DEVOPS FOR AIOT
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING, SINGAPORE POLYTECHNIC

LABORATORY 6: INTRODUCTION TO DOCKER

Objectives

By the end of the laboratory, students will be able to

- Use Docker to containerize their Python projects with all required dependencies into a Docker container

Activities

- Installation and setup of VMware Workstation
- Introduction to Docker Desktop and command line interface
- Familiarization with instructions in Dockerfile
- Containerization of a Python application

Review

- Docker image is created locally for a Python Flask Web Application
- Pushed Docker image to Docker Hub

Equipment:

- Git / GitHub
- VMware Workstation Pro
- Ubuntu Linux 18.04.6 LTS

1 Installation and Setup of VMware Workstation Pro

In this lab exercise we will install VMware Workstation Pro and setup a Linux OS Virtual Machine which we will use to run Docker and Kubernetes.

- Go to Broadcom.com.
- In the upper right corner, select 'Support Portal'.
- Either log in by clicking 'Go To Portal' or 'Register' for a basic Broadcom account.
- Once logged in, go to support.broadcom.com if you're not redirected there.
- Click the dropdown to choose the VMware Cloud Foundation division.
- On the left, click 'My Downloads'.
- Search for “Workstation”.
- Click the product name (VMware Workstation Pro).
- Notice the dropdown for the Personal Use edition (it is the exact same binaries as the Commercial one).
- Select the latest version for personal use.
- Download and install.

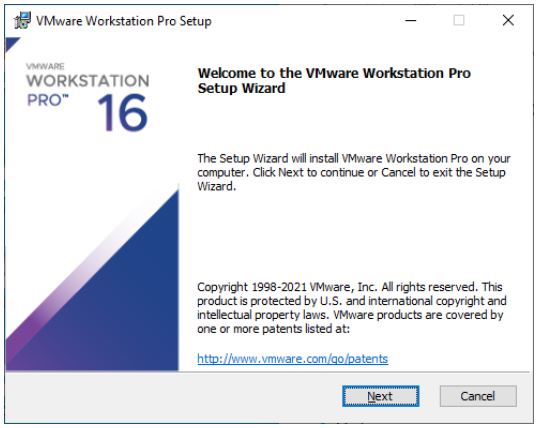

Figure 1

- After the installer runs for the first time, if “Microsoft VC Redistributable” is not currently installed on the PC, the installer will first install this and then request for a restart.

- Select “Yes” to restart the PC (Figure 2) and continue the actual VMware workstation after the restart.

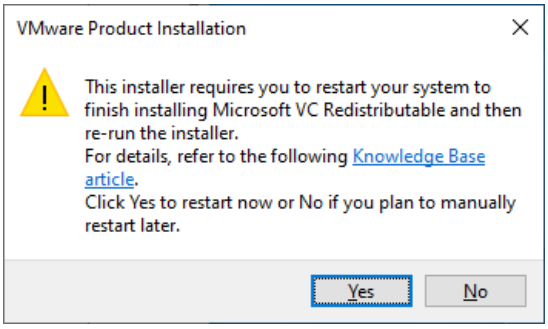

Figure 2

- After restarting PC, run the installer "VMware-workstation-full-?????.exe" again to continue the installation of VMware workstation.

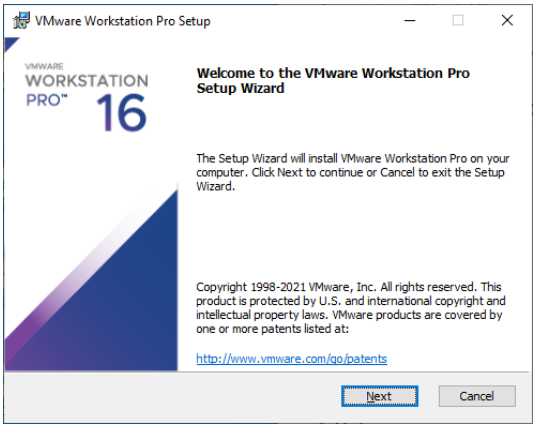

Figure 3

- Click "Next" to continue.

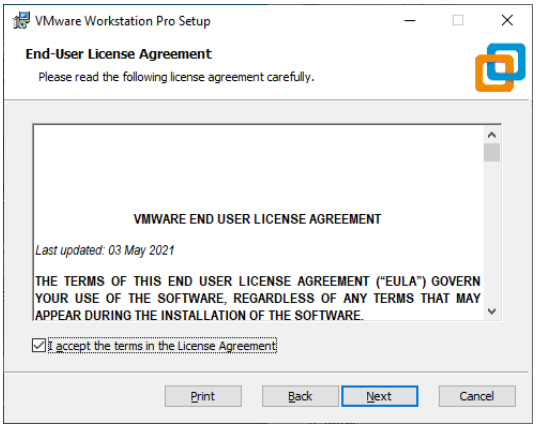

Figure 4

- Click "Next" to continue.

