Fiche de Procédure Docker

L’installation de Docker

On choisira de se connecter en ssh à la machine virtuelle

```
allavenavr@A110-23-LINUX:~$ ssh etu1@172.20.107.3
```

On suit ensuite la documentation de docker pour mettre en place le 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
```

Une fois fait on installe docker :

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

Fiche de Procédure Docker

Pour voir si tout c'est bien passé on exécute cette commande :

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

```
root@debian-console:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Fiche de Procédure Docker

Prise en main de Docker

Récupération d'une image / verification images

Docker image :

Command	Description
`docker image history`	Show the history of an image
`docker image import`	Import the contents from a tarball to create a filesystem image
`docker image inspect`	Display detailed information on one or more images
`docker image load`	Load an image from a tar archive or STDIN
`docker image prune`	Remove unused images
`docker image rm`	Remove one or more images
`docker image save`	Save one or more images to a tar archive (streamed to STDOUT by default)
`docker image tag`	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
`docker image ls`	List images
`docker image pull`	Download an image from a registry
`docker image push`	Upload an image to a registry

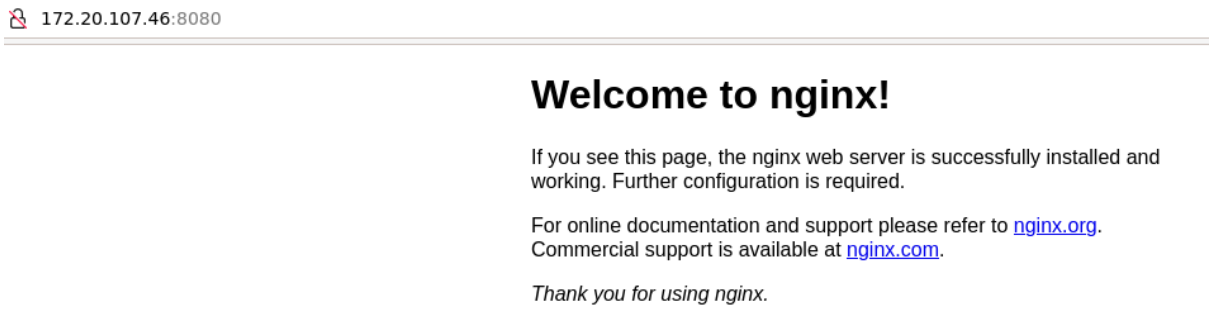
Fiche de Procédure Docker

Créer et lancer un conteneur à partir d'une image

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

```
-d → pour démarrer en arrière plan
-p → pour les ports
nginx → l'image
```

En cherchant sur mon navigateur l'IP de la machine et le port on peut voir que ça fonctionne



Lister les conteneurs

```
docker ps -a
```

Le -a affiche tous les conteneurs (par défaut seuls les conteneurs actifs sont affichés)
Un conteneur démarré aura le statut Up
Un conteneur stoppé aura le statut Exited(0)

Exécuter des commandes dans le conteneur

```
docker exec -it <container_id> bash
```

Fiche de Procédure Docker

Après avoir exécuté cette commande on peut exécuter des commandes bash

Arrêt du conteneur

```
docker stop <container_id>
```

Redémarrer le conteneur

```
docker start <container_id>
```

Suppression d’un conteneur

```
docker rm <container_id>
```

Suppression d’une image

```
docker rmi <image_name>
```

Création de nos propres images Docker

Création d’une image perso

créer un fichier Dockerfile et l’éditer

On met d’abord l’image qui sera utilisée comme base

```
FROM <nom de l’image(ex: debian ou nging)>:latest
```

On ajoute ensuite ce que l’image doit faire (exemple “Hello World!”) :

Fiche de Procédure Docker

```
CMD ["echo", "Hello World!"]
```

Construire l’image

```
docker build -t <nom de l’image> .
```

-t Donne un nom à l’image
. Indique que Docker doit utiliser le Dockerfile dans le répertoire courant

Exécuter l’image

```
docker run <nom de l’image>
```

Fiche de Procédure Docker

Personnaliser une image avec le contenu d’un fichier HTML

Créer et modifier un fichier html

```
touch index.html
nano index.html
```

Créer un nouveau Dockerfile pour personnaliser une image (ici nginx)

Utiliser NGINX comme base
```
FROM nginx:latest
```

Copier le fichier HTML personnalisé dans le répertoire de NGINX
```
COPY index.html /usr/share/nginx/html/index.html
```

Construire l’image

```
docker build -t <nom de mon image> .
```

Lancer un conteneur basé sur mon image (cf Page3)

```
docker run -d -p 8080:80 <nom de mon image>
```

Voir le résultat

```
http://<IP de la VM>:8080
```

le contenu du fichier html est affiché !!!

