Objectifs :

Être capable d’installer et d’utiliser Docker.

Être capable de créer ses propres Dockerfile et Docker Compose.

Matériel à avoir :

Une machine Debian avec au moins 2048 Mo de mémoire vive.

Cours :

## Introduction à Docker et à la virtualisation de conteneurs

Docker est un outil de virtualisation qui permet de créer, déployer et gérer des applications dans des conteneurs logiciels. Les conteneurs sont des unités d'exécution qui emballent le code de l'application avec toutes ses dépendances et configurations nécessaires pour qu'il fonctionne correctement sur n'importe quel environnement.

Contrairement à la virtualisation traditionnelle, qui nécessite une machine virtuelle pour chaque application, Docker utilise une approche différente en utilisant un seul noyau pour exécuter plusieurs conteneurs. Cela permet de gagner en efficacité, en réduisant la taille des images de conteneurs et en accélérant le démarrage et l'arrêt des conteneurs.

Les conteneurs Docker sont isolés les uns des autres et de l'hôte, ce qui signifie qu'ils ne peuvent pas interférer les uns avec les autres ou avec le système hôte. Cela permet de créer des environnements de développement et de production fiables, reproductibles et évolutifs.

Dans ce cours, nous allons apprendre à installer Docker, à utiliser l'interface de ligne de commande Docker, à créer des images de conteneurs, à exécuter et gérer des conteneurs, et à automatiser le processus de construction d'images avec Dockerfile. Nous allons également aborder des sujets tels que le partage de fichiers entre l'hôte et le conteneur, l'utilisation de volumes persistants, la création de réseaux entre les conteneurs, et l'intégration de Docker avec les outils de développement.

Maintenant que nous avons une idée de ce qu'est Docker et de la virtualisation de conteneurs, nous allons passer à l'installation de Docker.

## Installation de Docker

Dans cette section, nous allons apprendre à installer Docker sur Debian.

Tout d'abord, nous devons mettre à jour le système pour nous assurer que nous avons les dernières mises à jour et correctifs de sécurité:

sudo apt-get update

sudo apt-get upgrade

Ensuite, nous allons installer les paquets nécessaires pour construire et exécuter des conteneurs Docker:

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

Une fois que nous avons installé les paquets nécessaires, nous allons ajouter la clé GPG de Docker à notre système:

curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -

Ensuite, nous allons ajouter le dépôt Docker à notre système:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb\_release -cs) stable"

Maintenant que nous avons ajouté le dépôt Docker, nous pouvons installer Docker:

sudo apt-get install docker-ce docker-ce-cli containerd.io

Une fois que Docker est installé, nous pouvons vérifier la version en utilisant la commande suivante:

docker --version

Cela devrait afficher la version de Docker installée sur notre système.

Ensuite, nous allons démarrer le service Docker et l'activer pour qu'il démarre automatiquement lors du démarrage du système:

sudo systemctl start docker

sudo systemctl enable docker

Enfin, nous allons vérifier que Docker est correctement installé et configuré en exécutant un conteneur Docker de test:

sudo docker run hello-world

Cela devrait télécharger et exécuter un conteneur Docker de test, affichant un message de bienvenue.

Félicitations, vous avez installé Docker sur Debian avec succès!

## Utilisation de l'interface de ligne de commande Docker

Dans cette section, nous allons apprendre à utiliser l'interface de ligne de commande Docker (CLI) pour interagir avec Docker et gérer les conteneurs et les images.

Voici une liste de commandes Docker courantes:

\* docker run: Exécute une image de conteneur.

\* docker ps: Affiche les conteneurs actuellement en cours d'exécution.

\* docker stop: Arrête un conteneur en cours d'exécution.

\* docker rm: Supprime un conteneur.

\* docker images: Affiche les images de conteneur disponibles.

\* docker rmi: Supprime une image de conteneur.

\* docker pull: Télécharge une image de conteneur à partir d'un registre.

\* docker exec: Exécute une commande dans un conteneur en cours d'exécution.

