TP Docker

Partie 1 :

1 – Téléchargez la dernière image ubuntu (latest) à partir du docker hub registry
$ docker pull ubuntu :latest

2 – Listez les images disponibles
$ docker images

3 – Créer un conteneur basé sur l'image Ubuntu (Instanciation de l'image ubuntu). Voici la liste des arguments que nous utiliserons dans cette partie :
-t : Fournit un terminal virtuel TTY au docker
-i : qui permet d'attacher le terminal à l'entrée standard du processus que l'on lance. Permet d'écrire (STDIN) dans le conteneur (couplé à -t)
-d : Exécute le conteneur en arrière plan
-v : Permet de monter un répertoire local sur le conteneur
-p : Permet de binder un port sur le conteneur vers un port sur le host
-e : Permet l'ajout d'une variable d'environnement
--name : Donne un nom au conteneur
--rm : Détruit le conteneur une fois terminé
-w : Choisit le répertoire courant (dans le conteneur)
--link : Permet de faire un lien entre deux conteneurs

Dans notre cas nous avons besoin d'une Tty (option -t) et du mode interactif (option -i) pour interagir avec notre conteneur basé sur l'image Ubuntu. Ces 2 options sont généralement passées ensemble et elles permettent de "connecter" son terminal au conteneur (au processus lancé dans le conteneur plus précisément).

Tentons alors l'exécution de ces deux options :
$ docker run -ti --name ubuntucontainer ubuntu /bin/bash

Cette commande permet de lancer le conteneur et ensuite vous dirigez vers un pseudo terminal (TTY) avec une invite de commande (mode interactif).

Tapez quelques commandes linux comme ls, pwd, df –h, …..

Lorsque l'on lance la commande docker run, Docker va d'abord créer un conteneur à partir de l'image choisie, puis le démarrer pour le mettre dans un état running. Quand le processus à l'intérieur du conteneur aura terminé de s'exécuter, le conteneur va s'arrêter et deviendra exited.

4- Faites **$ Docker ps** pour vérifier les conteneurs actifs. Est ce que le conteneur **ubuntucontainer** figure dans la liste ?
……

Enfin tapez la commande **exit** pour sortir du conteneur. Est ce que le conteneur **ubuntucontainer** figure dans la liste ?

5 – Listez les conteneurs non actifs (**docker ps –a**). Que remarquez vous ?
……

Mode détaché :
Il est possible de lancer un conteneur en mode détaché, c'est d'ailleurs le cas le plus fréquent. Par exemple si l'on veut faire tourner un service web et une base de données en arrière-plan sur le serveur. Pour cela on utilise simplement l'option -d.

Par exemple la commande
docker run -d **ubuntu** sleep 30
va lancer, en arrière-plan, un processus sleep 30 dans un conteneur. Une fois le processus terminé le conteneur s'arrêtera.

Partie 2 (Gérer le conteneur)

Maintenant que l'on sait démarrer nos conteneurs, il serait intéressant de pouvoir contrôler leur cycle de vie. En effet un conteneur Docker peut-être dans plusieurs états :

- running indique que le conteneur est en cours d'exécution
- exited indique le conteneur a terminé son exécution
- paused indique que le conteneur est mis en pause dans son exécution
- d'autres états de transition (created, restarting, removing) ou d'erreur (dead) qui ne seront pas approfondis ici

Pour démarrer un conteneur qui est arrêté, on utilisera tout simplement docker start NOM_CONTENEUR.
De la même manière c'est avec docker stop que l'on peut stopper le conteneur prématurément:

$ docker start ubuntucontainer
Tapez maintenant : **$ docker ps**
Que remarquez vous ?

6 – Il est possible de lancer des commandes directement dans un conteneur en cours d'exécution, en parallèle du processus courant.
Ceci est possible grâce à la commande docker exec -it NOM_CONTENEUR COMMANDE, elle est très souvent utilisé lorsqu'il est nécessaire d'ouvrir un shell dans un conteneur.

