Lab 02: Docker-compose

The exercises marked with * are obligatory to complete before the upcoming weeks lecture.

The purpose of this lab is to get you started with another main tool of this course, Docker-compose. The lab is structured in a way, where the student is assisted through the tasks.

Remember to write useful commands in your cheat sheets.

Table of content

Exercise 01*

In this exercise the student is going to spin up a container running an Apache server, expose a port and see the resulting website on the host machine.

Ports

A port is a communication endpoint found within the computer. All forms of connections made to a computer are routed through a port, where the communication follows the standard defined in protocols (more about protocols in lab 04). Ports on remote hosts are accessed with <host ip>:<port> and ports on your own machine are accessed with localhost:<port>

Normally any port can be assigned to any process, but the IANA has a list of well-known ports that are used for for example:

Port number	Assignment	Description
22	Secure Shell (SSH)	Secure communication between two devices.

80	HyperText Transfer Protocol (HTTP)	Web pages
443	HTTP Secure (HTTPS)	Encrypted web pages

In this exercise, an Apache server is set up in a container, displaying a premade site.
The Docker image that should be used can be found on DockerHub and is called `cocodolan/php:apache`

Your tasks:

1. Make a run command that **exposes** port 7000 host machine, to container port 80 and runs the `cocodolan/php:apache` image.
2. Visit the forwarded port through a browser on your host machine via localhost.
   As long as you get some response in the console after visiting localhost, the exercise is complete.

The exercise will return something like the following message:

```
[Thu Jul 04 13:29:48.343170 2019] [mpm_prefork:notice] [pid 1] AH00163:
Apache/2.4.25 (Debian) PHP/7.2.19 configured -- resuming normal operations
[Thu Jul 04 13:29:48.343489 2019] [core:notice] [pid 1] AH00094: Command line:
'apache2 -D FOREGROUND'

172.17.0.1 - - [04/Jul/2019:13:20:20 +0000] "GET / HTTP/1.1" 200 370 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/12.1.1 Safari/605.1.15"
```

Exercise 02*

In this exercise the student is going to create a Dockerfile, build an image from that Dockerfile, expose a port and see the resulting website on the host machine. The container should run a php server.

The .php files for this exercise are located in the L2E2 folder on BlackBoard.

Help: Use your skills and your cheat sheet from Lab 01 to complete this exercise.

Your tasks:

1. Create a Dockerfile that does the following.
   a. Uses the `php:apache` image.
   b. Insert `RUN docker-php-ext-install mysqli`
   c. Copies the included files to the default root folder that Apache2 uses (/var/www/html).
2. Build the Dockerfile. **Remember** to give the build a tag.

3. Run the above built image. **Remember** to **expose** the port.
4. Visit the forwarded port through a browser on your host machine. (Localhost)
    a. The functionality of the site is not working, due to there being no database connected. But as long as you get a site that shows something, it works.

The exercise will return the following message:

```
[Fri Sep 06 14:36:34.862957 2019] [mpm_prefork:notice] [pid 1] AH00163:
Apache/2.4.38 (Debian) PHP/7.3.9 configured -- resuming normal operations
[Fri Sep 06 14:36:34.863025 2019] [core:notice] [pid 1] AH00094: Command line:
'apache2 -D FOREGROUND'

172.17.0.1 - - [06/Sep/2019:14:36:55 +0000] "GET / HTTP/1.1" 302 254 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/76.0.3809.132 Safari/537.36"
172.17.0.1 - - [06/Sep/2019:14:36:55 +0000] "GET /insert.php HTTP/1.1" 200 473
"-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/76.0.3809.132 Safari/537.36"
```

Exercise 03*

In this exercise the student is going to spin up an ubuntu container, and install mySQL on it. The student is going to try a list of commands on the mySQL server.

A mySQL server is a database that uses the SQL language to manage the server and the data it consists of.

As we assume that you have basic knowledge about the SQL language, we will be focusing on the mysql CLI and some basic configuration commands instead of the language itself.

MySQL CLI

MySQL is a database server, using the database language SQL.

Command-Line Interface (CLI) is a way of interacting with different programs through the Terminal/Cmd. The mySQL CLI is called by using the command `$ mysql -u root -[YOUR-PASSWORD]`.

When inside the MySQL CLI, you will see `mysql>` written. Inside the CLI the SQL-language is used, meaning that the SQL commands you are familiar with will work.
Remember to add “;” after every command.

