TP 1 : Conteneurisation - Docker

Objectif : Le but de ce TP est de consolider vos connaissances sur la conteneurisation et les commandes docker pour gérer les images et les conteneurs

I. Gestion des images

Pour ce TP vous avez besoin d'une VM sous linux. Première chose à faire c'est d'installer le package docker avec la commande : `sudo apt-get install docker.io`

À noter que l'interface `docker0` est ajoutée automatiquement. C'est le nom d'un pont virtuel créé par Docker sur votre système hôte lors de l'installation de Docker. Ce pont est utilisé pour connecter les conteneurs Docker entre eux et les isoler du reste du système hôte. Par défaut, Docker utilise le sous-réseau IPv4 172.17.0.0/16 pour la mise en réseau des conteneurs

Lorsqu'un nouveau conteneur est démarré, Docker lui attribue une adresse IP dans le sous-réseau associé à `docker0`, ce qui permet aux conteneurs de communiquer entre eux via ce pont. Les conteneurs peuvent également communiquer avec le monde extérieur via le système hôte, qui agit comme une passerelle vers le réseau hôte.

Dans cette section, nous allons voir comment manipuler les images Docker. Nous nous appuierons sur des images existantes soit disponibles dans les dépôts officiels, soit créer spécifiquement pour ce TP et disponibles sur un dépôt privé.

1. *Téléchargement des images*

La première manipulation est de télécharger une image depuis un dépôt nommé registry. La commande pour télécharger une image depuis le registry officiel (https://hub.docker.com) est : `docker pull <image>:<tag>`

Dans cette commande, `<image>` correspond au nom de l'image. `<tag>` permet de choisir quelle version à installer. Ce paramètre est optionnel, par défaut, il s'agit de latest. Les tags permettent d'identifier une image par rapport au système et la version du logiciel.

a. Télécharger l'image du docker hello-world depuis le registry officiel
b. Donner la commande pour afficher les images docker disponible sur votre système. Tester la commande avec l'option -q, qu'est-ce que vous obtenez ?
c. C'est quoi le rôle du tag ?
d. Quelle commande permet de supprimer une image donnée par son nom ou par son identifiant

2. *Gestion des conteneurs*

Une fois téléchargée, l'image peut être utilisée pour exécuter un conteneur.

a. Lancer un conteneur avec l'image hello-world. Quelle commande avez-vous utilisé ?
b. Lancer un conteneur en mode interactif avec l'image python et le tag **slim**. Vous précisez le nom python pour le conteneur. C'est quoi l'invite de commande que vous obtenez ? comment sortir ?

- Lorsque l'image n'est pas présente, elle est automatiquement téléchargée.

c. L'option `--env VAR=VALUE` (ou `-e VAR=VALUE`) permet de passer des variables d'environnement au Docker, c'est l'option privilégiée pour passer des paramètres au

programme à exécuter dans le conteneur. Commencer par récupérer l'image `alpine` puis lancer un conteneur à partir de cette image en mode interactif en définissant la variable `$MESSAGE` avec la valeur 'Hello from Alpine'. Comment vérifier la prise en considération de votre variable ?

d. Certaines images nécessitent l'utilisation de données disponibles sur votre poste ou bien des données qui doivent persister après l'arrêt du conteneur. Dans ce cas, un répertoire peut être partagé avec le conteneur en utilisant l'option `--volume <dir>:<dest>` où <dir> est le chemin absolu du répertoire sur votre poste et <dest> le nom du répertoire dans le conteneur. Lancer un conteneur à partir de l'image alpine en précisant un répertoire partagé entre votre conteneur et votre VM.

