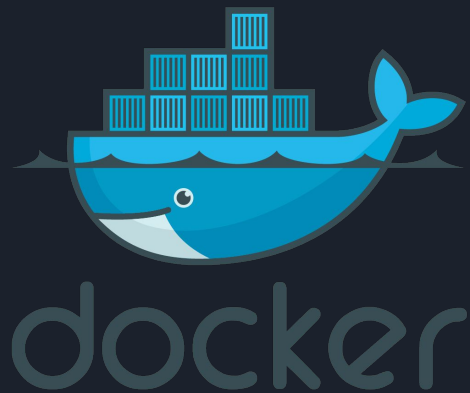


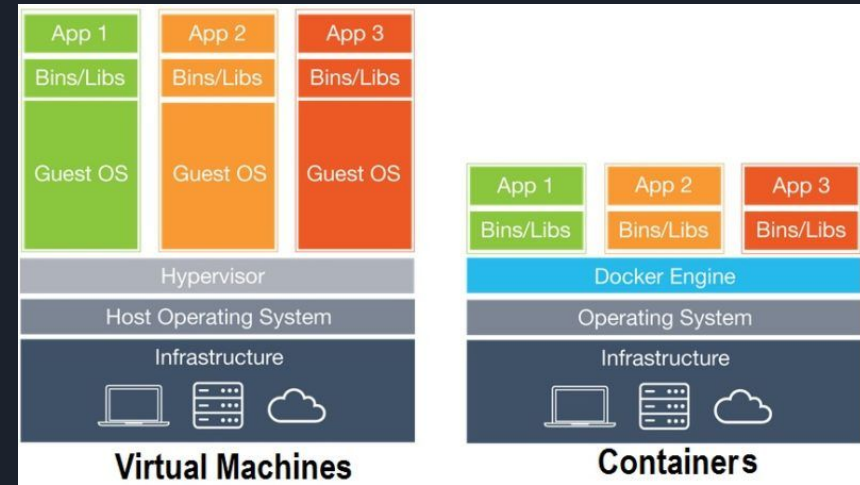
Prise en main de Docker



Présenté par Luca et Martin

Introduction

Pourquoi utiliser Docker?



- MV : exécuter de nombreux systèmes d'exploitation sur un seul et même système
- *Avantage container* : les *containers* se partagent le même noyau de système d'exploitation et isolent les processus de l'application du reste du système

Plus efficient en terme de consommation de ressources système :

- virtualiser le hardware comme l'hyperviseur vs
- virtualiser le système d'exploitation



Les commandes de base

Commandes de monitoring

`docker ps (-a)`

`docker images (-a)`

`docker network ls`

`docker-compose ps`



Les commandes de runtime

On a maintenant besoin d'images et de conteneurs pour tester les commandes précédentes

```
docker-compose up (-d) (--build)
```

```
docker-compose stop
```

```
docker run (-d) (-p <hostPort>:<containerPort>) (--name <NAME>) <IMGNAME>/<IMGID>
```

```
docker start <ID>/<NAME>
```

```
docker stop <ID>/<NAME>
```



Les commandes de suppression

Ces commandes permettent de supprimer vos conteneurs et vos images
Libérer de l'espace disque

```
docker rm <ID>/<NAME>
```

```
docker-compose rm
```

```
docker rmi <ID>/<NAME>
```



Qu'est ce qu'un dockerfile et comment se constitue-t-il?

Un Dockerfile est un fichier texte avec lequel vous allez pouvoir donner à Docker les instructions nécessaires pour pouvoir créer une image.

Ils décrivent les différentes étapes de création d'un conteneur totalement personnalisé.

Un Dockerfile se rédige grâce à des instructions précises tel que : FROM, RUN, ADD, WORKDIR, EXPOSE, VOLUME, CMD


Comment créer une image Docker?

Une image est un modèle composé de plusieurs couches, ces couches contiennent notre application ainsi que les fichiers binaires et les bibliothèques requises.

