Exercice de rappel : Exécution d'un conteneur Docker existant

Recherchez et exécutez le conteneur Docker « hello-world » à partir de Docker Hub.

Présentation des Dockerfiles :

Un Dockerfile est un fichier texte qui contient une série d'instructions pour construire une image Docker. Il s'agit d'un composant essentiel dans le processus de création d'images personnalisées dans Docker.

C'est utilisé pour automatiser le processus de construction d'une image Docker, en spécifiant les dépendances, les configurations et les commandes nécessaires pour créer un environnement de conteneur cohérent et fonctionnel.

Voici quelques éléments clés dans la compréhension d'un Dockerfile :

Directive FROM : La première instruction d'un Dockerfile spécifie l'image de base à utiliser pour construire l'image Docker. Par exemple, FROM ubuntu:20.04 indique que l'image de base sera basée sur Ubuntu 20.04.

Directive COPY : Cette instruction copie des fichiers et des répertoires depuis le système de fichiers de l'hôte vers l'image Docker en cours de construction. Par exemple, COPY app.py /app copie le fichier "app.py" de l'hôte vers le répertoire "/app" dans l'image Docker.

Directive RUN : Cette instruction exécute des commandes dans l'image Docker en cours de construction. Par exemple, RUN apt-get update && apt-get install -y python3 met à jour les paquets de l'image et installe Python 3.

Directive WORKDIR : Cette instruction définit le répertoire de travail à utiliser pour les instructions suivantes. Par exemple, WORKDIR /app définit le répertoire de travail à "/app" dans l'image Docker.

Directive EXPOSE : Cette instruction spécifie les ports sur lesquels le conteneur Docker écoute lorsqu'il est en cours d'exécution. Par exemple, EXPOSE 80 indique que le conteneur Docker écoute sur le port 80.

Directive CMD : Cette instruction spécifie la commande par défaut à exécuter lorsqu'un conteneur Docker basé sur l'image est démarré. Par exemple, CMD ["python", "app.py"] exécute la commande "python app.py" lorsque le conteneur démarre.

Ces directives et bien d'autres permettent de définir l'environnement et le comportement d'une image Docker. Une fois que vous avez rédigé un Dockerfile, vous pouvez le passer à la commande docker build pour construire l'image Docker correspondante.

TD 1- Création d'un conteneur personnalisé

Créez un conteneur Docker personnalisé à l'aide d'un Dockerfile. Le conteneur devrait être basé sur une image existante et effectuer une action spécifique. Par exemple, créez un conteneur basé sur une image Python3 qui exécute un script Python simple qui affiche "Hello, Docker!".

Création d'un fichier script.py

```
user@user-VirtualBox:/app$ ls
Dockerfile  script.py
user@user-VirtualBox:/app$ sudo nano script.py
```

```
  GNU nano 6.2
print("Hello, Docker!")
```

print("Hello, Docker!")

Création d'un fichier Dockerfile :

```
user@user-VirtualBox:/app$ ls
Dockerfile  script.py
user@user-VirtualBox:/app$ sudo nano Dockerfile
```

```
  GNU nano 6.2
FROM python:3.8-slim-buster
COPY script.py /script.py
CMD [ "python", "/script.py" ]
```

FROM python:3.8-slim-buster

COPY script.py /script.py

CMD ["python", "/script.py"]

Pour construire et exécuter le conteneur personnalisé :

```
user@user-VirtualBox:/app$ ls
Dockerfile  script.py
user@user-VirtualBox:/app$ docker build -t my-container .
[+] Building 0.8s (7/7) FINISHED
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 122B
 => [internal] load metadata for docker.io/library/python:3.8-slim-buster
 => [internal] load build context
 => => transferring context: 60B
 => CACHED [1/2] FROM docker.io/library/python:3.8-slim-buster@sha256:89ad1c2cd09bda5bc85ada7eb93b5db57d32dc0105b7c942d272d68f376f67c3
 => [2/2] COPY script.py /script.py
 => exporting to image
 => => exporting layers
 => => writing image sha256:76c28afb96b4a385bd23acfbe5fd11f627e400f88e45e19b9fc71488d39f6e13
 => => naming to docker.io/library/my-container
user@user-VirtualBox:/app$
```

docker build -t my-container .

```
user@user-VirtualBox:/app$ ls
Dockerfile  script.py
user@user-VirtualBox:/app$ docker run my-container
Hello, Docker!
user@user-VirtualBox:/app$
```

docker run my-container

TP 1- A vous :

Créez un dockerfile d'un script python qui affiche l'heure et exécutez-le.

TP 2 – Changez d'image :

Refaites un des deux exercices précédents, mais en utilisant cette fois l'image « ubuntu:20.04 »

Vous allez rencontrer des difficultés si vous n'adaptez pas, pour cela référez vous aux différentes directives listées dans le cours sur la présentation des dockerfiles