TD 2 :

Dans cette partie, nous allons voir les commandes de base de Docker en utilisant l'interface en ligne de commande de Docker :

Préparation :

Lancer le démon Docker : `sudo service docker start`

Instruction

Étape 1 : Manipulation

1. **image hello-world :**
    a. Instancier et exécuter une image hello-world : `docker run hello-world`
    b. Expliquer le processus concret lors de l'exécution de `docker run hello-world`.
    c. Ré-instancier l'image hello-world, et décrire ce qui se passe. Pourquoi y a-t-il une différence ?
2. **Image Ubuntu :**
    a. Télécharger l'image d'Ubuntu avec le tag `latest` : `docker pull ubuntu:latest`
    b. Lister les images disponibles : `docker images`
    c. Lancer une instance de l'image `ubuntu:latest` : `docker run ubuntu:latest`
    d. Lancer une instance d'Ubuntu en mode interactif (`-it` ou `--interactive`) : `docker run -it ubuntu:latest`
    e. Exécuter la commande `exit` depuis l'intérieur du conteneur.
    f. Décrire ce qui s'est passé lors de ces étapes.
    g. Lister tous les conteneurs, y compris les arrêtés (`--all`) : `docker ps --all`
    h. Relancer un conteneur Docker arrêté : `docker start [nom_ou_id_du_conteneur]`
3. **Image Ubuntu en mode interactif :**
    a. Lancer un conteneur Ubuntu avec la commande `bash` : `docker run -it ubuntu bash`
    b. Lister les conteneurs disponibles : `docker ps`
    c. Stopper le conteneur : `docker stop [nom_ou_id_du_conteneur]`
    d. Vérifier que le conteneur a bien été arrêté.
    e. Nettoyer vos conteneurs : `docker rm [nom_ou_id_du_conteneur]`
4. **Image Ubuntu en mode détaché :**
    a. Lancer un conteneur Ubuntu avec `bash` en mode détaché : `docker run -d ubuntu bash`
    b. Lancer un conteneur avec une variable d'environnement `ma_variable` : `docker run -e "ma_variable=valeur" ubuntu`
    c. Afficher cette variable dans le conteneur.
    d. Expliquer le comportement du conteneur lorsqu'on sort de celui-ci. S'est-il arrêté ? Pourquoi ?
5. **Inspect :**
    a. Inspecter le conteneur pour voir les propriétés du conteneur lancé : `docker inspect [nom_ou_id_du_conteneur]`
    b. Arrêter le conteneur et le ré-inspecter. Qu'est-ce qui a changé ?

Argument Utilisation :

- `-it` ou `--interactive` : Permet d'interagir avec le conteneur via la commande passée.
- `-d` ou `--detach` : Permet de lancer le conteneur en arrière-plan.
- `-n` ou `--name` : Permet de choisir le nom du conteneur.
- `--rm` : Force la suppression du conteneur après son arrêt.
- `-e` : Permet de définir une variable d'environnement dans le conteneur.

Étape 2 : Network

1. **Docker – Réseaux :**
    a. Télécharger l'image d'Elasticsearch avec le tag 7.2.0 : **`docker pull elasticsearch:7.2.0`**
    b. Lancer une instance d'ElasticSearch avec les propriétés suivantes :
        i. En mode détaché : **`-d`**
        ii. Forcer la suppression lors de l'arrêt du conteneur : **`--rm`**
        iii. Nommer le conteneur **`my_es_container`** : **`--name my_es_container`**
        iv. Définir une variable d'environnement **`discovery.type=single-node`** : **`-e discovery.type=single-node`**
    c. Essayer d'accéder à http://localhost:9200 (ou faire un **`curl -X GET -i`**)
    d. Quel est le résultat obtenu ? Pourquoi ?
    e. Refaire le test mais à l'intérieur du conteneur.
    f. Chercher l'IP du conteneur (**`docker inspect [nom_ou_id_du_conteneur]`** et rechercher **`IPAddress`**)
    g. Essayer d'y accéder à http://IP_du_conteneur:9200 (ou faire un **`curl -X GET -i`**)
    h. Rediriger les ports 9200 et 9300 du conteneur vers les ports 9201 et 9301 de la machine hôte.
    i. Essayer d'y accéder à http://localhost:9201 (ou faire un **`curl -X GET -i`**)
    j. Quel est le résultat obtenu ? Pourquoi ?
    k. Relancer le conteneur avec les mêmes ports. Que s'est-il passé ? Pourquoi ?
2. **Communication entre conteneurs**
    a. Télécharger l'image **`datascientest/curl:latest`** : **`docker pull datascientest/curl:latest`**
    b. Lancer et utiliser ce conteneur :
        i. **`docker run --rm datascientest/curl:latest http://example.org`**
        ii. Que fait ce conteneur ?
    c. Essayer d'effectuer une requête sur cette adresse : **`http://127.0.0.1:9200`**
        i. La connexion est-elle passée ? Pourquoi ?
    d. Essayer d'effectuer une requête sur cette adresse : **`http://127.0.0.1:9201`**
        i. La connexion est-elle passée ? Pourquoi ?
    e. Tenter de faire cette requête sur l'adresse réelle du conteneur Elasticsearch : **`http://IP_du_conteneur:9200`**
3. **Network**
    a. Afficher la liste des réseaux : **`docker network ls`**
    b. Inspecter le réseau **`bridge`** : **`docker network inspect bridge`**

i. Quelles informations intéressantes retrouve-t-on ?
ii. Dans la clé `IPAM` (IP Address Management), on retrouve le `subnet` c'est-à-dire que le réseau `bridge` peut donner aux conteneurs qu'il contient, les adresses allant de 172.17.0.2 à 172.17.255.255.
iii. 172.17.0.1 est le gateway (point d'entrée) utilisé par le démon Docker pour gérer ce sous-réseau.

c.

