TRAVAUX PRATIQUES
DOCKER : DOCKERFILE

I. introduction

Dans ce TP, nous allons concevoir des conteneurs à l'aide d'un fichier Dockerfile permettant de modifier l'image de base de notre conteneur et de préciser un ensemble d'élément

II. premier fichier Dockerfile

Dans un répertoire, vous allez créer un fichier nommé **Dockerfile**, vous allez ajouter les ligne suivantes

```
FROM alpine
WORKDIR /tmp
COPY bonjour.txt .
ENTRYPOINT cat bonjour.txt
```

Créer aussi un fichier bonjour .txt dans le même répertoire à l'aide de la commande

```
echo "bonjour le monde!! " > bonjour.txt
```

Nous allons créer l'image à l'aide de ce fichier Dockerfile par la commande

```
docker build -t premier .
```

ne pas oublier le point à la fin de la commande qui indique le répertoire ou se situe le Dockerfile à builder. Cette commande génère l'image *premier* dont nous pouvons instancier un conteneur comme avant

```
docker run --rm premier
```

Détaillons ce premier fichier.

(1) L'instruction **FROM** permet de définir l'image sur laquelle se construit notre propre image.

(2) L'instruction **WORKDIR** correpond à l'option –**workdir** ou **-w** de docker pour changer le répertoire courant de l'application.

(3) L'instruction **COPY** permet de faire une copie du ficher bonjour.txt de la machine hôte vers le conteneur, comme un **docker cp**

(4) Enfin la commande **ENTRYPOINT** permet de préciser la commande à exécuter au démarrage du conteneur, c'est à dire celle qui sera en PID 1

III. contexte d'exécution

les instructions RUN, CMD et ENTRYPOINT qui vont être présenté après peuvent avoir 2 format d'écriture correspondant à deux modes d'exécution différents

(1) format terminal

```
CMD <commande>
```

qui exécute cette commande dans un terminal sous la forme

```
/bin/sh -c <commande>
```

CMD ["executable","param1","param2",...]

(2) format exécution qui exécute cette commande sans passer par un shell

IV. commande RUN

nous allons modifier le précédent fichier Dockerfile afin d'effectuer la création du fichier directement lors de la création de l'image. ous allons utiliser la commande **RUN** qui permet d'exécuter des commandes lors de la création. Donc dans le fichier Dockerfile, vous allez commentez les 2 lignes **WORKDIR** et **COPY** et y ajouter une instruction

```
3 RUN cd /tmp
4 RUN echo "bonjour le monde !!" > bonjour.txt
```

Vous pouvez à présent recréer une nouvelle image à l'aide de ce Dockerfile. Que se passe t'il lors de l'exécution ? Que faut il corriger pour que cela fonctionne ?

Détaillons la commande RUN. Cette commande permet d'exécuter une action au sein de notre conteneur dans un shell (nous verrons qu'il est possible d'exécuter des commandes sans appel au shell). Il y a donc un changement de répertoire et la création du fichier en 2 commandes séparées. Vous avez pu remarquer que la création de l'image passe donc par 2 couches distinctes ajouter à notre image de base (rappel sur l'UFS dans le cours), chacune correspondant à l'enregistrement sur une couche en lecture seule des modifications apportée à la couche d'écriture du conteneur intermédiaire. Nous pouvons réunir ces deux commandes en une seule instruction **RUN** en utilisant le double &

```
5 RUN cd /tmp \
6 && echo "bonjour le monde !!" > bonjour.txt
```

Que cela change t'il au niveau de l'UFS.

V. L'instruction CMD

Cette instruction permet d'exécuter une ou plusieurs commande lors du démarrage du conteneur. Si une instruction ENTRYPOINT existe, l'instruction CMD qui la suit voit ses éléments ajouter à la commande définie en ENTRYPOINT

Dans l'exemple précédent, remplacer l'instruction ENTRYPOINT par cette la même commande mais en CMD. Que constatez vous ? Si vous instanciez un conteneur avec la commande suivante que se passe t'il

```
7 docker run --rm --entrypoint "ps aux" #nom_image
```

VI. exposition des ports

Nous allons à présent partir d'une image Nginx. Il faudra donc exposer le port 80 du conteneur afin d'accéder au serveur web. Dans un nouveau répertoire, vous allez aussi créer un fichier html d'accueil sur serveur et allez le copier dans le répertoire par défaut de Nginx (/usr/share/www/.. à vérifier sur débian 9) Vous n'avez pas besoin de préciser un ENTRYPOINT, nous utiliserons celui défini par défaut dans l'image nginx de base. Une fois l'image créée, démarrez un conteneur de cette image.

VII. monter un volume

Toujours avec le même Dockerfile, vous allez à présent ajouter un instruction **VOLUME** pour préparer un volume permettant d'accéder au fichier html copié au préalable pour le modifier. Cette instruction **VOLUME** ne prend que le chemin de montage du volume dans le conteneur, le volume monté sera passé en argument de la commande **docker run** démarrant le conteneur

Tester plusieurs configurations de volumes comme lors du précédent TP

VIII. mise en place d'une application django.

Vous avez à votre disposition dans moodle une application Django fonctionnelle. Ecrivez le Dockerfile permettant de la déployez à partir de l'archive zip fourni et qui démarre l'application à partir du serveur interne à Django.