# Mémento du Docker

**Attention :**

**1)** Penser à utiliser « sudo » si nécessaire.

## Définitions

• **Image Docker :**

— système de fichiers sans les processus,

— contient tout ce qu’on y a installé, mais est dans un état inerte.

• **Conteneur :**

— exécution d’une image,

— processus ou ensemble de processus isolés du reste du système,

— permet de faire de la virtualisation légère : il ne virtualise pas les ressources mais créé une isolation de processus,

— partage les ressources CPU, RAM et disque avec l’ensemble du système mais ne réserve pas la quantité de CPU, RAM et disque attribuée auprès du système hôte (à la différence des machines virtuelles).

• **Volume :** permet de :

— conserver des données quand un conteneur est supprimé,

— partager des fichiers/dossiers entre conteneurs ou entre hôte et conteneurs.

Remarques :

— Un conteneur n’a pas besoin d’être démarré pour partager son volume.

— Un changement dans le volume ne sera pas conservé lorsque l’image sera mise à jour.

— Les volumes sont persistants jusqu’à ce qu’aucun conteneur ne les utilise.

• **Couche de réseau :** permet aux conteneurs Dockers de communiquer entre eux et avec le monde extérieur.

• **Trois réseaux par défaut :**

— **bridge :** réseau de communication entre conteneurs, réseau par défaut si on n’en choisi pas lors de la création d’un conteneur,

— **host :** réseau qui reprend les informations de la physique, extension de la physique,

— **null**: isole le conteneur.

• **Dockerfile :**

— fichier qui permet de construire une image Docker adaptée aux besoins une fois compilé,

— contient toutes les instructions pour créer une image comme des métadonnées ou les commandes à exécuter pour installer un logiciel,

— peut être inclus dans d’autres Dockerfiles et être à la base de plusieurs images différentes.

• **Docker compose :**

— outil de gestion de package Docker,

— permet de définir plusieurs conteneurs en un seul fichier dans un fichier de configuration écrit en yaml.

— déclenche le tout en une seule commande.

• **Registry :** registre d’images, sorte de google drive où l’on stocke les images. Ex :: celui d’Amazon, celui de GitLab, etc.

## Commandes de base

### Installation de Docker

• **Installation de yum** (sur Centos)**:** yum install -y yum-utils

• Suivre les instructions (Centos) : https://docs.docker.com/engine/install/centos/

• Suivre les instructions (Debian) : https://docs.docker.com/engine/install/debian/

• **Vérifier la version de Docker :** docker -v

• **Obtenir plus d’informations sur la version de Docker :** docker version

### Démarrage de Docker

• **Démarrer Docker :** systemctl start docker

• **Démarre Docker à chaque démarrage :** systemctl enable docker

• **Vérifier si Docker est démarré :** systemctl status docker

### Téléchargement d’images

• **Télécharger une image avec une version** (tag) **spécifique :** docker pull image:version (« :version » est optionnel, par défaut, télécharge la dernière version « latest »)

• **Télécharger une image qui se trouve sur un compte utilisateur :** docker pull utilisateur/image

### Création des conteneurs à partir d’une image

• **Télécharger une image et créer un conteneur :** docker run -dit image /bin/bash

• **Créer un conteneur avec port :** docker run -dit -p port\_conteneur:port\_choisi identifiant\_image (on choisit le port\_conteneur que l’on veut)

Ex : docker run -dit -p 1234:80 cd6d8154f1e1, docker run -dit -p 10000-10010:80 guicotine/apachegui:v1 apachectl -D FOREGROUND

• **Créer un conteneur et lui donner un nom :** docker run -dit --name surnom image

• **Créer un conteneur avec le service apache2 actif :** docker run -dit -p port\_conteneur:port\_choisi identifiant\_image apachectl -D FOREGROUND

### Utilisation des conteneurs et des images

• **Démarrer un conteneur existant :** docker start identifiant\_conteneur

• **Arrêter un conteneur existant :** docker stop identifiant\_conteneur

• **Arrêter un conteneur existant et le redémarrer :** docker restart identifiant\_conteneur

• **Arrêter un conteneur quand on est dedans en ligne de commandes :** exit

• **Accéder à un conteneur en exécution** (en ligne de commandes) **:** docker attach identifiant\_conteneur (on est dans la seule instance de bash du conteneur, si on utilise « exit », le conteneur se fermera)

• **Accéder à un conteneur en exécution** (en ligne de commandes) **:** docker exec -it identifiant\_conteneur /bin/bash (on a ouvert une deuxième instance de bash donc si on utilise « exit », le conteneur ne se fermera pas)

• **Supprimer tous les conteneurs de force :** docker rm -f $(docker ps -aq)

• **Supprimer une image de force :** docker rmi -f image

• **Supprimer toutes les images de force :** docker rmi -f $(docker images -q)

• **Créer une nouvelle image à partir du container :** docker commit identifiant\_conteneur nom:version (avec comme nom nom\_utilisateur/nom\_conteneur)

• **Déployer un conteneur :** docker run nom:version service apache2 start (avec comme nom nom\_utilisateur/nom\_conteneur)

