Lab 01: Docker

The exercises marked with * are obligatory to complete before the coming week's lecture.

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

IMPORTANT: Throughout this course we recommend you all to make a cheat sheet with Linux, Docker and other commands that you learn. This will be very useful in the project.

Table of content

Exercise 01*

In this exercise the student is going to run the Docker Hello World container.

In Docker, a container is spinned up by using the `run` command. When using the command, it is required for you to specify the image you would like to spin up. If the image is not to be found in your library, Docker will look for the image on Docker Hub.

The `run` command:

```
$ docker run <DOCKER-IMAGE>
```

On Docker Hub a build has been created, named `hello-world`. The build is used to test if docker has been installed correctly and to give the user a feeling of how the docker CLI (Command-Line Interface) works.

Your task:
In this exercise, your task is to create a container that runs the above-mentioned `hello-world` image.

The exercise will return the following message:

```
$ docker run <DOCKER-IMAGE>

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/
```

Exercise 02*

In this exercise the student is going to create a Dockerfile, specifying configurations for a home-made Hello World image.

In Docker, each image is an immutable file created from a Dockerfile. It is a template used to create containers with a predefined environment.
Docker also features containers which are a lightweight and portable encapsulation of the environment described in the Docker image.

As a metaphor, images can be seen as classes, whereas containers can be seen as objects.

Important: A Docker container has a running (currently executing) process that defines its lifetime.

Creating a Dockerfile

A Dockerfile is simply a file called “Dockerfile” without any extension. Inside a Dockerfile a specific syntax is used to create and describe the environment that is spinned up with a container.

For starters, there are two things to notice inside a Dockerfile. 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, the instructions in a Dockerfile follow the syntax `[INSTRUCTION] [ARGUMENTS]`. That means that the instructions are written with capital letters and the arguments are written in non-capital letters.

```
# A Dockerfile often begins with the FROM instruction, telling the Docker engine which image this image is built from.
# In this case ubuntu is used as the argument.
FROM ubuntu

# The RUN instruction is an instruction that takes a BASH command as an argument. In this way simple functionality is included in the Dockerfile.
# In this case, the text “Hello, World!” is inserted into a txt-file called ‘helloWorld’.
RUN echo “Some text” > helloWorld.txt

# The CMD instruction is the instruction that determines the container's lifetime. This instruction is not runned when building the image. When a container is spinned up from an image, the following command is runned as the executing command. When the command finishes its execution, the container will shut down.
# In this case, the text inside the ‘helloWorld.txt’-file is printed to the screen.
CMD cat helloWorld.txt
```

As a rule of thumb, if you do not have a specific idea for the image you want to create, then develop your Dockerfile from the ubuntu image (using `FROM ubuntu`).

Building a Docker image

Building a docker image can be done by using the `build` command.
`$ docker build <PATH-TO-DOCKERFILE>`

An important option to know when building Docker images is the tag option (`-t`) which is used to tag the Docker image with a specific name. This makes it easier to run the container, as the name is chosen by us, and not the Docker engine.

An example of using the tag option:
`$ docker build -t <TAG> <PATH-TO-DOCKERFILE>`

Furthermore, it is important to note that as default the Docker Engine expects the Dockerfile to be named “Dockerfile” as well. If you want to build a Dockerfile with another name, you will have to include the file option (-f).
`$ docker build -f <FILENAME> <PATH-TO-DOCKERFILE>`

Running a Docker container

Spinning up and running a Docker container follows the same way as described in the previous exercise.

Your tasks:

1. Create a Dockerfile that prints the following message to the screen: ‘Hello, World! 42.’.
2. Build the above created Dockerfile. **Remember** to give the build a tag.
3. Run the above builded image.

The exercise will return the following message:

```
Hello, World! 42.
```

Exercise 03*

In this exercise the student is going to use the included Dockerfile to build an image and spin up a container.

This Dockerfile is different from the previous files, as a new instruction is introduced.

Dockerfile

In this Dockerfile a new instruction is introduced. The `COPY` instruction is used to copy files or folders from the host machine to the Docker container.

```
FROM ubuntu

# Here, the COPY instruction copies the content of the /L1E3 folder from the
```

```
host machine to the / (root) destination of the container.
COPY L1E3/ /home/

CMD bash
```

Running a Docker container

Spinning up and running a Docker container follows the same way as described in the previous exercise.

This time, an extended version of the `run` command is used.
`$ docker run <OPTIONS> <DOCKER-IMAGE> <ARGUMENT>`

Another option that is important to know is the option for using interactive processes (`-it`). Combining this with the argument `bash`, makes it possible for us to enter the container and interact with it under runtime.

Your tasks:

1. Build the included Dockerfile. **Remember** to give the build a tag.
2. Run the above builded image. **Remember** to run it in a way, which makes it possible for you to enter the container.
3. Install `python3` and run the included `print.py` file.
4. Edit the Dockerfile, so `python3` is installed beforehand and runs the Python script.

The exercise will return the following message:

```
200
```

Exercise 04*

In this exercise the student is going to copy the included .php files to an image, expose a port and see the resulting website on the host machine. The files can be found in the resources.

Running a Docker container

Spinning up and running a Docker container follows the same way as described in the previous exercise.

`docker run -p <HOST-PORT>:<CONTAINER-PORT> <DOCKER-IMAGE>`

Another option that is important to know is the option for exposing ports (`-p <HOST-PORT>:<CONTAINER-PORT>`).

This option is exposing port <CONTAINER-PORT> from the container, and connecting it with port <HOST-PORT> on the host machine. This means that a visit to the host machine's (localhost) port <HOST-PORT> will be redirected to the container's port <CONTAINER-PORT>.

There are many cases where this function comes in handy. An example would be, if you wanted to host an apache web server in a container. It would be boring, if it was not possible for outside machines to access the information that the server serves.

In this case we are using an apache server that communicates through HTTP. This is done through port 80.

Your tasks:

1. Examine the included Dockerfile. Place your files, so it complies to the structure dictated by the Dockerfile.
2. Build the included Dockerfile. **Remember** to give the build a tag.
3. Run the above builded image. **Remember** to expose the port.
4. Visit the forwarded port through a browser on your host machine.

The exercise will return the following message in the terminal:

```
[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 05*

In this exercise the student is going to create a script in Python and a Dockerfile where the script is copied to the container. Python shall be run inside the container and as the executed process.

Help: Look at the previous Dockerfiles to get an idea about how the Dockerfile is configured.

When the executed process is set to be a process that has a definite lifetime, the container is stopped when the process dies. In this case it means that the container will automatically stop when the script has been executed.

Help: When running Python in a Docker container, use the flag -u to print while the container is running.

Your tasks:

1. Create a Python script that makes some form of output that is printed to the terminal.
    a. Inspiration: Make a for-loop that counts. Use the `time.sleep(int count)` method to make the executing Thread sleep.
2. Create a Dockerfile that copies the script to the container, installs Python and runs the script.

The exercise will return the following message:

```
SCRIPT-OUTPUT
```

Exercise 06*

In this exercise the student is going to explain different concepts to/with the group. It is important to grasp the concepts of Docker and some base-knowledge.

Your tasks:

1. Explain what a Dockerfile is.
    a. How does it work?
    b. Which instructions can be used?
    c. What do the above-mentioned instructions do?
2. Explain how a Docker image is built.
    a. Which options do you know?
    b. What are they used for?
3. Explain how a Docker container is runned.
    a. Which options do you know?
    b. What are they used for?
4. Compare cheat sheets and share your pearls of wisdom.