Vous êtes maintenant à l'intérieur de votre conteneur ubuntu via un shell.
Profitons-en pour télécharger l'outil git :
$ docker exec -it ubuntucontainer apt- apt-get install -y git
Vérifier :
$ docker exec -it ubuntucontainer git --version

7- Stopper puis supprimer le conteneur Ubuntu

On peut aussi supprimer rapidement les conteneurs en statut *Exited* avec la commande `docker container rm NOM_CONTENEUR`. Il est aussi possible d'indiquer à Docker de supprimer automatiquement un conteneur lorsque celui-ci se termine, grâce à l'option `--rm` dans la commande `docker run`

8 – Essayez d'utiliser l'option **--rm** lors de la création d'un docker. Faites des commandes **docker ps** pour voir ce qui se passe.

Partie 3 (manipulation port mapping)

Lancez un conteneur Apache (image httpd:latest) avec le nom « monServeurWeb » et qui tourne en mode arrière-plan. Pensez à utiliser aussi la redirection de port 8080 du host vers le port exposé par le conteneur (port 80)

……

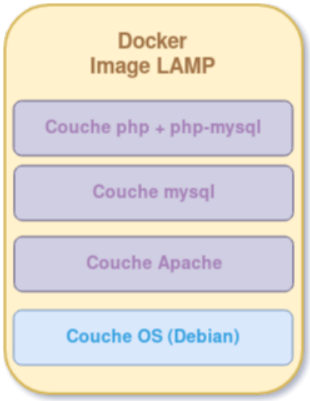
Partie 4 : DockerFile

Nous allons créer notre propre stack LAMP (Linux Apache MySQL PHP) au moyen de Docker.

Voici les différentes couches de cette image :

- Une couche OS pour exécuter notre Apache, MySQL et Php, je vais me baser sur la distribution Debian.
- Une couche Apache pour démarrer notre serveur web.
- Une couche php qui contiendra un interpréteur Php mais aussi les bibliothèques qui vont avec.
- Une couche Mysql qui contiendra notre système de gestion de bases de données.

Voici le schéma de notre image :



1 - Création des sources et du Dockerfile

Commencez par créer un dossier et téléchargez les sources de l'image :

$ mkdir LAMP puis $ cd LAMP

$ wget https://devopssec.fr/documents/docker/dockerfile/sources.zip

$ unzip sources.zip

Ensuite dans la racine du dossier que vous venez de créer, créez un fichier et nommez le **Dockerfile**.
Voici l'arborescence que vous êtes censé avoir :

- *db* : contient un fichier *articles.sql*, qui renferme toute l'architecture de la base de données.
- *app* : comporte les sources php de notre l'application web

Dans le fichier **Dockerfile**, rajoutez le contenu suivant :

```
# ---------- DÉBUT COUCHE OS ----------
FROM debian:stable-slim
# ---------- FIN COUCHE OS ----------

# MÉTADONNÉES DE L'IMAGE
LABEL version="1.0" maintainer="GI3"

# VARIABLES TEMPORAIRES
ARG APT_FLAGS="-q -y"
ARG DOCUMENTROOT="/var/www/html"
# ---------- DÉBUT COUCHE APACHE ----------
RUN apt-get update -y && \
   apt-get install ${APT_FLAGS} apache2
# ---------- FIN COUCHE APACHE ----------

# ---------- DÉBUT COUCHE MYSQL ----------
RUN apt-get install ${APT_FLAGS} mariadb-server
COPY db/articles.sql   /
# ---------- FIN COUCHE MYSQL ----------
# ---------- DÉBUT COUCHE PHP ----------
RUN apt-get install ${APT_FLAGS} \
   php-mysql \
   php && \
   rm -f ${DOCUMENTROOT}/index.html && \
   apt-get autoclean -y

COPY app ${DOCUMENTROOT}
```

```
# ---------- FIN COUCHE PHP ----------

# OUVERTURE DU PORT HTTP
EXPOSE 80

# RÉPERTOIRE DE TRAVAIL
WORKDIR  ${DOCUMENTROOT}

# DÉMARRAGE DES SERVICES LORS DE L'EXÉCUTION DE L'IMAGE
ENTRYPOINT service mysql start && mysql < /articles.sql && apache2ctl -D FOREGROUND
```