Création d'un stack LAMP à l'aide de docker :

- Une couche php qui contiendra un interpréteur Php
- Une couche Mysql qui contiendra notre SGBD.
- Une couche Apache pour démarrer notre serveur web.
- Une couche OS pour exécuter notre Apache, MySQL et Php





Une bonne pratique veut que l'on sépare l'image de l'application web de l'image de la base de données.

Dans le dockerfile :

```
# ----- 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
```


- Création de la couche OS (basée sur l'image debian-slim)

```
FROM debian:stable-slim
```

- Ajout des métadonnées de mon image

```
LABEL version="1.0" maintainer="AJDAINI Hatim <ajdaini.hatim@gmail.com>"
```

- création de 2 variables temporaires inhérentes au Dockerfile

```
ARG APT_FLAGS="-q -y"  
ARG DOCUMENTROOT="/var/www/html"
```

- Création de la couche Apache

```
RUN apt-get update -y && \  
apt-get install ${APT_FLAGS} apache2
```

- téléchargement du service mysql et ajout du fichier articles.sql pour le futur nouveau conteneur

```
RUN apt-get install ${APT_FLAGS} mariadb-server  
COPY db/articles.sql /
```

- Installation de l'interpréteur php ainsi que le module php-mysql

```
RUN apt-get install ${APT_FLAGS} \  
    php-mysql \  
    php && \  
    rm -f ${DOCUMENTROOT}/index.html && \  
    apt-get autoclean -y  
  
COPY app ${DOCUMENTROOT}
```

- Ouverture du port HTTP

```
EXPOSE 80
```

- Initialisation du répertoire de travail

```
WORKDIR /var/www/html
```



```
ENTRYPOINT service mysql start && mysql < /articles.sql && apache2ctl -D FOREGROUND
```

Lors du lancement du conteneur, le service mysql démarrera et construira l'architecture de la base de données grâce au fichier articles.sql .

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.

- 
- Construction d'une image docker depuis un Dockerfile

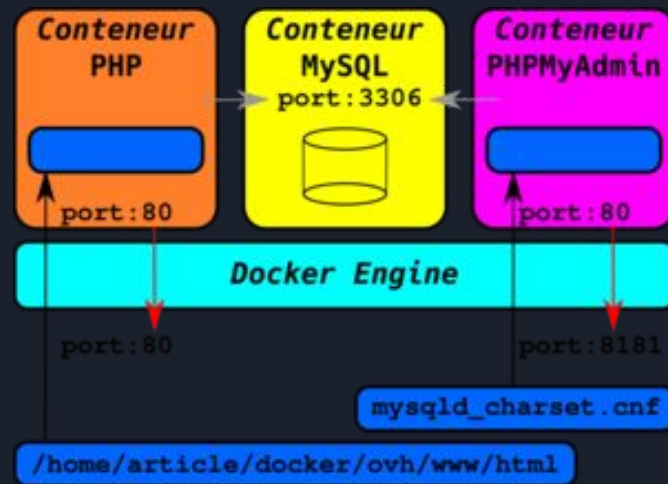
```
docker build -t my_lamp .
```

- Exécution de l'image personnalisée

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

Exemple d'un environnement avec Docker : environnement LAMP

Environnement de développement LAMP
(Linux+Apache+MySQL+PHP)



Le conteneur MySQL

```
db:  
image: mysql  
environment:  
  MYSQL_ROOT_PASSWORD: Secret  
  MYSQL_DATABASE: mydb  
  MYSQL_USER: mydb_user  
  MYSQL_PASSWORD: mydb_password  
volumes:  
  - /home/article/docker/ovh/mysqld_charset.cnf:/etc/mysql/conf.d/mysqld_charset.cnf
```



Conclusion

- Docker est une solution efficace de déploiement d'une application, de manière adaptable, sur n'importe quel serveur.
- Ses avantages résident aussi dans la préparation d'environnement de développement faciles à déployer grâce aux conteneurs.