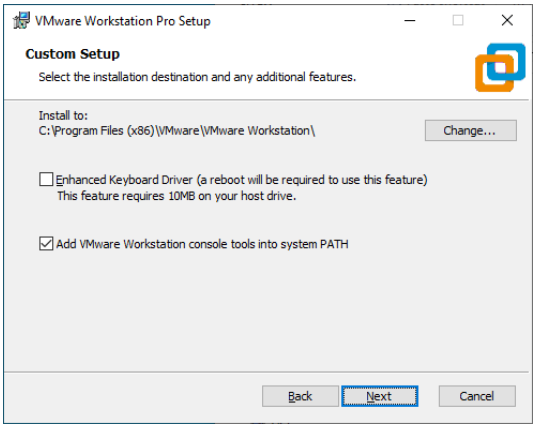

Figure 5

- Click “Next” to continue.

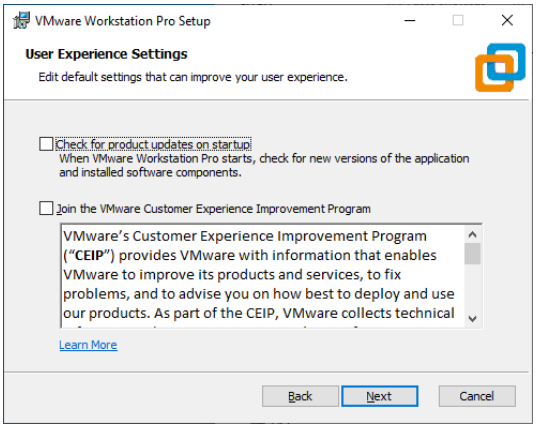

Figure 6

- **Unselect both options** for the following,
  - “Check for product updates on startup”
  - “Join the VMware Customer Experience Improvement Program”
- Click “Next” to continue.

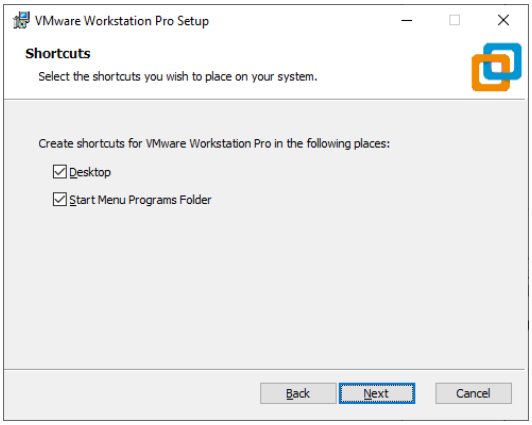

Figure 7

- Click “Next” to continue.

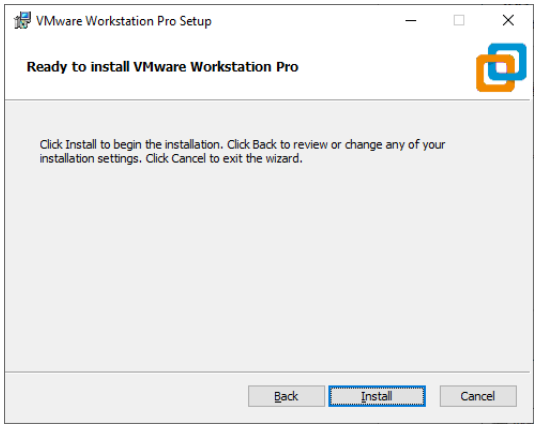

Figure 8

- Click “Install” to start the installation.

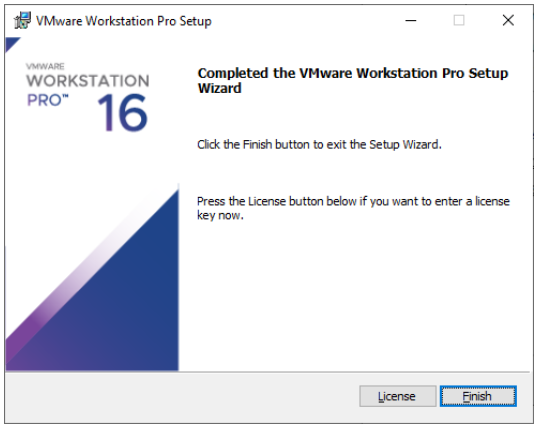

Figure 9

- Click “Finish” to complete the installation.
- For the license make sure to select for “personal use”.

1.1. Import Virtual Machine (VM)

Using VMware Workstation we can import pre-configured Virtual Machines installed with on different Operating Systems like Linux, Windows, etc.

For this Lab exercise we will import a pre-configured Linux VM with Docker and Kubernetes pre-installed.

