Docker workshop (ING)

Docker installation

Install Docker on your local machine using their instructions.

https://docs.docker.com/engine/installation/windows/

https://docs.docker.com/engine/installation/mac/

General information can be found at: https://docs.docker.com/engine/understanding-docker/

Make shure docker is running and the command line terminal is open.

Corporate proxy config @ ING

Create a .bat file to start up the docker toolbox.

Create .bat file

```
@echo off
set HTTP_PROXY=http://user:pass@proxynlwp.europe.intranet:8080
set HTTPS_PROXY=http://user:pass@proxynlwp.europe.intranet:8080
set NO_PROXY=localhost;127.0.0.*
cd "C:\Program Files\Docker Toolbox\"
c:
@start.sh
```

Daarna heb ik de profile aangepast van boot2docker (/var/lib/boot2docker/profile), daar heb ik dit in gezet:

```
export PROXY="http://user:pass@proxynlwp.europe.intranet:8080"
export HTTP_PROXY=$PROXY
export HTTPS_PROXY=$PROXY
```

Working with containers

Apply the following docker actions to learn the basics with docker.

1. Use “docker help’ and “docker <cmd> help” for information about the possibilities of docker. Important commands are ‘docker images’ to get a list with the current images, ‘docker ps’ to get a list with de current active containers and ‘docker ps -a’ to get a list with all the containers including the inactive.
2. Use docker run to start a container from image nginx and make it available at port 88. Check if the container is running by retrieving the website in your browser. You need the flag -d to run the server as deamon (the command prompt becomes unresponsive) and the flag -p hostport:guestport for port mapping. Nginx serves at port 80. Use “—name” to give your container a manageable name instead of a random ID. Use a browser to access the nginx webserver “http://<dockerhost>:88/”.
3. Use docker exec -ti <container-id> bash to open a shell in the active container. Check the filesystem with ls, with ps -ef the active processes, with hostname the hostname and with hostname -i the assigned ip-address of the container. (end shell with exit)
4. Stop the docker container with docker stop. Make sure that the webpage is not available.
5. With docker ps -a you can see the containers (active and not active).
6. Every time you execute docker run a new container gets created. After a run you can run the same container with the assigned name (docker start name) to prevent the creation of a huge amount of containers. With the flag -d the container runs as a deamon and you need to stop it explicitly. Restart the container now.
7. Check the local available docker images with ‘docker images’.
8. Check the image layering with ‘docker history’.
9. Remove the previously started docker container with ‘docker rm’. Use the -f flag if the container is still running.
10. Remove the nginx image with docker rmi. This only works if the containers based on the image are removed. If this is not the case, use docker ps -a and docker rm to remove the remaining.

11. We are going to do something more exiting, we are going to serve our own (static) webpage from the nginx container. Get the static pages from /web-app/static_webpage.zip. We need a new container, so we use docker run. Unzip static_webpage.zip to a location of your choice, for example /tmp/webpage1/.
12. Start a new container with a port mapping and a mount to /usr/share/nginx/www/ use “-v hostfolder:guestfolder:mountoptions”. In this case you can use “ro” for mountoptions, so it’s read-only. Use /usr/share/nginx/html as guestfolder for nginx. Do NOT use relative paths. Check the webpage with the browser.

Creating images

In this part of the workshop we will be deploying a three tier application that comprises of a web server hosting an angular web application, a Java based REST service and a mysql database. In this part of the workshop we will create a Docker image for the frontend and backend layer. There is no dedicated Docker image for de database layer needed as the database tables of the application will be created by the hibernate layer of the REST service upon startup, and the database can be created by the docker image it self.

Step 1. Run MySQL

The Java REST service expects a mysql database with the name cddb_quintor and a user with the same name and quintor_pw as password. Use the standard version 5.6 mysql image from Docker hub (https://hub.docker.com/_/mysql/) for this. There are environment variables available to let the mysql docker container create the database and an user. Give the mysql container the name cddb_mysql so it can be linked later on. Bind the mysql port (3306) to a local port and check that the database is available.

Step 2. Create an docker image of the Java REST service

The Java REST service is a simple JavaEE web app that is ready available in the three-tier-app/backend folder. This WAR file can be run with tomcat 8.5, so create a Docker image based on https://hub.docker.com/_/tomcat/ using a Docker file. Link the cddb_mysql container to mysql so the application can connect to mysql. Give the backend container the name cddb_backend so it can be linked later on. Bind the tomcat port (8080) to a local port and check that the REST service is available using a browser or other tool (http://<dockerhost>:<bindport>/cddb/rest/).

Step 3. Create an Docker image of the Angular web app

The web application is a simple Angular application compising only of static files to be served. This can be done easily using nginx. To make the REST endpoints easily accessible form the browser an nginx configuration is provided (nginx.conf) to create a reverse proxy that proxies the REST backend service. Therefor link the cddb_backend container to cddb_backend so the nginx reverse proxy can connect to the Java backend. Bind the nginx port (80) to a local port and check that the web application is available and working using a browser (http://<dockerhost>:<bindport>/cddb/rest/).

(Optional) Step 4.

Use volume mounting to store the mysql data locally. After removing the mysql container the stored data should be available again when a new container is create with the same volume mount.

(Optional) Use Docker Compose to run the three tier application at once

Docker compose give you the possibility to run multiple containers that are depended on each other at once. Command line options that are needed to link containers, mount volumes and expose ports can be defined is one configuation file.

Create a docker compose yaml configuration file to start up the three tier application at once. Documentation can be found here: https://docs.docker.com/compose/overview/