**TD6 CLOUD COMPUTING**

**Étape 1:**

Pour cette étape, nous pouvons ajouter le code suivant à notre application Flask pour lire le contenu d'un fichier texte et l'afficher sur une page Web:

@app.route('/file\_content')

def file\_content():

with open('/path/to/your/file.txt', 'r') as f:

content = f.read()

return content

Ce qui nous donne :Une image contenant texte

Description générée automatiquement

***import os et import time : importent les bibliothèques Python nécessaires pour notre application.***

***from flask import Flask : importe la classe Flask, qui est utilisée pour créer notre application Flask.***

***app = Flask(\_\_name\_\_) : crée une instance de l'application Flask.***

***def read\_file(): : définit une fonction qui lit le contenu du fichier "content.txt".***

***@app.route('/') : définit une route pour notre application Flask.***

***content = read\_file() : lit le contenu du fichier "TEST.txt".***

***return f'Hello, World! Here is the content of the file: {content}' : renvoie une chaîne de caractères contenant le contenu du fichier "TEST.txt".***

***if \_\_name\_\_ == '\_\_main\_\_': : vérifie si le fichier est exécuté directement.***

***app.run(host='0.0.0.0', debug=True) : lance l'application Flask en écoutant les requêtes sur toutes les interfaces réseau et en activant le mode de débogage.***

Dans ce code, nous créons une application Flask qui lit le contenu d'un fichier texte nommé "TEST.txt" à la racine du projet. Le contenu de ce fichier est renvoyé sur la page d'accueil de l'application Flask. Le code est également écrit pour permettre à l'application de se recharger automatiquement lorsqu'un changement est apporté au fichier "TEST.txt".  
  
Nous avons aussi le Dockerfile :

Une image contenant texte

Description générée automatiquement

***FROM python:3.9-slim-buster : spécifie l'image de base que nous utiliserons pour construire notre conteneur. Ici, nous utilisons l'image Python 3.9 slim.***

***WORKDIR /app : définit le répertoire de travail à l'intérieur du conteneur.***

***COPY requirements.txt . : copie le fichier requirements.txt du répertoire local vers le répertoire de travail du conteneur.***

***RUN pip install -r requirements.txt : installe les dépendances Python nécessaires pour exécuter notre application.***

***COPY app.py . : copie le fichier app.py du répertoire local vers le répertoire de travail du conteneur.***

***CMD [ "python", "./app.py" ] : définit la commande qui sera exécutée lorsque le conteneur est démarré. Ici, nous exécutons l'application Python contenue dans le fichier app.py.***

Le chemin d'accès au fichier peut être un chemin absolu sur notre système d'hébergement ou un chemin relatif dans le conteneur Docker. Pour permettre les mises à jour en temps réel du contenu du fichier, nous pouvons utiliser un bind mount en ajoutant l'option -v lors du démarrage du conteneur Docker:

docker run --name td6 -p 5005:5000 -v /Users/alexis/Desktop/TEST.txt:/app/TEST.txt my-flask-app

***docker run : commande pour démarrer un conteneur Docker.***

***--name td6 : spécifie un nom pour le conteneur Docker.***

***-p 5005:5000 : mappe le port 5000 du conteneur Docker sur le port 5005 de l'hôte local.***

***-v /Users/alexis/Desktop/TEST.txt:/app/TEST.txt : monte le fichier /Users/alexis/Desktop/TEST.txt de l'hôte local vers le répertoire /app à l'intérieur du conteneur Docker.***

***my-flask-app : spécifie le nom de l'image Docker à utiliser pour démarrer le conteneur.***

Ainsi, cette commande lance un conteneur Docker à partir de l'image spécifiée en montant le fichier /Users/alexis/Desktop/TEST.txt sur le répertoire /app à l'intérieur du conteneur et en mappant le port 5000 du conteneur sur le port 5005 de l'hôte local. Le conteneur est nommé td6.

**Étape 2:**

Pour rendre notre base de données persistante, nous pouvons utiliser un volume Docker. Pour ce faire, nous pouvons ajouter l'option -v lors du démarrage du conteneur MongoDB pour spécifier un nom de volume:

docker run -p 27017:27017 -v my-mongodb-data:/data/db mongo

Pour migrer la base de données vers une autre instance du même moteur de base de données, vous pouvez sauvegarder le volume Docker en utilisant la commande docker volume create et docker run pour créer un nouveau conteneur Docker en utilisant le volume créé:

docker volume create --name my-mongodb-data-backup

docker run -v my-mongodb-data:/backup -v my-mongodb-data-backup:/volume-backup busybox tar cvf /volume-backup/backup.tar /backup

Cela crée un nouveau volume Docker nommé my-mongodb-data-backup et sauvegarde le contenu du volume my-mongodb-data dans le fichier backup.tar dans le nouveau volume.

Pour restaurer la base de données à partir de la sauvegarde, vous pouvez exécuter la commande docker run avec les options -v et --entrypoint pour extraire le fichier de sauvegarde et l'importer dans la base de données MongoDB:

docker run -v