Docker

Étape 1 – L’installation de Docker

Pour cette fiche de procédure, on se connectera en SSH à une machine virtuelle.
On part du principe que la machine virtuelle est déjà prête.

```
ssh etu1@172.20.xx.x
# (Faire un `ip -a` pour connaître l’IP de la VM)
su -
```

Ajout du dépôt APT de Docker :

```
apt-get update
apt-get install ca-certificates curl

install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo \"$VERSION_CODENAME\") stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null

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

Tester l'installation :

```
sudo docker run hello-world
```

Étape 2 – Découverte des commandes Docker

Récupérer une image :

```
docker pull <nom-image>
```

Lister les images téléchargées :

```
docker images
```

Créer et lancer un conteneur :

```
docker run -d -p 8080:80 <nom-image>
docker ps -a  # ou docker ps
```

Détails :

- `-d` : exécution en arrière-plan
- `-p` : liaison des ports

Conteneur accessible sur : `http://172.20.xx.x:8080`

Accéder au conteneur :

```
docker exec -it <id-conteneur> bash
```

Arrêter, redémarrer, supprimer :

```
docker stop <id-conteneur>
docker start <id-conteneur>
docker rm <id-conteneur>
docker rmi <nom-image>
```

Étape 3 – Créer ses propres images Docker

Exemple 1 – Hello World (Debian)

Créer un fichier Dockerfile :

```
touch Dockerfile
nano Dockerfile
```

Contenu :

```
FROM debian:latest
CMD echo "Hello World!"
```

Construire l'image :

```
docker build -t my-hello-world .
```

Exemple 2 – Image NGINX personnalisée

Créer un fichier HTML :

```
echo "<h1>Hello from NGINX!</h1>" > index.html
```

Modifier le Dockerfile :

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
```

Construire l'image :

```
docker build -t my-nginx .
```

Créer un conteneur :

```
docker run -d -p 8080:80 my-nginx
```

Accessible sur : `http://172.20.xx.x:8080`

Étape 4 – Automatiser avec Docker Compose

Exemple simple avec NGINX

Créer un fichier `docker-compose.yml` :

```
services:
  nginx:
    image: nginx:latest
    ports:
      - "8080:80"
```

Commandes utiles :

```
docker-compose pull
docker-compose logs -f
sudo docker-compose up -d
```

Exemple WordPress + PostgreSQL

```
services:
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: Nom/Ip du serveur
      WORDPRESS_DB_USER: Utilisateur
      WORDPRESS_DB_PASSWORD: Mot de passe
      WORDPRESS_DB_NAME: Nom de la base
    volumes:
      - wordpress_data:/var/www/html

  db:
    image: postgres:latest
    environment:
      POSTGRES_DB: Nom/Ip du serveur
      POSTGRES_USER: Utilisateur
      POSTGRES_PASSWORD: Mot de passe
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  wordpress_data:
  db_data:
```

Résumé des commandes

Commande	Description
`docker run <image>`	Lance un conteneur à partir d'une image
`docker build -t <nom> .`	Construit une image depuis un Dockerfile
`docker pull <image>`	Télécharge une image
`docker images`	Liste les images locales
`docker ps`	Liste les conteneurs actifs
`docker ps -a`	Liste tous les conteneurs
`docker stop <id>`	Arrête un conteneur
`docker rm <id>`	Supprime un conteneur
`docker rmi <image>`	Supprime une image
`docker exec -it <id> bash`	Accès shell dans un conteneur
`docker-compose up`	Lance les services définis
`docker-compose down`	Supprime les ressources lancées
`docker-compose pull`	Télécharge les images du fichier Compose
`docker-compose logs -f`	Affiche les logs en temps réel
`docker-compose ps`	Affiche l'état des services