Cloud-native Software-defined Mobile Networks
Assignment #5
Network Service Management and Orchestration

Sharif University of Technology

Department of Electrical Engineering

Due Date:20.06.2020

Problem 1

In this problem we are going to get familiar with containers and the most popular container run time, Docker. You will also get ready for the following problems.

Getting started

Install docker and run an Ubuntu image using `docker run` command. You should be able to execute some commands in the shell of the running container (what is the difference between a container and an image?). In another terminal run `docker ps` and see your container information. Exit the shell of the container. Again run `docker ps`. Do you see the container you just exited? (note: there is nothing to deliver in this task, so have fun *dockering*!)

As you saw in the previous task, containers have only one main process and when it is finished, they no longer live. Now run an Nginx image (Nginx is a well-known web server). When we run Nginx in a machine and out of containers, we can access it via port 80 of the server. But it is not true when we run it in a container. You can access it through port 80 of the container but not the machine hosting it. Find a way to expose port 80 of your container at port 8080 of the host. This way when we send a request to port 8080 of the host, the request is sent to port 80 of the container and we can see the welcome page of Nginx. Verify it either by your browser or using `curl http://localhost:8080` (there is nothing to deliver here, too).

Most of the times we need to connect containers to each other. As you know containers are isolated processes and they can not see other ones in the system by default. So we need to execute some commands to achieve layer 3 connectivity. Use docker to create a network named same as your last name. Run two containers in that network one of which must be named iperfclient_YOUR_STUDENT_NUMBER and the other must be named iperfserver_YOUR_STUDENT_NUMBER. These containers are meant to perform iperf testing. One of them will run iperf server and the other will run iperf client. You can either use your previously pulled Ubuntu image and then install iperf or use images with iperf pre-installed. Now perform iperf testing between the two. The client must be able to reach iperf server using its name.

Deliverable

- Screenshot of the shell of iperf client successfully connecting iperf server using its name (iperfserver_YOUR_STUDENT_NUMBER)
- Screenshot(s) of output of `docker inspect network` (may have more than one page)

Problem 2

In this problem you will install OpenBaton on docker and also will use docker as a VIM.

Part 1 - OpenBaton up and running

Go to this link and install OpenBaton on docker using docker compose. Choose one installation that includes all of the necessary components. Verify your installation by checking the web UI at your browser or curl command. Now use instructions in OpenBaton's main website and connect OpenBaton NFVO to docker as its VIM. You should be able to see docker images listed in the web UI.

Deliverable

- Screenshot of docker images listed in the web UI

Part 2 - OpenBaton components

Check new docker images pulled when installing OpenBaton. List these new images and make sure you know what each one does.

Deliverable

- Answers to this question:
  What is the role of these components in OpenBaton:
  1. rabbitmq
  2. openbaton-nfvo
  3. generic-vnfm
  4. openstack4j

Problem 3

In this problem you will develop a simple NFV MANO using Python language. The NFVO receives a network service descriptor in JSON format and based on that it will create a network service. The JSON file is provided for you. For sake of simplicity the network service will have only two VNFs and a link between them, which implies that the VNFs do not reside in a same docker network.
Your implementation must contain two files: `NFVO.py` and `VNFM.py`. It is recommended to use Python classes but a functional implementation is also acceptable. Here is the API that your code must provide:

- `NFVO.get_nsd()`
  This function properly lists on-boarded NSD names.
- `NFVO.get_images()`
  This properly lists images existing in the local machine.
- `NFVO.get_ns()`
  This properly lists running network services.
- `NFVO.launch_ns(nsd_name, ns_name)`
  This launches a network service. It gets two arguments, namely, the NSD name provided by the template and the name of the network service to be launched. If the NSD is not on-boarded, it should return an error message.
- `NFVO.delete_ns(ns_name)`
  This deletes a network service by taking its name. If the name does not exist, it should return an error message.
- `NFVO.onboard_nsd(file_path, nsd_name)`
  This takes a file path for the JSON file as NSD and saves it for further operations. It returns error message if the NSD name already exists.
- `NFVO.remove_nsd(nsd_name)`
  This takes an NSD name and removes it. It should return an error message if the NSD name does not exist.

- `VNFM.create_network(network_name)`
  This creates a network in docker. Gives an error message if the network name already exists.
- `VNFM.create_vnf(image_name, container_name, network_name)`
  This creates a VNF in the given network using the given image. If the image and/or the network does not exist, it returns an error message. It also returns an error message if the container name exists.
- `VNFM.get_images()`
  This gives a list of images pulled by docker to local machine.
- `VNFM.delete_vnf(container_name)`
  This deletes a container. It returns an error message if there is no container with the name given.
- `VNFM.delete_network(network_name)`
  This deletes a network in docker. If the network does not exist or there is a container in the network, an error message must return.

Important notes:

- The only means for your code to interact with the user is NFVO. Also NFVO does not interact with the VIM (docker).
- VNFM is the only entity which runs docker commands. You can use `os` or `subprocess` modules (not necessary but recommended) in Python to run commands in the command line.
- You can add more functions to your code for extra modularity.
- All of the error messages must be written into a log file preceded with a string specific for each function's error. VNFM writes into `vnfm.log` and NFVO writes into `nfvo.log` in the directory they are in.

Deliverable

- Two python scripts: one for NFVO and one for VNFM