

Docker

Mise en situation

- Mise en place de Docker pour le labo3 RES 2018
- Réalisé depuis une machine Linux

Init

- Créer un dossier à la racine du projet (mais peut être créé n'importe où, il faut juste le retrouver facilement)
 - Ou il y aura le fichier *Dockerfile* et le dossier *src* ou il y a le *.jar* (ou les exécutable que docker devra lancer) - Ceci est juste une bonne manière qui facilite les prochaines opérations
- **1 - Créer un fichier Dockerfile**

```
#Docker server Java
FROM java:8

ADD src /opt/src/

WORKDIR /opt/src/

ENTRYPOINT ["java", "-jar", "QuizRouletteServer-code-1.0-SNAPSHOT-standalone.jar"]
```

- **FROM** : image sur lequel notre image va se baser (*java* est l'image de base et *8* la version)
 - pour mettre une image différente de java, mettre après les : le nom de la version/ vm
Exemple:
- ```
FROM java:openjdk-8-jre-alpine
```
- Alpine est une vm très rapide et très légère
  - Voir site: [www.hub.docker.com](http://www.hub.docker.com)
  - **ADD**: ajoute ce qu'il y a dans le dossier *src* (local au dossier ou il y a le Dockerfile) à l'intérieur du conteneur au chemin: */opt/src*
  - **WORKDIR**: Point d'entrée dans le container par défaut
  - **ENTRYPOINT**: lance une commande (très ressemblant et peut être remplacé par **CMD**). Chaque mot de la commande doivent être séparé. Différence entre **ENTRYPOINT** et **CMD**:  
<https://stackoverflow.com/questions/21553353/what-is-the-difference-between-cmd-and-entrypoint-in-a-dockerfile> (savoir la différence, potentiellement dans un test :3 ...) ou  
<http://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/>
    - RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.

```
RUN ["apt-get", "install", "python3"]
```

- CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.

```
CMD ["java", "-jar", "server.jar"]
```

Lors du lancement du container il est pas possible de passé des arguments à notre programme

mais peut être écrasser en donner une autre commande:

```
docker run -p 1313:1313 labo2-server-java java -jar server.jar 1313
```

Ou `java -jar server.jar "1313"` va écraser le CMD du Dockfile

- ENTRYPOINT configures a container that will run as an executable.

```
ENTRYPOINT ["java", "-jar", "server.jar"]
```

Si notre `server.js` peut prendre des arguments (exemple: numéro de port). il est possible de lui spécifié lors du `docker run`

```
docker run -p 1313:1313 labo2-server-java 1313
```

ou "1313" est l'argument passé au jar. Ceci n'est pas possible avec *CMD*

- **Build l'image (depuis dossier ou il y a le Dockerfile):**

```
docker build -t labo2-server-java .
```

- **-t** pour spécifier un nom à notre nouvelle image.
- **labo2-server-java** : nom de l'image que je voulais créé
- **.** dossier dans lequel est mon Dockerfile (ici dans le dossier courant)

- **Voir si l'images à bien été créé:**

```
docker images
```

- **Run l'image**

```
docker run -p 1313:1313 labo2-server-java
```

- **-p** : pour spécifier les ports (port-machine-locale:port-container)
  - Le *port de la machine* local permet de ce connecter depuis l'extérieur
  - Le *port du container* est le port sur lequel le processus du container écoute
    - On peut par exemple avoir plusieurs instance d'un serveur qui écoute sur le meme port mais doit avoir un *port machine* différent pour les différencier et ce connecter à

chaqu'un.