### Informations sur les conteneurs et les images

• **Voir la liste les images présentes sur le serveur :** docker images

• **Voir la liste des conteneurs en cours de fonctionnement :** docker ps

• **Voir la liste des conteneurs arrêtés :** docker ps -a

• **Voir le dernier conteneur exécuté :** docker ps -l

• **Obtenir des informations sur le réseau d’un conteneur :** docker network inspect identifiant\_conteneur

• **Obtenir des informations sur un conteneur :** docker inspect conteneur

• **Obtenir des informations sur une image :** docker inspect image

### Apache

• **Installer Apache :** apt install apache2

• **Vérifier le statut d’Apache :** service apache2 status

• **Démarrer le service apache2 :** service apache2 start

### Volume

• **Créer un volume nommé :** docker volume create --name volume

• **Monter un volume nommé :** docker run -dit -v volume:/chemin\_ciblé\_sur\_conteneur conteneur (si le volume possède des données, il écrase celles du conteneur, s’il n’en a pas, il récupère celle du conteneur s’il en possède)

Ex : docker run -dit -p 10000-10010:80 -v vol\_nextcloud:/var/www/html nextcloud

• **Monter un volume depuis le système hôte :** docker run -dit -v $PWD/volume:/ chemin\_ciblé\_sur\_conteneur conteneur (le volume prendra toujours le répertoire physique quel que soit son état : il écrase les données du conteneur)

Ex : docker run -it v /html1/:/var/www/html/ apache,

docker run -dit -p 10005:80 --name mount1 -v $PWD/html/:/usr/local/apache2/htdocs httpd

Remarque : $PWD prend automatiquement le chemin absolu du dossier courant.

• **Voir la liste les volumes :** docker volume ls

• **Récolter des informations sur un volume :** docker volume inspect volume (l’emplacement exact du volume, où l’on peut créer par exemple « index.html », sera dans « mountpoint »)

• **Supprimer un(des) volume(s) de force :** docker volume rm -f volume1 volume2

• **Supprimer tous les volumes qui ne sont pas en cours d’exploitation :** docker volume prune

• **Ajouter un fichier index.html à un volume :** sudo vi chemin\_vu\_avec\_inspect/index.html

### Réseau

• **Créer un réseau avec un type** (bridge, host, null)**:** docker network create --driver type réseau

• **Voir la liste des réseaux :** docker network ls

• **Voir les machines présentes sur un réseau :** docker network inspect réseau

• **Connecter un conteneur à un réseau :** docker run -dit --name conteneur --network réseau image

• **Supprimer tous les réseaux qui ne sont pas en cours d’exploitation :** docker network prune

### Dockerfile

• **Créer un Dockerfile :** nano Dockerfile

• **Définir l’image sur laquelle les instructions vont se baser** (première instruction du Dockerfile)**:** FROM image

• **Indiquer la personne qui maintient le Dockerfile :** MAINTAINER personne

• **Exécuter une(des) commande(s) sur l’image courante et faire un commit du résultat :** RUN commande1 && commande2 && commande3

• **Définir la commande exécutée au lancement d’un container :** CMD

• **Indiquer la commande par défaut qui sera exécutée au démarrage du conteneur :** ENTRYPOINT

Remarques :

— Il ne peut y avoir qu’un CMD et qu’un ENTRYPOINT dans un Dockerfile.

— Pour CMD et ENTRYPOINT, toujours préférer la syntaxe en tableau.

Ex : CMD "/bin/service -d" → CMD ["/bin/service", "-d"]

— CMD simule un « docker run » et peut être surchargé par cette commande alors qu’ENTRYPOINT simule un « docker run » et ne peut pas être surchargé par cette commande.

— ENTRYPOINT, c’est le binaire de lancement et CMD, les arguments du binaire. Ex :

ENTRYPOINT [ "python" ]

CMD [ "/student\_age.py" ]

• **Définir le(s) port(s) qui devra(ont) être exposé(s) vers l’extérieur :** EXPOSE port

• **Affecter une valeur à une variable d’environnement :** ENV

• **Copier de nouveaux fichiers, dossiers locaux ou des fichiers distants (via URL) et les ajouter au système de fichier du conteneur selon le chemin indiqué**(utilisé uniquement pour les téléchargements depuis le net et la gestion des archives de l’hôte vers le conteneur) **:** ADD

• **Copier de nouveaux fichiers, dossiers locaux et les ajouter dans le système de fichier du conteneur selon le chemin indiqué** (purement dédié pour copier des fichiers locaux lors de la construction d’une image, utilisé quand ADD ne l’est pas)**:** COPY chemin\_d’accès\_local chemin\_sur\_l’image

Remarque : si on utilise « . », en local c’est là où on est et sur l’image c’est le WORKDIR qu’on a défini. Ex : COPY . .