The CLI includes a list of commands, made to maintain the mysql server. The following list is a extract containing the commands you should know as a minimum:

Command	Description
show databases;	Shows all databases inside the mysql-server.
use DATABASE;	Making a database usable.
show tables;	Shows the tables in the selected database.
help;	Listing all MySQL commands.
exit;	Exits the MySQL CLI.

Your tasks:

1. Spin up an interactive ubuntu container.
2. Install MySQL.
    a. The package is called `mysql-server (Hint: apt-get)`.
    b. Start the mysql service using `$ service mysql start`
    c. Configure mysql using the command `mysql_secure_installation`.Select a password for the `root` user.
3. Use the `mysql` command to interact with the mysql database.
   NOTE: When accessing the database, you will have to include the username and password using the -u and -p option. `$ mysql -u username -p`
4. Show all databases. Create a database.
5. Show all tables. Create a table.
6. Insert a row to the table. Show the table.
7. Remove all data from the table.

Exercise 04*

In this exercise the student is going to create a Dockerfile that initiates a mySQL server with preconfigured settings. A Docker Image is built from that Dockerfile and a container that is spun up, using the built image.

Your tasks:

1. Create a Dockerfile that does the following:
    a. Uses the mysql image
    b. Insert `ENV MYSQL_ALLOW_EMPTY_PASSWORD yes`
    c. Copy the `init.SQL` file from the resources into the folder in the container with the route `/docker-entrypoint-initdb.d/`. Then the `init.SQL` file is executed automatically by the mysql image.

2. Build the Dockerfile. **Remember** to give the build a tag.
3. Run the image built from the Dockerfile above.
4. Open another terminal and use the `exec` command to access the mysql container.
5. Access the mySQL with the username `root` and no password.
6. Check that the database and table is created.
   As long as they are created, the exercise is complete.

Exercise 05*

In this exercise the student is going to create a docker-compose.yml file that composes a system that uses the Dockerfiles made in Exercise 02 and Exercise 04.

Docker-compose

Compose is a tool made for defining and running a multi-container system. This is done using a YAML file (.yml) called a Compose file, where the containers and the environment is configured.

The Compose file is used to define services, networks and volumes.
We will be focusing on services only, which tell docker how the containers are configured from the images (or builds).

Using docker-compose is divided into three steps:

1. Define a single container environment in a Dockerfile that does some individual work.
2. Define services for each container in a Compose file, so the multi-container system can be runned as an isolated environment.
3. Run the YAML file using `$ docker-compose up` while inside the folder with the YAML file.

YAML file

A YAML file is a general file format often used for configuration files. In Docker-compose the file is used to configure services with the purpose of running them together.
A specific syntax is used inside the YAML file to create and describe the environment that is spun up with the containers.

For starters, there are two things to notice inside a YAML file. Firstly, that # used in the beginning of a line is treated as a comment. This means that the Docker engine skips over the lines beginning with a #.
Secondly, that instructions in a YAML file often follow the syntax `[KEY]: [VALUE]`

```
# A docker-compose YAML file always starts by defining the version of
```

```
docker-compose that is used.
# In this case version 3 is used.
version: '3'

# Using the key services, we indicate that the following lines will define the
services included in the environment.
services:

# After the services key, the services are defined. Here you include a key,
holding the name of your service.
# In this case, we have named the service ‘sql’ as it is a description of what
the service contains.
   sql:

# After a service has been made, a number of options is available for use, to
specify what the service should be built from and which properties the service
should have.
# Build is used to specify the file-path to where the Dockerfile for the SQL
service is located. The path should be relative to the docker-compose.
      build: [PATH-TO-SQL-DOCKERFILE]

   web:
      build: [PATH-TO-WEB-DOCKERFILE]

# Depends_on is telling Docker, that the web-service is dependent on the
sql-service which in return starts the services in dependency order.
      depends_on:
      - sql
Ports work just like -p 8080:50, making the container’s port 80 accessible on
the host port 5050.
      ports:
      - '5050:80'
```

Your tasks:

1. Create a docker-compose.yml that has the following
    a. A db service
        i. That builds from a sql-subfolder wherein the SQL Dockerfile is located.
        ii. Because of some version incompatibility between mySQL and PHP insert:
        `command: --default-authentication-plugin=mysql_native_password`
        iii. Make sure that the db service’s name matches the server name in the config.php in the web service folder
    b. A web service
        i. That depends on the db service
        ii. That build from a web-subfolder wherein the Web Dockerfile is located.

iii. That exposes port 80 to the hosts port 7000.

2. Run the docker-compose
3. Check localhost now, and try to insert some data!