Experiment 2 - Docker on Linux/Windows/Play-with-Docker

Deliverables: The following screenshots are to be submitted:

Note: The screenshots should be in .jpg format and must be submitted in a single word or pdf document on Edmodo.

- **1a.jpg:** Screenshot of running docker hello-world.
- **2a.jpg:** Screenshot of C Program successfully run inside the container.
- **2b.jpg:** Screenshot of the image pushed to Dockerhub.
- **3a.jpg:** Sample.html showing the web page on the browser.
- **3b.jpg:** Screenshot of docker container running nginx
- **3c.jpg:** Screenshot of python application successfully writing and reading from the MongoDB database
- **3d.jpg:** Screenshot showing mongodb being run within the network(docker command has to be clearly highlighted)
- **3e.jpg:** Screenshot showing python file being run within the network and successfully writing and reading from MongoDB(docker command has to be clearly highlighted)
- **4a.jpg:** Screenshot of python-mongodb application running as a docker-compose application(logs of the application)
- **4b.jpg:** Screenshot of 3 python application writes and reads from MongoDB after scaling the python application.

Few key points to note:

1. All required Dockerfile(s) have been given.
2. It is very important to go through all reference material given in the pre-reading and installation guide, as this will help you understand and debug the lab tasks.
3. **Ensure docker has been installed** before starting this lab.
4. Apart from the attached resources, you can always refer to the official docker documentation. The docker documentation is well maintained and should help you through all your tasks.https://docs.docker.com/
5. Additional resources have been given at the end of the manual.
6. When you run docker commands in the foreground, you cannot access the command prompt in which case press `[Ctrl+C]` and continue with the next step or run docker in the background mode by using `-d option`.

Task 1: Installing Docker Engine

Note! If you are using Play-with-docker, ensure you add a new instance before starting the lab.

Verify that Docker Engine is installed correctly by running : *docker run hello-world*

Take screenshot of running docker hello-world and name it as 1a.jpg

Task 2: Docker images and docker files

Sub tasks (Ensure you have created an account on Docker Hub before starting this task):

1. Pull the following images onto your docker instance using *docker pull <image-name>*
    - Ubuntu 18.04 : *docker pull ubuntu*
    - Nginx : *docker pull nginx*
    - Python : *docker pull python*
    - MongoDB : *docker pull mongo*
2. Open a text editor and create a C program and name it program.c

    Use the following C program as a base, only modify your SRN:

    ```
    #include<stdio.h>

    int main()

    {

        printf("Running this inside a container !\n");

        printf("My SRN is <YOUR SRN HERE>\n");
    ```

3. Create your own Docker image by writing a Dockerfile (template shown below). The image you will create will run a simple C program, after installing the GCC compiler. The Dockerfile must:
    a. Specify the base image as **ubuntu:18.04**
    b. Update the “apt” repository and install the GCC compiler
    c. Copy the program.c file from your instance to the docker image.
    d. Compile the C program.
    e. Run the ./a.out *command*

Dockerfile:

```
FROM ubuntu:18.04

RUN apt-get update

RUN apt-get install gcc -y

COPY program.c program.c

RUN gcc program.c

CMD ["./a.out"]
```

Save and name the file as Dockerfile.

2. After you have your Dockerfile, build the image using *docker build -t <myimage-name> .*

   myimage-name is the name you will give for your newly created docker image. **Do not forget** to **include the period** after the image name. The *period* indicates the use of Dockerfile in the local repository. You can replace this with the path to your Dockerfile.

3. Run the container using *docker run <myimage-name>*

Take a screenshot of the C Program successfully running inside the container (2a.jpg).

4. To push the image to Docker hub:
   a. Login from the terminal using

      docker login -u "myusername" -p "mypassword" docker.io

      Note: Avoid password with special characters like $ as it will be interpreted by shell and the above login command will not work.

   b. Create a tag using:

      docker tag <myimage-name> <myusername>/<myimage-name>:<version-number>

   c. Push the image using

      docker push <myusername>/<myimage-name>:<version-number>

   Login to docker hub and the image you pushed will appear in your repository.

Take a screenshot of the image pushed to Dockerhub (2b.jpg).