- Download the Virtual Machine compressed zip file from the following location:

  URL to download Ubuntu Linux VM here → VMs

- Unzip the zip file into a new folder in your laptop.

- To import a Virtual Machine image, in VMware Workstation go to “File → Open” then select the file “**ET0735 - DevOps for AIoT [Ubuntu Linux].ovf**” (See Figure 10).

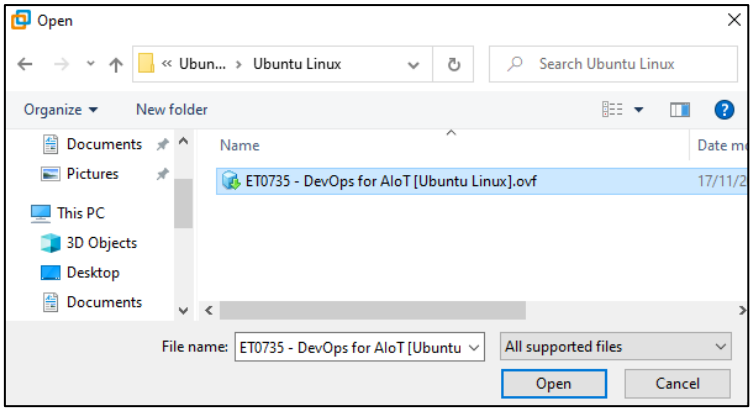

Figure 10

- Follow the settings shown in Figure 11 to import the Linux Virtual Machine.
- Select a suitable "Storage path for the new virtual machine" on your own laptop.

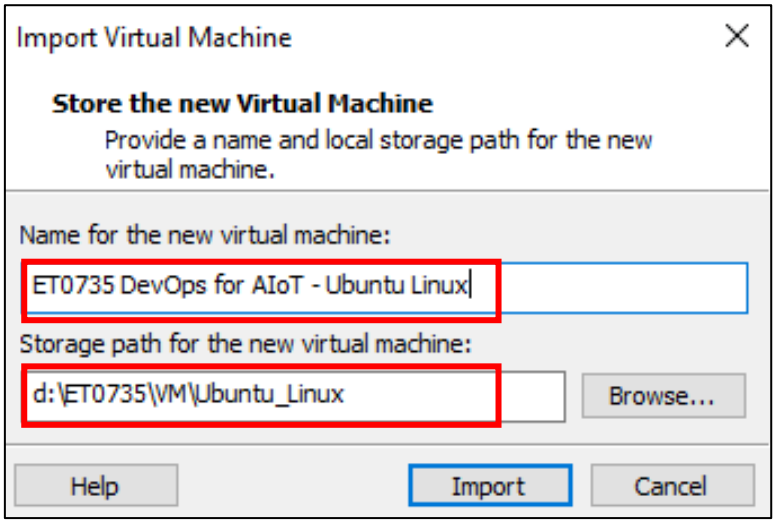

Figure 11

1.2. Virtual Machine Network Settings

To access the Internet from the VM, we need to first configure the VM instance's Network Settings to **NAT** (Network Address Translation) which allows the VM instance to access the network and Internet via your host machine (your laptop).

Right click on your VM instances and then select "Settings" to view the "Virtual Machine Settings".

Select “Network Adapter” and then select “NAT: Used to share the host’s IP address” as shown in Figure 12.

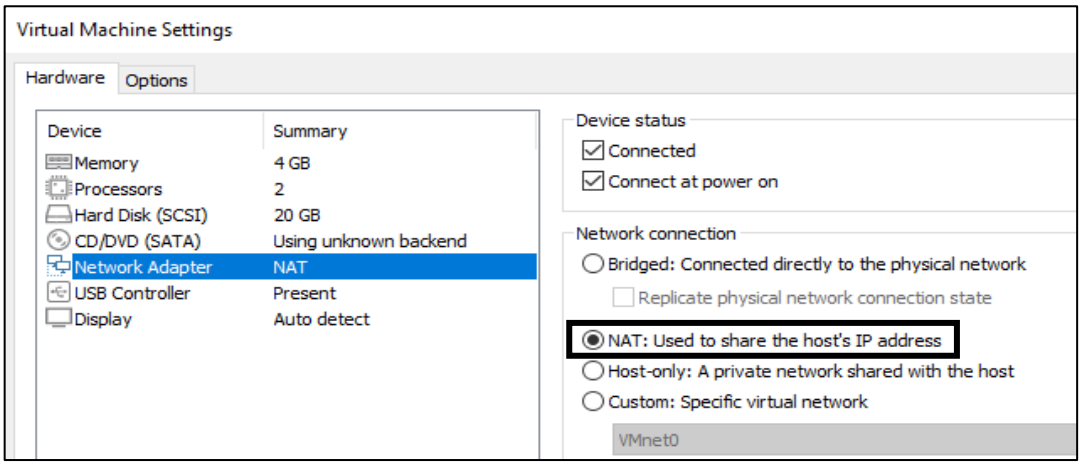

Figure 12

1.3. Running the Linux Virtual Machine

We are running Ubuntu Linux in the Virtual Machine, use the Username and Password below to log in,

Username = devops-admin

Password = password

2 Ubuntu Linux Terminal

To run the Docker commands in this lab, we will need to use the Ubuntu Terminal which allows us to run commands on the Linux Bash terminal.

