LAB 3 - Manipulation des Dockerfile

Exercice 1 : Empaquetage de Git avec un Dockerfile

Commençons par revoir la création d'une image permettant d'empaqueter l'utilitaire git dans une image basée sur la dernière version du Ubuntu

1. Créez un fichier Dockerfile avec le contenu suivant :

```
FROM ubuntu:latest
MAINTAINER "devops@gk.fr"
RUN apt-get update && apt-get install -y git
ENTRYPOINT ["git"]
```

2. Créer l'image correspondant au Dockerfile avec le tag **`ubuntu-git-latest`**

```
docker build -t ubuntu-git-latest .
```

Exercice 2 : ENTRYPOINT et CMD

Le but de cet exercice est de vérifier les différences entre ENTRYPOINT et CMD.

1. Créez un fichier Dockerfile-v1 contenant les instructions suivantes :

```
FROM alpine
ENTRYPOINT ["ping"]
```

- Créez ensuite une image, nommée ping:1.0, à partir de ce fichier.

```
docker build -t ping:1.0 -f Dockerfile_v1 .
```

- Lancez maintenant un container basé sur l'image ping:1.0

```
docker run ping:1.0
```

- Observez le résultat
- Lancez un nouveau container en lui donnant l'adresse IP d'un DNS Google (8.8.8.8), en ajoutant également l'option -c 3 pour limiter le nombre de ping envoyés.

```
docker run  ping:1.0   8.8.8.8  -c3
```

- Observez de nouveau le résultat

2. Créez un fichier Dockerfile-v2 contenant les instructions suivantes :

```
FROM alpine
CMD ["ping"]
```

- Créez une image, nommée ping:2.0, à partir de ce fichier.

```
docker build -t ping:2.0 -f Dockerfile_v2 .
```

- Lancez maintenant un container basé sur l'image ping:2.0

```
docker run  ping:2.0
```

- Observez le résultat
- Lancez un nouveau container en lui donnant l'adresse IP d'un DNS Google (8.8.8.8), en ajoutant également l'option -c 3 pour limiter le nombre de ping envoyés.

```
docker run  ping:2.0   8.8.8.8  -c3
```

- Observez de nouveau le résultat

3. Créez un fichier Dockerfile-v3 contenant les instructions suivantes :

```
FROM alpine
ENTRYPOINT ["ping"]
CMD ["-c3", "localhost"]
```

- Créez une image, nommée ping:3.0, à partir de ce fichier.

docker build -t ping:3.0 -f Dockerfile_v3 .

- Lancez maintenant un container basé sur l'image ping:3.0

```
docker run  ping:3.0
```

```
docker run  ping:3.0 8.8.8.8
```

- Observez le résultat

Exercice 3 : Dockerizez un serveur web simple

Le but de ce premier exercice est de Dockerizez un serveur web simple de type nginx dans un conteneur basique.

1. Développez (en se basant sur le code suivant) un serveur HTTP qui expose le endpoint /ping sur le port 80 et répond par PONG.

Fichier 1 : pong.js

```
var express = require('express');
var app = express();
app.get('/ping', function(req, res) {
        console.log("received");
        res.setHeader('Content-Type', 'text/plain');
        res.end("PONG");
});
app.listen(80);
```

Fichier 1 : package.json

```
{
 "name": "pong",
 "version": "0.0.1",
```

```
 "main": "pong.js",
 "scripts": {
 "start": "node pong.js"
},
 "dependencies": { "express": "^4.14.0" }
}
```

2. Créez le fichier Dockerfile qui servira à construire l'image de l'application. Ce fichier devra décrire les actions suivantes

    - image de base: alpine:3.10
    - installation du runtime du langage choisi
    - installation des dépendances de l'application
    - copie du code applicatif
    - exposition du port d'écoute de l'application
    - spécification de la commande à exécuter pour lancer le serveur

```
FROM alpine:3.10
#MAJ des packages et install nodeJS
RUN apk update && apk add nodejs-npm

#Spécifier le répertoire courant de notre conteur
WORKDIR /app

#Copier les fichiers sources dans le dossier courant (/app)
COPY . .

# installer le Framework node JS en
RUN npm install

# Expsoer le port 80 du serveur web HTTP
EXPOSE 80

#Démarrer le Framework nodeJS
CMD ["npm","start"]
```

