Exercise 8

Using Docker

Prior Knowledge
Unix command-line
Apt-get package manager

Learning Objectives
Be able to instantiate docker containers
Be able to modify docker containers and save them
Interacting with the docker hub
Creating a dockerfile

Software Requirements

- Docker
- Ubuntu
- Nano text editor

Introduction
This lab consists of two parts. The first part is just playing around with Docker to understand how stuff works. The things we are going to do are not typical docker usage as we are investigating the way the system works

The second part involves creating a Dockerfile, which is a sort of build file. This is the more usual usage of Docker and will stand you in good stead for many projects.

PART A – understanding the Docker model

1. Let's start by running a CentOs image inside our Ubuntu VM.

2. From the Ubuntu command-line, type:

```
sudo docker pull centos
```

3. You should see something like:

```
Using default tag: latest
latest: Pulling from library/centos

a3ed95caeb02: Pull complete
1534505fcbc6: Pull complete
Digest:
sha256:039c32c0924e4f5da9c4952a86929fe00e10c9473c3948335ff13d
00550993dd
Status: Downloaded newer image for centos:latest
```

4. We will take a look at what this means shortly, but first lets try it out.

   sudo docker run -ti centos /bin/bash

   You should see:
   [root@c22c9c908236 /]#

   Did you notice how fast it started?! This is not your usual VM.

   > ***Hint:***
   > *-ti basically means run this container in interactive mode. For more explanation see: https://docs.docker.com/engine/reference/run/*

   Let's refer to this window as the *docker window.*

5. Now type
   ls /home/oxsoa

   This will fail, because we are now in a mini virtual machine. Now try
   apt-get
   Again it fails. But what about yum?
   Why does yum succeed? Because yum is the package manager for CentOS and now we are in a CentOS world. (Actually we won't use yum or apt-get *within* the docker… we'll come to how that works shortly).

6. Start a separate window. Let's refer to this as the *control window.* Now type

   ```
   sudo docker ps
   ```

7. You will see something like:

   ```
   CONTAINER ID        IMAGE               COMMAND             CREATED
   STATUS              PORTS               NAMES
   c22c9c908236        centos:latest       "/bin/bash"         10 minutes ago
   Up 10 minutes                           drunk_engelbart
   ```

8. Docker has given your container instance a random name (in my case drunk_engelbart). You can now see how this instance is doing:

   ```
   sudo docker stats drunk_engelbart
   ```

   Obviously change *drunk_engelbart* to the name of your container!

9. Notice how little memory each container takes.

10. Now **Ctrl-C** to exit that command.

11. Now go onto http://hub.docker.com and signup. You need a valid email address to complete signup. I think you might want to do this in your own name because it's a useful system.

12. Now login from your Ubuntu host:
    ```
    sudo docker login
    ```

13. Back in the control window, type

    ```
    sudo docker commit <your_container_name> <yr_dock_id>/mycentos
    ```
    e.g.
    ```
    sudo docker commit drunk_engelbart pizak/mycentos
    ```

14. Now list the images you have locally
    ```
    sudo docker images
    ```
    You will see something like:
    ```
    REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
    SIZE
    pizak/mycentos      latest              9f154062124f        21 minutes ago      172.3 MB
    centos              latest              ce20c473cd8a        5 weeks ago         172.3 MB
    ```

15. Actually it would be useful to give that image a version name:
    e.g.
    sudo docker tag pizak/mycentos pizak/mycentos:1

16. Repeat the sudo docker images command.

17. Now let's push that image up to the docker hub:
    e.g.
    ```
    sudo docker push pizak/mycentos:1
    ```

    Replace your userid as needed.

18. The system will whirr away and upload some stuff. Eventually you will see something like:

    ```
    The push refers to a repository [pizak/mycentos] (len: 1)
    9f154062124f: Image already exists
    ce20c473cd8a: Image successfully pushed
    4234bfdd88f8: Image already exists
    812e9d9d677f: Image already exists
    168a69b62202: Image successfully pushed
    47d44cb6f252: Image already exists
    Digest:
    sha256:f751347496258e359fdc065b468ff7d72302cbb6f2310adee802b6c5ff92615d
    ```

19. Now let's go back to the original docker window, where your image is still running. Make a new file in home like this:

    ```
    [root@482fe4e23a8b /]# cd home
    [root@482fe4e23a8b home]# echo hi > hi
    [root@482fe4e23a8b home]# ls
    hi
    ```

Now in your control terminal you can commit this change:

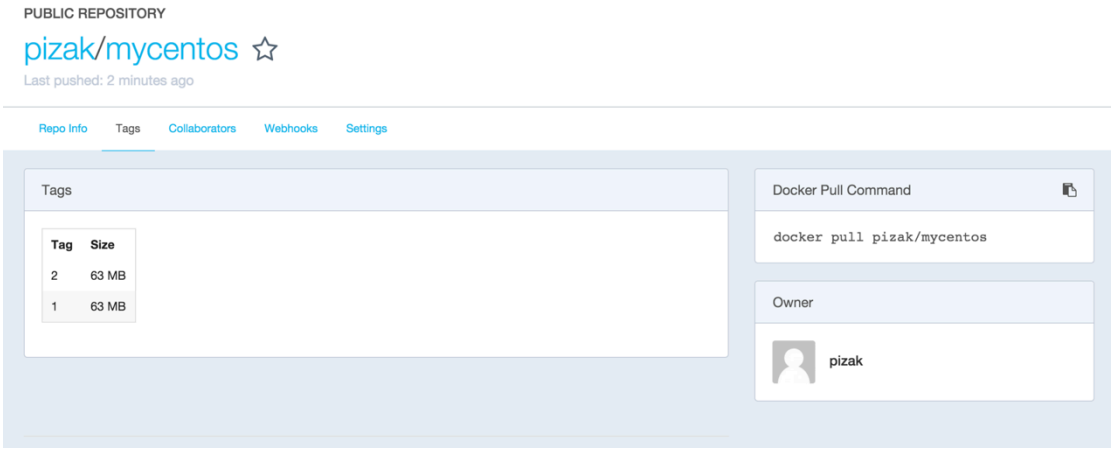
```
sudo docker commit drunk_engelbart pizak/mycentos:2
```

20. Let's push that image you've just made up to the Docker hub:

    ```
    sudo docker push pizak/mycentos:2
    ```

21. Notice how this time only a few bytes were uploaded. This is because of the layered file-system that docker uses to only save incremental changes. It is one of the major benefits of the docker system.

22. Go to the docker website http://hub.docker.com and view your repositories. In particular look at the tags tab:



23. You can now pull this docker image and create a container anywhere you like. Let's try some stuff out. From your *docker window* first exit the container by typing exit or Ctrl-D.

24. Now let's start v1 of your container:

    sudo docker run -ti pizak/mycentos:1 /bin/bash

    Try looking at the home directory:
    ls /home

    Now exit and load version 2
    sudo docker run -ti pizak/mycentos:2 /bin/bash
    ls /home

25. To prove that this is saved in the docker repo, do the following:

    First delete all the images locally that were tagged with your userid:
    (Replace pizak with your userid)

    ```
    sudo docker rmi -f $(sudo docker images -q pizak/*)
    ```

26. Now try to start v2 again. You will see that docker automatically re-downloads this and then runs it. Check that your file exists in the /home directory.

27. Stop the running image.

28. The one thing we haven't yet seen is how to get a docker image to do something vaguely useful.

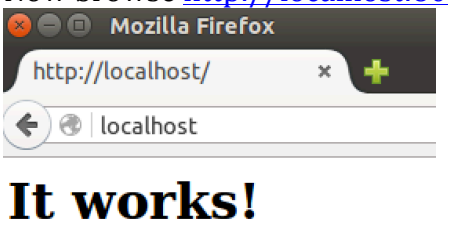
29. First check you have nothing running locally on port 80. Browse to http://localhost:80 It should fail.

30. Now in your docker window, type:
    ```
    sudo docker run –p 80:80 httpd
    ```

31. You should see a bunch of stuff like this:

```
Unable to find image 'httpd:latest' locally
latest: Pulling from httpd
ef2704e74ecc: Pull complete
1d6f63d023f5: Pull complete
…
781a5fd1cabf: Pull complete
bbd8adcb3ad5: Pull complete
6f953eead92f: Pull complete
afa235ca0577: Pull complete
f6d0a9cc3857: Pull complete
3fdd2b382f43: Pull complete
httpd:latest: The image you are pulling has been verified.
Important: image verification is a tech preview feature and should
not be relied on to provide security.
Digest:
sha256:fe40d6cb973ad7acbbc5fa99867efc03474649250a54da002fddaa88c6a
5ff2f
Status: Downloaded newer image for httpd:latest
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using 172.17.0.11. Set the 'ServerName'
directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using 172.17.0.11. Set the 'ServerName'
directive globally to suppress this message
[Fri Nov 20 14:08:08.239803 2015] [mpm_event:notice] [pid 1:tid
140576655767424] AH00489: Apache/2.4.17 (Unix) configured --
resuming normal operations
[Fri Nov 20 14:08:08.239940 2015] [core:notice] [pid 1:tid
140576655767424] AH00094: Command line: 'httpd –D FOREGROUND'
```

32. Now browse http://localhost:80 again and you should see.



33. *Are you wondering what –p 80:80 means?*
    It means expose port 80 from within the container as port 80 in the host system.