Right click anywhere on the Ubuntu desktop and select “Terminal” as shown below,

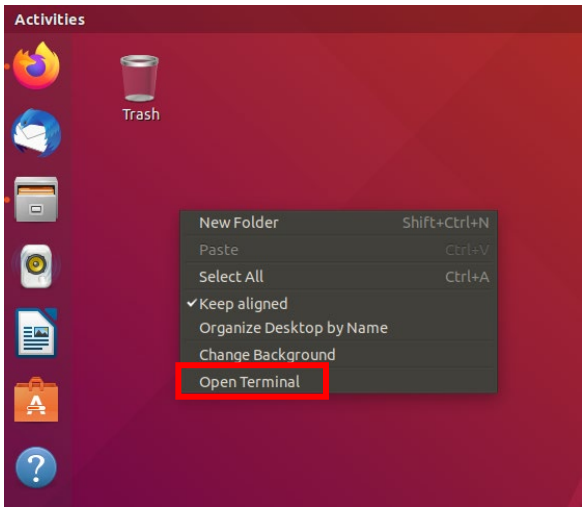

Figure 13

After selecting "Open Terminal", the Linux Bash terminal should open as shown in Figure 14.

We will use this Linux Bash Terminal for the rest of the exercises in this lab to run all the Git and Docker commands

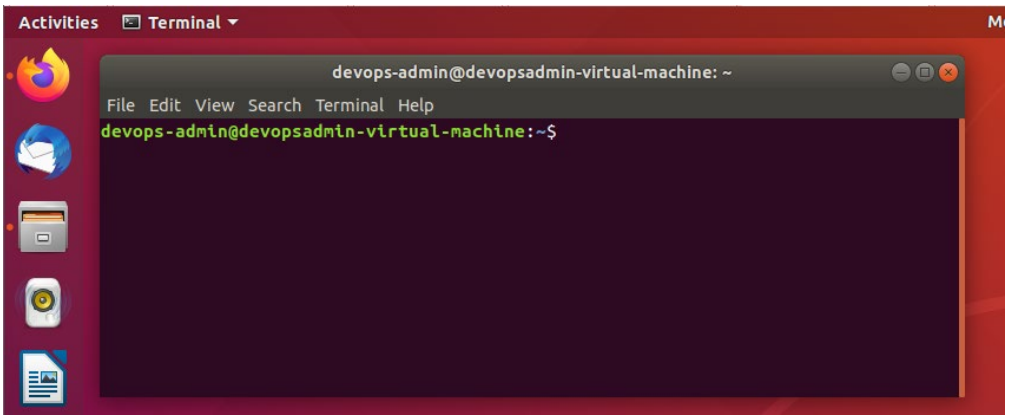

Figure 14

3 Python Flask Web Server

Before learning how to use Docker to containerize a Python application we will first run a Python Flask application which runs a simple localhost web-server locally.

3.1. Python Flask Web Application

In Git, clone the repository in the URL below to your Ubuntu Linux VM local directory at **/home/devops-admin/ET0735**

https://github.com/ET0735-DevOps-AIoT/Lab6_flask_app.git

- After cloning the repository, change to the directory **/home/devops-admin/ET0735/Lab6_flask_app** in the Linux Terminal.
- Run the Python file in the Command Prompt with the following command,

```
python3 app/app.py
```

- After running the Python Flash application above, check if the Command Prompt shows the **ModuleNotFoundError: No module named 'flask'** error as show in Figure 15.

```
devops-admin@devopsadmin-virtual-machine:~/ET0735/labs/Lab6_flask_app$ python3 app/app.py
Traceback (most recent call last):
  File "app/app.py", line 1, in <module>
    from flask import Flask, render_template
ModuleNotFoundError: No module named 'flask'
```

Figure 15

- If you encounter the error in Figure 15, try to resolve the error using the Python "pip3" tool to install the missing Python package "Flask".

- Write down in the box below the command that you used to install the Python package "Flask" to resolve the error.

python3 -m pip install flask

- After the error has been resolved, use the same command to run the Python Flask application again to see whether it now runs successfully:

```
python3 app/app.py
```

- The Python Flash script "app.py" uses the Flask library to create a simple web server which displays the HTML page located in app/templates/index.htm.

3.2. Running Python Flask Web Application

Now that all the Python dependencies have been resolved, we can now check that our Python Flask Web Server application is running.

