ubuntu:v1.0
sha256:e5e1dab62a80d04df697a557cf826c9da5f2b19280aa7d620295ca99006f36d7
aghi@Exodus:~$ 
aghi@Exodus:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
my_ubuntu v1.0 e5e1dab62a80 6 seconds ago 260MB
nginx latest 07ccdb783875 2 weeks ago 160MB
ubuntu latest 97bed23a3497 3 weeks ago 78.1MB
debian latest 61d0976aceca 3 weeks ago 120MB
aghi@Exodus:~$ 
aghi@Exodus:~$ docker run -d --name ubuntu01 --hostname ubuntu01 my_ubuntu:v1.0
3361197161d05c9e5aff1376b494ffa88c869e3374cf77e95698999d71dd093a
aghi@Exodus:~$ 
aghi@Exodus:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
3361197161d0 my_ubuntu:v1.0 "sleep infinity" 5 seconds ago Up 4 seconds
ubuntu01
b6ca72a185bc ubuntu:latest "sleep infinity" 10 minutes ago Up 10 minutes
ubuntu00
aghi@Exodus:~$ 
aghi@Exodus:~$ docker exec -it ubuntu01 bash
root@ubuntu01:#
root@ubuntu01:~# fish
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
root@ubuntu01:~# 

aghi@Exodus:~$ docker commit --message "add fish & net tools (ip,ping)" --author aghilas ubuntu00 my_ubuntu:v1.0
sha256:b822b85db473505eb4e18b8c4aa22eb9e8ea827c841f05a91ba65a2b897de2f
aghi@Exodus:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
my_ubuntu v1.0 b822b85db47 11 seconds ago 260MB
<none> <none> eed0339f0b7f 2 minutes ago 260MB
<none> <none> 9bcfd54ba940 3 minutes ago 260MB
<none> <none> e5e1dab62a80 4 minutes ago 260MB
<none> <none> cef04442454d 4 minutes ago 260MB
<none> <none> e5e1dab62a80 14 minutes ago 260MB
nginx latest 07ccdb783875 2 weeks ago 160MB
ubuntu latest 97bed23a3497 3 weeks ago 78.1MB
debian latest 61d0976aceca 3 weeks ago 120MB
aghi@Exodus:~$ 

aghi@Exodus:~$ docker inspect b822b85db47
[{"Id": "sha256:b822b85db473505eb4e18b8c4aa22eb9e8ea827c841f05a91ba65a2b897de2f",
 "RepoTags": [
   "my_ubuntu:v1.0"
 ],
 "RepoDigests": [],
 "Comment": "add fish &0026 net tools (ip,ping)",
 "Created": "2025-10-22T14:29:24.899401334Z",
 "DockerVersion": "28.5.1",
 "Author": "aghilas",
 "Architecture": "arm64",
 "Os": "linux",
 "Size": 260427,
 "GraphDriver": {
   "Data": [
     {"LowerDir": "/var/lib/docker/overlay2/93abff010bb856426219f9443bcd96837bf7ea31fe7fe10e23f901f31b0b23/diff",
      "MergedDir": "/var/lib/docker/overlay2/f50fd1d352364a4631c734e78c1337964b95f376a246ab70ae92066c17f515c8/merged",
      "UpperDir": "/var/lib/docker/overlay2/f50fd1d352364a4631c734e78c1337964b95f376a246ab70ae92066c17f515c8/diff",
      "WorkDir": "/var/lib/docker/overlay2/f50fd1d352364a4631c734e78c1337964b95f376a246ab70ae92066c17f515c8/work"
    },
    {"Name": "overlay2"
  },
  "RootFs": {
    "Type": "layers",
    "Layers": [
      "sha256:073ec47a8c22dcaa4d6e5758799ccfe2f9bde943685830b1bf6fd2395f5eabc"
    ]
  },
  "Metadata": {
    "LastTagTime": "2025-10-22T16:29:24.907843713+02:00"
  },
  "Config": {
    "Cmd": [
      "sleep",
      "infinity"
    ],
    "Image": "sha256:b822b85db473505eb4e18b8c4aa22eb9e8ea827c841f05a91ba65a2b897de2f"
  }
}]
```

Historique de modification d'une image

