Résumé Virtualisation

# Leçon

## Images et containers

Une **image** inclue tout ce qui est strictement nécessaire à l'exécution d'une application :

* Le code
* L'exécution
* Les variables d'environnement
* Les bibliothèques
* Les fichiers de configuration
* Utilisable pour créer des **containers** (instanciation d’une image)

## Mapping

Pour rendre persistantes les données contenues :

* “Mapper” un dossier de l’hôte dans le conteneur et enregistrer toutes les données que l’on souhaite voir disponible et pérenne dans le dossier mapper dans le conteneur.
* Créer un volume, et stocker les informations dans ce volume

Lorsqu’on héberge un site sans mapping :

* La modification des fichiers sources est fastidieuse
* La sauvegarde des informations du site sera elle aussi fastidieuse

## Dockerfile

Permet de créer sa propre image.

# Commandes

## Containers et images

### Créer/Supprimer

* Docker container run <image> 🡪 le créé à partir d’une image
  + -p *PORTLOCAL:PORTCONTAINER* → précise les ports
  + -d → exécute en asynchrone
  + –*name* <nom> → donne un nom
  + -v *DOSSLOCAL | DOSSCONTAINER →* permet de mapper le dossier serveur dans le dossier hote / volume
  + --name *SERV*
* Docker container rm <image> 🡪 le supprime

### Mettre en pause

* Docker container pause
* Docker container unpause

### Démarrer/Stopper

* Docker container start 🡪 Initialise toutes les données a neuf
* Docker container stop 🡪 supprime le conteneur, ce qu’il contient, y compris les données stockées, fichiers de configuration modifiés directement dans le conteneur

### Afficher

* Docker container ps

### Exécuter une commande

* Docker container exec *container* commande
  + –it 🡪

## Volumes

### Créer

* Docker volume create *volumeName*

## Réseaux

### Créer

* Docker network create *nomReseau*

### Se connecter

* Docker network connect *nomReseau*

## Bases de données

Depuis une session utilisateur dans le container :

### Connexion

* mysql -u NOMUTILISATEUR –p
* quit

### Affichages

* show tables
* show databases

### Manipulations BD

* create NOMDB → créé la base de données
* use NOMDB → utilise la base de données

## Doxyfile

* Docker build <adresse> 🡪 fabrique une image
  + –t <image> 🡪 indique le nom de l’image

## Autre

### Installer nano

apt-get install nano

# TD-TP

## TD1

1. on lance un container nginx **docker run –p 8000:80 –d nginx**

Pour trouver le dossier contenant les fichiers sources on doit ouvrir le fichier de configuration :

1. On ouvre une session de terminal utilisateur **docker exec –t –i webserver /bin/bash**
2. On accède ensuite au fichier de config dans /etc/nginx/conf.d/default.conf :

Une image contenant texte

Description générée automatiquement

Les fichiers source se trouvent donc dans /usr/share/nginx/html

→ Solution : Créer un conteneur tout en mappant un dossier spécifique du conteneur à un dossier du système hôte :

1. Docker run -p 8000:80 -d -v f:\BUT\docker\webserver:/usr/share/nginx/html --name webserver nginx

## TD2

Volumes

1. **docker volume create monVolume**
2. **docker container run -p 8000:80 -d --name serv -v monVolume:/usr/share/nginx/html nginx**
3. On créée un fichier html **notepad index.html**
4. **docker cp index.html serv:/usr/share/nginx/html**
5. On supprime puis relance le docker **docker stop, docker rm puis docker run**

Les fichiers restent les mêmes puisque c’est le même volume

Dockerfile

1. docker build -t image\_a\_moi . 🡪 On créé l’image

## TP1

1. On cherche sur docker hub un container apache et php 7 en tapant “apache php” et en cochant sponsor OSS à gauche.

Une image contenant texte

Description générée automatiquement

1. On regarde les images crées **docker images**
2. On mappe httpd dans le serveur nommé webserver dans le dossier C:\Temp\docker\webserver\. **docker run -p 9002:80 -d -v C:\Temp\docker\webserver\:/usr/local/apache2/htdocs --name webserver httpd**

## TP2

1. On lance un container webdevops/php-apache-dev
2. On trouve dans le fichier de config le dossier des fichiers source : /app
3. On relance le container en le mappant dans C:/Temp : **docker run -p 8080:80 --name container -v C:/Temp:/app webdevops/php-apache-dev**
4. On créé le réseau myworkflow : **docker network create myworkflow**
5. Ensuite on ajoute le container à ce réseau : **docker network connect myworkflow db**
6. on lance un container mariadb (car mysql fonctionne pas bien) en précisant le mdp (c’est obligatoire) et son réseau : **docker run -p 8081:80 -e MYSQL\_ROOT\_PASSWORD=toor --network myworkflow --name db -d mariadb**
7. on se connecte au container avec **docker exec -it db bin/bash**
8. ensuite on se connecte à la base de donnée avec **mysql -u root -p**