Remarque :
il faut savoir qu'un conteneur **se ferme automatiquement à la fin de son processus principal**. Il faut donc un processus qui tourne en premier plan pour que le conteneur soit toujours à l'état running, d'où le lancement du service Apache en premier plan à l'aide de la commande **apache2 -D FOREGROUND**.

2- Construction et Exécution de notre image

Voici la commande pour qui nous permet de construire une image docker depuis un Dockerfile :

$ docker build -t <IMAGE_NAME> .

Ce qui nous donnera :

$ docker build -t my_lamp .

Ensuite, exécutez votre image personnalisée :

$ docker run -d --name my_lamp_c -p 8080:80 my_lamp

Visitez ensuite la page suivante http://localhost:8080/

3 - Publier son image dans le Hub Docker

a - Créer un repository public dans docker hub

b - L'étape suivante est de se connecter au hub Docker à partir de la ligne de commande

$ docker login
Il va vous demander, votre nom d'utilisateur et votre mot de passe, et si tout se passe bien vous devez avoir le message suivant : Login Succeeded

c- Récupérer ensuite l'id ou le nom de votre image :
$ docker images ls

d - Ensuite il faut rajouter un tag à l'id ou le nom de l'image récupérée. Il existe une commande pour ça :
$ docker tag my_lamp account/lamp:first

E - Maintenant pousser votre image vers le Hub Docker grâce à la commande suivante :
$docker push account/lamp:first

Amélioration de notre image LAMP avec le concept de Volume

L'image avec stack LAMP, malheureusement, ce n'est pas top niveau persistance de données car, lors d'un redémarrage du conteneur, nous allons rencontrer les deux problèmes suivants

- Les données de notre base ne sont pas sauvegardées.
- Les modifications des sources de notre application ne seront pas appliquées.

Pour résoudre ce problème, nous allons utiliser les volumes ! Créer un nouveau dossier et Commencez par télécharger le projet :
$ wget https://devopssec.fr/documents/docker/volumes/project.zip

$ unzip project.zip

$ cd project

Dans ce projet, il s'agit du même Dockerfile, mais juste on a changé les fichiers sources en rajoutant un formulaire.
Dans le fichier Dockerfile, mettre cette instruction en mode commentaire :
```
# COPY app ${DOCUMENTROOT}
```

Créer votre image :

$ docker build -t my_lamp2 .

Concernant la base de données, nous allons créer un volume nommé "mysqldata" :

$ docker volume create mysqldata

Lister les volumes :

$ docker volume ls

Nous allons aussi changer le dossier source en utilisant le concept de volume. Lançons donc notre conteneur :

$ docker run -d --name my_lamp2_c -v $PWD/app:/var/www/html -v mysqldata:/var/lib/mysql -p 8080:80 my_lamp2

La commande $PWD prendra automatiquement le chemin absolu du dossier courant, donc faites bien attention à exécuter votre image depuis le dossier du projet où mettez le chemin complet si vous souhaitez lancer votre commande depuis n'importe quelle autre chemin.

Remarque :
Supprimer tous les volumes locaux non inutilisés
docker volume prune
Supprimer un conteneur Docker avec le/les volumes associés
docker rm -v <CONTAINER_ID ou CONTAINER_NAME>
-v ou --volume : supprime les volumes associés au conteneur

Annexe

Rappel pour les instructions DockerFile les plus communément utilisées.