```
aghi@Exodus:~$ docker history ba822b85db47
IMAGE      CREATED      CREATED BY
COMMENT
ba822b85db47  5 minutes ago  sleep infinity          182MB
add fish & net tools (ip,ping)
97bed23a3497  3 weeks ago   /bin/sh -c #(nop) CMD ["/bin/bash"]    0B
<missing>     3 weeks ago   /bin/sh -c #(nop) ADD file:249778a1782b02a1c...  78.1MB
<missing>     3 weeks ago   /bin/sh -c #(nop) LABEL org.opencontainers....  0B
<missing>     3 weeks ago   /bin/sh -c #(nop) LABEL org.opencontainers....  0B
<missing>     3 weeks ago   /bin/sh -c #(nop) ARG LAUNCHPAD_BUILD_ARCH    0B
<missing>     3 weeks ago   /bin/sh -c #(nop) ARG RELEASE               0B
aghi@Exodus:~$
```

Dockerfile

Un **Dockerfile** est l'élément fondamental pour **automatiser la création d'images Docker**. Bien conçu (comme une recette de cuisine ou un makefile), il permet de garantir des **déploiements rapides, fiables et sécurisés**.

Les étapes :

- Bien choisir une image de départ
- Une suite d'instruction
- Processus principale lancé dans le conteneur

Structure du Dockerfile :

- C'est un fichier plat (pas d'indentation)
- Commentaire avec **#**
- Instruction en **majuscule**
- Arguments en **minuscule**
- **Parsers Dockerfile :**
 - **# syntax=docker/dockerfile:1.7** (Docker saura qu'il doit interpréter ce Dockerfile avec le parseur officiel docker/dockerfile version 1.7)
 - **# escape=** (utilisée pour changer le caractère d'échappement dans un Dockerfile)
- **L'ordre doit toujours être :**
 1. # syntax=...
 2. # escape=...
 3. Puis les instructions Docker (FROM, RUN, etc.)
- Le **.dockignore** : Exclure les fichiers temporaires, secrets, dépendances locales.

Exemple :

```

aghil@Exodus:~/docker/img2$ tree . && echo -e "\n\n" && cat .dockerignore && echo -e "\n\n" && cat Dockerfile
.
└── Dockerfile

0 directories, 1 file

# Empêche d'envoyer ces fichiers au build
Dockerfile
.dockerignore
*.log

#####
## le message "Hello docker! " s'affiche au moment du build ##
#####

# syntax=docker/dockerfile:1.7
# escape=\
FROM ubuntu:latest
RUN apt update && \
    echo "Hello docker!" aghil@Exodus:~/docker/img2$ [REDACTED]

```

The screenshot shows a terminal window with the following content:

```

img2 > Dockerfile > ...
1 ##### #####
2 ## le message "Hello docker! " s'affiche au moment du build ##
3 ##### #####
4
5 # syntax=docker/dockerfile:1.7
6 # escape=\
7
8 FROM ubuntu:latest
9 RUN apt update && \
10 | echo "Hello docker!"

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● aghil@Exodus:~/docker/img2$ docker build -t hello .
[+] Building 179.8s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 340B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 115B
=> [1/2] FROM docker.io/library/ubuntu:latest
=> [2/2] RUN apt update && echo "Hello docker!"
=> exporting to image
=> => exporting layers
=> => writing image sha256:8032dd87d8b3347be8eefc25c9abc993c7267a25043aa4fe21674178a6a73250
=> => naming to docker.io/library/hello

```

A red arrow points from the terminal window to the Docker build output, highlighting the progress bar and log messages.

Les instructions Dockerfile

L'instruction FROM

Est **obligatoire** dans tous Dockerfile., Elle indique **l'image de base** sur laquelle l'image va être construite.

Syntaxe :

FROM <image>[:<tag>] [AS <alias>]

L'instruction LABEL

Sert à **ajouter des informations descriptives** à l'image : auteur, version, description, mainteneur, etc.

Syntaxe :

LABEL clé="valeur" clé2="valeur2"

L'instruction ONBUILD

Permet de définir une instruction qui sera exécutée lorsque l'image sera utilisée comme image de base pour une autre image. Cela peut être utile pour créer des images de base qui incluent des instructions supplémentaires à exécuter lors de la construction d'une image dérivée.

Syntaxe :

ONBUILD <INSTRUCTION>

The screenshot shows a terminal window with two Dockerfiles and their build logs. The top Dockerfile defines an image with a git dependency:

```
img_deb > Dockerfile > ...
1 ##### ...
2 ## debian avec : git ...
3 ##### ...
4 ...
5 # syntax=docker/dockerfile:1.7 ...
6 # escape=\
7 ...
8 FROM debian:stable ...
9 LABEL auteur="aghi" \
10 | description="une image debian avec git installer"
11 ONBUILD RUN apt update && apt install git -y
```

The build log shows the ONBUILD command being executed:

```
[+] Building 0.7s (5/5) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 414B
=> [internal] load metadata for docker.io/library/debian:stable
=> [internal] load .dockerignore
=> transferring context: 2B
=> CACHED [1/1] FROM docker.io/library/debian:stable@sha256:5bcdced7b0c02512895dd702dfc903f3249755de86f72ab86e12b514ca9e4505e
=> exporting to image
=> exporting layers
=> writing image sha256:2d04e9f407e97192876a9a43989c8bad165c47e97f789ff7f2883a32256e3af0
=> naming to docker.io/library/deb_utils
```

The bottom Dockerfile creates a child image based on deb_utils:

```
img_deb > child > Dockerfile > ...
1 ##
2 ## Création d'une image dérivée
3 ...
4 FROM deb_utils:latest
5 ...
6 CMD ["bash"]
```

The build log for the child image shows the ONBUILD command being run:

```
[+] Building 132.6s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> transferring dockerfile: 112B
=> [internal] load metadata for docker.io/library/deb_utils:latest
=> [internal] load .dockerignore
=> transferring context: 2B
=> CACHED [1/1] FROM docker.io/library/deb_utils:latest
=> [2/2] ONBUILD RUN apt update && apt install git -y ↙
=> exporting to image
=> exporting layers
=> writing image sha256:aee27d32cd2e4f32488087c331e455ac4ef409e064b7c315026041018e0d3404
=> naming to docker.io/library/deb_git
```

- **CMD [“bash”]** : définit la commande par défaut à exécuter quand le conteneur démarre

```
aghi@Exodus:~/docker/img_deb$ docker run -it deb_git:latest
root@6c09a4f668b3:/#
root@6c09a4f668b3:/# git -v
git version 2.47.3
root@6c09a4f668b3:/#
```

L'instruction ENV

Permet de définir des variables d'environnement qui seront utilisés dans les commandes exécutées avec l'instruction **RUN**. Ces variables sont persistantes et seront disponibles au moment de l'exécution de l'image de conteneur.

```
ENV NOM_VARIABLE= "val1" VARIABLE2="valeur2"
```

L'instruction WORKDIR

Définit le répertoire de travail pour toutes les instructions qui la suivent dans le Dockerfile. Si le répertoire n'existe pas, il sera créé.

```
WORKDIR /path/to/workdir
```

The screenshot shows a terminal window with a Dockerfile and its build logs. The Dockerfile content is:

```
1 #####           #####
2 ## WORKDIR & ENV      ##
3 #####           #####
4
5 # syntax=docker/dockerfile:1.7
6 # escapes\
7 FROM debian:stable
8
9 ENV AUTEUR="Aghi"
10 ENV APP_HOME="/opt/mon_app"
11
12 WORKDIR $APP_HOME
13 CMD ["bash"]
```

The terminal shows the build process:

```
● aghi@Exodus:~/docker/img3$ docker build -t deb_aghi .
[+] Building 1.7s (6/6) FINISHED
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 371B
--> [internal] load metadata for docker.io/library/debian:stable
--> [internal] load .dockerignore
--> => transferring context: 2B
--> CACHED [1/2] FROM docker.io/library/debian:stable@sha256:5bcdced7b0c02512895dd702dfc903f3249755de86f72ab86e12b514ca9e4505e
--> [2/2] WORKDIR /opt/mon_app
--> => exporting to image
--> => exporting layers
--> => writing image sha256:3ff702fff43b5067b7160d096cf14131bcc3b27a26ceb4926cee7b806e7eaf
--> => naming to docker.io/library/deb_aghi
○ aghi@Exodus:~/docker/img3$
```

The build log shows the WORKDIR instruction being processed. A red arrow points to the line where it changes the working directory to /opt/mon_app. The terminal then shows the container running and the user switching to root, changing the working directory to /opt/mon_app, and printing the environment variable \$AUTEUR which contains 'Aghi'.

```
○ aghi@Exodus:~/docker/img3$ docker run -it deb_aghi:latest
root@8da6f0cba96c:/opt/mon_app# 
root@8da6f0cba96c:/opt/mon_app# pwd
/opt/mon_app
root@8da6f0cba96c:/opt/mon_app#
root@8da6f0cba96c:/opt/mon_app# echo $AUTEUR
Aghi
root@8da6f0cba96c:/opt/mon_app#
```

L'instruction USER

Définit l'**utilisateur** ou l'**UID** et éventuellement le **groupe d'utilisateurs** ou le **GID** à utiliser pour le reste de l'étape en cours. L'utilisateur spécifié est utilisé pour les instructions **RUN** et, au moment de l'exécution de l'image de conteneur.

```
USER <user>[:<group>]
```

ou

```
USER UID[:GID]
```

L'instruction COPY

L'instruction COPY permet d'intégrer des fichiers et ou des dossiers pour les ajouter au système de fichiers de l'image dans le répertoire <dest>.

```
COPY [--chown=<user>:<group>] [--chmod=<perms>] <src>... <dest>
```

ou

```
COPY [--chown=<user>:<group>] [--chmod=<perms>] [<src>, ... <dest>]
```

L'instruction ADD

Fonctionne comme l'instruction **COPY** sauf qu'il permet d'intégrer des URL de fichiers distants et/ou des archives qui seront décompressés à la volée et les ajoute au système de fichiers de l'image dans le répertoire <dest>.

```
ADD [--chown=<user>:<group>] [--chmod=<perms>] [--checksum=<checksum>] <src>... <dest>
```

ou

```
ADD [--chown=<user>:<group>] [--chmod=<perms>] [<src>, ... <dest>]
```

L'instruction ARG

Est la seule instruction qui **peut précéder** l'instruction **FROM**. Elle permet de définir des **variables** qui peuvent être transmises au moment de la construction de l'image.

```
ARG <name>[=<default value>]
```

L'instruction RUN

L'instruction RUN permet de lancer des commandes aux moments de la construction de l'image.

