



SOFE 4610U:
Internet of Things

Lab #5: Using Docker for Automated System Deployment on the Raspberry Pi

Contents

1. Objectives	3
2. Important Notes	3
What is Docker?	3
3. Lab Activity	3
3.1 Software Setup	3
3.2 Create a Docker file to run your own application	6
3.3 Web server.....	7
3.4 Display DHT11 sensor data on the web page built on task3	7
4. Deliverables	8

1. Objectives

- Understand the design goals of Docker
- Setup Docker on the Raspberry Pi
- Download and run a Docker image from Docker Hub
- Design your own custom Docker Image for automated deployment of your IoT solution

2. Important Notes

- Work in groups of **two or three** students
- All reports must be submitted as a PDF on blackboard, if source code is included submit your report as PDF and source files as an archive (e.g. zip, tar.gz)
- Save the submission as <lab_number>_<first student's id> (e.g. lab1_100123456.pdf)
- If you cannot submit the document on blackboard then please contact the TA with your submission: David Lennick <david.lennick@ontariotechu.net>

What is Docker?

The slogan for Docker is “Build, Ship, and Run Any App, Anywhere”. This is the goal of Docker – to package an application, and all its dependencies so that it can be deployed to a wide variety of system architectures, easily. For example, a web application may use Apache HTTP server, MySQL database, and PHP scripting language. Traditional setup of this web application is complicated as it requires the installation of all individual components and their manual configuration. For example, when installing a web application, one may have to install the MySQL database server and then run a script to setup the required tables in that database. Docker simplifies this process; by building the application and all its dependencies into an independent Linux container, it can easily be moved from platform to platform without worrying about conflicts with the software installed on the host.

3. Lab Activity

3.1 Software Setup

1. First to install the docker on Raspberry Pi

```
$curl -sSL https://get.docker.com | sh
```

*This will take around 5-7 minutes

```
$curl -fsSL https://get.docker.com -o get-docker.sh && sh get-docker.sh
```

* This might take 2 mins

```
$docker --version
```

* To check the docker's version

2. Add the user "pi" to the docker group to access the Docker engine.

Run the following

```
$sudo usermod -aG docker pi  
  
$sudo reboot
```

3. Try to run a container

```
$docker run hello-world
```

Now you run, you should see it recommends that you should run something more ambitious, such as Ubuntu.

```
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (arm32v7)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/
```

The following command will download Ubuntu bash using

```
$docker run -it ubuntu bash
```

*it's from the command suggested after hello-world

*It might take a min or 2 to download the Ubuntu

then you are root prompt

```
root@xxx#ls
```

```
root@50e9743805d7:/# ls
bin    dev    home   media  opt    root   sbin   sys    usr
boot   etc    lib    mnt    proc   run    srv    tmp    var
```

root@xxx#more /etc/os-release

it could show as 20.04.3 LTS as the latest Ubuntu.

```
pi@raspberrypi:~$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
c0cc98128e2b: Pull complete
Digest: sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f
Status: Downloaded newer image for ubuntu:latest
root@0703747cb8d0:/# ls
bin boot dev etc home lib media mnt opt proc root run sbin srv sys tmp usr var
root@0703747cb8d0:/# more /etc/os-release
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
root@0703747cb8d0:/#
```

Now the ubuntu container is running. Open another terminal tab to check

```
docker ps
```

```
pi@raspberrypi:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
0703747cb8d0   ubuntu   "bash"    4 minutes ago   Up 4 minutes           determined_pike
pi@raspberrypi:~$
```

You can see it is running the ubuntu container

However, if you run `uname -a`, we are still on the raspberry pi.

root@xx#exit to get out of the container

Actually, you can run the application suck top to the cpu running

```
$ docker run -it ubuntu top
```

```
$docker images
```

There are many images out there for you to run, please check out the following URL for more information

<https://hub.docker.com/search?q=&type=image>

3.2 Create a Docker file to run your own application

Dockerfile defines what goes on in the environment inside your container and allow you to build you own application into your container. We choose C program as an example.

1. Make your own application's directory, for example "sudo mkdir docker-example"
2. New a file named Dockerfile

```
1. FROM gcc:4.9
2. WORKDIR /docker-example
3. COPY . .
4. RUN gcc -o hello hello.c
5. CMD ["/hello"]
```

3. Create hello.c file, simply printing "Hello"

Now, the directory should look like the figure.

```
pi@raspberrypi:~/docker-example $ ls -la
total 16
drwxr-xr-x  2 pi pi 4096 Sep  4 12:17 .
drwxr-xr-x 22 pi pi 4096 Sep  4 12:17 ..
-rw-r--r--  1 pi pi   86 Sep  4 12:17 Dockerfile
-rw-r--r--  1 pi pi  118 Sep  4 12:17 hello.c
pi@raspberrypi:~/docker-example $ pwd
/home/pi/docker-example
pi@raspberrypi:~/docker-example $
```