- `-d`: permet de faire run en arrière plan (utile pour des scripts)
- `--rm` permet de dire que dès qu'il est stoper ce supprimer

- **Autre commande utile:**

- Pour récupérer l'id de container ou pour voir quelle container sont en cours d'exécution on peut lancer:

```
docker ps
```

- Possible de voir tous les containers même stopper

```
docker ps -a
```

- Lorsqu'un container est en cours d'exécution, il est possible de lui passer des commandes grâce à:

```
docker exec -it <id du container ou nom> <cmd>
```

Utile pour lancé un bash par exemple:

```
docker exec -it 804b9b257085 /bin/bash
```

*Attention* selon l'image (exemple alpine) il n'y a pas de bash mais on peut lancer un `/bin/sh`

- Pour kill un container on peut:

```
docker kill <id>
```

Pour les stopper tous les container en cours d'exécution:

```
docker kill $(docker ps -q)
```

**ATTENTION:** ca stop le container mais le supprime pas (on peut les voir avec `docker ps -a`)

- Pour les remove:

```
docker rm <container>
```

## Utile

- A chaque fois qu'on compile le projet, il faut recopier le jar dans le dossier `docker-server/src/` et rebuild le Dockerfile (recopie le jar dans le contener(**ADD**)) et le **run**.
- Cette manipulation étant ennuyeuse, faire un script à la racine du projet est une solution de facilitée:

```

echo "copy of jar file in docker-server/src"
cp QuizRouletteServer-build/QuizRouletteServer-code/target/QuizRouletteServer-
code-1.0-SNAPSHOT-standalone.jar docker-server/src/

echo "build of Dockerfile"
docker build -t labo2-server-java ./docker-server

echo "run image"
docker run -p 8080:1313 labo2-server-java

```

- Il sera ensuite possible d'ajouter d'autres images à *copier, rebuild* et *run* (exemple: le client).

*Script avec client et création de dossier:*

```

mkdir docker-server/src
echo "copy of jar file in docker-server/src/"
cp -f QuizRouletteServer-build/QuizRouletteServer-code/target/QuizRouletteServer-
code-1.0-SNAPSHOT-standalone.jar docker-server/src/

mkdir docker-client-node
echo "copy of client.js (client) in docker-client-node/src/"
cp -f QuizRouletteClient/client.js docker-client-node/src/

mkdir docker-client-node/data
echo "copy of data"
cp -f data/RES.csv docker-client-node/data/

echo "build of Dockerfile server"
docker build -t labo2-server-java ./docker-server

echo "build of Dockerfile client"
docker build -t labo2-client-node ./docker-client-node

echo "run image server"
docker run -d -p 1313:1313 labo2-server-java

echo "run image client"
docker run -p 8080:8080 labo2-client-node

```

## Dockfile Cmd

- **ADD:** Ajoute des fichiers locaux à la machine dans le container.
- **COPY:** Même chose que ADD mais en plus transparent (souvent préféré)
- **FROM:** Utilise le repo officiel pour la base de notre image. La doc Docker recommande les image *Alpine*
- **LABEL:** Ajoute un label à l'image pour permettre d'organiser les images par projet
- **RUN:**

RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.

```
RUN ["apt-get", "install", "python3"]
```

- **CMD:**

CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.

```
CMD ["java", "-jar", "server.jar"]
```

Lors du lancement du container il est pas possible de passé des arguments à notre programme mais peut être écrasser en donner une autre commande:

```
docker run -p 1313:1313 labo2-server-java java -jar server.jar 1313
```

Ou `java -jar server.jar "1313"` va écraser le CMD du Dockfile

- **ENTRYPOINT:**

ENTRYPOINT configures a container that will run as an executable.

```
ENTRYPOINT ["java", "-jar", "server.jar"]
```

Si notre `server.js` peut prendre des arguments (exemple: numéro de port). il est possible de lui spécifié lors du `docker run`

```
docker run -p 1313:1313 labo2-server-java 1313
```

ou "1313" est l'argument passé au jar. Ceci n'est pas possible avec *CMD*

- **EXPOSE:** informe docker que ce container écoute le réseau sur un port spécifique en runtime
- **WORKDIR:** Définit le dossier de travail pour toute commande *RUN*, *CMD*, *ENTRYPOINT*, *COPY* et *ADD*

Source: <https://docs.docker.com/engine/reference/builder/#from>