```
RUN apk update
```

```
RUN apk add nginx
```

```

# syntax=docker/dockerfile:1.7
# escape=\
FROM debian:stable

# L'argument est utilisé uniquement pendant le build (pas après)
ARG USERNAME=aghi
ARG APP_HOME=/opt/aghi

ENV APP_HOME=${APP_HOME}

COPY requirements.txt ${APP_HOME}/

RUN apt update && apt install git -y

# Création d'un utilisateur non-root
RUN useradd -m -d /home/${USERNAME} ${USERNAME}

# Définir le répertoire de travail
WORKDIR ${APP_HOME}

# Changer le propriétaire des fichiers
RUN chown -R ${USERNAME}: ${APP_HOME}

# Exécution en tant qu'utilisateur non-root
USER ${USERNAME}

CMD ["bash"]

```

```

$ aghi@Exodus:~/docker/img4$ docker run -it deb_exp:latest
aghi@ba075749ee1d:/opt/aghi$
aghi@ba075749ee1d:/opt/aghi$ aghi@ba075749ee1d:/opt/aghi$ pwd
/opt/aghi
aghi@ba075749ee1d:/opt/aghi$ id
uid=1000(aghi) gid=1000(aghi) groups=1000(aghi)
aghi@ba075749ee1d:/opt/aghi$ aghi@ba075749ee1d:/opt/aghi$ sudo apt update
bash: sudo: command not found
aghi@ba075749ee1d:/opt/aghi$ aghi@ba075749ee1d:/opt/aghi$ echo $APP_HOME
/opt/aghi
aghi@ba075749ee1d:/opt/aghi$ git -v
git version 2.47.3
aghi@ba075749ee1d:/opt/aghi$ 

```

L'instruction VOLUME

Crée un point de montage avec le nom spécifié et le marque comme contenant des **volumes** montés en externe à partir d'un hôte natif ou d'autres conteneurs.

Permet aussi de partager des données entre plusieurs conteneurs.

```
VOLUME ["/data"]
```

L'instruction EXPOSE

Informe le moteur de conteneur que le conteneur écoute sur les ports réseau spécifiés au moment de l'exécution. Vous pouvez spécifier le protocole TCP ou UDP, TCP étant la valeur par défaut.

```
EXPOSE <port> [<port>/<protocol>...]
```

Les instructions CMD et ENTRYPOINT

Ces deux instructions permettent de définir les commandes qui seront exécutées au moment du lancement de l'image de conteneur.

La principale différence entre **CMD** et **ENTRYPOINT** est que les commandes fournies par **CMD** peuvent être remplacées, alors que celles fournies par **ENTRYPOINT** ne le peuvent pas à moins de les forcer avec le bon argument dans la CLI de votre moteur de conteneur.

```
CMD ["executable", "param1", "param2"]
```

```
ENTRYPOINT ["executable", "param1", "param2"]
```

```
# syntax=docker/dockerfile:1.7
# escape=\
FROM debian:stable
RUN apt update && apt install -y --force-yes apache2
EXPOSE 80 443
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

```
aghil@Exodus:~/docker/img_web$ docker run -it deb web:latest
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 10.10.0.2. Set the 'ServerName' directive global
ly to suppress this message
```



L'instruction SHELL

Permet de définir le **shell** à utiliser pour exécuter les commandes dans le conteneur. Par défaut, Docker utilise **/bin/sh -c**, mais on peut le remplacer par un autre **shell** si nécessaire.

```
SHELL ["/bin/bash", "-c"]
```

L'instruction STOP SIGNAL

Permet de définir le signal qui sera envoyé au conteneur lorsqu'il sera arrêté. Cela peut être utile pour personnaliser le comportement d'arrêt du conteneur en fonction de l'application qu'il exécute.

```
STOP SIGNAL SIGKILL
```

L'instruction HEALTHCHECK

Permet de définir une commande qui sera exécutée périodiquement pour **vérifier** si le conteneur est en bonne santé. Si la commande échoue, le conteneur sera marqué comme "**non sain**". Cela peut être utile pour surveiller l'état de l'application à l'intérieur du conteneur et prendre des mesures si nécessaire.

```
HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
CMD curl -f http://localhost/ || exit 1
```

Bonnes Pratiques pour Écrire un Dockerfile de Qualité

- **Écrire des Dockerfiles Simples et Lisibles**
- **Minimiser la Taille de l'Image**
- **Utiliser des `.dockerignore`**
- **Séparer les Phases de Construction et d'Exécution**
- **Gérer les Variables d'Environnement avec Précaution**
- **Utiliser des Outils de Contrôle**
 - **Hadolint** : est un outil qui analyse vos Dockerfiles et vous donne des recommandations pour améliorer leur qualité. Il vérifie si vous suivez les bonnes pratiques et vous alerte sur les erreurs courantes.
 - **Dive** : est un outil d'analyse qui vous permet de plonger dans les couches de votre **image Docker** pour comprendre comment elle est construite. Il vous aide à identifier les couches inutiles et à optimiser la taille de votre image.
 - **Dockle** : est un outil d'analyse de sécurité pour les **images Docker**. Il vérifie les meilleures pratiques de sécurité et vous alerte sur les mauvaises pratiques que vous pourriez avoir dans vos images. Il est particulièrement utile pour s'assurer que vos images sont sécurisées et conformes aux normes de sécurité des **benchmarks CIS Docker**.