COMP.SE.140 – Docker-compose hands on

Version history

V0.1 30.08.2023	Initial version for course staff only
V0.2 06.09.2023	Corrections and clarifications
V1.0 12.09.2023	Version published to students

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 should be used for the two applications.

Learning goals

- Learn some hands on with Docker and Docker Compose. This is needed in the next steps of the course.
- Understand the runtime context of Docker containers, for instance networks and volumes.
- See the value of virtualization for application development.

Task definition

In this exercise we will build a simple system composed of two small services implemented in different programming languages. The services are small programs running in separate containers. Thus, the target is the following:

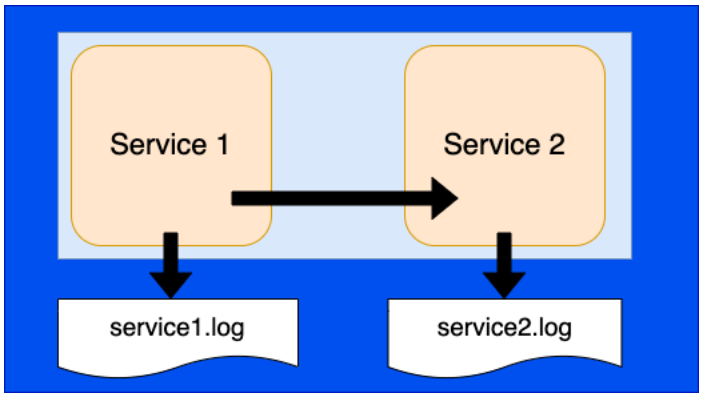

Figure 1. Architecture of the target system.

The files “service1.log” and “service2.log” stored in folder “logs” in the hosting Linux and made available to services with the “volume” directive in docker-compose file. In the beginning of the execution both services should

- Create service1.log / service2.log if it does not exists
- Ensure that the files are empty in the beginning.

Service/application 1 should repeatedly 20 times in 2 second intervals:

- Compose a **text** from a counter (initialized to 1) current time and address+port of service2. Follow this example syntax precisely:

```
1 2022-10-01T06:35:01.373Z 192.168.2.22:8000
```

- Write the above text to file “service1.log” (each writing to a separate line)
- Send the text with HTTP protocol to service 1.
- If the sending fails, catch the exception and write the error message to “service1.log”

- Increase the counter with 1

After the 20 rounds

- Write “STOP” to file “service1.log”
- send STOP to service 2
- Close the file “service1.log”
- Exit the service

Service/application 2 should

- Wait for 2 seconds
- Establish an HTTP server that listens in port 8000
- As a response to incoming message create a new text that adds the remote address (address of service 1). An example:
  `1 2022-10-01T06:35:01.373Z 192.168.2.22:8000 192.168.2.21:78390`
- Write that text to file “service2.log”
- If the received text was “STOP” close file “service2.log” and exit.

After execution the files “service1.log” and “service2.log” should be readable in local folder “logs”.

As the service should run in separate containers, you should write *Dockerfiles* for the both services and *docker-compose.yaml* to start both containers and connect then with a private network.

For fluent testing by teaching staff you should do your best to ensure that that both docker images and running containers have unique names from your other students.

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.

By remote address/port we mean the address of the host that sent the request.

The built images should have the application installed. Do not “install” it by bringing it on a volume.

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
- Output of “**`docker container ls`**” and “**`docker network ls`**” (executed when the services are up and running.) in a text file
- Source codes of the applications.

Please do not include extra files in the repository.

These files are returned with some git service. Courses-gitlab repositories have been created to course-gitlab, but you do not need to do that. Any git-repo that the staff can access 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 a minute
$ docker-compose down
$ more logs/service1.log
```

Grading

The points from this exercise depend on timing and content:

- Maximum 8 points are given.
- missing the first deadline (30.09.2023): points reduced by 1 points / starting day.
  The absolute deadline is 07.10.2023.
- 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 (~page) 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/research.
- 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.

If you use Python, note that some libraries do not provide access to remote address. Do not use such library!

Useful material:
https://docs.docker.com/compose/,
https://docs.docker.com/compose/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
```