- Les répertoires sont automatiquement créés s'ils n'existent pas auparavant.

e. Créer un fichier dans le répertoire de partage au niveau du conteneur puis quitter ce dernier et relancer le encore. Est-ce que votre fichier existe toujours dans ce répertoire ?

f. De nombreux services que nous allons dockeriser sont des services réseaux, ils disposent d'un port d'écoute qu'il faut rendre accessible depuis l'extérieur. Ici, l'extérieur correspond à notre machine. L'option `-p` de la commande docker run permet d'exposer le port du conteneur sur le port de notre machine. Récupérer tout d'abord l'image httpd depuis ub.docker.com puis exécuter la commande docker pour que l'application soit accessible sur le port 8080 de votre machine. Personnaliser également le nom du conteneur. Sur un navigateur aller sur le lien : http://localhost:8080 vous devez avoir le message « it works ! ». Tester aussi avec l'adresse ip du conteneur.

g. Vous remarquerez qu'en lançant le docker dans votre console, l'invite de commande reste bloquer. Quelle option il faut utiliser pour éviter ce comportement ?

h. Il est possible d'envoyer des commandes vers un conteneur en cours d'exécution avec la commande `docker exec -it <nom ou id du conteneur> <cmd>`. Agissez sur le conteur httpd pour modifier la page index.html en rajoutant votre nom. Tester votre travail en rechargeant la page web. Vous devez voir votre nom apparaitre. (Vous devez installer un éditeur avant pour pouvoir modifier le fichier qui se trouve dans /usr/local/apache2/htdocs)

i. Quelle commande il faut utiliser pour lister les conteneurs en cours d'exécution ? Pour lister également les conteneurs arrêtés, quelle option il faut ajouter l'option ?

j. Arrêter puis supprimer le conteneur httpd que vous avez lancer précédemment. Quelles commandes vous avez utilisées ?

k. Quelle commande permet de supprimer tous les conteneurs inactifs ?

l. Si vous voulez supprimer tous les conteneurs affichés avec la commande `docker ps -a`, vous pouvez utiliser la commande suivante : `docker rm $(docker ps -a -q)`

m. D'autres commande à tester :
   - `docker cp` : permet de copier des fichiers vers ou depuis le conteneur
   - `docker rename` : permet de renommer un conteneur à postériori

II. Docker-compose

L'objectif dans ce 2e partie de TP est de réaliser une stack de services interconnectés avec docker-compose.

Docker Compose est un outil qui permet de décrire, déployer et gérer une stack de services interconnectés. Ces services sont bien évidement déployé sous forme de conteneur docker. La configuration de Docker Compose est donné par un fichier `docker-compose.yaml` qui prend les éléments que nous avons vus dans la commande docker run. Un exemple de la configuration de ce fichier est illustré dans la figure suivante :

```
version: "3"

services:
    nomservice1:
        image: "image pour le docker"
        ports:
            - "<port_hote>:<port_conteneur>"
            - "<port2_hote>:<port2_conteneur>"
        volumes:
            - "<dir>:<dest>"
            - "<dir2>:<dest2>"
        environment:
            - "<variable>=<valeur>"
            - "<variable2>=<valeur2>"
        env_file:
            - fichier
        restart: always
```

Dans ce fichier, vous pouvez retrouver les éléments suivants :

- ✓ version : le numéro de version du format de fichier docker-compose.yaml
- ✓ nomservice1 : le nom du service
- ✓ image : l'image Docker utilisée, au format vu dans la première partie : `<registry>/<image>:<tag>
- ✓ ports : la liste des ports à publier (option --publish de la commande docker)
- ✓ volumes : la liste de répertoires et fichiers à monter (option --volume de la commande docker)
- ✓ environment : la liste des variables d'environnement (option --env)
- ✓ env_file : la liste des fichiers contenant de variables d'environnement (option --env-file)
- ✓ restart : le mode de redémarrage du service

1. *Première configuration*

1- Créer un fichier yaml avec le contenu suivant :

```yaml
version: '3.8'
services:
  mywebserver:
    image: httpd
    ports:
      - 80:80
  mydb:
    image: mysql
    ports:
      - 3306:3306
    environment:
      - MYSQL_ROOT_PASSWORD=Pass123!
```

2- Quelle commande vous devez utiliser pour faire exécuter le contenu du fichier yml ?

