Docker Worksheet Solutions

Alex Wright & Matthew Mostert
FEEG6003

May 6, 2017

The following are the commands required to complete the worksheet provided.

Exercise 1. To run the hello-world container, enter into the command line

```
$ docker run tutum/hello-world
```

In a new terminal

```
$ docker ps
```

lists all active (i.e. running) containers. To show all containers including inactive one, simply add the relevant flag:

```
$ docker ps -a
```

To stop any container, with a respective Docker ID, type

```
$ docker stop containerID(/containerName)
```

To run the container as a daemon, and to reroute the output of the container such that we can see it running, use the following

```
$ docker run -d -p 8080:80 -name hello tutum/hello-world
```

where the $-name$ flag labels the container *hello*, this tends to be easier than manipulating containers using their Docker ID.

When we look at the address *localhost* : 8080 using firefox, we see the output of the hello-world container. Note: the host name displayed is the container ID.

Run another hello-world container, with a different name tag, and reroute the port to 8081. We can see here how the same container has been run multiple times, with different IDs.

A quick way so stopping all active containers is the following command:

```
$  docker stop $(docker ps -q)
```

or, if necessary,

```
$ sudo docker stop $(sudo docker ps -q)
```

where the

```
$ docker ps -q
```

command returns a list of all the active containers' IDs. This sort of command is very useful for clearing up inactive containers that can build up very quickly if you are not careful!

```
$ docker rm $(docker ps --filter "status=exited")
```

will remove all containers that are currently not being used.

Exercise 2. As stated in the exercise, the layers are retrieved using the command line via

```
$ docker pull dataquestio/python3-starter
```

Each line that is outputted to the terminal represents a single layer of the container. For example, one may be the scientific python package, another may be the numba module, and so on. As you can see there are a lot of layers to this container, so the download may take some time.

Create a folder for some output, for example

```
$ mkdir exampleOutput
```

To run a Jupyter notebook inside the container, we simply run all the layers we have just downloaded as a daemon, remembering to reroute the output so that we can interact with it,

```
$ docker run -d -p 8888:8888 -v
  /home/feeg6003-docker/Documents/Docker/exampleOutput:/home/ds/notebooks
  dataquestio/python3-starter
```

As before, we check the output is correct by using firefox and entering *localhost* : 8888 into the address bar (the particular choice of port is irrelevant, as long as it is available for the browser).

When we print the current working directory of the notebook we see the path */home/ds/notebooks*. This is the location from which the container is running that notebook, hence why it is different from the location that we run the container - the container is running on a virtual machine, not our local machine. When we save the notebook, as we have specified a location on our local machine using the $-v$ flag, any changes are stored in the path specified.

To make a change on the file on our local machine, call

```
$ nano exampleOutput/path.ipynb
```

There are two blocks that are of interest here. The first is labelled *outputs*, this contains the output of any cell in the notebook that has been executed. In the first case this should be the directory the notebook is running in (on the Docker VM).

The second block is labelled *source*. To change this from what we originally set is as, replace the following line:

```
$ "print(os.getcwd())"
```

with

```
$ "print(\"Someday you will find me, caught between a landslide...\")"
```

Save and exit the editor and refresh the browser you used to create the notebook. As you can see, changes made locally on your machine (i.e. not directly using the container) are still picked up and applied.

If we want to import data from our local machine for use inside the container, there are multiple ways of doing this.

1. In the terminal use the command

   ```
   $ cp /home/feeg6003-docker/Documents/Data/exams.csv
     /home/feeg6003-docker/Documents/Docker/exampleOutput/
   ```

   In this case the path does not have to be absolute.

2. The command

   ```
   $ docker cp /home/feeg6003-docker/Documents/Docker/Data/cetdl1772on.dat
     (containerID):/home/ds/notebooks/
   ```

3. Go to the home page of the Jupyter notebook, and click the upload button on the top right to transfer files from your local machine.

   Any available files will be shown in the home page of the Jupyter notebook.