Activité n°5: Création des images et les Dockerfiles

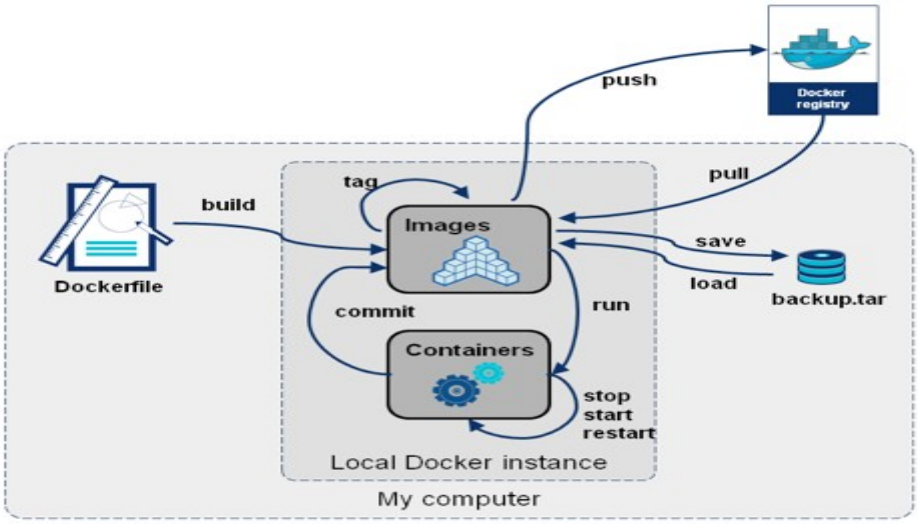
Création d’une image à partir d’un conteneur

1. Démarrez le conteneur tenstenvvar.
2. Ajoutez git et vim à votre conteneur.
3. Quittez votre conteneur.
4. Exécutez la commande `$docker ps` et localisez l’id de votre conteneur
5. Créez l’image "m2cloud" à partir de votre conteneur. Voir la documentation docker commit: *https://docs.docker.com/engine/reference/commandline/commit/*
6. Exécutez `$docker image ls` ou `$docker images` et vérifiez que votre image figure bien dans la liste des images.
7. Lancez un conteneur test à partir de l’image que vous avez crée.
8. Exécutez un shell bash interactif sur votre conteneur et vérifiez que vim et git sont bien installées
9. Quitter le shell exécuté sur votre conteneur et Essayez de supprimer l’image m2cloud?Est-ce possible?Que faut-il faire?

Création d’une image avec Dockerfile

Le moteur Docker comprend des outils qui automatisent la création d'images de conteneur. Rien ne vous empêche de créer des images de conteneur manuellement en exécutant la commande docker commit, mais l'adoption d'un processus de création d'image automatisé présente de nombreux avantages, parmi lesquels:

- stockage d’images de conteneur sous forme de code;
- nouvelle création rapide et précise d’images de conteneur à des fins de maintenance et de mise à niveau;
- intégration continue entre les images de conteneur et le cycle de développement.



C’est quoi un DockerFile

- Dockerfiles sont des fichiers utilisés pour créer des images Docker par programmation.
- Ils vous permettent de créer rapidement et de manière reproductible une image Docker.
- Les fichiers Docker contiennent des instructions pour créer une image Docker. Chaque instruction est écrite sur une ligne et est donnée sous la forme **<INSTRUCTION><argument(s)>** .
- Souvent, une instruction de fichier Dockerfile doit occuper plusieurs lignes. Pour cela, vous pouvez utiliser un caractère d'échappement. Le caractère d’échappement de fichier Dockerfile par défaut est une barre oblique inverse \.
- Les fichiers Docker sont utilisés pour créer des images Docker à l'aide de la commande docker build: **docker build -t nomimage:version .**

Listes des instructions

- ➔ **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 ou un autre conteneur 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).

