Exercise 2 Documentation

The services are built with Node.js, which is a (server) runtime engine for Javascript. Both these servers (services) are providing an HTTP interface on different ports (81 and 82). The reason for having separate ports is based on how TCP connections work, explained later. Since HTTP is an application layer protocol, that resides on top of TCP protocol, which is on transportation level, it has to comply with some of the restrictions caused by the TCP protocol.

TCP uses a combination of IP address (IP protocol is a network-layer protocol, one layer below TCP which again TCP resides on) and a port number. Thus, each unique combination of an IP address and a port is delivering one connection endpoint. When there is a connection, there are two endpoints in it: the sender and the receiver – server and client.

Consequently, when testing locally, both services 1 and 2 are bind to the host address (OS internal loopback address, 127.0.0.1 alias localhost), therefore requiring a different port, to allow both services to create a new listening TCP socket on the localhost connection at the same time, to listening for those ports and addresses be possible.

In this particular case, the service2 is acting as a server, the service1 is acting as both server and client, server for the host and a client for the service2. Finally, the host running both services is acting as client for the service1. When the host connects to the service1, as how TCP connections work, as explained earlier, it reserves some random port from temporary dynamic port area (Ephemeral ports 49152 – 65535) as the connection source port and uses its source address in the same network where the service1 is.

In this case, the Docker network is bridged with the host in a way that explicitly forwared ports from the container(s) are accessible from the host, in localhost address. After reserving the address, it tries to establish a connection to the service1. This is the reason for the connection source port being some “random” high port number, seen in the response message.

Now the service1 is listening its own address and port, it receives the connection request on its open port, 8001, and accepts the incoming connection request (3-way handshake). After that, it sees that the client (host) sent a GET request to / (server root), it has knowledge to respond to it with the code written (report connection source and destination ips and ports) back to the client (host).

Also, before responding to the host, it creates a new connection, as a client, to the service2, connected internally with the service1, having the only port, 82, open inside the container network. The service1 sends also a GET request to the service2, which responds the same way to the service1. After receiving the respond, service1 concatenates the respond from service 2 and from itself and responds back to host with the complete message.

Now that both services are in their own container, they could use the same port for their services, since they are running in their own subsystems, in the docker network, having different IP addresses. Since I did some testing with the code within the same local machine, using the machine’s own loopback address that is not routed outside the machine, I had to use different port numbers for the services (81 and 82).