After running the "flask-app" in the step above, you should see the console output in the Command Prompt as shown in Figure 16.

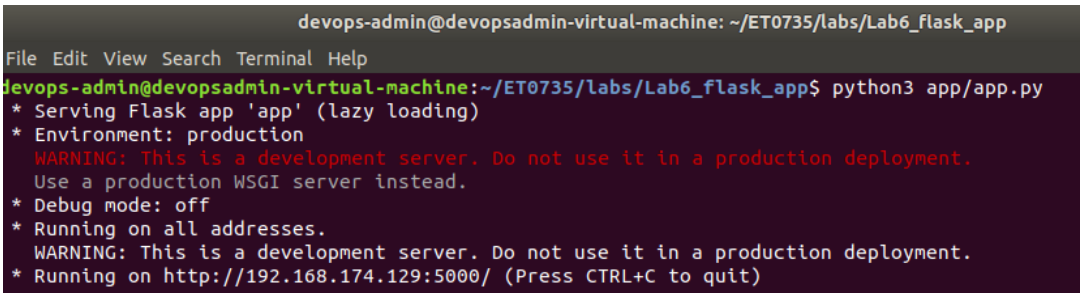

Figure 16

- To check whether the "flask-app" is running correctly, open the Firefox Web Browser and enter the localhost IP address 127.0.0.1:5000 and you should see the webpage shown in Figure 17.

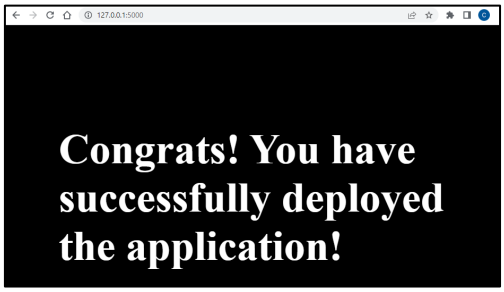

Figure 17

- Notice that after running the Python "flask-app", the Linux Bash Terminal cannot run any other commands as it continues running and blocks the terminal.

- To quit the Python "flask-app", press the Ctrl + C key combination.

4 Docker Images and Containers

Now that we have tested our Python Flask application natively on our local machine, we will proceed to generate a Docker Image and Container to run our "flask-app" application.

4.1. Docker Configuration files and Setup

In the Python Flask GitHub repository you have just cloned, notice that other than the app.py source file, in the directory "flask-app" there are also two other files "Dockerfile" and "requirements.txt".

To containerize an application, Docker needs to know all the dependencies and external libraries the application would require to compile and run.

In the Dockerfile shown in Figure 18, each line is an instruction for Docker to run when containerizing the application.

Dockerfile

```
1  FROM python:3.8-slim-buster
2
3  WORKDIR /flask-app
4
5  COPY requirements.txt .
6
7  RUN pip install -r requirements.txt
8
9  COPY ./app ./app
10
11 CMD ["python", "./app/app.py"]
```

Figure 18

The "requirements.txt" file is referenced by Docker when it runs the Python "pip" tool to install all the required Python libraries for the Flask application to run.

The "requirements.txt" file contains the list of all the necessary Python libraries and their respective versions that should be installed with "pip"

requirements.txt

```
1  click==7.1.2
2  Flask==1.1.2
3  itsdangerous==1.1.0
4  Jinja2==2.11.3
5  MarkupSafe==1.1.1
6  Werkzeug==1.0.1
```

Figure 19

4.2. Building a Docker Image

First step is to build a Docker Image from our "flask-app" application which contains all the dependencies needed by our Python code to run a Flask Web Server and Application.

- In the Command Prompt, change directory to **/home/devops-admin/ET0735/Lab6_flask_app**

- Run the following Docker command below **(Please note that there is a dot . character at the end of the command)**

```
docker build -t flask-app .
```

- After running the command above, Docker will build and generate a Docker image called "flask-app".

