KIV/PIA Labs - Docker Introduction

This lab provides introduction into Docker container engine. You will learn how to run, create and configure docker images and how to prepare your Java application to run inside docker container.

Basic terms

- image - binary prescription of what the runtime should be
- container - actual instance of an image (similar relationship as between class and object)
- tag - version identifier (images are versioned as any other distributable artifact in SW development)

Testing your docker installation

First we need to test if our docker service is running. Run the following:

```
docker run hello-world
```

The program should show image download information and afterwards something like:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Running and accessing docker image

In this section we will try to start a container with Tomcat Servlet Container and access its default page.

There is a global repository of public images: Docker Hub
The repository contains Tomcat image among others.

Docker containers run in their internal network and by default cannot be reached via host's IP/domain name (localhost in our case). Docker engine allows mapping between host's network and the network in which containers are running. We will start the Tomcat with port mapping to **localhost:8666**:

```
docker run --name=mytomcat -p 8666:8080 tomcat:8
```

The option **-p 8666:8080** says: map *host* port *8666* to *container* port *8080*.

Visiting **http://localhost:8666/** in your browser should take you to the tomcat's landing page.

Managing running processes

To see all your running containers, run `docker ps`. You should see similar output:

|CONTAINER ID |IMAGE | COMMAND| CREATED| STATUS | PORTS| NAMES
|-
|b53f0f62f191 |tomcat:8 |"catalina.sh run"|10 seconds ago | Up 8 seconds | 0.0.0.0:8666->8080/tcp | mytomcat

To **kill** a container use one of the following:

```
# docker kill <name>
docker kill mytomcat

# docker kill <id>
docker kill b53
```

Killed containers remain in your machine until removed (and can be started again).
To remove a container:

```
# docker container rm <name>
docker container rm mytomcat

# docker container rm <id>
docker container rm b53
```

Building Own Image

Own images can be created via simple text-file configuration - a Dockerfile. Dockerfiles are created by extending existing images and providing own file, commands and configuration.

We will extend the **tomcat** image we have just used in the previous section.

1. Build application in **app_nodb** folder using Maven. As a result, you will have **.war** file in the *target* folder.
2. Add the following text into the Dockerfile in this folder:

```
FROM tomcat:8

RUN rm -r /usr/local/tomcat/webapps/*

COPY ./app_nodb/target/*.war /usr/local/tomcat/webapps/ROOT.war
```

1. Run `docker build -t pia-example:first .`
2. Start container from your newly created image: `docker run --rm --name=mytomcat -p 8666:8080 pia-example:first`
3. Check the application is running at the **http://localhost:8666/**.

Note: the **--rm** flag removes the container automatically after the container is killed.

Multi-Stage Builds

Now we will get rid of the explicit Maven build before the actual docker image creation.

1. Replace the contents of **Dockerfile** with the following:

```
# Stage 1 - build the app
FROM maven:3.6-jdk-8

COPY ./app_nodb /build
WORKDIR /build
RUN mvn clean install

#Stage 2 - Final image
FROM tomcat:8

RUN rm -r /usr/local/tomcat/webapps/*

COPY --from=0 /build/target/*.war /usr/local/tomcat/webapps/ROOT.war
```

1. Build the new image: `docker build -t pia-example:second .`

Setting-up runtime environment

In this section we will use docker to build our application and setup a running mysql instance for it.

To configure container startup we will use **Docker Compose** - an orchestration tool from Docker.

1. Replace the contents of **Dockerfile** with the following:

```
# Stage 1 - build the app
FROM maven:3.6-jdk-8

COPY ./app /build
WORKDIR /build
RUN mvn clean install

#Stage 2 - Final image
FROM tomcat:8

RUN rm -r /usr/local/tomcat/webapps/*

COPY --from=0 /build/target/*.war /usr/local/tomcat/webapps/ROOT.war
```

1. Put the following text into docker-compose.yml:

```
version: '3'
services:
  mywebapp:
    image: pia-example:third
    build: .
    ports:
     - "8666:8080"
    environment:
     - jdbc.url=jdbc:mysql://mydb:3306/pia?createDatabaseIfNotExist=true
     - jdbc.user=root
     - jdbc.pwd=example
  mydb:
    image: "mariadb:10.4"
    environment:
      - MYSQL_ROOT_PASSWORD=example
    volumes:
      - ./conf/db/:/docker-entrypoint-initdb.d/
```

1. Run build and both containers with `docker-compose up --build`
2. Wait for the following log to show up (may take some time):

```
mydb_1_9555b81fed22 | 2018-12-02 14:04:40 0 [Note] mysqld: ready for connections.
mydb_1_9555b81fed22 | Version: '10.4.0-MariaDB-1:10.4.0+maria~bionic'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  mariadb.org binary distributio
```

1. Access your application at http://localhost:8666/
   and test that everything works.

License

Base of the JPA setup has been created by Karel Zibar during one of the courses at the University.