Tasks and Assignments Course 02267

Hubert Baumeister

Version 1

1 Assignment (group): Deploy and run Simple DTU Pay in a docker container and using Jenkins ASSIGNMENT_SIMPLE_DTU_PAY

This assignment is based on the task for creating Simple DTU Pay with access to the bank. The idea is to put together all topics from week 1 and create a Simple DTU Pay service that accesses the bank, runs in a Docker container and is deployed and tested using Jenkins on the Linux VM.

1.1 Part 1: Dockerize Simple DTU Pay using the bank

Move the execution of the server into the docker image. The `Dockerfile` you will be using will look like this:

```
FROM eclipse-temurin:21 as jre-build
WORKDIR /usr/src
COPY target/quarkus-app /usr/src/quarkus-app
CMD java -Xmx64m \
    -jar quarkus-app/quarkus-run.jar
```

In the first step, we are using the Java 21 run time (JRE) from the Eclipse Temurin project. We don't need the full JDK here, and we want to keep our images as small as possible. We are going to run several docker images on a machine with 4GB memory, so we want to make sure that we can get as many running docker containers as possible.

Then we set the work directory to `/usr/src` on the docker image and copy the compiled binaries to the docker image (`quarkus-app` contains the jar file but also all libraries that the jar file depends on).

In the next step we start the server using the `CMD` command. When you run an image in a container, this command will be executed. When the command terminates, the container will terminate. This is the same command as previously with basically one difference. You should limit the heap size of Java to 64MB (i.e. `-Xmx64m`), which keeps the memory used by the container low and should be sufficient for our purposes of running several microservices in one VM at the same time.

Here[1] is the documentation for the contents of the Dockerfile.

The simplest way to build a new docker image is to run

`docker build --tag demo .`

in the directory containing the Dockerfile. This creates an image called `demo`. However, we will later use `docker compose build` to create the image.

1.1.1 Running the docker image

The simplest way to run a docker image is to execute `docker run demo`. However, this won't give you access to port 8080 where the Web server is running. You have to map the internal port 8080 to the external port 8080 (or whatever) using `-p external:internal`. Thus we need to run the image by `docker run -p 8080:8080 demo`. Now you can access your Web services through `localhost:8080`. Note that to do this, no other program has to listen on the port 8080.

[1] `https://docs.docker.com/engine/reference/builder/`

1. Using ‘docker compose‘ Another way to run the image is to use `docker compose`. This is the prefereed way as it allows easy control of running several docker images at a time. It also creates a private network between the containers defined in the docker-compose file, so that you can access those containers by using their service name. In addition, docker compose allows for hot-swapping of container whose images have changed. docker compose works together with a `compose.yml` file which is shown here:

   ```
   services:
     demo:
       build: .
       image: demo
       container_name: demo
       ports:
        - "8080:8080"
   ```

   Running `docker compose up` in the directory with the `compose.yml` file will run the demo image and map host port 8080 to internal port 8080. If you run the command twice nothing will happen unless you rebuild the demo image. In this case the old container (called `demo`) will be stopped, recreated from the new image, and started. The `build: .` part is optional. If it is present it indicates the directory from which to build the image, i.e. the directory containing the Dockerfile plus the files to be copied to the image. Thus, instead of running `docker build --tag demo .`, you can just run `docker compose build`.

   A useful trick is to start the docker container as demons, i.e, `docker compose up -d` and then run `docker compose logs -f` which shows you the logs of all the containers started by docker compose. It is then easy to stop and restart the logging without terminating the containers.

   To shut down the container and delete it, run `docker compose down`.

1.1.2 Running the client

Your client works as usual. Here[2] is the simple REST demo from the previous task; however, now using docker and docker compose.

1.1.3 More information

To get more information about available docker commands use `docker -h`. If you want to know more about a given docker command, e.g., `build`, you run `docker build -h`. The same principle also applies for `docker compose`. If you then need to know more about an option use google to search for the Docker documentation.

1.2 Part 2: Jenkins Project

Create a Jenkins project for your Simple DTU Pay using the bank application and build and run the tests on the Linux virtual machine via Jenkins.

[2] Project simple_rest_demo_docker in the solutions directory of the Git repository `https://gitlab.gbar.dtu.dk/huba/02267-student-repo/`