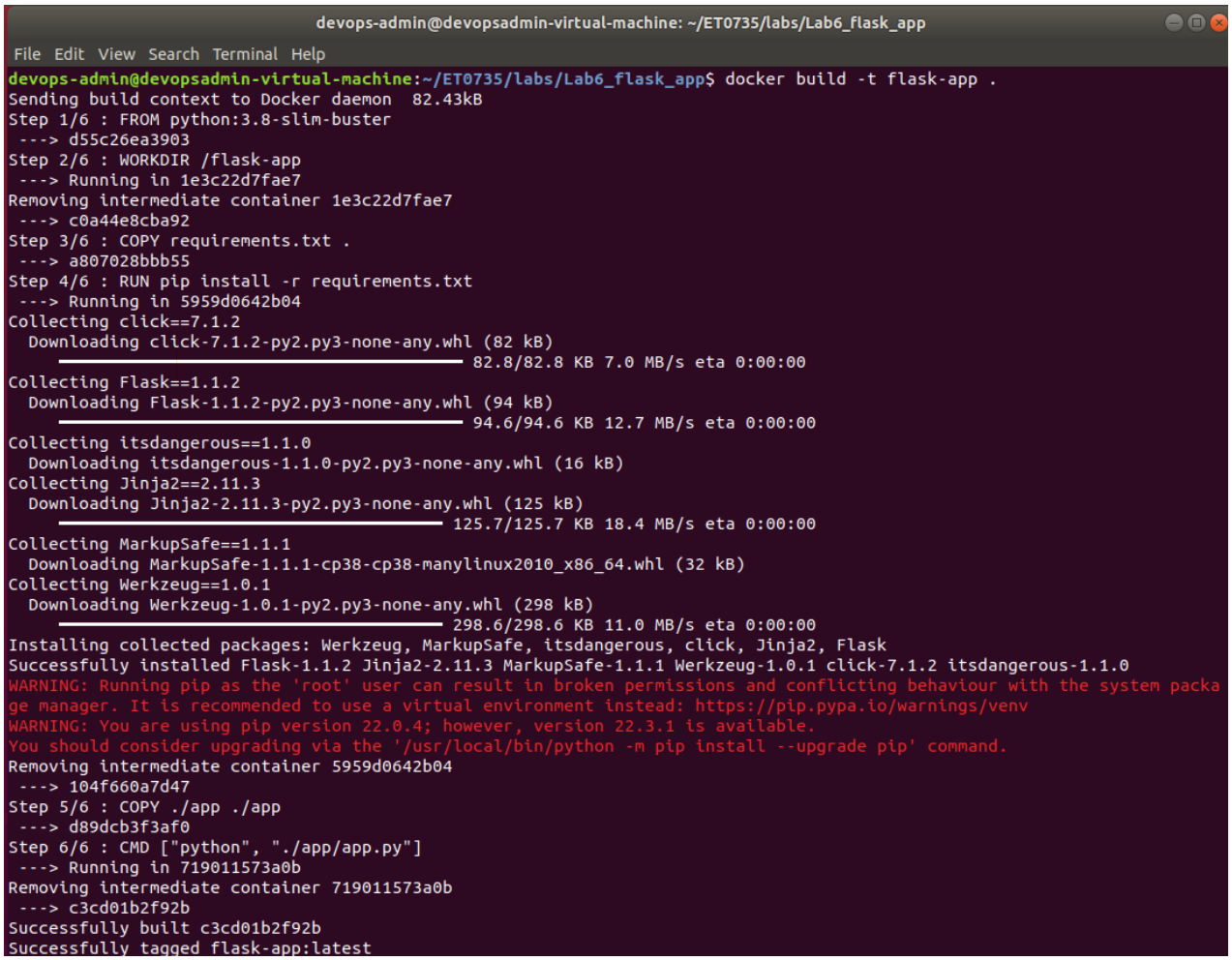

Figure 20

- To check that Docker has successfully generated the Docker Image "flask-app", run the command below which lists all the Docker images locally:

```
docker image ls
```

```
devops-admin@devopsadmin-virtual-machine: ~/ET0735/labs/Lab6_flask_app
File  Edit  View  Search  Terminal  Help
devops-admin@devopsadmin-virtual-machine:~/ET0735/labs/Lab6_flask_app$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
flask-app           latest              c3cd01b2f92b        About a minute ago   128MB
```

Figure 21

4.3. Running a Docker Container

Now that we created a Docker Image, the next step is to run the Docker container and start our Python "flask-app" Web Application.

- In the Command Prompt, enter the command below which runs the Docker container "flask-app".

  ```
  docker run flask-app
  ```

- Notice in Figure 22 that the Command Prompt is again unresponsive and cannot run any other command while the "flask-app" application is running.

```
devops-admin@devopsadmin-virtual-machine: ~/ET0735/labs/Lab6_flask_app
File  Edit  View  Search  Terminal  Help
devops-admin@devopsadmin-virtual-machine:~/ET0735/labs/Lab6_flask_app$ docker run flask-app
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figure 22

- Test if the Docker container hosting the Python "flask-app" application is running correctly by entering the local IP address **127.0.0.1:5000**.

- Notice that despite already running the Docker container for "flask-app" the application still does not seem to run correctly as depicted by the web browser response ("This site can't be reached") as shown in Figure 23.

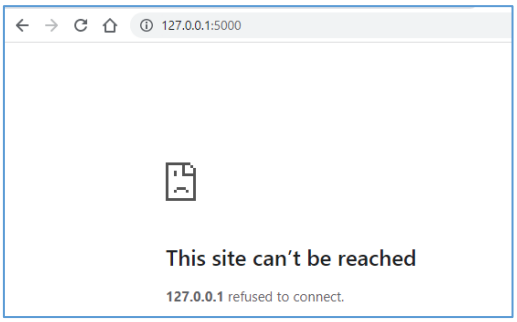

Figure 23

- The Docker command `docker run flask-app` runs our Docker container for the "flask-app", however it runs the Web Application using the internal container HTTP Port 5000 which is not yet mapped to our local machine Port 5000.

- To map the internal Docker port 5000 to our local machine's port 5000, we need to run the following Docker command:

  ```
  docker run -d -p 5000:5000 flask-app
  ```

- In the console output after running the Docker Container "flask-app", notice now that after running the Docker container it returns to the Linux Bash Terminal and does not block other commands. This is because of the **-d** flag which stands for "**detached mode**".

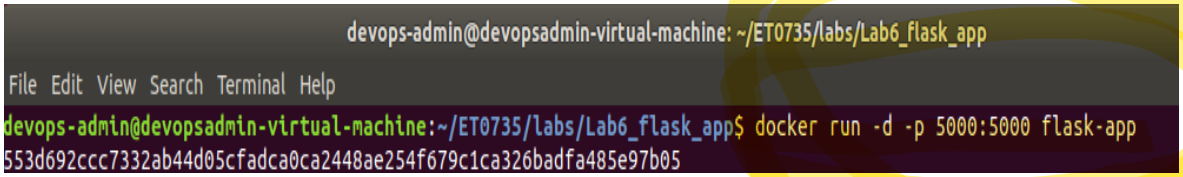

Figure 24

- Finally open the web browser and enter the IP address 127.0.0.1:5000 and check that the "flask-app" can now be displayed in the web browser.

5 Pushing Docker Images to Docker Hub

We now need to create a new Docker Hub account which we can use to push and store our Docker images to the cloud.

5.1. Registering for Docker Hub account

Similar to GitHub, Docker Hub is a cloud based service that we can use to push and upload (store) our Docker containers which can be deployed later on different platforms.

Before using Docker Hub, we first need to register and create a free Docker Hub account that we will later use to upload our Docker containers for deployment

Go the URL below and register for a new Docker Hub account **using your SP ichat email address:**

https://hub.docker.com/

After registering for a Docker Hub account, log in to Docker Hub in the web browser.

5.2. Log in to remote Docker Hub account in Linux Bash Terminal

Before we can "push" our locally created Docker image to our Docker Hub account, we first need to log in via Docker command line to Docker Hub.

Login to the Docker Hub account we have just created by running the following Docker command and enter your Docker Hub credentials for username and password when prompted,

```
docker login
```

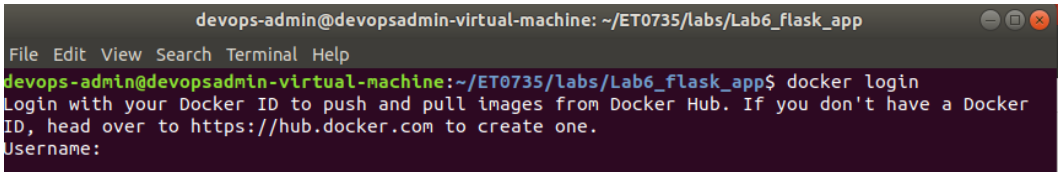

Figure 25

After successfully logging into Docker Hub from the command line, you should see the console output in Figure 26.

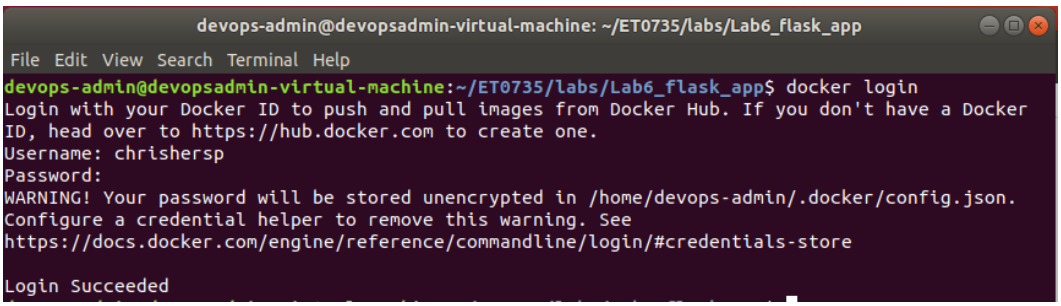

Figure 26

5.3. Tagging Docker Image to Docker Hub namespace

In this step we will now use the Docker command line to "tag" our local Docker Image to an image that follows the namespace required by Docker Hub.

- Change to the directory of the Python "flask-app".
- Run the following Docker command to tag the "flask-app" docker image

  ```
  docker image tag flask-app {your namespace}/flask-app
  ```

  where {your namespace} is your Docker Hub user ID.

- After creating the Docker image tag, run the command below to verify that the image tag has been successfully created as shown in Figure 27.