3- Visualisez les composants démarrés en utilisant la commande : `docker-compose -f votre-fichier-yml ps`

N.B le nom de fichier attendu par défaut par la commande docker-compose est soit docker-compose.yml, docker-compose.yaml, compose.yml ou compose.yaml. Si un autre nom est utilisé alors il faut utiliser l'option -f pour préciser le nom du fichier

4- Quelle autre commande permet de voir les conteneurs actifs ?
5- Quelle commande vous permet de visualisez également les logs de tous les services ?
6- Connecter vous sur le conteneur mysql afin de consulter la base de données crée. Quelle commande vous allez utiliser ?

Une fois dedans, connecter-vous à la BD avec la commande `mysql -u root -p` puis utiliser la commande `show databases;` vous devez avoir le résultat suivant :

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

7- Arrêter vos deux conteneurs avec la commande `docker-compose -f votrefichier.yaml down`

2. *Configuration de services interconnectés*

L'intérêt de Docker Compose est le déploiement de stacks de services interconnectés. Il est donc intéressant que le fichier docker-compose.yaml propose plusieurs services dépendants. Pour cela, nous définissons plusieurs services et ajoutons la clé depends_on à ceux dépendants des autres.

```yaml
services:
  web:
    image: monappli
    depends_on:
      - db
      - redis
  redis:
    image: redis
  db:
    image: postgres
```

Docker Compose s'assurera alors que le service web sera démarré uniquement lorsque les deux conteneurs *redis* et *db* le seront.

Docker Compose vérifie que le conteneur est démarré, il peut y avoir un délai avant que l'application le soit effectivement.

Pour fonctionner correctement, le service web doit être configuré avec les accès aux bases de données (hostname, user, password). Ceci peut être fait soit par un fichier de configuration soit par des variables d'environnement.

1- Modifier votre fichier yaml pour mettre une dépendance entre le conteneur httpd et mysql.

2- Ajoutez dans votre stack l'application *adminer* permettant de visualiser le contenu de la base de données. (https://hub.docker.com/_/adminer) puis accédez à l'application avec votre navigateur.

III. Réaliser et publier son image Docker

Dans cette partie vous allez voir comment réaliser l'image docker d'une application et la publier sur un registry privé. Avant de continuer arrêtez et supprimez tous les conteneurs et images créés auparavant.

1. Créer un fichier Dockerfile

Pour créer sa propre image, il faut partir d'une image existante complétée avec des instructions nécessaires à la configuration de son application. Ces instructions sont stockées dans un fichier nommé Dockerfile. Il s'agit d'un fichier texte dans lequel la première instruction `FROM` permet de définir l'image de base (le tag de l'image peut également être précisé). Ce fichier peut également contenir une instruction `ENTRYPOINT` pour définir le point d'entrée (et les paramètres) de notre application.

Voici une liste non exhaustive des commandes les plus couramment utilisées dans le Dockerfile :

- FROM : Spécifie l'image de base à utiliser pour construire l'image.
- RUN : Exécute une commande lors de la compilation, puis enregistre les modifications dans l'image.
- COPY : Copie des fichiers ou dossiers du système hôte vers le conteneur.
- WORKDIR : Définit le répertoire de travail pour les commandes suivantes.
- EXPOSE : Spécifie les ports que le conteneur écoutera.
- ENV : Définit une variable d'environnement dans le conteneur.
- LABEL : Ajoute une étiquette à l'image pour la décrire.
- USER : Définit l'utilisateur qui exécutera les commandes dans le conteneur.
- VOLUME : Crée un point de montage avec le nom spécifié
- CMD : Spécifie la commande à exécuter lorsque le conteneur est démarré.

