Lab 3. Maîtrise de Docker sous Ubuntu

Guide Complet de Déploiement d'Applications

Partie 1 : Commandes de base Docker

1. Installation de Docker

```
sudo apt install docker.io
```

2. Démarrage du service Docker

```
sudo systemctl start docker
```

3. Vérification de l'installation de Docker

```
docker --version
```

4. Lister les images Docker

```
docker images
```

5. Lister les conteneurs en cours d'exécution

```
docker ps
```

6. Créer un conteneur interactif

```
docker run -it ubuntu /bin/bash
```

7. Démarrer un conteneur en arrière-plan

```
docker run -d nginx
```

8. Exécuter une commande dans un conteneur en cours d'exécution

```
docker exec -it container_id /bin/bash
```

9. Arrêter un conteneur en cours d'exécution

```
docker stop container_id
```

10. Supprimer un conteneur arrêté

```
docker rm container_id
```

Partie 2 : Création d'une image à partir d'un conteneur

1. Créer un conteneur de base

```
docker run -d --name my-container ubuntu
```

2. Personnaliser le conteneur

```
docker exec -it my-container /bin/bash
```

3. Créer une image à partir du conteneur personnalisé**

```
docker commit my-container my-image:v1
```

4. Lister les images pour vérifier la nouvelle image**

```
docker images
```

5. Supprimer le conteneur

```
docker stop my-container

docker rm my-container
```

6. Exécuter un conteneur basé sur la nouvelle image

```
docker run -it my-image:v1 /bin/bash
```

7. Effectuer des modifications dans le conteneur basé sur l'image et intaller python dans le container

```
apt-get install python3
```

8. Créer une nouvelle version de l'image

```
docker commit my-container my-image:v2
```

9. **Lister les images pour vérifier la nouvelle version**

```
docker images
```

10. Supprimer le conteneur basé sur l'ancienne image

```
docker stop my-image:v1

docker rm my-image:v1
```

Partie 3 : Utilisation de `docker build`

Pour cette partie, nous allons utiliser un exemple concret en créant une image Docker pour un serveur web Nginx personnalisé.

1. Créer un répertoire pour votre projet**

```
mkdir mon_projet_nginx
```

```
cd mon_projet_nginx
```

2. Créer un fichier Dockerfile

```
gedit Dockerfile
```

Ce fichier contiendra les instructions pour la construction de l'image Docker.

```
# Utilisez une image de base Nginx

FROM nginx:latest

# Copiez un fichier de configuration personnalisé dans le conteneur

COPY nginx.conf /etc/nginx/nginx.conf
```

3. Créer un fichier de configuration Nginx personnalisé

Créez un fichier de configuration Nginx personnalisé appelé "nginx.conf" dans le même répertoire que votre Dockerfile avec les configurations souhaitées.

```
# Exemple de configuration Nginx personnalisée
server {
    listen 80;
    server_name example.com;
    location / {
        root /usr/share/nginx/html;
        index index.html;
    }
}
```

4. Construire une image à partir du Dockerfile

Utilisez la commande suivante pour construire une image à partir du Dockerfile situé dans le répertoire actuel :

```
docker build -t mon-nginx-personnalise:1.0 .
```

5. Lister les images pour vérifier la nouvelle image

```
docker images
```

6. Exécuter un conteneur basé sur l'image construite

```
docker run -d -p 8080:80 mon-nginx-personnalise:1.0
```

7. Vérifier l'exécution du conteneur

Ouvrez un navigateur web et accédez à **`http://localhost:8080`** pour vérifier si Nginx fonctionne avec votre configuration personnalisée.

8. Arrêter et supprimer le conteneur

```
docker stop container_id

docker rm container_id
```

9. Nettoyer les conteneurs et images non utilisés

Pour nettoyer les conteneurs et images non utilisés, exécutez les commandes suivantes :

```
docker container prune

docker image prune
```

10. Supprimer l'image locale

```
docker rmi mon-nginx-personnalise:1.0
```

Partie 4 : Utilisation de Docker Compose

Pour cette partie, nous allons utiliser un exemple concret en créant un environnement Docker Compose pour une application web Python basée sur Flask et un serveur de base de données MySQL. Assurez-vous d'avoir Docker Compose installé sur votre système.

