SWEN 438 *Special Topic: DevOps*

Laboratory 3: Docker

The purpose of this laboratory is to get started building and running Docker containers, as well as pushing them to the Container Registry on GitLab.

Getting Started

To begin with, create a suitable repsitory on GitLab (e.g. entitled `Lab_3`). Next, enable the Container Registry in your repo by selecting `Settings` → `General` → `Visibility, project features, permissions`. The click the `Container registry` slider to enable the registry which is disabled by default. You will also need to click the `save changes` button. Upon doing so you will be able to see `Container registry` as an option under `Packages & Registries`

If you are working on the lab machines, you will need to `ssh` into a machine that is Docker-enabled (e.g. `co246a-1` .. `co246a-8`).

If you are working on your own machine, you will need to *install* Docker.

Once you have installed Docker, you will need to login to the ECS GitLab server. Run the following command in the command prompt (or equivalent) and enter your GitLab details.

`docker login gitlab.ecs.vuw.ac.nz:45678`

Part 1: Writing a Docker Image

Inside your repo, create a new file named `Dockerfile` the pathfile to your Dockerfile should be: `https://gitlab.ecs.vuw.ac.nz/course-work/swen438/2021/<username>/Lab_3` *Note* that you will need to change `<username>` to your ECS username and `Lab_3` to the name of your project should you choose a different name.

Inside your repo you should create two directories: `busybox` and `pacman`. For Parts 1, 2 & 3 you will be in the `busybox` file directory. For Part 4 you will be in the `pacman` file directory.

Using a text editor, you will edit your `Dockerfile` to create your image, remember you can pull from an existing *base* image or you can create your image from scratch. It should look something like this:

```
FROM busybox
LABEL maintainer="<your name>"

CMD ["/bin/ls"]
```

Part 2: Building and Running a Docker Image

Running the following command will allow you to build your Docker Image with the tag `latest`:

```
docker build --tag gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3 .
```

Note: that you will need to ensure you specify the directory by including `.` as part of your build command.

Note if you don't specify a `--tag` name using `image:tagname` then docker automatically names the image `latest`. This will become problematic when we want to push multpile containers to our registry, i.e. one for Busybox and one for Pacman. It would be best practise to tag *both* of our images what they are:

```
docker build --tag gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3:bu
sybox .
```

Upon running the `build` command you will see the output of the build that should look something like this:

```
$ docker build --tag gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3:
busybox .
[+] Building 3.7s (6/6) FINISHED
 => [internal] load build definition from Dockerfile                       0.1s
 => => transferring dockerfile: 106B                                       0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load metadata for docker.io/library/busybox:latest          3.2s
 => [auth] library/busybox:pull token for registry-1.docker.io             0.0s
 => CACHED [1/1] FROM docker.io/library/busybox@sha256:0f354ec1728d9ff32e  0.0s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:24bcd6cc8d64a775fbfbd59f8fd96bfa15e61f3401dbe  0.0s
 => => naming to gitlab.ecs.vuw.ac.nz:45678/../<username>/lab-3               0.0s
```

You should now be able to see your freshly made image by executing `docker images` in the command prompt:

```
$ docker images
REPOSITORY                                          TAG     IMAGE ID       CREATED
SIZE
gitlab.ecs.vuw.ac.nz:45678/../<username>/Lab_3  busybox  24bcd6cc8d64   2 minutes ago    1
.24MB
```

We can run our docker image by running the following command:
`docker run gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3:busybox`

And we can check to see if the container is running by using `docker ps` if the container *is* running it will be listed, however if it is *not* we won't be able to see it. The command `docker ps -a` gives us all the images that have run and their status.

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

$ docker ps -a
CONTAINER ID   IMAGE   COMMAND       CREATED          STATUS                     PORTS  NAM
ES
0408bc297ae4   lab-3   "/bin/ls"   7 seconds ago    Exited (0) 6 seconds ago         zealo
us_wilson
```

Part 3: Push to Container Registry

Running the following command will push your image to the `Container registry` within your GitLab project:
`docker push gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3`

Upon running the `push` command you will see the output of the push that should look something like this:

```
$ docker push gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3:busybox
Using default tag: latest
The push refers to repository [gitlab.ecs.vuw.ac.nz:45678/<username>/lab-3]
5b8c72934dfc: Pushed
latest: digest: sha256:bcc8a71fc4b3b8e2671d5a6bca1e9af9ae3e7e1951bcd2fe21402dd7dd327ee1 si
ze: 526
```

By navigating to your container inside your GitLab repository in a web browser you should see the container you have created.
`https://gitlab.ecs.vuw.ac.nz/course-work/swen438/2021/<username>/Lab_3/container_registry`

Part 4: Running Pacman Server

We're now going to setup the pacman server to run inside a docker container. This requires a little bit of preparation. You need:

1. **(Server Jar)**. The first step is to create an executable Jar for the pacman server. You can do this by exporting the Pacman server from Eclipse as a Jar file. Remember to select the "Main" class on the last part.
2. **(Client Jar)**. You can simply download this from the SWEN438 homepage.
3. **(Board File)**. You will need an example board file. For example, `classic.txt`. However, you will want to modify this board file to remove any features you have not yet implemented (e.g. powerups `*` and other ghosts `IPC`).

Note: if you have trouble exporting the server executable jar file from Eclipse, it can be downloaded from here.

Copy these files to you repository root and check that they work on your local machine as follows:

```
$ java -cp pacman-server-v1.0.jar pacman.server.Main -server 1 classic.txt
LISTENING ON PORT 32768
AWAITING 1 CLIENTS
```

At this point, the server is running locally and you can kill this process (since the test was successful). The next step is to create a Docker image which will run our PacMan server. The following illustrates a suitable Dockerfile:

```
FROM openjdk:11

# Copy over files
COPY pacman-server-v1.0.jar /
COPY classic.txt /
# Set default command to run
CMD ["java","-cp","pacman-server-v1.0.jar","pacman.server.Main","-server","1","classic.txt
"]
```

This `Dockerfile` is based from the image `openjdk:11` which includes a Java Virtual Machine. The `Dockerfile` then copies over the files we need to run the server, and sets the default command. You should then build this image (as above) and you can run it as follows:

```
$ docker run gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>/Lab_3:pacman
LISTENING ON PORT 32768
AWAITING 1 CLIENTS
```

This time, leave the server running and e.g. switch to another terminal tab. We can try to run the client on our local machine as follows:

```
$ java -jar pacman-client-v1.0.jar localhost:32768
TRYING localhost:32768
I/O error: Connection refused (Connection refused)
...
```

This has failed because we cannot "see" the PacMan server running inside Docker. To make it visible from the outside, we must map a port from our local machine to the Docker image. The simplest way to do that is through the `-p` command-line switch as follows:

```
$ docker run -p 32768:32768 gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>
/Lab_3:pacman
LISTENING ON PORT 32768
AWAITING 1 CLIENTS
```

This time, we can connect to the server from our client as follows:

```
$ java -jar pacman-client-v1.0.jar localhost:32768
TRYING localhost:32768
CONNECTED TO localhost:32768
CLIENT RUNNING
I/O Error: null
java.io.EOFException
        at java.base/java.io.DataInputStream.readInt(DataInputStream.java:397)
...
```

We can see that the client has connected, but things have gone wrong. Looking on the server side, we notice the

following:

```
$ docker run -p 32768:32768 gitlab.ecs.vuw.ac.nz:45678/course-work/swen438/2021/<username>
/Lab_3:pacman
LISTENING ON PORT 32768
AWAITING 1 CLIENTS
ACCEPTED CONNECTION FROM: /172.17.0.1
ALL CLIENTS ACCEPTED --- GAME BEGINS
Exception in thread "main" java.lang.NoClassDefFoundError: pacman/client/util/Connection
```

The problem is that the server requires the `pacman-client-v1.0.jar` in order to work correctly. Therefore, you should update your `Dockerfile` to copy over this file and rebuild the image. You will need to update the `CMD` line to replace `"pacman-server-v1.0.jar"` with `"pacman-client-v1.0.jar:pacman-server-v1.0.jar"`. At this point, you should be able to run the server and connect using the client!

Upon finishing this part of the lab you should push your container to the GitLab container registry by following the steps in Part 3. And push your Dockerfile and other documents to the repository.

Submission and Marking

For this first lab, please submit a file `submission.txt` containing a link to your repository via the ECS Submission system https://apps.ecs.vuw.ac.nz/submit/SWEN438/Laboratory_3.

Full marks will be earned for meeting the following criteria:

1. BusyBox image is in Container Registry and can be run
2. Pacman image is in Container Registry and can be run