Full Stack Development
Lab Docker 4

Building Containers

1. Lab objectives

This lab reviews the basics of building Docker containers using Docker build.

2. Setup

You should start this lab with no containers. If you have any containers running, you should stop them, then run **docker container prune** to remove all the stopped containers

3 Commit a container as an image

Run an Ubuntu image interactively. Inside the container, create a new file and name it distinctively so that you will know it was created by you. In the example below, the **hosts** file has just been copied to **zippy**.

```
D:\Docker>docker run -it --name ubbi ubuntu

root@04e34b49bcf9:/# cp /etc/hosts zippy
root@04e34b49bcf9:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  zippy
root@04e34b49bcf9:/#
exit
```

Get the ID of the stopped container and then commit it as an image. In the example below, the new image is named ubbi:1.0. Confirm that this image exists.

```
D:\Docker>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS                          PORTS     NAMES
04e34b49bcf9   ubuntu    "bash"    About a minute ago   Exited (0) About a minute ago             ubbi

D:\Docker>docker commit 04e34b49bcf9 ubbi:1.0
sha256:6ce2dad011fada8866fdc210fbc9f3c966ffbc65a370066847062689fbc85264

D:\Docker>docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
ubbi         1.0       6ce2dad011fa   10 seconds ago   77.8MB
nginx        latest    88736fe82739   3 days ago       142MB
ubuntu       latest    a8780b506fa4   2 weeks ago      77.8MB
```

Now run the new image and confirm that the file you created previously is now part of the image.

```
D:\Docker>docker run -it  ubbi:1.0

root@4ad189ca9c33:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  zippy
root@4ad189ca9c33:/#
exit
```

4. Build an image from a Dockerfile

Clone the repository from GitHub

https://github.com/ExgnosisClasses/HelloWorldMicroService

```
D:\Docker>git clone https://github.com/ExgnosisClasses/HelloWorldMicroService.git
Cloning into 'HelloWorldMicroService'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 18 (delta 6), reused 8 (delta 2), pack-reused 0
Receiving objects: 100% (18/18), done.
Resolving deltas: 100% (6/6), done.
```

This is a Python application for a hello world sort of web service. You will be covering what the code means in later sections when you build the Java equivalent application. For now, the Python app is more compact.

Notice that the image says the application will be using container port 5000.

app.py

```python
from flask import Flask
from flask import request

import os

app = Flask(__name__)

@app.route("/")
def hello():
    return "Welcome to the hello web service"

@app.route("/hello")
def helloanon():
   return "Hello anonymous user"

@app.route('/hello/<username>')
def helloname(username):
    if username == 'Jack' :
        return "HIT THE ROAD JACK!!!"
    else :
     return 'Hello {}'.format(username)

@app.route('/hello/<int:userid>')
def hellouserid(userid):
    return 'Hello user unit number {:d}'.format(userid)

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True,host='0.0.0.0',port=port)

```

Examine the Dockerfile

Dockerfile

```
FROM python:3.10
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["app.py"]

```

1. The base image is a Python image.
2. The contents of the directory the build is being executed in are copied to a the directory ***/app*** in the container that is then identified as the working directory
3. The Python pip command is used to install the dependencies need to run the application

4. The ENTRYPOINT command specifies that the Python shell is executed at start up.
5. The CMD specifies what the Python shell should execute.

Now run the build process. Make sure you are in the same directory as the Docker file. You can use whatever name you want as the image tag.

Notice the various intermediate containers that are started and stopped. These are pruned automatically after the build is completed, as well as any intermediate images. Make sure that you remember to include the period at the end of the command.

```
D:\Docker\HelloWorldMicroService>docker build -t hithere:latest .
[+] Building 21.1s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 158B                                                0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.10                      1.0s
 => [internal] load build context                                                   0.0s
 => => transferring context: 33.33kB                                                0.0s
 => [1/4] FROM docker.io/library/python:3.10@sha256:
… lots of stuff …
=> [2/4] COPY . /app
=> [3/4] WORKDIR /app
=> [4/4] RUN pip install -r requirements.txt
=> exporting to image
=> exporting layers
=> writing image sha256:b184c4b9d5868e179df3d346c2ea90d54b044b42d6d2854f1bb3da7fad609a9a
=> naming to docker.io/library/hithere:latest

D:\Docker\HelloWorldMicroService>docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
hithere      latest    b184c4b9d586   4 minutes ago   933MB
```

Run the resulting image on port 80 and confirm it works.

```
D:\Docker>docker run -d -p 80:5000 hithere
47211bd9fb155e9b7bea360ee812a19d485eb6a9d501841f3d5cd2f1353e47ad
```

Now use exec to go into the running container and look at what its contents are. Because of the copy command, all of the contents of the build directory, including the readme file are in the container.

```
D:\Docker>docker ps
CONTAINER ID   IMAGE     COMMAND           CREATED          STATUS          PORTS                    NAMES
47211bd9fb15   hithere   "python app.py"   21 minutes ago   Up 21 minutes   0.0.0.0:80->5000/tcp   adoring_williams

D:\Docker>docker exec -it 47211bd9fb15 bash
root@47211bd9fb15:/app# ls
Dockerfile  README.md  app.py  requirements.txt
root@47211bd9fb15:/app#
exit
```

Can you modify the Dockerfile so that only python code file and requirements.txt files are in the container?

End Lab