34. Now kill that container (Ctrl-C) and start it again in detached mode.
    This is how you would normally run a docker workload.
    ```
    sudo docker run –d –p 80:80 httpd
    ```

35. To find your docker runtime try
    ```
    sudo docker ps
    CONTAINER ID        IMAGE               COMMAND
    CREATED             STATUS              PORTS
    NAMES
    f9ed00d6c251        httpd:latest        "httpd-foreground"   5
    seconds ago       Up 4 seconds        0.0.0.0:80->80/tcp
    reverent_lalande
    ```

    and finally to stop it
    ```
    sudo docker kill reverent_lalande
    ```

Recap:
In this section we have learnt basic docker commands including run, ps, image, commit, push and pull. We have learnt about the layered file system, and also about the docker repository.
We have looked at exposing network ports, how to start detached workloads and how to kill them.

In particular, notice how the docker containers seem like processes, but with the complete configuration neatly packaged and contained within a single packaged system that can be versioned, pushed and pulled. This model is ideal for creating and managing *microservices.*

PART B – creating a Dockerfile

36. While I can imagine it might be possible to create docker images by modifying them like we have and then saving them, this is not a repeatable easy to use approach. Instead we want to build a dockerfile in a repeatable way.

37. I have already created a simple Dockerfile and an app to run in it.

38. From the terminal window
    mkdir ~/ex8
    cd ~/ex8
    git clone https://github.com/pzfreo/auto-deploy-node-js.git
    cd auto-deploy-node-js

39. The Dockerfile looks like this:

```
FROM alpine:3.3
MAINTAINER Paul Fremantle (paul@fremantle.org)

RUN apk --update add nodejs  && \
      ln -s /usr/bin/nodejs /usr/local/bin/node && \
      npm install express && \
      npm install -g forever && \
      mkdir -p /home/root/js

ADD simpletest.js /home/root/js
EXPOSE 8080
ENTRYPOINT forever /home/root/js/simpletest.js
```

The first line says that this Dockerfile builds on another, in this case alpine version 3.3. Alpine is a very small Linux distribution and this makes for a small docker image.

The RUN line does the following:

a. Updates the package manager and installs nodejs
b. Makes nodejs available under the name node (for old software)
c. Installs the node package express (Express.js framework)
d. Installs the node package forever globally (installs the forever command line utility)
e. Makes a directory for our code /home/root/js

Then the ADD line copies the file from the local directory into the image (simpletest.js)
EXPOSE says that this will offer port 8080
ENTRYPOINT defines the default command line to run to start our node.js app.
Here is the reference for all this:
https://docs.docker.com/engine/reference/builder/

40. Take a look at the node.js app (simpletest.js):

```
var express   =    require("express");
var app       =    express();

app.get("/",function(req,res){
    data = {a:'1',b:'2'}
    res.json(data);
});

app.listen(8080);
```

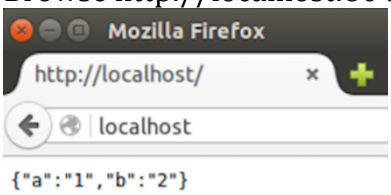
I think this is pretty self-explanatory. If you want to understand Express, take a look here:
http://expressjs.com/

41. Build it:

From the command line (in the correct directory) execute:
`sudo docker build –t <your_docker_id>/nodeapp:1 .`

42. Once it has built, try running it:
`sudo docker run –d –p 80:8080 <yrdockerid>/nodeapp:1`

43. Browse http://localhost:80 and you should see:



Hint: you may need to force a reload

44. Finally, push your newly created docker image into the cloud:
`sudo docker push <yrdockerid>/nodeapp:1`

45. Congratulations! You have completed this lab.

46.

47. **Extension**

If you have a github or bitbucket account, you can put the Dockerfile into the repository and automatically build it. Have a go.

Some rough hints:
Create a new public repository and place the Dockerfile in the root directory.
In http://hub.docker.com go to Settings (click on your username)
/Choose Linked Accounts and Services
Link to your Github account. Choose the "Public and Private"
Now click on Create (next to search) and Create Automated Build.
Select your github repository.
Enter a description. Click Create.
Now check the build status in the Build details tab. It takes about 3 minutes to build. If its not building you can manually trigger it from the build settings.
Try doing an update to your dockerfile (maybe a spare comment) and then git push.