• **Ajouter un(des) volume(s) aux conteneurs crées à partir d’une image :** VOLUME /volume

• **Désigner l’utilisateur qui exécutera (RUN) une instruction ou exécutera un ENTRYPOINT :** USER utilisateur

• **Assigner le répertoire de travail pour l’exécution des commandes sur l’ENTRYPOINT et/ou l’instruction CMD lorsque le conteneur est lancé à partir de l’image :** WORKDIR chemin

• **Ajouter des déclencheurs aux images** (les déclencheurs seront exécutés lorsque l’image sera utilisée comme base pour quelqu’un d’autre, cela permet de préparer certains éléments de terrain qui pourraient manquer comme certains paquets)**:** ONBUILD.

Ex : ONBUILD RUN/ADD commande

• **Préparer une variable qui va agir au sein du Dockerfile :** ARG FLAG="variable"

• **Utiliser la variable :** ${FLAG}

• **Créer l’image à partir du Dockerfile :** docker build -t nom\_image -f chemin\_Dockerfile .

### Docker compose

• **Installer docker compose :**

sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose• **Voir la version de docker compose** (permet de vérifier si l’installation a fonctionné) **:** docker compose -v

 • **Créer un fichier docker-compose.yml :** nano docker-compose.yml

Remarque : surtout ne pas utiliser de tabulations dans un fichier docker-compose car ça génère des erreurs.

Remarque : Le fichier docker-compose.yml ressemblera à :

Une image contenant texte

Description générée automatiquement

• **Démarrer l’ensemble des conteneurs en arrière-plan :** docker compose up -d

• **Démarrer l’ensemble des conteneurs en les buildant et les recréant :** docker compose up -d -build --force-recreate

• **Voir le statut de l’ensemble de la stack :** docker-compose ps

• **Afficher les logs de la stack :** docker-compose logs -f --tail 5

• **Arrêter l’ensemble des services d’une stack :** docker-compose stop

• **Couper les conteneurs :** docker-compose down

• **Valider la syntaxe du fichier docker-compose.yml :** docker-compose config

• **Afficher des informations sur nos conteneurs :** docker-compose logs

• **Afficher l’activité processus des conteneurs :** docker-compose top

• **Exécuter le Dockerfile :** docker run -d --name nom\_conteneur -p 8000:5000 nom\_image

Ex : docker run -d --name api -p 8000:5000 -v $PWD/student\_age.json:/data/student\_age.json student-list:v1

#### Version

• **Définir la version de Docker Compose utilisée :** version: "3.9"

#### Services

• **Définir les conteneurs :**

services:

conteneur1:

…

conteneur2:

…

##### Image

• **Définir l’image utilisée pour le conteneur :** image: image:version

##### Build

build:

• **Ajouter le répertoire où on se trouve :**

context: .

• **Ajouter un nom au conteneur et une version :**

tags:

- "nom:version"

• **Rendre un conteneur dépendant d’un autre** (pour qu’un service soit déployé après un autre et qu’il soit arrêté avant l’autre)**:**

depends\_on :

- conteneur

• **Limiter le pool de ressources d’un conteneur :**

deploy:

resources:

limits:

cpus: '0.50'

memory: 50M

reservations:

cpus: '0.25'

memory: 20M

##### Environment

• **Ajouter des variables d’environnement qui accompagneront le déploiement du conteneur :**

environment:

- USER: user

- PASSWORD: password

Remarque : il est possible d’employer un fichier de configuration à charger (.env est chargé par défaut) :

Une image contenant table

Description générée automatiquement

##### Ports

• **Définir un port :**

ports:

- "port\_extérieur:port\_intérieur/protocole" (l’option protocole est optionnelle)

##### Volumes

• **Définir un volume :**

volumes:

- nom\_volume:/chemin\_répertoire

Remarque : il faut déclarer les volumes configurés dans un service à la fin du docker-compose.yml

##### Networks

• **Attribuer des réseaux à un conteneur :**

networks:

- réseau01:

ipv4\_address: 172.18.0.123

ipv6\_adress: 2001:3984:3989:10

- réseau02

Remarque : il faut déclarer les réseaux configurés dans un service à la fin du docker-compose.yml

##### Restart

• **Gérer le redémarrage d’un service** (n’en choisir qu’un)**:**

restart: no / always / on-failure / unless-stopped

#### Volumes

• **Définir les volumes :**

volumes:

volume01:

external: true (indique que le volume existe déjà)

volume02: (on n’indique pas « external:true » alors un volume sera créé en se servant du nom du répertoire de projet)

#### Networks

• **Définir les réseaux :**

networks:

réseau01:

external: true (indique que le réseau existe déjà)

réseau02: (on n’indique pas « external:true » alors un réseau sera créé)

• **Définir un réseau par défaut pour l’ensemble des conteneurs issus de ce projet qui existe déjà sur notre hôte :**

networks:

default:

name: réseau

external: true

### Autre

• **Vérifier l’ip du serveur :** ip -a

• **Voir l’espace disque :** df

• **Se mettre dans le groupe « docker » pour pouvoir utiliser les commandes sans « sudo » :** sudo usermod -aG docker utilisateur

Remarque : il faut quitter le serveur ou sortir de l’utilisateur en cours et y revenir pour que le changement soit effectif.

• **Retrouver un élément :** find . -name élément.extension