TP Docker et Jenkins

Introduction

Dans ce projet, vous allez travailler avec Docker et Jenkins pour automatiser l'exécution d'un projet Python simple. Le script Python, `sum.py`, a pour but de prendre deux arguments en ligne de commande, de les additionner et d'afficher le résultat. Le projet consiste à créer un fichier `Dockerfile` qui permettra de construire un environnement Docker pour exécuter ce script, ainsi qu'à utiliser les commandes Docker pour construire et exécuter le conteneur.

Le fichier `sum.py` est le suivant :

Listing 1 – sum.py

```
import sys

# V rifier si deux arguments sont fournis
if len(sys.argv) != 3:
    print("Erreur: Deux arguments sont necessaires.")
    sys.exit(1)

# Interpr ter les arguments
try:
    arg1 = float(sys.argv[1])
    arg2 = float(sys.argv[2])
except ValueError:
    print("Erreur:Les arguments doiventetre des nombres.")

sys.exit(1)

# Calculer la somme
resultat = arg1 + arg2

# Afficher le r sultat
print(resultat)
```

Votre objectif est de cr´eer un fichier `Dockerfile` pour configurer cet environnement Python dans Docker, puis d'utiliser Docker pour construire et ex´ecuter le conteneur.

Partie 1 :

Question 1 : Écriture du Dockerfile

Rédigez un fichier `Dockerfile` qui utilise l'image officielle Python et copie le script `sum.py` dans le conteneur Docker.

Indications :

— Utilisez l'image `python:3.13.0-alpine3.20`.

— Copiez le script `sum.py` dans le répertoire `/app` à l'intérieur du conteneur.

— Assurez-vous que le conteneur reste actif après son démarrage, car cela sera utile pour l'intégration avec Jenkins.

Question 2 : Construction de l'image Docker

Donnez la commande à utiliser pour construire l'image Docker en utilisant le `Dockerfile` que vous avez créé dans la question précédente.

Indication :

— Utilisez l'option `-t` pour donner un nom à l'image Docker.

Question 3 : Exécution du conteneur Docker

Quelle est la commande à utiliser pour exécuter un conteneur Docker à partir de l'image que vous avez construite ?

Indication :

— Utilisez l'option `-it` pour exécuter le conteneur de manière interactive.

— Assurez-vous d'ajouter les arguments nécessaires pour exécuter le script `sum.py` avec deux nombres comme arguments.

En répondant aux questions ci-dessus, vous aurez réalisé un projet Docker simple qui automatise l'exécution d'un script Python. Ce projet pourra ensuite être intégré dans un pipeline Jenkins pour une automatisation plus avancée.

Partie 2 : Création du Jenkinsfile

Dans cette partie, vous allez créer un pipeline Jenkins qui exécute plusieurs étapes : construction de l'image Docker, exécution du conteneur, et test d'un script Python (`sum.py`) à l'intérieur du conteneur. Le fichier de test utilisé (`test_variables.txt`) contient des valeurs à tester avec le script `sum.py` pour vérifier la somme de deux nombres. Voici les instructions étape par étape pour chaque phase du pipeline.

Étape 1 : Définir l'agent et l'environnement

Instructions :

— Commencez par définir un agent global avec `agent any`.

— Ensuite, dans la section `environment`, créez les variables d'environnement suivantes :

  — `CONTAINER_ID` : variable qui stockera l'ID du conteneur Docker exécuté.

  — `SUM_PY_PATH` : chemin vers le fichier `sum.py` sur votre machine locale.

  — `DIR_PATH` : chemin vers le répertoire contenant le `Dockerfile`.

— `TEST_FILE_PATH` : chemin vers le fichier `test_variables.txt` qui contient les valeurs de test.

Étape 2 : Stage de Construction (Build)

Instructions :

— Créez un stage appelé `Build`.
— Dans ce stage, construisez l'image Docker à partir du `Dockerfile` à l'aide de la commande `docker build`.

Étape 3 : Stage d'Exécution (Run)

Instructions :

— Créez un stage appelé `Run`.
— Utilisez la commande `docker run` pour démarrer un conteneur à partir de l'image construite.
— Stockez l'ID du conteneur dans la variable `CONTAINER_ID`.

Commandes utiles :

```
def output = bat(script: 'COMMANDE DOCKER A COMPLETER', returnStdout: true)
def lines = output.split('\n')
CONTAINER_ID = lines[-1].trim()
```

Étape 4 : Stage de Test (Test)

Instructions :

— Créez un stage appelé `Test`.
— Lisez le contenu du fichier `test_variables.txt`, qui contient des lignes avec trois valeurs : deux nombres à additionner et leur somme attendue.
— Pour chaque ligne du fichier :
  — Exécutez `sum.py` à l'intérieur du conteneur Docker avec les deux premiers arguments (avec la commande `docker exec ...`).
  — Comparez la somme renvoyée par le script avec la somme attendue.
  — Si le résultat est correct, affichez un message de réussite ; sinon, générez une erreur.

Commandes utiles :

```
def testLines = readFile(TEST_FILE_PATH).split('\n')

for (line in testLines) {
    def vars = line.split(' ')
    def arg1 = vars[0]
    def arg2 = vars[1]
    def expectedSum = vars[2].toFloat()

    def output = bat(script: "COMMANDE DOCKER A COMPLETER", returnStdout: true)
```

```
def result = output.split('\n')[-1].trim().toFloat()

if (result == expectedSum) {
    echo "MESSAGE A COMPLETER"
} else {
    error "MESSAGE A COMPLETER"
}
```

Étape 5 : Section `post`

Instructions :

— Après les tests, assurez-vous de toujours stopper et supprimer le conteneur Docker, quel que soit le résultat du pipeline.

Explication du fichier `test_variables.txt`

Le fichier `test_variables.txt` contient des variables de test sous forme de lignes où chaque ligne est composée de trois valeurs :

— Le premier nombre (`arg1`).
— Le deuxième nombre (`arg2`).
— La somme attendue de ces deux nombres.

Voici un exemple de contenu du fichier :

```
5 10 15
3.5 7.2 10.7
-2 4 2
0 0 0
-3 -7 -10
1.1 2.2 3.3
```

Chaque ligne est utilisée pour tester la fonction d'addition dans le fichier `sum.py` à l'intérieur du conteneur Docker. Si la somme renvoyée par le script Python correspond à la somme attendue, le test est considéré comme réussi.

Étape 6 : Stage de Déploiement (Deploy to DockerHub)

Instructions :

— Créez un stage nommé `Deploy`.
— Dans ce stage, vous allez pousser l'image Docker que vous avez construite vers DockerHub.
— Assurez-vous que vous êtes connecté à DockerHub en utilisant la commande `docker login`.
— Utilisez ensuite la commande `docker tag` pour taguer l'image avant de la pousser vers votre dépôt DockerHub.
— Enfin, poussez l'image avec la commande `docker push`.