Docker

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

1. Installation

Installation des paquet nécessaire pour permettre à apt d'utiliser des dépots par https:

```
# apt install ca-certificates curl gnupg lsb-release
```

On ajoute la clé GPG de docker:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

On ajoute les dépôts:

```
$ echo \
  "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

On met à jour la liste des paquets et on install docker:

```
# apt update
# apt install docker-ce docker-ce-cli containerd.io
```

2. Commandes

Télécharger une image:

```
# docker pull <Image_Name>[:Version]
```

Liste des images disponibles : https://hub.docker.com/search

Création d'un conteneur à partir d'une image:

```
# docker run [options] <Image_Name>[:Version]
```

Options	Description
-t	Allouer un pseudo TTY (terminal virtuel)
-i	Garder un STDIN ouvert (l'entrée standard plus précisément l'entrée clavier)
-d	Exécuter le conteneur en arrière-plan et afficher l'ID du conteneur
--name	Nommer le conteneur
--expose	Exposer un port ou une plage de ports (on demande au firewall du conteneur de nous ouvrir un port ou une plage de port)
-p	Mapper un port déjà exposé, (permet de faire une redirection de port)

Exemple de création d'un conteneur avec un réseau bridge:

```
# docker run -dit --name <NomConteneur> --network bridge -p 8080:80
<Image_Name>
```

Création d'une interface réseau:

```
# docker network create --driver <Type> <NomInterface>
```

Connecter un conteneur à un réseau docker:

```
# docker network connect <NomReseau> <NomConteneur>
```

Lister les conteneurs:

```
# docker ps
```

Gestion des conteneurs:

```
# docker start <NomConteneur>
# docker stop <NomConteneur>
# docker pause <NomConteneur>
# docker kill <NomConteneur>
# docker system prune      ##Nettoyer le système
```

3. Dockerfile

Le Dockerfile permet de créer des images en ajoutant des paquets à des images de base.

Liste des commandes:

Syntaxe	Description
`FROM <Image_Name>[:Version]`	Définit l'image de base qui sera utilisée par les instructions suivantes.
`ARG <NOM>="<contenu>"`	Variables temporaires qu'on peut utiliser dans un Dockerfile.
`ARG <NOM>="<contenu>"`	Variables d'environnements utilisables dans votre Dockerfile et conteneur.
`RUN <commande(s)>`	Exécute des commandes Linux ou Windows lors de la création de l'image.
`COPY <src> <dst>`	Permet de copier des fichiers depuis notre machine locale vers le conteneur Docker.
`ADD <src> <dst>`	Même chose que COPY mais prend en charge des liens ou des archives (si le format est reconnu, alors il sera décompressé à la volée).
`WORKDIR <Chemin>`	Définit le répertoire de travail qui sera utilisé pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.
`EXPOSE <NumPort>`	Ouvre un port.
`VOLUMES <CheminIntern>`	Crée un point de montage qui permettra de persister les données.
`ENTRYPOINT <commande(s)>`	Point d'entrée de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur.
`CMD <commande(s)>`	Spécifie les arguments qui seront envoyés au ENTRYPOINT, (on peut aussi l'utiliser pour lancer des commandes par défaut lors du démarrage d'un conteneur).

ex:

```
FROM debian:latest

LABEL version="1.0" maintainer="test"

RUN apt update -y && \
      apt install -y -q apache2

RUN apt install -y -q php && \
      apt autoclean -y

EXPOSE 80

WORKDIR /var/www/html

VOLUME /var/www/html

ENTRYPOINT apache2ctl -D FOREGROUND
```

Création de l’image:

```
# docker build -t <NomImage> <cheminDockerfile>
```