Task 3: Exposing ports,docker networks

1. Download the sample HTML file from the Lab 2 drive folder , and modify your SRN.
2. Create a Dockerfile and then a docker image having an **nginx** base image, and copying the html file into the default folder in the container.

Dockerfile:

```
FROM nginx

COPY my.html /usr/share/nginx/html
```

Save and name the file as Dockerfile.

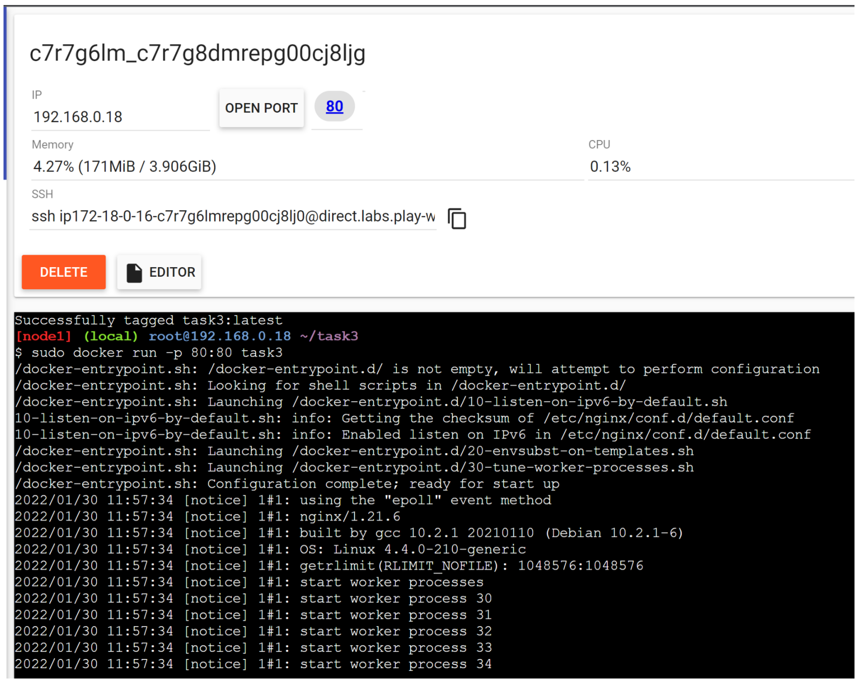
3. Build the docker image using *docker build -t <myimage-name>* **.**
4. Run the docker container using the previously created docker image and expose the HTTP port using

   docker run -p 80:80 <myimage-name>

Take a screenshot of docker container running nginx (3b.jpg).

5. Access the nginx server displaying the webpage by typing out the following url [*http://localhost:80/my.html*](http://localhost:80/my.html)

   If you are using PWD, access it via the `Open Port` button and access port 80. Append my.html to the end of the url.


c7r7g6lm_c7r7g8dmrepg00cj8ljg

IP: 192.168.0.18

Memory: 4.27% (171MiB / 3.906GiB)

CPU: 0.13%

SSH: ssh ip172-18-0-16-c7r7g6lmrepg00cj8lj0@direct.labs.play-w

```
Successfully tagged task3:latest
[node1] (local) root@192.168.0.18 ~/task3
$ sudo docker run -p 80:80 task3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/01/30 11:57:34 [notice] 1#1: using the "epoll" event method
2022/01/30 11:57:34 [notice] 1#1: nginx/1.21.6
2022/01/30 11:57:34 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/01/30 11:57:34 [notice] 1#1: OS: Linux 4.4.0-210-generic
2022/01/30 11:57:34 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/01/30 11:57:34 [notice] 1#1: start worker processes
2022/01/30 11:57:34 [notice] 1#1: start worker process 30
2022/01/30 11:57:34 [notice] 1#1: start worker process 31
2022/01/30 11:57:34 [notice] 1#1: start worker process 32
2022/01/30 11:57:34 [notice] 1#1: start worker process 33
2022/01/30 11:57:34 [notice] 1#1: start worker process 34
```

Take a screenshot of my.html showing the web page on the browser (3a.jpg).