Le fichier suivant permet de construire une image basée sur le système linux *alpine* et dont le point d'entrée est la commande date.

```
#Dockerfile
FROM alpine

ENTRYPOINT ["date"]
```

La sous-commande `build` permet de compiler le fichier Dockerfile en une nouvelle image, l'option -t permet de nommer notre image : `docker build -t afficher-date .`

N'oubliez pas le point `(.)` à la fin de la commande. La commande docker build attend le chemin du répertoire contenant le fichier Dockerfile (ici le répertoire courant).

a. Dans un répertoire afficher-date, créez le fichier Dockerfile ci-dessus et compilez-le. Vérifiez l'existence de l'image afficher-date et exécutez-la. Quelles commandes vous avez utilisé ?

L'instruction ENTRYPOINT n'est pas obligatoire, dans ce cas le point d'entrée sera celui de l'image de base, pour l'image Alpine, il s'agit de /bin/sh -c date.

b. Modifier votre Dockerfile pour ajouter la commande `RUN` avec la commande `apk update` qui met à jour la liste des fichiers disponibles dans les dépôts APK pour l'image alpine. Vous rajouter aussi une commande pour installer le package vim avec la

commande `apk add vim`. Vous rajouter une 3e commande pour créer le répertoire rep1 dans le dossier home. Vous mettez en commentaire la commande `ENTRYPOINT`

c. Générer une nouvelle image à partir de votre Dockerfile puis lancer un conteneur à partir de votre image et vérifier bien la présence du dossier rep1 et de l'éditeur vim.

2. *Dockerfile – un peu plus loin*

Vous allez maintenant créer un 2e fichier dockerfile nommé mydockerfile avec les instructions suivantes :

- L'image de base sera ubuntu
- Mettre à jour les dépôts (apt-get update)
- Installer apache et vim
- Créer le dossier backup dans le dossier /home
- Copier le fichier myfile dans le dossier backup (le fichier myfile est à créer sur votre VM avec un contenu de votre choix)

a- Créer une image nommée newimage à partir de votre fichier mydockerfile. Utiliser l'option -f de la commande docker build
b- Lancer un conteneur depuis l'image que vous venez de créer.
c- Connectez-vous à votre conteneur et vérifier bien la présence du fichier myfile puis apporter des modifications dans ce fichier.
d- Arrêter le conteneur et vérifier si le fichier myfile sur la machine hôte contient bien les modifications apportées. Si ce n'est pas le cas, proposer une solution pour rendre les modifications persistantes.
e- Relancer le conteneur et vérifier qu'il charge bien le fichier myfile avec le bon contenu.
f- Sur un navigateur accéder à l'url http://@IP_de-votre-conteneur est-ce que vous avez la page par défaut d'apache ? Si ce n'est pas le cas faite le nécessaire

3. *Publier son image sur un registry privé*

Pour publier une image, il faut tout d'abord se connecter à Docker Hub (créer un compte si vous ne l'avez pas déjà) à l'aide de la commande **docker login**. Ensuite la commande docker push permet de pousser une image locale vers un registre distant (Docker Hub ou un registre privé). Voici un exemple de commande pour pousser une image vers Docker Hub:

```
docker tag my_image_name my_dockerhub_username/new_image_name
docker push my_dockerhub_username/my_image_name
```

Dans cette commande, `my_image_name` est le nom de l'image locale que vous avez construite à l'aide de docker build et `new_image_name` c'est le nom que vous allez donner à l'image qui sera publiée. Le `my_dockerhub_username` est votre nom d'utilisateur Docker Hub. Vous devez d'abord "tagger" l'image avec votre nom d'utilisateur Docker Hub, puis exécuter la commande docker push pour pousser l'image vers Docker Hub.