  ```
  docker image ls
  ```

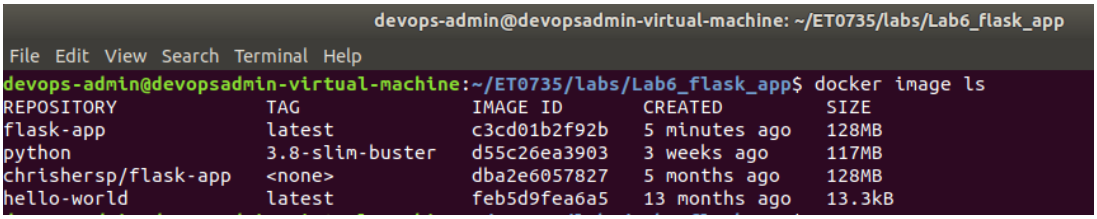

Figure 27

5.4. Pushing Docker Image to Docker Hub

Now that we have tagged our Docker Image for the "flask-app" to the required Docker Hub namespace, we are ready to push the image to Docker Hub.

- Run the following Docker command to push the "flask-app" Docker Image to Docker Hub

  ```
  docker push {your namespace}/flask-app
  ```

  Note: Replace {your namespace} above with your own Docker Hub Username.

- After running the Docker push command, check that the "push" action completes without any error in the Linux Bash Terminal as shown in Figure 28.

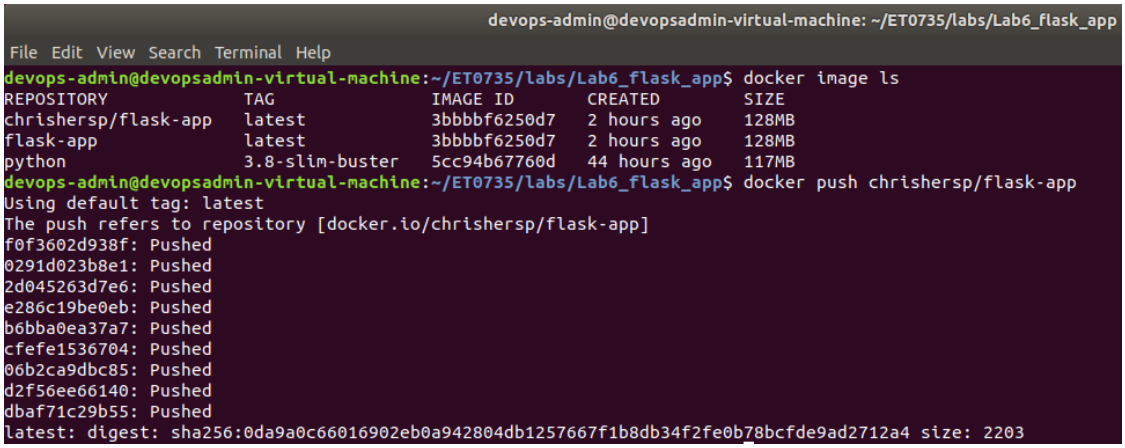

Figure 28

5.5. View Docker Images on Docker Hub

After pushing the "flask-app" to Docker Hub, we need to check that the image is successfully pushed.

- At the Docker Hub web-page, click "Repositories" to view your Docker Hub images that you have pushed as shown in Figure 29.

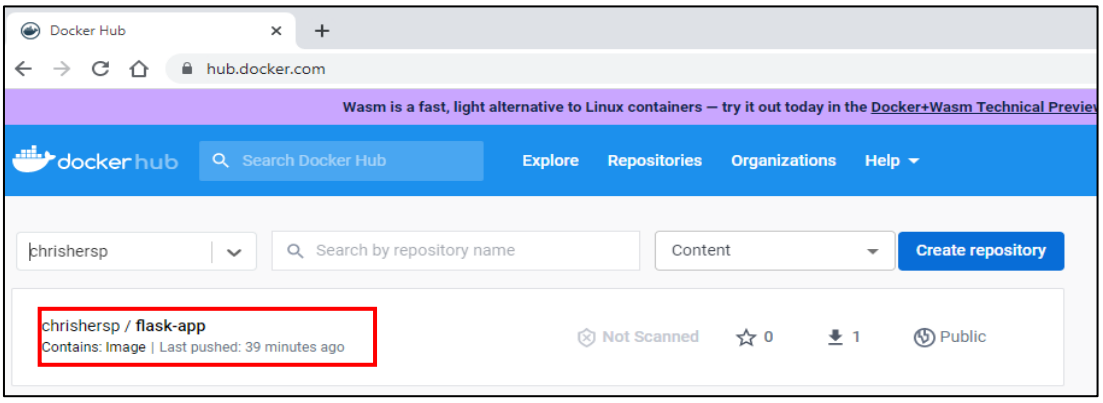

Figure 29