Installation docker compose :

```
sudo apt install docker-compose
```

1. Créer un répertoire pour votre projet

```
mkdir mon_projet_flask

cd mon_projet_flask
```

2. Créer un fichier `docker-compose.yml'

Créez un fichier `docker-compose.yml` dans le répertoire du projet. Ce fichier définira la configuration de votre application il contient les instructions suivantes :

```
version: '3'

services:

  web:

    image: python:3.8-slim

    command: python app.py

    volumes:
```

```
    - ./app:/app
  ports:
    - 5000:5000
 db:
  image: mysql:5.7
  environment:
    MYSQL_ROOT_PASSWORD: my-secret-pw
    MYSQL_DATABASE: mydb
    MYSQL_USER: myuser
    MYSQL_PASSWORD: mypassword
```

3. Créer un répertoire `app` pour votre application Flask

```
mkdir app
```

4. Créer un fichier `app.py` pour votre application Flask

Créez un fichier `app.py` dans le répertoire `app` avec votre code Flask.

```
from flask import Flask
import mysql.connector
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

5. Exécuter l'application avec Docker Compose

Dans le répertoire principal de votre projet, exécutez la commande suivante pour démarrer les services définis dans le fichier `docker-compose.yml` :

```
docker-compose up -d
```

6. Vérifier l'application Flask

Ouvrez un navigateur web et accédez à `http://localhost:5000` pour vérifier votre application Flask en cours d'exécution.

7. Arrêter les services Docker Compose

Pour arrêter les services, exécutez la commande suivante dans le répertoire principal de votre projet :

```
docker-compose down
```

8. Supprimer les conteneurs et volumes Docker Compose

Si vous souhaitez supprimer complètement les conteneurs et volumes créés par Docker Compose, utilisez la commande suivante :

```
docker-compose down -v
```

Partie 5 : Déployer une application Python avec Docker

Dans cette partie, nous allons déployer une application Python Flask simple à l'aide d'une image réelle de Python en utilisant Docker Compose. Suivez ces 10 étapes pour accomplir cette tâche :

1. Créer un répertoire pour votre projet

Créez un répertoire dédié pour votre projet Docker :

```
mkdir mon_projet_python

cd mon_projet_python
```

2. Créer un fichier `Dockerfile`

Créez un fichier Dockerfile dans le répertoire de votre projet. Ce fichier contiendra les instructions pour construire l'image Docker pour votre application Python Flask.

gedit Dockerfile

```
# Utilisez une image de base Python

FROM python:3.8-slim

# Définissez le répertoire de travail dans le conteneur

WORKDIR /app

# Copiez le fichier des dépendances Python

COPY requirements.txt .

# Installez les dépendances Python

RUN pip install --no-cache-dir -r requirements.txt

# Copiez le code source dans le conteneur

COPY . .

# Commande par défaut pour exécuter l'application

CMD ["python", "app.py"]
```

3. Créer un fichier `requirements.txt`

Créez un fichier `requirements.txt` pour lister les dépendances de votre application Python Flask. Par exemple :

```
flask==2.0.1
```

4. Créer un répertoire `app` pour votre application Python

```
mkdir app
```

5. Créer un fichier `app.py` pour votre application Python Flask

Créez un fichier `app.py` dans le répertoire `app` avec votre code Python Flask. Voici un exemple simple :

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def hello():

   return 'Hello, World!'

if __name__ == '__main__':

   app.run(host='0.0.0.0')
```

6. Créer un fichier `docker-compose.yml`

Créez un fichier `docker-compose.yml` dans le répertoire principal de votre projet pour définir la configuration de votre application et des services associés.

```
version: '3'

services:

 app:

   build:

     context: .

     dockerfile: Dockerfile

   ports:

     - 5000:5000
```

7. Construire l'image Docker

Exécutez la commande suivante pour construire l'image Docker à partir du Dockerfile :

```
docker-compose build
```

8. Exécuter l'application avec Docker Compose

Démarrez l'application en exécutant le service défini dans le fichier `docker-compose.yml` :

```
docker-compose up -d
```

9. Vérifier l'application Flask

Ouvrez un navigateur web et accédez à `http://localhost:5000` pour vérifier que votre application Python Flask fonctionne correctement.

10. Arrêter et nettoyer les conteneurs Docker Compose

Pour arrêter les services Docker Compose, exécutez la commande suivante dans le répertoire principal de votre projet :

```
docker-compose down
```

Si vous souhaitez également supprimer complètement les conteneurs et volumes créés par Docker Compose, utilisez la commande suivante :

```
docker-compose down -v
```