Fiche de Procédure Docker

Automatisation du déploiement des conteneurs

Créer et modifier un fichier docker-compose.yml

```
touch docker-compose.yml
nano docker-compose.yml

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

Récupérer les images sans démarrer les conteneurs

```
docker compose pull
```

Lancer le conteneur au premier plan

```
docker compose up
```

Lancer le conteneur en arrière-plan

```
docker compose up -d
```

Voir les logs en temps réel

```
docker compose logs -f
```

Fiche de Procédure Docker

Stopper le conteneur (arrêt complet)

```
docker compose down
```

Cette fois, au lieu de simplement le stopper, la commande arrête et détruit les conteneurs

Fiche de Procédure Docker

Créer une configuration pour WordPress et PostgreSQL

Créer ou modifier le fichier docker-compose.yml pour WordPress et PostgreSQL :

```
services:
  wordpress:
    image: wordpress:latest
    ports:
    - "8080:80"
    environment:
      WORDPRESS_DB_HOST: Nom/Ip de l'hébergeur
      WORDPRESS_DB_USER: Nom d’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:
```

Fiche de Procédure Docker

Services :

- db : Service pour PostgreSQL. Il utilise l'image officielle `postgres`, et nous définissons le mot de passe pour l'utilisateur `postgres` via une variable d'environnement.
- `wordpress` : Service pour WordPress. Il utilise l'image officielle `wordpress` et se connecte à la base de données PostgreSQL via les variables d'environnement spécifiant l'hôte de la DB et les identifiants de connexion.

Volumes :

- `db_data` : Volume pour persister les données de PostgreSQL.
- `wp_data` : Volume pour persister les fichiers de WordPress (comme les plugins, les thèmes, etc.).

Fiche de Procédure Docker

Résumé des Commandes

Commande	Description
`docker pull <image>`	Télécharge l'image Docker depuis Docker Hub (ex. `docker pull nginx`).
`docker images`	Affiche la liste des images Docker téléchargées sur le système.
`docker run -d -p <port_hôte>:<port_conteneur> <image>`	Lance un conteneur en arrière-plan et mappe un port du conteneur à un port de l'hôte (ex. `docker run -d -p 8080:80 nginx`).
`docker ps`	Liste les conteneurs en cours d'exécution.
`docker ps -a`	Liste tous les conteneurs (en cours et arrêtés).
`docker exec -it <container_id> bash`	Ouvre un shell Bash dans un conteneur en cours d'exécution.
`docker stop <container_id>`	Arrête un conteneur en cours d'exécution.
`docker start <container_id>`	Redémarre un conteneur précédemment arrêté.
`docker rm <container_id>`	Supprime un conteneur arrêté.
`docker rmi <image_id>`	Supprime une image Docker.
`docker build -t <nom_image> .`	Construit une image Docker depuis un Dockerfile.
`docker-compose up`	Lance tous les services définis dans un fichier `docker-compose.yml`.
`docker-compose up -d`	Lance tous les services en arrière-plan.
`docker-compose down`	Arrête tous les services et supprime les conteneurs.

Fiche de Procédure Docker

`docker logs <container_id>`	Affiche les logs du conteneur spécifié.
`docker-compose logs`	Affiche les logs de tous les services gérés par Docker Compose.
`docker-compose exec <service_name> bash`	Ouvre un shell dans le service spécifié géré par Docker Compose.
`docker container prune`	Supprime tous les conteneurs arrêtés.
`docker image prune`	Supprime toutes les images inutilisées.
`docker volume prune`	Supprime tous les volumes inutilisés.