Il est possible de préciser votre nom d'utilisateur lors de la construction de l'image avec :

```
docker build -t my_dockerhub_username/my_image_name:tag .
```

Dans le cas où vous avez affecté plusieurs tags (par exemple latest et un autre contenant le numéro de version), vous pouvez pousser l'ensemble des tags avec l'option -all :

```
docker push --all-tags my_dockerhub_username/new_image_name
```

a- Créer un dockerfile avec uniquement l'instruction From alpine
b- Créer une image à partir du dockefile en précisant le nom utilisateur de votre compte docker hub ainsi avec un tag à votre prénom.
c- Ajoutez un 2e tag à votre image avec la valeur 1.0.
d- Poussez l'image avec ces 2 tags sur votre registre docker hub.
e- Connectez-vous au registry pour vérifier qu'ils sont bien présents.

IV. Script d'automatisation bash pour créer et exécuter un conteneur Docker

Dans cette partie, vous allez écrire un script bash run_flask.sh et y ajouter des commandes permettant de créer et exécuter un conteneur Docker d'une application web basé sur flask. L'objectif c'est d'avoir une application web qui se lance automatiquement au démarrage du conteneur, qui charge son contenu (pages html et css) à partir des fichiers fournis lors de la création du conteneur.

Commencer tout d'abord par récupérer au niveau de votre VM le dossier Flask_app présent sur moodle, contenant les éléments suivants :

```
Flask_app/
├── flask_app.py
├── static
│   └── style.css
└── templates
    └── index.html
```

Le fichier flask_app.y a comme contenu :

```
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)

@sample.route("/")
def main():
   return render_template("index.html")
if __name__ == "__main__":
   sample.run(host="0.0.0.0", port=8080)
```

Ce code python permet à Flask de renvoyer le contenu de la page index.html lorsqu'une requête http est reçu sur le port 8080.

À présent vous allez créer votre script bash `run_flask.sh` qui doit effectuer les tâches suivantes :

1- Ajoutez le 'she-bang' et les commandes pour créer une structure de répertoires avec tempdir comme dossier parent pour stocker les fichiers du site Web.

2- Copiez les répertoires du site Web et le fichier flask_app.py dans les répertoires respectifs.

3- Maintenant vous devez créer un Dockerfile en utilisant les commande bash ‘echo’. Pour cela vous allez rediriger la sortie des echo vers le fichier Dockerfile qui sera utilisé plus tard pour construire le conteneur.

   a. Vous avez besoin de Python dans votre conteneur.

   b. Votre application a besoin de Flask, ajoutez donc la commande pour installer Flask dans le conteneur.

   c. Votre conteneur aura besoin des dossiers du site Web et du script flask_app.py pour exécuter l'application. Ajoutez donc les commandes nécessaires pour les ajouter à un répertoire dans le conteneur. Dans cet exemple, vous allez créer /home/myapp comme répertoire parent dans le conteneur Docker.

   d. Utilisez la commande Docker EXPOSE pour exposer le port 8080 à utiliser par le serveur Web.

   e. Enfin, ajoutez la commande permettant de faire exécuter le script Python.

4- Ajoutez les commandes dans votre scripte pour basculer vers le répertoire tempdir et créer dedans votre image avec le nom `tp1_np` (avec np sont vos initiales en minuscule)

5- Ajoutez la commande docker run à votre scripte bash pour démarrer le conteneur. Vous nommez le conteneur avec le nom `run_tp1_np`. Votre conteneur doit être lancé en mode détaché.

6- Ajoutez la commande docker ps -a pour vérifier que votre conteneur est en cours d'exécution. Cette commande sera la dernière exécutée par le script bash.

Une fois votre script bash est prêt, lancer-le :

- Accéder à votre conteneur qui est en cours d'exécution. Quelle commande vous devez utiliser ? Notez bien l'invite de commande qui devient root@containerID avec l'ID du conteneur correspond à l'ID affiché dans la sortie de docker ps -a.
- Depuis un navigateur accéder à votre application, c’est quoi le message d’accueil que vous obtenez ?
- Arrêtez puis supprimez votre conteneur avec les commandes docker stop et docker rm puis personnaliser le fichier index.html ainsi que le fichier css puis regénérer un nouveau conteneur.