Voici quelques exemples d'utilisation de la CLI Docker:

Exécuter un conteneur Nginx:

docker run -d -p8080:80 nginx

Cela exécutera un conteneur Nginx en arrière-plan, en mappant le port HTTP (80) du conteneur sur le port HTTP (8080) de l'hôte.

Afficher les conteneurs en cours d'exécution:

docker ps

Cela affichera une liste des conteneurs actuellement en cours d'exécution, y compris leur ID, leur nom, leur image, leur état, leur port et leur création.

Arrêter un conteneur:

docker stop <container\_id>

Cela arrêtera le conteneur avec l'ID spécifié.

Supprimer un conteneur:

docker rm <container\_id>

Cela supprimera le conteneur avec l'ID spécifié.

Afficher les images de conteneur disponibles:

docker images

Cela affichera une liste des images de conteneur disponibles, y compris leur nom, leur tag, leur ID, leur taille et leur création.

Supprimer une image de conteneur:

docker rmi <image\_id>

Cela supprimera l'image de conteneur avec l'ID spécifié.

Télécharger une image de conteneur:

docker pull <image\_name>:<tag>

Cela téléchargera une image de conteneur à partir d'un registre, en utilisant le nom et le tag spécifiés.

Exécuter une commande dans un conteneur en cours d'exécution:

docker exec -it <container\_id> <command>

Cela exécutera une commande dans un conteneur en cours d'exécution, en utilisant l'ID spécifié. Par exemple pour ouvrir le bash dans un conteneur :

docker exec -it <container\_id> /bin/bash

Félicitations, vous savez maintenant comment utiliser l'interface de ligne de commande Docker pour gérer les conteneurs et les images!

## Création d'une image de conteneur

Dans cette section, nous allons apprendre à créer une image de conteneur à l'aide de Dockerfile.

Un Dockerfile est un fichier de configuration qui décrit les étapes nécessaires pour construire une image de conteneur. Il s'agit d'un langage simple et déclaratif qui permet de définir les dépendances, les configurations et les commandes nécessaires pour construire une image de conteneur.

On peut créer un fichier nommé « Dockerfile » dans n’importe quel dossier, de préférence vide, qui nous servira de dossier de travail.

Voici un exemple de contenu de fichier Dockerfile pour un serveur LAMP:

FROM debian:latest

RUN apt-get update && apt-get install -y apache2 mariadb-server php libapache2-mod-php php-mysql

COPY . /var/www/html

EXPOSE 80

CMD ["apache2ctl", "-DFOREGROUND"]

Voici une explication des instructions utilisées dans ce Dockerfile:

\* FROM debian:latest: Spécifie la version de la base d'image à utiliser pour construire l'image de conteneur.

\* RUN apt-get update && apt-get install -y …etc: Exécute des commandes dans le conteneur pour installer les dépendances nécessaires.

\* COPY . /var/www/html: Copie les fichiers du répertoire courant vers le répertoire /var/www/html du conteneur.

\* EXPOSE 80: Spécifie le port à exposer sur le conteneur. On ne pourra mapper ce port avec un port de l'hôte qu'à l'exécution du conteneur.

\* CMD ["apache2ctl", "-DFOREGROUND"]: Spécifie la commande à exécuter lors du démarrage du conteneur.

Pour construire une image de conteneur à partir d'un Dockerfile, utilisez la commande suivante:

docker build -t <image\_name>:<tag> .

Cela construira une image de conteneur à partir du Dockerfile dans le répertoire courant et lui attribuera le nom et le tag spécifiés.

## Exécution d'un conteneur

Dans cette section, nous allons apprendre à exécuter un conteneur Docker à l'aide de l'image de conteneur que nous avons créée dans la section précédente.

Pour exécuter un conteneur Docker, utilisez la commande suivante:

docker run -d -p <host\_port>:<container\_port> <image\_name>:<tag>

Voici une explication des options utilisées dans cette commande:

\* -d: Exécute le conteneur en arrière-plan.