4. Generate a docker image

```
docker build --tag docker-example-c-hello .
```

```
pi@raspberrypi:~/docker-example $ docker build --tag docker-example-c-hello .
Sending build context to Docker daemon 3.072kB
Step 1/5 : FROM gcc:4.9
4.9: Pulling from library/gcc
e925dd4ffa2a: Pull complete
c9bfbf7dfc78: Pull complete
015138dd660d: Pull complete
d88b2b5023e5: Pull complete
4d0d77a38079: Pull complete
996bfab2b29c: Pull complete
d27243b445c7: Pull complete
2f949e025be6: Pull complete
Digest: sha256:6356ef8b29cc3522527a85b6c58a28626744514bea87a10ff2bf67599a7474f5
Status: Downloaded newer image for gcc:4.9
----> 69a20488ee66
Step 2/5 : WORKDIR /docker-example
----> Running in 86e971ef46ae
Removing intermediate container 86e971ef46ae
----> 9ceb0cabe245
Step 3/5 : COPY . .
----> 72034223cf41
Step 4/5 : RUN gcc -o hello hello.c
----> Running in aa829c9217cb
Removing intermediate container aa829c9217cb
----> b4c59fad484d
Step 5/5 : CMD ["/hello"]
----> Running in f8b263c88c03
Removing intermediate container f8b263c88c03
----> bfdded3e7c6c
Successfully built bfdded3e7c6c
Successfully tagged docker-example-c-hello:latest
pi@raspberrypi:~/docker-example $ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
docker-example-c-hello  latest         bfdded3e7c6c   12 seconds ago  1.19GB
```

5. Run the container

```
docker run -it docker-example-c-hello
```

```
pi@raspberrypi:~/docker-example $ docker run -it docker-example-c-hello
Hello!pi@raspberrypi:~/docker-example $
```

3.3 Web server

We choose Python program as an example and import Flask as web framework.

1. Create your own Python application, with a Dockerfile

The directory should look like as figure:

```
pi@raspberrypi:~/docker-example-python-web-server $ ls -la
total 16
drwxr-xr-x  2 pi pi 4096 Sep  4 15:59 .
drwxr-xr-x 23 pi pi 4096 Sep  4 12:51 ..
-rw-r--r--  1 pi pi 169 Sep  4 12:51 app.py
-rw-r--r--  1 pi pi 159 Sep  4 12:51 Dockerfile
pi@raspberrypi:~/docker-example-python-web-server $ pwd
/home/pi/docker-example-python-web-server
pi@raspberrypi:~/docker-example-python-web-server $
```

The app.py and Dockerfile could find at https://github.com/tanglz/TA_IOT/tree/main/docker-example-python-web-server

2. Build docker image

```
docker build --tag docker-example-python .
```

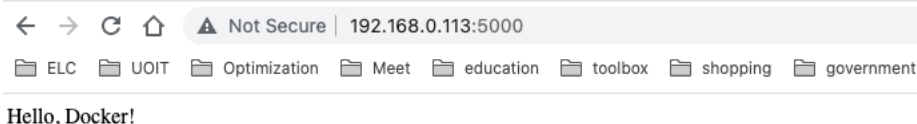
3. Run docker container

```
docker run -p5000:5000 -it docker-example-python
```

4. Check http server(please note the <http://172.17.0.3:5000> might vary depend on what you get)
Curl <http://172.17.0.3:5000>

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS                               NAMES
e9a3094f9379   docker-example-python  "python3 -m flask ru..." 29 seconds ago Up 27 seconds 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp  romantic_kirch
pi@raspberrypi:~ $ curl http://172.17.0.3:5000/
Hello, Docker!pi@raspberrypi:~ $
```

5. Check on your laptop, where 192.168.0.113 is your raspberry pi address.



The screenshot shows a web browser window with the address bar displaying "192.168.0.113:5000". The page content shows "Hello, Docker!". The browser's address bar also shows "Not Secure" and the page has a loading indicator. Below the address bar, there are several tabs: "ELC", "UOIT", "Optimization", "Meet", "education", "toolbox", "shopping", and "government".

3.4 Display DHT11 sensor data on the web page built on task3

1. From previous labs, we have saved sensor data to a csv file, and plot the data.
2. Save the plot as a .png image file. If you have trouble creating the .png file, it can be given to you.
3. Create a html file in this application, display the sensor data image on the html page
4. Start server, display data on <http://ip:5000/data.html>

4. Deliverables

Complete all lab tasks and document with step-by-step instructions such that a person with only basic knowledge of the Raspberry Pi could reproduce your work. Use of screenshots is highly recommended.

Please note, all lab reports will have title pages, introduction, content of the lab tasks and conclusion.