Introduction to Cloud Computing – Exercise 2

Scope: The lab is dedicated to working with a multi-container environment.

1. Multi-containerization

In the previous lab, we installed Docker on our test environment and got acquainted with creating and running containers. In this lab, we'll look at solutions that require splitting and running on several containers. We will also work with managing multi-container environments.

The main task of this lab is to run a prepared example of Food Trucks, simulating an example of real use of Docker.

The application is written in Python (Flask) and uses Elasticsearch for searching.

First, let's download the repository

```
$ git clone https://github.com/prakhar1989/FoodTrucks
$ cd FoodTrucks
$ tree -L 2
```

There is an app in the flask-app folder. The utils folder contains additional utilities for loading data into Elasticsearch. The directory also contains YAML files and a Dockerfile. Later in the lab, we'll look at the individual files.

Once we have the repository downloaded, we can start breaking the application into containers. Our application consists of a server and the Elasticsearch service. The natural division is to create two containers in this case - one for the application, and the other for the Elasticsearch service. In this way, our solution will be easily scalable.

The next step is to create an Elasticsearch container. Find available Elasticsearch containers in the docker hub.

How many images were found and why?

It should be mentioned here that the Elastic company that creates the Elasticsearch solution has its own repository from which we should download the latest versions of Elasticsearch.

Execute the command

```
$ docker pull docker.elastic.co/elasticsearch/elasticsearch:6.3.2
```

When the container was successfully downloaded, run it in development mode by specifying the ports and indicating that Elasticsearch should run on one Docker instance.

```
$ docker run -d --name es -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"
docker.elastic.co/elasticsearch/elasticsearch:6.3.2
```

If you find that there are free memory issues, additional memory throttling flags should be applied, more here.

What will be the name of our container after launch?

View running containers to ensure that the container has been created.

To view the ES (Elasticsearch) logs in the console, use the command

```
$ docker container logs es
```

Once Es is initialized try querying our container with the curl command

```
$ curl 0.0.0.0:9200
{
  "name" : "ijJDAOm",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "a_nSV3XmTCqpzYYzb-LhNw",
  "version" : {
    "number" : "6.3.2",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "053779d",
    "build_date" : "2018-07-20T05:20:23.451332Z",
    "build_snapshot" : false,
    "lucene_version" : "7.3.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
```

```
  "tagline" : "You Know, for Search"
}
```

When we got a response, it means that our ES container is working. The next step is to create a container for our server. The first step is to create a Dockerfile (currently it is created in the downloaded files). In the previous lab, we used the python:3 base image. In this lab, we will create our Dockerfile based on pure Ubuntu, as we will also need Nodejs to complete the task. Created dockerfile should look like below (change MAINTAINER in original file to student)

```
# start from base
FROM ubuntu:18.04

MAINTAINER Student

# install system-wide deps for python and node
RUN apt-get -yqq update
RUN apt-get -yqq install python3-pip python3-dev curl gnupg
RUN curl -sL https://deb.nodesource.com/setup_10.x | bash
RUN apt-get install -yq nodejs

# copy our application code
ADD flask-app /opt/flask-app
WORKDIR /opt/flask-app

# fetch app specific deps
RUN npm install
RUN npm run build
RUN pip3 install -r requirements.txt

# expose port
EXPOSE 5000

# start app
CMD [ "python3", "./app.py" ]
```

What is the -yqq flag for?

We can now build our container. Replace YOUR_NAME with your Docker Hub username.

```
$ docker build -t YOUR_NAME/foodtrucks-web .
```

The first run will take more time because Docker will be an ubuntu image. A later call to docker build will be much shorter. When everything is built try to run your container

```
$ docker run -P --rm YOUR_NAME/foodtrucks-web
```

What is the result?

2. Docker network

As you can see, there was a test. This is because one container cannot communicate with another container. Docker has the appropriate tools for this, but first we will try to solve this problem ourselves. Execute the command

```
 $ docker container ls
```

The output shows that the ES container is running at 0.0.0.0:9200. In order for our server's container to have access to the ES container, it should be indicated at which address the ES container is running. Take a look at the code of our Flask app (app.py). Find the line of code where the connection to ES is declared.

Once you find a suitable line, you should complete that line with the correct value. However, the value you know (0.0.0.0) is not correct, why?

To find the right value, we need to familiarize ourselves with the Docker network. During installation, Docker creates three networks.

```
$ docker network ls
```

A "bridge" network is the network where containers run by default. This means that our ES container is running on this network.

```
 $ docker network inspect bridge
```

Is the ES container visible on the network as a result of the command? In what place?

You should also get the address of the ES container in the bridge network. To see if our server container has access to ES on the indicated IP address, run the container in interactive mode

```
$ docker run -it --rm YOUR_NAME/foodtrucks-web bash
```

Then invoke the curl command with the correct ES address.

What's the result?

We managed to connect two containers with each other, but this solution has two problems:

- How can we tell a container server that hostname es belongs to an ES container? (IP addresses change, containers should be after the hostname)

- The bridge network is shared by all containers, which violates security and isolation in Docker.

To solve this problem, we can use a ready-made Docker tool. Create your own network

```
$ docker network create foodtrucks-net
```

Once created, view the available networks again

```
$ docker network ls
```

The network create command creates a new network that can be used to communicate between containers. This type of network can connect containers while maintaining isolation between networks. There are several types of networks that can be created, but in this lab we will focus on this one.

Once we have the network ready, we can restart the ES container in our network, the -- net flag is used for this. First, let's stop the running container.

```
$ docker container stop es
$ docker container rm es
```

Then restart the container

```
$ docker run -d --name es --net foodtrucks-net -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:6.3.2
```

Then see the properties of the created network

```
$ docker network inspect foodtrucks-net
```

If everything went well, you should now see your network name in "name". Then start our server

```
$ docker run -it --rm --net foodtrucks-net YOUR_NAME/foodtrucks-web bash
```

Now run the curl command again, but using the hostname

```
$ curl es:9200
```

What's the score?

Once we have a connection, we can run our server inside the container. View the available files with the ls command and then invoke

```
Python3 app.py
```

Take a snapshot of the result.

If the server started correctly, we should get information at what address our server is running. Exit the container. At this point, we can run our container properly.

```
$ docker run -d --net foodtrucks-net -p 5000:5000 --name foodtrucks-web
YOUR_NAME/foodtrucks-web
```

See if the container is running with the docker container ls command and then invoke it

```
 $ curl -I 0.0.0.0:5000
```

You can go to http://0.0.0.0:5000 take a screenshot of the page.

Even though it seems like a lot of work to be done, the entire lab process can be minimized to four commands. Below is an example bash file

```
 #!/bin/bash

# build the flask container
docker build -t YOUR_NAME/foodtrucks-web .

# create the network
docker network create foodtrucks-net

# start the ES container
docker run -d --name es --net foodtrucks-net -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:6.3.2

# start the flask app container
docker run -d --net foodtrucks-net -p 5000:5000 --name foodtrucks-web
prakhar1989/foodtrucks-web
```

Close and remove all running containers and then invoke the bash file.