\* -p <host\_port>:<container\_port>: Mappe le port de l'hôte au port du conteneur.

\* <image\_name>:<tag>: Spécifie l'image de conteneur à utiliser pour exécuter le conteneur.

Par exemple, pour exécuter un serveur LAMP en arrière-plan et mapper le port HTTP (80) du conteneur sur le port HTTP (8080) de l'hôte, utilisez la commande suivante:

docker run -d -p8080:80 lamp-image:latest

Cela créera et démarrera un conteneur Docker à partir de l'image de conteneur lamp-image:latest et mappera le port HTTP (80) du conteneur sur le port HTTP (8080) de l'hôte.

Si tout s’est bien passé, vous pouvez à présent ouvrir l’IP de votre machine sur votre navigateur de la façon suivante :

http://<ip>:8080

Et le site web (ou la page par défaut d’Apache) devrait apparaitre.

Enfin, pour arrêter le conteneur on peut suivre les instructions suivantes :

Pour afficher les conteneurs en cours d'exécution, utilisez la commande suivante:

docker ps

Pour arrêter un conteneur en cours d'exécution, utilisez la commande suivante:

docker stop <container\_id>

Note : On peut n’utiliser que les premiers caractères de l’ID, pas la peine de la taper en entier, ça serait fastidieux.

## Partage de fichiers entre l'hôte et le conteneur

Dans cette section, nous allons apprendre à partager des fichiers entre l'hôte et le conteneur Docker.

Il existe plusieurs méthodes pour partager des fichiers entre l'hôte et le conteneur:

\* Utilisation de volumes: Les volumes permettent de monter un répertoire de l'hôte dans le conteneur. Les modifications apportées au répertoire de l'hôte sont automatiquement répercutées dans le conteneur.

\* Utilisation de bind mounts: Les bind mounts permettent de monter un répertoire de l'hôte dans le conteneur, en spécifiant le répertoire de l'hôte à monter. Les modifications apportées au répertoire de l'hôte sont automatiquement répercutées dans le conteneur.

\* Utilisation de copie de fichiers: Il est possible de copier des fichiers du conteneur vers l'hôte et vice versa à l'aide de la commande docker cp.

Nous allons exclusivement aborder les volumes aujourd'hui.

Voici un exemple d'utilisation de volumes pour partager des fichiers entre l'hôte et le conteneur:

Créez un répertoire sur l'hôte pour partager des fichiers:

mkdir /home/bastien/site

Copiez ensuite à l’intérieur les fichiers de votre site web. Un simple index.html de test suffit.

Exécutez un conteneur avec un volume:

docker run -d -p80:80 -v /home/bastien/site:/var/www/html httpd

Cela exécutera un conteneur Apache2 en arrière-plan, en mappant le port HTTP (80) du conteneur sur le port HTTP (80) de l'hôte et en montant le répertoire /home/bastien/site de l'hôte dans le répertoire /var/www/html du conteneur.

Maintenant, vous pouvez placer des fichiers dans le répertoire /home/bastien/site de l'hôte et ils seront automatiquement répercutés dans le répertoire /var/www/html du conteneur.

Félicitations, vous savez maintenant comment partager des fichiers entre l'hôte et le conteneur Docker en utilisant des volumes!

## Création d'un réseau entre les conteneurs

Dans cette section, nous allons apprendre à créer un réseau entre les conteneurs Docker pour permettre aux conteneurs de communiquer entre eux.

Les réseaux permettent de créer des sous-réseaux virtuels dans lesquels les conteneurs peuvent communiquer entre eux.

Voici un exemple de création d'un réseau et de connexion de conteneurs à ce réseau:

Créez un réseau:

docker network create my-network

Exécutez un premier conteneur et connectez-le au réseau:

docker run -d --name container1 --network my-network httpd

Exécutez un deuxième conteneur et connectez-le au réseau:

docker run -d --name container2 --network my-network httpd

Maintenant, vous pouvez communiquer entre les conteneurs en utilisant leur nom:

Connectez-vous au conteneur1:

docker exec -it container1 /bin/bash

On installe la commande ping :

apt install iputils-ping

Ping conteneur2:

ping container2

Félicitations, vous savez maintenant comment créer un réseau entre les conteneurs Docker et permettre aux conteneurs de communiquer entre eux!

