Deployment -- Exercises

Systems Engineering Laboratory 1

1 Introduction to Docker

In the first part of the lab we try out the Docker tooling, start our first containers build our own first images and finish with setting environment variables and volumes in the container. The goal is to get well aligned with the core concepts and tools of the Docker technology.

If necessary you can install additional Ubuntu packages in the containers (`apt-get update` and `apt-get install package`). For example:

- `net-tools` for using `ifconfig`
- `man` for manual pages
- `mysql-client` for command line MySQL client

We propose to use multiple terminal windows in order to have some connected to some container and other one can be used to execute commands on the host.

1.1 Docker basics

1. Start a base image (`ubuntu:xenial`) and check its system parameters, like memory and CPU. What is the correlation between the container's resource limits and the hosts resource limits?
2. Check the properties of the running container from the host, execute `docker ps` and `docker inspect`.
3. Check the IP address in the container (using `ip addr show` or `ifconfig`) and also on the host using the `docker inspect`.
4. Install an `nginx` webserver inside the container and check the default web content from the host (`apt-get update` and `apt-get install nginx`).
5. Check the default webpage of the server by nginx using a `curl` CLI tool on the host.
6. Exit the container and check its state from the host. (`docker ps -a`)
7. Restart the previous container and start the nginx again. (`service nginx start`)

1.2 Docker image construction

In the following the Docker images will be created:

- `retelab/manual-java`: created manually, contains JDK and starts a bash shell.

- `retelab/java`: created automatically using Dockerfile, contains JDK and starts a bash shell.
- `retelab/hello-world`: created automatically using Dockerfile based on the `retelab/java` and starts the `Hello world` application.
- `retelab/java-gradle-project`: it is based on the `retelab/java`, mounts any Gradle-based Java project as a volume and executes it.

1.2.1 Manual image construction

1. Based on the `ubuntu:xenial` image prepare a new base image for your Java-based projects:
   - Start a new `ubuntu:xenial` container.
   - Install JDK (`openjdk-8-jdk` Ubuntu package).
   - Create a `Hello world` Java application inside the container. Use only the `javac` compiler and a simple text editor such as `mcedit`, `vim`, `nano` or `emacs`. Hints:

     ```
     # compile the project
     $ javac HelloMain.java
     # run the generated file
     $ java HelloMain
     ```

   - Execute the `Hello world` application.
   - Save the running container as a new Docker image, named `retelab/manual-java` (`docker commit`).
2. Start a new container based on the new `retelab/manual-java` image and test your `Hello world` application again.

1.2.2 Automatic image construction

Hint: create a new subfolder for each image definition and put the `Dockerfile` and other related content into that folder.

1. Using the `Dockerfile` syntax create the same image definition as before: based on the Ubuntu base image, install Java, add and compile the `Hello world` application (Hint: using the `ADD` Dockerfile command, you can add files into a container). Build this as a new image `retelab/java` (`docker build`).
2. Start a new container based on the new `retelab/java` image and test your `Hello world` application again.
3. Start a new container based on the new `retelab/java` and provide a `COMMAND` *command-line argument* to `docker run` and run the container using the `Hello world` application instead of the `bash` shell.
4. Build a new image `retelab/hello-world` based on the `retelab/java` image (defined by a new `Dockerfile`) that starts the `Hello world` application instead of the `bash` shell (a `CMD` command in the `Dockerfile`). (Hint: take care on the `WORKDIR` in order to execute the commands in the right directory)

1.3 Environment variables

1. Extend your a `Hello world` application to also print the value of the `RETE_ENV` Linux environment variable. Rebuild your two images after the changes.
2. Start a new container based on the `retelab/java`, set the `RETE_ENV` environment variable (`export RETE_ENV=production`) and test your `Hello world` application again.
3. Start a new container based on the `retelab/hello-world` by adding the `RETE_ENV` variable to the `docker run` command (as command line parameter and not using the export command) and check the output.
4. Clean up all the running containers. (Use command line tools, like `docker ps`, `grep`, `cut`, `xargs`, `docker rm -f`. Check the manuals if necessary (`man command`))

2 Docker Compose

2.1 Deploying using Docker containers and shared volumes

1. Clone the repository from https://github.com/FTSRG-ReteLab/docker-lab and run the project with the following Gradle command:

   ```
   $ ./gradlew run
   ```

   It will fail as the MySQL server is not available.

2. Start a new container running the MySQL server with username `root` and password `retelab`:

   ```
   $ docker build -t mysql-backend mysql-backend
   $ docker run -d mysql-backend
   ```

   This contains the `golf` dataset. The container does not have interactive shell, it starts in the background (-d parameter). Check if the container is running (`docker ps`) and check its IP address.

3. Start and attach to a new process (bash shell) inside the running MySQL container using `docker exec -ti` and check the running processes using `ps ax`. Test the command line MySQL client (`mysql -u root -pretelab -h localhost`). (Hint: do not use `docker attach` to connect to the container, since it attaches the terminal to the main process of the container, which is a MySQL server process running in the background. Let's start a new bash process using `docker exec`).

4. Test the MySQL connection from outside the MySQL container (e.g. from the host or from an other `ubuntu:xenial`-based empty Ubuntu container). Install the MySQL client **on the host machine** or in a new Ubuntu container:

   ```
   $ sudo apt-get install mysql-client
   ```

5. To join to the MySQL server, use the following command:

   ```
   $ mysql -u root -pretelab -h IP_ADDRESS
   ```

6. Define a new Docker image `retelab/java-gradle-project` using `Dockerfile` based on the previous `retelab/java` image. Add a new directory to the filesystem (`mkdir /project`), set it as workdir (`WORKDIR` command in the `Dockerfile`) and set the default command (`CMD`) to `./gradlew run`. Build the image.
7. Start a new container based on the `retelab/java-gradle-project`, add the git project's root dir to the container's `/project` directory as a volume (`-v` parameter). Take care on the connection string: modify the MySQL server address to the IP of your MySQL container.
8. Clean up all the running containers.

2.2 Docker Compose-based deployment

1. Using the `docker-compose` technology define the infrastructure in the form of a `docker-compose.yml` file.
   - Add a MySQL server service.
   - Add a Java Gradle project service
   - Mount the `mysql-client` project as a volume to the Java Gradle project service (`volumes` property in the `docker-compose.yml`).
   - Define the network connection between the containers (using the `links` property in the `docker-compose.yml`). Take care on the connection string: modify the MySQL server address to the name of the link to your MySQL service in the `docker-compose.yml` file. This name can be used as hostname.
2. Fire up the infrastructure with a single command (`docker-compose up`).

2.3 Excercises for iMSC

1. Extend the previously created docker-compose deployment with an additional web service, that serves a Wordpress website. You can use the offical php-apache docker image to serve the web content and the existing MySQL instance to serve a new database for the Wordpress.
   - Add the new service to the docker-compose definition
   - Download and uncompress the Wordpress installer from the offical Wordpress.org website
   - Create a new MySQL database and user for the ne service
   - Define a volume for the web content, add the WP files to it and mount to the right location in the new web container
   - Finish the WP installation on its web UI.