TP 09 - Introduction to Docker

Estimated Time : **60** *minutes*

- Le présent TP se veut présenter docker et les commandes de bases

8.1 - Stop - Start Doker Daemon

- Start et stop du docker daemon

```
$ service docker stop
$ service docker start
```

8.2 - Docker sans sudo

```
$ sudo groupadd docker
$ sudo gpasswd -a $USER docker
```

- Ensuite logout/login ou rédemarrer
- Pour vérifier :

```
$ docker version
```

8.3 - Docker Hello World

La commande suivante démarre l'image busybox

```
$ docker run busybox echo hello world
```

8.4 - Rentrer dans un container

```
$ docker run -it ubuntu bash
$ apt-get update && apt-get install curl -y
$ curl -L www.google.com
```

- `exit` pour quiter

Après avoir quitté, démarrer un nouveau container ubuntu et constatez que Curl n'est

pas là.

8.5 - Rechercher une image

- Pour rechercher une image sur les repository Docker

```
$ docker search ubuntu
```

Constaté qu'il y a des images officielles et un système de vote Depuis un webbrowser vous pouvez explorer les repositories en utilisant l'url suivante : https://hub.docker.com/explore/

8.6 - Télécharger une image

- Pour télécharger une image et la mettre dans son cache :

```
$ docker pull ubuntu
```

- Remarquez l'analogie avec git
- Pour voir les images déjà téléchargées

```
$ docker images
```

- La commande suivante pull une image du repo debian

```
$ docker pull debian:jessie
```

8.7 - Suivre les changements

```
$ docker run -it ubuntu bash
```

- Ajouter un fichier dans l'image :

```
$ apt-get update && apt-get install python -y
```

- Puis quiter le container avec `exit`
- Recupérer l'id de votre container en utilisant la commande suivante :

```
$ docker ps -lq
```

- Regarder les changements :

```
$ docker diff <yourContainerId>
```

8.8 - Créer une nouvelle image

- Créer une nouvelle image :

```
$ docker commit <yourContainerId>
<newImageId>
```

- Le système docker à un système de tag comme git

```
$ docker tag <newImageId> myimg
```

Nous allons utiliser cette image après assurez vous de bien lui avoir donner ce nom

8.9 - Rergarder l'historique d'une image

- Pour regarder l'historique d'une image

```
- docker history ubuntu
```

8.10 - Docker et mapping de ports

- Démarrer un serveur HTTP dans docker

```
$ docker run -d -p 80 myimg python -m SimpleHTTPServer 80
```

- Noter le container id
- Notes :
  - -d pour daemon
  - -p pour que docker alloue un port aléatoire au port 80 dans le containeur
  - python -m SimpleHTTPServer 80 pour démarrer un serveur http sur le port 80
- Vérifiez que le containeur est démarré

```
$ docker ps
```

- Pour retrouver le port mapper automatiquement sur votre host :

```
$ docker port <containeur_id> 80
```

- Utiliser Firefox pour aller sur le port retrouvé
- Notes : Essayer de retrouver le port avec les commandes suivantes :
    - docker inspect -f container_id
    - docker inspect -f "{{ json .NetworkSettings.Ports }}" container_id
    - docker ps

8.11 - Rentrer dans un containeur qui est démarré

- Essayer de créer un fichier dans le containeur qui fait tourner votre serveur, en utilisant la commande suivante :

```
$ docker exec -it container_id bash
```

- Notes :
    - touch toto.txt créé un fichier.

8.12 - Stopper et effacer un containeur

```
$ docker stop container_id
$ docker rm container_id
```

- Notes :
    - stop et effacer en même temps

```
$ docker rm -f container_id
```

8.12 - Industrialiser votre image : DockerFile

- Créer un dossier server_http (`mkdir server_http`)

- Créer un fichier nommé Dockerfile dedans
- Dans se fichier Dockerfile écrivez :

```
# L'image de base
FROM ubuntu:14.04

# Le mainteneur de l'image
MAINTAINER votre nom <votre@email>

# On installe python
RUN apt-get install python -y

# On créé un fichier
RUN echo "ma première image docker" > helloWorld.txt

# On rend le port 80 accessible à l'exterieur du containeur
EXPOSE 80

# La commande par defaut lorsque l'on run l'image
ENTRYPOINT exec python -m SimpleHTTPServer 80
```

- Il faut maintenant builder l'image :

```
$ docker build -t <nom_img> <repertoire_du_dockerfile>
```

- Regarder le résumé de votre build :

```
$ docker image <img_name>
```

```
$ docker -d -p 80:80 image <nom_img>
```

