COMP.SE.140 – Docker-compose hands on

Version history

1.0 21.09.2021	Initial version
1.1 23.09.2021	Clarification on "build"

Synopsis

The purpose of this exercise is to learn how to create a system of interworking services that started up and stopped together. This requires creation of your own Dockerfile 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).

Learning goals

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

Task definition

In this exercise we will build a simple system composed of two small services. The first service is exposed to outside world and the other is internal. Thus, the target is the following:

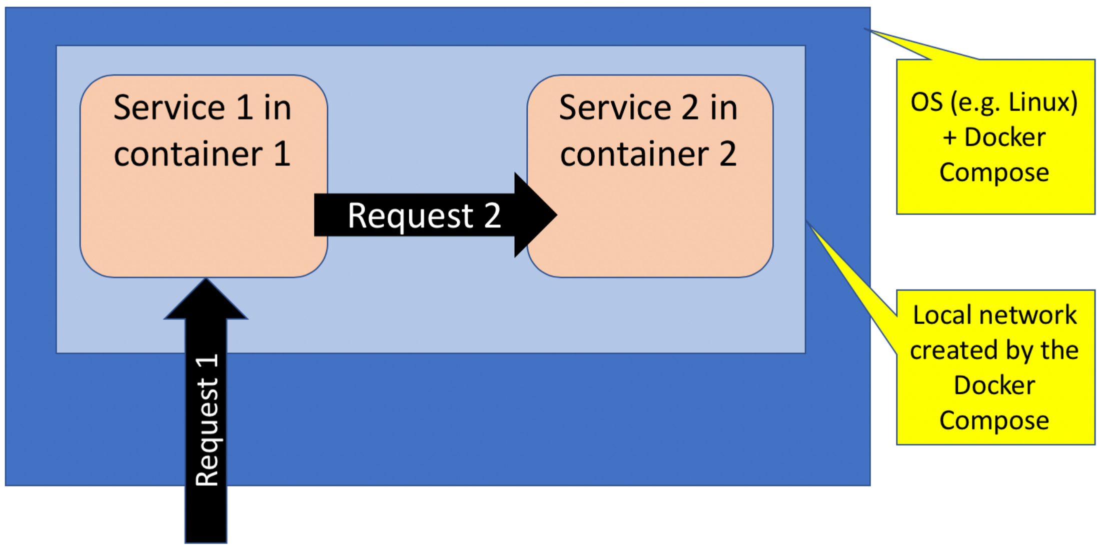
Service 1 in container 1
Service 2 in container 2
Request 2
Request 1
OS (e.g. Linux) + Docker Compose
Local network created by the Docker Compose

Figure 1. Architecture of the target system.

Service/application 1 should:

- As a response to incoming Request 1 send a HTTP GET request to Service2
- Compose a response from (4 lines of text)
  "Hello from " + <Remote IP address + port of incoming Request1[1] separated with colon (:)>
  " to " + <Local IP address and port of Service1 separated with colon (:)>
  The response from for the above request to Service2
- Return the composed response

Service/application 2 should

- As a response to incoming Request 2 compose a response (two lines) from
  "Hello from " + <Remote IP address and port of the incoming Request2 separated with colon (:)>
  " to " + <Local IP address and port of Service2 separated with colon (:)>
- Return the composed response.

[1] See the figure

By remote address/port we means the address of the host that sent the request. For example, in nodejs these can be queried with the following code:

```
http.createServer(function (req, res) {
    console.log("Req came from " + req.client.remoteAddress + ":" + req.client.remotePort);
    console.log("Req served at " + req.client.localAddress + ":" + req.client.localPort);
}).listen(8893);
```

In Golang you can use http.Request.RemoteAddr and http.Request.host. NOTE: Python has several HTTP libraries, some libraries do give access to this information. Do not use such libraries.

You should write *Dockerfiles* for the both services and *docker-compose.yaml* to start both containers so that Service1 is exposed in port number 8001. The docker-compose should also create a private network that allows Services 1 and 2 to communicate with each other but the only external access is the HTTP-port 8001 to Service 1.

The built images should have the application installed. Do not "install" it by bringing it with a volume.

The service1 is assumed to be under development, so the image is rebuilt each time the system is booted. (hint you may want to use "build:"-attribute for that service in *docker-compose.yaml*. Service2 is a reused service and you may pre-build the image. Image can be stored locally though.)

After the system is ready the student should return (in the git repository).

- Content of two Docker and docker-compose.yaml files
- A PDF document with the following content:
  - explanation why the addresses and port-numbers are like they are. (We want to ensure that you understand how your program works).
  - Output of "**`docker container ls`**" and "**`docker network ls`**" (executed when the services are up and running.
- 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 will be created by 25.09. You should prepare your system in a way that the course staff can test the system with the following procedure (on Linux):

```
$ git clone <the git url you gave>

$ docker-compose up –build

$ curl localhost:8001
<output should follow the above requirement; 4 lines
Hello from XXXX:P…
to …
Hello from …
to …
>
$ docker-compose down
```

Hints

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

Useful reading: 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=pinger .
```

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

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