
Les commandes de base Docker

1. Comprendre les tags

Un tag est le moyen qu'utilise Docker pour désigner une version spécifique d'une image au sein d'un dépôt (repository).

Le format complet pour identifier une image est

[Dépôt/]nom_image[:tag]

Exemple : nginx:1.27.0

- Dépôt : Implicitement docker.io (le Docker Hub).
- Nom de l'image : nginx (le serveur web).
- Tag : 1.27.0 (la version).

TAG	Signification	Remarques
slim	Version allégée , basée sur Debian ou Ubuntu.	Plus légère que la version standard, plus lourde qu'Alpine.
alpine	Basée sur la distribution Linux Alpine, extrêmement légère	Idéal pour la production où la taille est critique.
latest	Tag par défaut , utilisé si aucun tag n'est fourni	Ne signifie PAS forcément "la dernière version stable".

2. Gestion des images

Télécharger une image Docker depuis un registre (comme Docker Hub) vers votre machine locale s'effectue avec la commande **docker pull**

Syntaxe de base:

docker pull IMAGE[:TAG]

Exemples:

```
docker pull ubuntu
docker pull ubuntu:22.04
docker pull nginx:alpine
docker pull python:3.11-slim
docker pull node:18-alpine
```

Vérifier les images téléchargées:

`docker images`
ou `docker image ls`

```
PS C:\Users\medkh> docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.11-slim	040af88f5bce	28 hours ago	124MB
postgres	15-alpine	6df5377b1f74	4 days ago	273MB
nginx	alpine	d4918ca78576	3 weeks ago	52.8MB
ubuntu	latest	c3a134f2ace4	4 weeks ago	78.1MB
ubuntu	22.04	9fa3e2b5204f	5 weeks ago	77.9MB

REPOSITORY : Nom de l'image

TAG : Version/variante

IMAGE ID : Identifiant unique

CREATED : Date de création

SIZE : Taille

Inspecter une image:

Inspecter une image Docker est l'acte de récupérer des informations de bas niveau très détaillées sur la structure, les métadonnées et l'historique de cette image.

Ceci est réalisé avec la commande:

`docker inspect <nom_image>`

Historique des layers:

La commande `docker history` vous permet de visualiser l'historique d'une image Docker spécifique. Elle révèle comment l'image a été construite, couche par couche.

L'historique vous permet de voir **exactement** ce que contient une image (surtout les images téléchargées) sans avoir accès au Dockerfile original

Exemple: `docker history nginx:alpine`

Supprimer des images:

Ces commandes sont cruciales pour gérer l'espace disque en supprimant les images inutilisées.

Supprimer une image spécifique:

`docker rmi <image>`

Exemple:

Supprimer plusieurs images

`docker rmi ubuntu:22.04 nginx:alpine`

Supprimer toutes les images qui n'ont plus de tag associé (dangling):

`docker image prune`

Supprimer toutes les images inutilisées:

`docker image prune -a`

Exemple de Scénario d'image dangling

1. Vous avez une image nommée `myapp:latest` sur votre machine.
2. Vous modifiez votre `Dockerfile` et vous exécutez `docker build -t myapp:latest .`
3. La nouvelle image remplace l'ancienne image `myapp:latest`. L'ancienne image, qui n'est plus référencée par aucun tag, devient **"dangling"**.
4. La commande `docker image prune` va **supprimer cette ancienne image "dangling"**.

3. Gestion des conteneurs

Pour lancer un conteneur Docker, la commande principale à utiliser est **docker run**. Cette commande télécharge l'image (si elle n'est pas déjà présente localement), crée le conteneur et le démarre en une seule étape.

La syntaxe:

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Mode Détaché:

Pour lancer un service en arrière-plan:

```
docker run -d -p 8080:80 --name mon-serveur-web nginx
```

Détail des "drapeaux" (flags) :

- `-d` (Detached) : Lance le conteneur en arrière-plan. Sans ça, votre terminal serait bloqué.
- `-p 8080:80` (Port) : On connecte le port **8080** de votre ordinateur au port **80** du conteneur (le port par défaut du web).
- `--name mon-serveur` : On donne un petit nom au conteneur pour ne pas avoir à retenir son ID compliqué.

Vous pouvez maintenant accéder à <http://localhost:8080>

Lancer un terminal interactif:

```
docker run -it ubuntu bash
```

Explication des options

- `-i` ou `--interactive` : Garde STDIN ouvert
- `-t` ou `--tty` : Alloue un pseudo-terminal
- `ubuntu` : Image à utiliser
- `bash` : Commande à exécuter

À l'intérieur du conteneur exécutez les commandes suivantes:

```
ls /  
cat /etc/os-release  
whoami  
hostname  
ps aux
```

```

PS C:\Users\medkh> docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
20043066d3d5: Pull complete
Digest: sha256:c35e29c9450151419d9448b0fd75374fec4fff364a27f176fb458d472dfc9e54
Status: Downloaded newer image for ubuntu:latest
root@b22c162829de:/# ls /
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@b22c162829de:/# cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.3 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
root@b22c162829de:/# whoami
root
root@b22c162829de:/# hostname
b22c162829de
root@b22c162829de:/# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4588   3840 pts/0    Ss   12:59   0:00 bash
root          14  0.0  0.0   7888   3968 pts/0    R+   13:01   0:00 ps aux
root@b22c162829de:/#

```

- Lorsque l'on quitte bash avec la commande exit, le conteneur se termine.

En résumé : on passe les deux options -i et -t à docker run lorsque l'on souhaite se connecter au terminal du conteneur dont le processus est un *shell* comme *bash*.

Affichage de l'État et des Informations:

commande	Description
docker ps	Liste les conteneurs en cours d'exécution.
docker ps -a	Liste tous les conteneurs, y compris ceux qui sont arrêtés.
docker logs <id_ou_nom>	Affiche la sortie (logs) du conteneur. Ajoutez -f pour suivre les logs en temps réel.

Contrôle du Cycle de Vie

commande	Description
docker start <id_ou_nom>	Redémarre un conteneur qui est à l'état arrêté.
docker stop <id_ou_nom>	Arrête proprement un conteneur en cours d'exécution.
docker restart <id_ou_nom>	Arrête et redémarre un conteneur.
docker kill <id_ou_nom>	Force l'arrêt immédiat du conteneur (arrêt brutal).
docker pause <id_ou_nom>	Met en pause tous les processus du conteneur.
docker unpause <id_ou_nom>	Reprend l'exécution d'un conteneur mis en pause.

Suppression (Nettoyage):

commande	Description
<code>docker rm <id_ou_nom></code>	Supprime définitivement un conteneur (doit être arrêté au préalable).
<code>docker rm -f <id_ou_nom></code>	Force la suppression d'un conteneur, même s'il est en cours d'exécution.
<code>docker container prune</code>	Supprime tous les conteneurs qui sont arrêtés.

Autre Commandes:

commande	Description
<code>docker exec -it <id_ou_nom><commande></code>	Exécute une commande dans un conteneur en cours d'exécution. Exemple: <code>docker exec -it mon_app /bin/bash</code> (pour ouvrir un shell)
<code>docker run -rm <id_ou_nom></code>	Supprime automatiquement le conteneur lorsqu'il s'arrête. Exemple: <code>docker run -it --rm ubuntu /bin/bash</code>
<code>docker stats</code>	Affiche l'utilisation en temps réel des ressources (CPU, RAM, réseau) des conteneurs en cours d'exécution.
<code>docker system prune</code>	Nettoie le système : supprime tous les conteneurs arrêtés, les réseaux inutilisés, les images orphelines et le cache de construction. C'est essentiel pour libérer de l'espace disque.