6. Press Ctrl+C to stop the nginx server.

We will explore connectivity without docker networks and see how docker networks make it much easier to connect within containers. To demonstrate this we will create a simple application using a python client and a MongoDB NoSQL server.

Note: You don't need to know how to use mongodb, code to use mongodb has been provided to you.

7. Run the mongodb container in a detached mode, exposing the default port(27017) of mongodb using-

   docker run -dp 27017:27017 mongo

8. Download sample.py from the drive folder. Modify the SRN wherever mentioned.
9. The MongoDB container is running, but we need to find out the *IP address* of the mongodb container.
   a. Find the container ID of the mongodb container using *docker ps -a*
   b. Run *docker inspect <container-id>*. Find this IPAddress field and note this down. Modify this IP address in the sample.py file.

   Create a Dockerfile using the template given in task 2, which:
   a. Uses **python** base image
   b. Updates the apt-repository
   c. Installs **pymongo** using pip (pymongo 3.11.2).
   d. Copies the sample.py from the instance to the container.
   e. Runs the python *command* , to run the python file.

   Dockerfile:

```
FROM python

RUN apt-get update

RUN pip install pymongo

COPY sample.py sample.py

CMD [“python”, “sample.py”]
```

Save and name the file as Dockerfile.

10. Build the docker image using the above Dockerfile and run the container.

    docker build -t <myimage-name> .
    docker run <myimage-name>

Take a screenshot of python application successfully writing and reading from the MongoDB database (3c.jpg).

You should see that the data was correctly inserted and fetched from the database container.

11. Use *docker ps* to get the container id of the mongo image. Note the container id and stop running the container using

    docker stop <container-id>

The above tasks showed the connectivity between 2 containers *without* a network. This can cause problems as every time a container is created it *could possibly* have a different IP address. This is the issue docker networks tries to solve.

1. Create a docker bridge network , called **my-bridge-network.**
   docker network create my-bridge-network
2. Run a mongodb container again, but now with the following parameters;
   a. network : **my-bridge-network**
   b. name: **mongodb**
   c. Exposed ports: **27017**
   d. Image: **mongo:latest**

docker run -dp 27017:27017 --network=my-bridge-network --name=mongodb mongo:latest

Take a screenshot showing mongodb being run within the network(docker command has to be clearly highlighted) (3d.jpg).

3. Go back to sample.py , comment line 3 and uncomment line 4. We are now going to use the name of the database containers as the host name, leaving the ip address resolution to docker.
4. Build the python app docker image again as you did previously and run the container using the built image. You should see that insertion and retrieval have been done successfully.

   docker build -t <myimage-name> .

   docker run --network=my-bridge-network <myimage-name>

Take a screenshot showing python file being run within the network and successfully writing and reading from MongoDB(docker command has to be clearly highlighted) (3e.jpg).

Task 4: Docker compose

1. For the purposes of this lab make sure all the following files are in the same directory
   a. docker-compose.yml (present on the drive)
   b. Dockerfile (same file used for task 3)

    c. sample.py (same file used for task 3)
2. Go to the docker-compose.yml file and try to understand the syntax and what each line does.
3. Within the same directory, run: *docker-compose up*

Take a screenshot of python-mongodb application running as a docker-compose application(logs of the application) (4a.jpg).

What you see is that Docker compose has built your python application, started the MongoDB server, created links internally between the containers (network) and started both the containers together as a *unified application*.

The python container exits since it's done with its utility of writing and reading to the database. (Press CTRL-C to exit Docker compose if the shell prompt is not returned)

4. Now we will scale the python application, so that we have 3 containers of the python application, but keep only one container of the mongodb.

   docker-compose up --scale pycode=3

Take a screenshot of 3 python application writes and reads from MongoDB after scaling the python application (4b.jpg)

Additional Resources/Common bugs you might encounter:

1. How to debug and fix common docker issues.
2. Not able to build/run docker containers due to insufficient space: How To Remove Docker Images, Containers, and Volumes
3. Docker - Container is not running
4. Docker CLI & Dockerfile Cheat Sheet
5. Docker Build: A Beginner's Guide to Building Docker Images
6. Docker Container Tutorial #8 Exposing container ports
7. Overview of Docker Compose