## Utilisation de Docker Compose pour gérer des applications multi-conteneurs

Dans cette section, nous allons apprendre à utiliser Docker Compose pour gérer des applications multi-conteneurs.

Docker Compose est un outil qui permet de définir et d'exécuter des applications multi-conteneurs en utilisant un fichier de configuration.

Voici un exemple de fichier de configuration pour une application qui contient un serveur web et une base de données. ATTENTION il doit impérativement s’appeler docker-compose.yaml. Attention également aux espaces en début de ligne, ils servent d’indentation. NE PAS UTILISER TAB POUR INDENTER.

version: "3"

services:

web:

build: .

ports:

- "80:80"

depends\_on:

- db

environment:

- DB\_HOST=db

- DB\_PORT=3306

- DB\_NAME=mydb

- DB\_USER=myuser

- DB\_PASS=mypassword

db:

image: mysql:5.7

environment:

- MYSQL\_ROOT\_PASSWORD=mypassword

- MYSQL\_DATABASE=mydb

- MYSQL\_USER=myuser

- MYSQL\_PASSWORD=mypassword

Ce fichier de configuration définit deux services: web et db.

Le service web est construit à partir d'une image Docker personnalisée qui est construite à partir d'un Dockerfile qui se trouve dans le même répertoire que le fichier de configuration.

Le service web expose le port HTTP (80) et dépend du service db.

Le service web définit également des variables d'environnement pour la connexion à la base de données. Une variable d'environnement est définie pour le nom de la base de données, le nom d'utilisateur et le mot de passe de la base de données.

Le service db utilise l'image mysql:5.7 et définit des variables d'environnement pour la configuration de la base de données.

Explication détaillée du fichier :

1. La première ligne définit la version du format du fichier de configuration. Dans cet exemple, nous utilisons la version 3:

version: "3"

2. La section services définit les services qui composent l'application:

services:

3. Le premier service définit le serveur web. Il s'appelle web et est construit à partir d'une image Docker personnalisée:

web:

build: .

4. Le serveur web expose le port HTTP (80):

ports:

- "80:80"

5. Le serveur web dépend du service db:

depends\_on:

- db

6. Le serveur web définit des variables d'environnement pour la connexion à la base de données:

environment:

- DB\_HOST=db

- DB\_PORT=3306

- DB\_NAME=mydb

- DB\_USER=myuser

- DB\_PASS=mypassword

7. Le deuxième service définit la base de données. Il s'appelle db et utilise l'image mysql:5.7:

db:

image: mysql:5.7

8. La base de données définit des variables d'environnement pour la configuration de la base de données:

environment:

- MYSQL\_ROOT\_PASSWORD=mypassword

- MYSQL\_DATABASE=mydb

- MYSQL\_USER=myuser

- MYSQL\_PASSWORD=mypassword

Pour exécuter l'application, utilisez la commande suivante:

sudo docker compose up

Cela exécutera les deux services et les connectera à un réseau Docker Compose interne.

Pour exécuter l’application en arrière-plan :

sudo docker compose up -d

Pour arrêter l’application, plusieurs possibilités :

Si vous avez lancé le conteneur au premier plan, un simple CTRL+C suffit. Ne pas spammer, une fois suffit, même si ça semble long. Le conteneur se ferme proprement.  
Si vous avez lancé le conteneur en arrière plan, utiliser la commande suivante dans le dossier de son docker-compose.yaml :

sudo docker compose down

Enfin, vous pouvez également arrêter chaque conteneur de l’application à la main, comme n’importe quel conteneur, en regardant dans le docker ps.

Félicitations, vous savez maintenant comment utiliser Docker Compose pour gérer des applications multi-conteneurs!