COMP.SE.140 – Docker-compose hands on

Version history

V1.0 25.09.2024	First version
V1.1 29.09.2024	First corrections
V1.2 30.09.2024	Correction of correction

Synopsis

The purpose of this exercise is to learn (or recap) how to create a system of two interworking services that are started up and stopped together. This requires creation of your own Dockerfiles and docker-compose.yaml, and also creation the simple applications. The applications can be implemented in any programming language (shell script and HTML not allowed), but different programming language must be used for the two applications. You also need to play with some operating system concepts.

Learning goals

- Recap your hands on with Docker and Docker Compose. This is assumed to be known from earlier courses and will be needed in the next steps of the course.
- Understand the relation of containers to the operating system and networking.
- Ensure hands on with Linux.
- See the value of virtualization for application development.

Task definition

In this exercise we will build a simple system composed of two small services (Service1 and Service2) implemented in different programming languages. The services are small programs running in separate containers. Both of these applications collect information from the container:

- IP address of the container
- Running processes (e.g. output of "ps -ax" on Ubuntu)
- Available disk-space in the root file systems of the container (e.g. command "df")
- Time since last boot

The composition of two containers (services) works as a single service, so that one service works as an HTTP-server (waiting in port 8199) for external clients. Only one Service1 can be accessed from outside, so it needs to ask information from Service2 within the composition.

The response to the HTTP-request should be

```
Service
      - IP address information
      - list of running processes
      - available disk space
      - time since last boot
Service2
      - IP address information
      - list of running processes
      - available disk space
      - time since last boot
```

Use plain text or JSON formatting

Some notes

The IP address may be IP4 or IP6 address – depending on the system. For example the ":ffff:"-prefix provided by some libraries can be included.

Give the images unique names.

Submitting for grading

After the system is ready the student should return (in the git repository – in branch "exercise1").

- Content of two Docker and docker-compose.yaml files
- Source codes of the applications.
- Output of "**`docker container ls`**" and "**`docker network ls`**" (executed when the services are up and running.) in a text file "docker-status.txt"
- A short (max 300 words) text file describing your findings on what containers share with the host. Put that to "findings.txt"
- Optional "llm.txt" (see below)

Please do not include extra files in the repository.

These files are returned with some git service. Courses-gitlab repositories can created to course-gitlab if you request one. Any git-repo that the staff can access without extra effort is ok.

You should prepare your system in a way that the course staff can test the system with the following procedure (on Linux):

```
$ git clone -b exercise1 <the git url you gave>
$ docker-compose up --build
… wait for 10s
$ curl localhost:8199
$ docker-compose down
```

Grading

The points from this exercise depend on timing and content:

- Maximum 5 points are given.
- missing the first deadline (08.10.2024): points reduced by 0.5 points / starting day. The absolute deadline is 15.10.2024.
- how well the requirements (including technical instructions to the submit your project) are met: 6p
- following the good programming and docker practices: 2p

On using ChatGPT or similar AI (large language models - LLM) tools

The university-level guidelines say:

> *"If a student uses a language model in an assignment or a thesis, for example, as part of language editing, this must always be mentioned. When individual students describe their use of language models, we can share good practices. The use of a language model for language revision is justified, for example, to produce a grammatically or structurally fluent text (cf. proofreading and translation tools and similar tools)."*

In this exercise we interpret this as follows

- If language models are used, a separate report (llm.txt) must be written (included in the submission). This report includes:
  - The used LLM tool
  - Motivation/reason to use LLM
  - How and why LLM helped
  - What kind of mistakes LLM did
  - What were things that LLM was not able to provide
- You allow course staff to use this report in grading and use of it (after anonymization) for teaching development and research purposes.

- The course staff will investigate different ways to discover use of LLM – students using LLM without reporting it, will be discontinued from the course.

Hints

It might be a good idea to create and test the applications first.

Do not provide the link from the browser (the one you see when you access your repo with a GUI) – that does not work with git clone, and your points will be reduced. Check that "git clone -b exercise1 <the git url you gave>" really works.

Useful material:
https://docs.docker.com/compose/,
https://docs.docker.com/compose/networking/ (also https://www.simplilearn.com/tutorials/docker-tutorial/docker-networking)

Docker images are easy to access, if they are tagged when built

```
$ docker build --tag=service1 .
```

If Docker image is rebuilt, docker-compose should also be given a hint that rebuilt should override the existing one

```
$ docker-compose up --build
```