- `FROM` : Définit l'image de base qui sera utilisée par les instructions suivantes.
- `LABEL` : Ajoute des métadonnées à l'image avec un système de clés-valeurs, permet par exemple d'indiquer à l'utilisateur l'auteur du Dockerfile.
- `ARG` : Variables temporaires qu'on peut utiliser dans un Dockerfile.
- `ENV` : Variables d'environnements utilisables dans votre Dockerfile et conteneur.
- `RUN` : Exécute des commandes Linux ou Windows lors de la création de l'image. Chaque instruction `RUN` va créer une couche en cache qui sera réutilisée dans le cas de modification ultérieure du Dockerfile.
- `COPY` : Permet de copier des fichiers depuis notre machine locale vers le conteneur Docker.
- `ADD` : Même chose que COPY mais prend en charge des liens ou des archives (si le format est reconnu, alors il sera décompressé à la volée).
- `ENTRYPOINT` : comme son nom l'indique, c'est le point d'entrée de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur. Il prend la forme de tableau JSON (ex : CMD ["cmd1","cmd1"]) ou de texte.
- `CMD` : Spécifie les arguments qui seront envoyés au `ENTRYPOINT`, (on peut aussi l'utiliser pour lancer des commandes par défaut lors du démarrage d'un conteneur). Si il est utilisé pour fournir des arguments par défaut pour l'instruction `ENTRYPOINT`, alors les instructions `CMD` et `ENTRYPOINT` doivent être spécifiées au format de tableau JSON.
- `WORKDIR` : Définit le répertoire de travail qui sera utilisé pour le lancement des commandes `CMD` et/ou `ENTRYPOINT` et ça sera aussi le dossier courant lors du démarrage du conteneur.
- `EXPOSE` : Expose un port.
- `VOLUMES` : Crée un point de montage qui permettra de persister les données.
- `USER` : Désigne quel est l'utilisateur qui lancera les prochaines instructions `RUN`, `CMD` ou `ENTRYPOINT` (par défaut c'est l'utilisateur root).

Quelle est la différence entre ENV et ARG dans un Dockerfile ?

Ils permettent tous les deux de stocker une valeur. La seule différence, est que vous pouvez utiliser l'instruction `ARG` en tant que variable temporaire, utilisable qu'au niveau de votre Dockerfile, à l'inverse de l'instruction `ENV`, qui est une variable d'environnements accessible depuis le Dockerfile et votre conteneur. Donc privilégiez `ARG`, si vous avez besoin d'une variable temporaire et `ENV` pour les variables persistantes. `ARG` définira les environnements uniquement pendant la construction. L'instruction `ENV` permet de définir des variables d'environnements

qui pourront ensuite être modifiées grâce au paramètre de la commande `run --env <key>=<value>`

Quelle est la différence entre COPY et ADD dans un Dockerfile ?

Ils permettent tous les deux de copier un fichier/dossier local vers un conteneur. La différence, c'est que `ADD` autorise les sources sous forme d'url et si jamais la source est une archive dans un format de compression reconnu (ex : zip, tar.gz, etc ...), alors elle sera décompressée automatiquement vers votre cible. Notez que dans les best-practices de docker, ils recommandent d'utiliser l'instruction `COPY` quand les fonctionnalités du `ADD` ne sont pas requises.

Quelle est la différence entre RUN, ENTRYPOINT et CMD dans un Dockerfile ?

- L'instruction `RUN` est **exécutée pendant la construction de votre image**, elle est souvent utilisée pour installer des packages logiciels qui formeront les différentes couches de votre image.
- L'instruction `ENTRYPOINT` est **exécutée pendant le lancement de votre conteneur** et permet de configurer un conteneur qui s'exécutera en tant qu'exécutable. Par exemple pour notre stack LAMP, nous l'avions utilisée, pour démarrer le service Apache avec son contenu par défaut et en écoutant sur le port 80.
- L'instruction `CMD` est aussi **exécutée pendant le lancement de votre conteneur**, elle définit les commandes et/ou les paramètres de l'instruction `ENTRYPOINT` par défaut, et qui peuvent être surchargées à la fin de la commande `docker run`.