Exemple1:

Un fichier Dockerfile minimal ressemble à ceci:

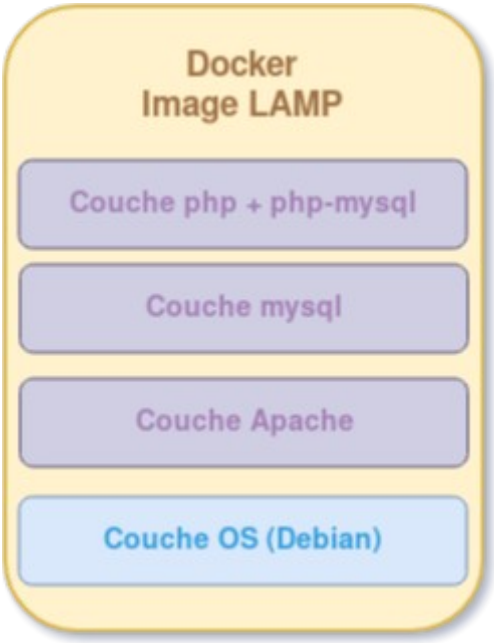
```
FROM alpine
CMD ["echo", "Hello StackOverflow!"]
```

Cela indiquera à Docker de créer une image basée sur Alpine (FROM), une distribution minimale pour les conteneurs et d'exécuter une commande spécifique (CMD) lors de l'exécution de l'image résultante.

Exemple2:«https://devopssec.fr/article/creer-ses-propres-images-docker-dockerfile»

Création d’un 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, on se base 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.



Créer un dossier et téléchargez les sources del'image:
https://devopssec.fr/documents/docker/dockerfile/sources.zip
Désarchivez le fichier zip, et mettez les dossiers suivants dans votre nouveau dossier :

- 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.

Ensuite dans la racine du dossier que vous venez de créer, créez un fichier et nommez le Dockerfile, puis rajoutez le contenu suivant :

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

# MÉTADONNÉES DE L'IMAGE
LABEL version="1.0" maintainer="AJDAINI Hatim <ajdaini.hatim@gmail.com>"

# 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
```

Manipulations

Refaites l'exercice de l'activité n°5 mais en utilisant un Dockerfile.

Dockerfile multi-stage

Le build multi-stage est une fonctionnalité ajoutée dans Docker à compter de la version 17.05 et qui permet de décrire dans un seul et même Dockerfile l'ensemble des opérations nécessaires à la construction d'une application finale.

Ce que va faire Docker est de créer des conteneurs intermédiaires qui ne sont ni plus ni moins que de nouveaux layers et permettre à l'utilisateur de les réutiliser. Cette possibilité de réutilisation est la clé du mécanisme de multi-stage.

Qu'est-ce qu'un fichier WAR ?

Le fichier WAR est une archive Java. Il peut contenir des données propres à différents ensembles :

- Java Server Pages ;
- Fichiers XML : descripteurs de déploiement… ;
- Servlets ;
- Détails de configuration de sessions ;
- Données Java ;
- Pages web statiques de type JavaScript ou encore HTML.

Dans quel cas utilise-t-on un fichier au format WAR ?

À partir d'un serveur d'applications un fichier WAR sert principalement au déploiement d'une application web.

Manipulations

On souhaite déployer une application war qui correspond à un site web. L'application se trouve dans un dépôt accessible à travers ce lien: https://github.com/priximmo/war-hello-maven

Package nécessaire au déploiement:

- git clone
- maven : mvn package
- tomcat : webserver

NB:

- Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs, etc.). Ce POM se matérialise par un fichierpom.xml à la racine du projet. Cette approche permet l'héritage des propriétés du projet parent. Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent. Ceci introduit le concept de réutilisation de configuration. Le fichier pom du projet principal est nommé pom parent. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus.
- Apache Maven (couramment appelé Maven) est un outil de gestion et d'automatisation de production des projets logiciels Java en général et JEE en particulier. Il est utilisé pour automatiser l'intégration continue lors d'un développement de logiciel.