8.13 - Un exemple d'intégration Maven

8.13.1 - Ajout du plugin `io.fabric:docker-maven-plugin`

- Nous allons utiliser le plugin maven de `io.fabric:docker-maven-plugin` pour construire une image Docker de chaque projet :
- Ajouter le maven plugin au `resanet-parent` :

```
<build>
<pluginManagement>
    <plugins>
        <plugin>
            <groupId>io.fabric8</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>0.14.2</version>
        </plugin>
        <!-- Configuration du plugin : 8.13.2 && 8.13.3 -->
    </plugins>
</pluginManagement>
</build>
```

- Notes :
    - nous le rajoutons au pluginManagement pour pas que le plugin s'exécute pour resanet-parent

8.13.2 - Configuration de l'execution du plugin

- On map le goal `build` du plugin - servant à builder une image docker - à la phase maven package :

```
<executions>
    <execution>
        <goals>
            <goal>build</goal>
        </goals>
        <phase>package</phase>
    </execution>
</executions>
```

8.13.3 - Création du template d'images

```
<configuration>
    <logDate>default</logDate>
    <autoPull>true</autoPull>
    <images>
        <!-- Image holding the artifact of this build -->
        <image>
            <name>${project.artifactId}-zenika:${project.version}</name>

            <!-- ..................................... -->
            <!-- Build configuration for creating images -->
            <!-- ..................................... -->
            <build>
                <!-- assembly qui mettera toutes les dépendances et les
jars dans le container dans le dossier /maven -->
                <assembly>
                    <descriptorRef>artifact-with-dependencies</descripto
rRef>
                </assembly>
                <!-- Image docker de base -->
                <from>java:8u40</from>
                <!-- Expose ports -->
                <ports>
                    <port>8080</port>
                </ports>
                <!-- Default command for the build image -->
                <entryPoint>java -cp '/maven/*' ${MAIN.CLASS} ${MAIN.ARG
S}</entryPoint>
            </build>
        </image>
    </images>
</configuration>
```

- Notes :
  - MAIN.CLASS et MAIN.ARGS sont des properties maven que l'on pourra spécifier dans chaque projets enfants
  - Si vous effectuez un mvn clean verify, vous remarquerez qu'aucune image est construite, il faut activer les plugins dans les pom enfants.

8.13.4 - Activation du plugin dans les projet enfants

- Dans les pom enfants ajouter les properties suivantes avec les bonnes valeurs :

```
<properties>
    <MAIN.CLASS>MAIN</MAIN.CLASS>
    <MAIN.ARGS></MAIN.ARGS>
</properties>
```

- La valeur du MAIN.ARGS de resanet-reservation est `<MAIN.ARGS>--datasource=jdbc:h2:tcp://resanet-db:1521/resanet</MAIN.ARGS>`
- Pour activer la configuration rajouter juste le plugin, la configuration sera héritée :

```
<build>
    <plugins>
        [...]
        <!-- docker activation -->
        <plugin>
            <groupId>io.fabric8</groupId>
            <artifactId>docker-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

- Effectuez un mvn clean verify.
- Vous pouvez démarrer les images docker avec leur nom (ie : ${project.artifactId}-zenika:${project.version})
- Attention `resanet-reservation` a besoin d'une base de données, il ne démarrera pas.

8.14 Créer un docker compose

- Le but but est de créer un docker-compose.yml pour démarrer toute l'application grâce à la commande `docker-compose.yml` :

8.14.1 Créer une image pour la database

- Pour créer l'image de la database, rajouter l'image suivante dans la configuration du docker-maven-plugin (dans le pom parent) :

```
<image>
    <alias>db</alias>
    <name>resanet-09-db:${project.version}</name>

    <!-- ..................................... -->
    <!-- Build configuration for creating database -->
    <!-- ..................................... -->
    <build>
        <from>oscarfonts/h2</from>
        <ports>
            <port>81</port>
        </ports>
        <!-- Default command for the build image -->
    </build>
</image>
```

Configurer le docker-compose.yml

- Créer un docker-compose.yml dans le repertoire `resanet-parent` avec le contenu suivant :

```
version: '2'
services:
    resanet-catalogue:
        image: resanet-catalogue-09-zenika:0.0-SNAPSHOT
        ports:
           - "127.0.0.1:8081:8001"

    resanet-reservations:
        image: resanet-reservations-09-zenika:0.0-SNAPSHOT
        ports:
           - "127.0.0.1:8080:8000"
        depends_on:
           - resanet-db


    resanet-db:
        image: resanet-09-db:0.0-SNAPSHOT
        ports:
            - "127.0.0.1:8082:1521"
            - "127.0.0.1:81:81"
        volumes:
            - ./h2-data:/opt/h2-data
```

- `docker-compose up` et normalement tout devrait demarrer !

Notes : - docker-compose down est a effectuer lorsque vous créer des nouvelles

images (changement dans le code + mvn clean install) - le maven-docker-plugin à un goal qui permettent de démarrer et de stoper les images, très utile pour démarrer les images avant les tests d'intégration et les éteindre après dans maven.