3. Construire l'image en la taguant pong:v1.0

```
docker build  -t pong:v1.0 .
```

4. Lancez un container basé sur cette image en exposant le port 80 du conteneur

```
docker run -d pong:v1.0
```

5. Inspecter le conteneur créé afin de déterminé l'IP privé alloué puis Tester l'application

```
docker inspect c8199bd54aca
```

```
curl 172.17.0.3/ping
```

Exercice 4 : Création d'une image à partir d'un container

1. Lancez un container basé sur une image alpine:3.8, en mode interactif, et en lui donnant le nom c1

```
docker run -ti --name c1 alpine:3.8
```

2. Lancez la commande curl google.com

```
/ # curl google.com
/bin/sh: curl: not found
```

3. Qu'observez-vous ?

4. Installez curl à l'aide du gestionnaire de package apk

```
/ # apk update && apk add curl
```

5. Quittez le container avec CTRL-P CTRL-Q (pour ne pas killer le processus de PID 1)

6. Créez une image, nommée myping, à partir du container c1

   Utilisez pour cela la commande ***commit*** (docker commit --help pour voir le fonctionnement de cette commande)

```
docker run -it myping:c1
```

7. Lancez un shell intéractif dans un container basée sur l'image myping et vérifiez que curl est présent

```
docker run -it myping:c1
```

Exercice 5 : Dokerfile pour une simple Application Web

1. Créer un répertoire local nommé WebApp

2. Télécharger les deux fichiers index.html et linux.png dans le répertoire déjà créé depuis le repository github suivant : **`https://github.com/medsalahmeddeb/docker`**

3. Créer un fichier Dockerfile dans le même répertoire qui permet d'automatiser les tâches suivantes en se basant sur la dernière image stable de Nginx :

    - Copier les deux fichiers index.html et linux.png dans le répertoire : **/usr/share/nginx/html**
    - Exposer les deux ports 80 et 443
    - Lancer la commande suivante **["nginx", "-g", "daemon off;"]** automatiquement avec le démarrage d'un conteneur en se basent sur l'image en question.

```
FROM nginx:1.14
COPY linux.png index.html /usr/share/nginx/html/
#COPY linux.png /usr/share/nginx/html

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

```
docker build -t webapp .
```

4. Tester l'accès à la page index.html en utilisant l'adresse IP de conteneur

```
docker inspect  020906ecaa47
```

Depuis un navigateur, lancez : http://172.17.0.7/

Exercice 6 : Script d'extraction de lien

1. Créer un répertoire local nommé PythonScript

2. Télécharger le script linkextractor.py dans le répertoire déjà créé

3. Créer un fichier Dockerfile dans le même répertoire qui permet d'automatiser les tâches suivantes en se basant sur l'mage python version 3

- Installer les deux bibliothèques beautifulsoup4 et requests (**pip install beautifulsoup4** et **pip install requests**)
- Le répertoire de travail pointe sur le chemin **/app**
- Copier le script linkextractor.py dans le répertoire de travail
- Rendre exécutable le script linkextractor.py
- Lancé le script avec le démarrage de conteneur

4. Lancer un conteneur en se basant sur l'image créé par ce Dockerfile en fournisant l'URL suivante comme argument d'entrée : http://example.com/

```
FROM python:3

MAINTAINER devPython@gk.fr
LABEL Projet  extration des liens

RUN pip install beautifulsoup4
RUN pip install requests

WORKDIR /app

COPY linkextractor.py /app

RUN ["chmod" ,"u+x", "linkextractor.py"]

ENTRYPOINT ["./linkextractor.py"]
```

```
docker build -t linkextractor .
```

```
docker run linkextractor http://example.com/
```