4. **Network Bridge :**
   a. Créer le réseau `my_network` avec un subnet `172.50.0.0/16` et un gateway `172.50.0.1` : `docker network create --subnet=172.50.0.0/16 --gateway=172.50.0.1 my_network`
   b. Inspecter le réseau : `docker network inspect my_network`
   c. Lancer un autre conteneur Elasticsearch sur ce réseau (`--network my_network`)
   d. Inspecter le réseau `my_network` : Quelle est l'IP du nouveau conteneur ?
   e. Faire une requête à cette adresse depuis l'intérieur du réseau `bridge`.
   f. Faire une requête à cette adresse depuis l'intérieur du réseau `my_network`.
   g. Comment faire pour pallier au problème des adresses IP non fixes attribuées aux conteneurs.
5. **Network Host :**
   a. Arrêter tous les conteneurs en fonctionnement.
   b. Lancer un conteneur Elasticsearch sur le réseau `host` (`--network host`)
   c. Inspecter le réseau `host` : Pas de `Subnet`, ni de `Gateway`.
   d. Inspecter le conteneur Elasticsearch : L'IP du conteneur est en fait 127.0.0.1.
   e. Essayer d'effectuer une requête sur cette IP : `http://127.0.0.1:9200`.
      i. La connexion est-elle passée ? Pourquoi ?

Etape 3 : Volume

1. **Volume entre layers**
   a. Lancer un conteneur, instance de `ubuntu:latest`, nommée `my_ubuntu` de manière interactive : `docker run -it --name my_ubuntu ubuntu:latest`
   b. Créer un fichier dans le conteneur :
      i. Par exemple, `echo "hello world from Docker" > /home/test.txt`
   c. Arrêter le conteneur, le relancer, et afficher le fichier :
      i. Le fichier est-il toujours présent ?
   d. Lancer l'image `datascientest/time-keeper:latest` :
      i. `docker run --name my_time_keeper datascientest/time-keeper:latest`
      ii. L'image écrit l'heure dans un fichier à l'intérieur d'un conteneur si ce fichier n'existe pas déjà. Si le fichier existe déjà, il n'est pas créé. Le résultat est imprimé dans les deux cas.
   e. Relancer le même conteneur (`start`) :
      i. Le fichier a-t-il été conservé entre les deux démarrages ?
      ii. Conserver un conteneur Docker arrêté n'est pas économe en termes d'espace occupé.
      iii. Les volumes sont plutôt utilisés pour la persistance des données.
2. **Volume entre layers**

a. Créer un volume : `docker volume create my_volume`
b. Lister les volumes existants : `docker volume ls`
c. Inspecter les volumes : `docker volume inspect my_volume`
d. Lancer un conteneur Ubuntu avec un montage sur le volume créé :
    i. `docker run -it --name my_ubuntu --mount type=volume,src=my_volume,dst=/home/my_folder --rm ubuntu`
e. Étape 1 :
    i. Depuis le conteneur, afficher le contenu de `/home`
    ii. Créer un fichier dans le dossier `/home/my_folder`
    iii. Depuis une autre console, de votre machine, affichez le contenu du dossier correspondant au volume.
f. Étape 2 : On peut créer un fichier depuis la machine hôte et le mettre dans ce dossier ; il apparaîtra dans le conteneur.

Étape 4 : Dockerfile

1. **Exercice :**
    a. Créez un dossier `my_docker_image` sur la machine hôte.
    b. Créez un fichier Dockerfile.
    ```
    FROM debian:latest
    RUN apt-get update && apt-get install python3-pip -y && pip3 install flask==2.0.3
    ```
    c. Créez un fichier `server.py` dans le dossier `my_docker_image`
    ```
    from flask import Flask
    server = Flask('my_server')
    @server.route('/')
    def index():
      return 'Hello World from a containerized server'
    if __name__ == '__main__':
      server.run(host='0.0.0.0', port=5000
    ```
    d. Ajouter ce fichier dans le répertoire de l'image Docker (ex : `/my_server/server.py`).
    e. Configurer le dossier `my_server` comme dossier de travail.
    f. Exposer le port 5000.
    g. Lancer par défaut, au démarrage du Docker, le fichier `server.py` (`python3 server.py`).
    h. Builder l'image avec le nom `my_image` et le tag `latest`.
    i. Vérifier le contenu de `my_docker_image`, et vérifier la création de votre image.
    j. Lancez le conteneur en créant une redirection entre le port 5000 de la machine hôte et le port 5000 du conteneur.
2. **Manipulation suplementaire**
    a. Créez une archive à partir de l'image créée (`save`).
    b. Supprimez l'image `my_image` (`rm`).
    c. Recréez l'image à partir de l'archive (`load`).
    d. Vérifiez la présence de l'image.
    e. Si vous avez un compte sur Docker Hub, vous pouvez créer des images qui comportent un nom en deux parties `username/imagename:tag` (ex : `docker image push username/imagename:tag`).