Exercise 9

Using Docker

Prior Knowledge
Unix command-line
Apt-get package manager
Amazon AWS / EC2 Console

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
Running a docker container on EC2

Software Requirements
- AWS
- Docker
- Ubuntu
- Nano text editor

Unfortunately we need to fix something inside Ubuntu before we can do this lab.

In a terminal window:
```
sudo nano  /etc/NetworkManager/NetworkManager.conf
```

Then make sure the dnsmasq line is commented out with a #:

```
[main]
plugins=ifupdown,keyfile,ofono
#dns=dnsmasq

[ifupdown]
managed=false
```

Finally, restart the network manager:

```
sudo restart network−manager
```

Introduction

This lab consists of three 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.

Finally we will load your newly created docker image up in EC2.

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
   ```

   (You may need to enter the oxclo password which is **oxclo**)

3. You should see something like:

   ```
   latest: Pulling from centos
   47d44cb6f252: Pull complete
   168a69b62202: Pull complete
   812e9d9d677f: Pull complete
   4234bfdd88f8: Pull complete
   ce20c473cd8a: Pull complete
   centos: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:c96eeb93f2590858b9e1396e808d817fa0ba4076c68b59395445cb
   957b524408
   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/oxclo

   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. 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
    ```

13. 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
    ```

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

15. Repeat the sudo docker images command.

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

    Enter your docker hub credentials when prompted.

17. 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
    ```

18. 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
    ```

19. Now in your control terminal you can commit this change:
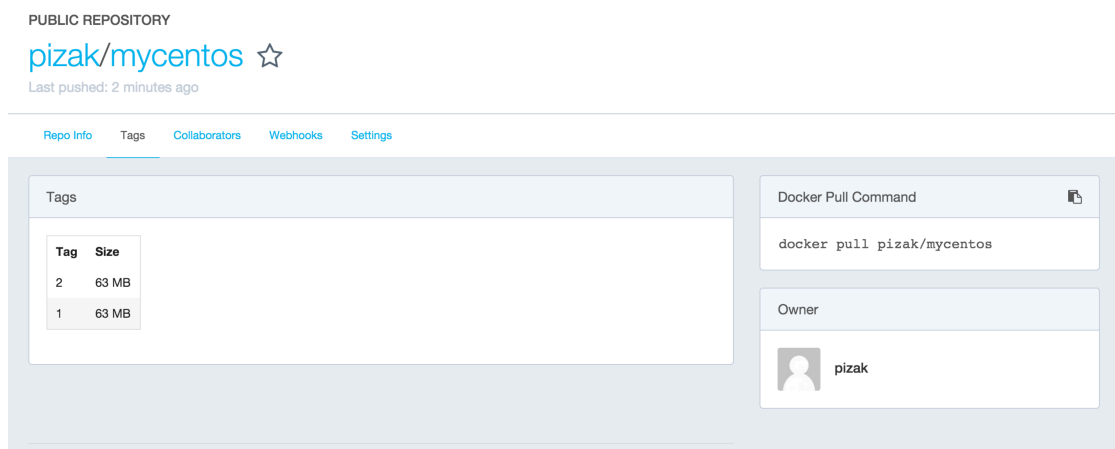
    ```
    sudo docker commit drunk_engelbart pizak/mycentos:2
    ```

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:

    PUBLIC REPOSITORY

    pizak/mycentos ☆
    Last pushed: 2 minutes ago

    Repo Info   Tags   Collaborators   Webhooks   Settings

Tags		Docker Pull Command	
Tag Size		docker pull pizak/mycentos	
2 63 MB		Owner	
1 63 MB		pizak	

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. Also note the special backwards single quote `)
    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. The one thing we haven't yet seen is how to get a docker image to do something vaguely useful.

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

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

30. 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'
    ```

31. Now browse http://localhost:80 again and you should see.
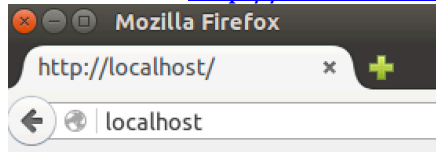
32. *Are you wondering what –p 80:80 means?*
It means expose port 80 from within the container as port 80 in the host system.

33. 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
```

34. 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

35. 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.

36. Make a new directory (e.g. ~/dockerbuild)
    Change to that directory and edit a file there called Dockerfile

37. Copy and paste the contents of this Dockerfile into it:
    http://freo.me/oxclo-dockerfile

38. Save it.

39. Build it:

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

40. While it is building, take a look at the docker file and also the reference guide:
    https://docs.docker.com/v1.8/reference/builder/

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

42. Browse http://localhost:80 and you should see:
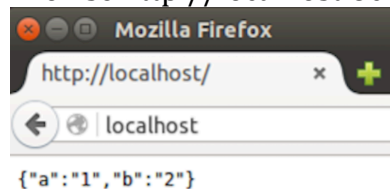
43. Finally, push your newly created docker image into the cloud:
    ```
    sudo docker push <yrdockerid>/nodeapp:1
    ```

    Congratulations, you have completed part B.

PART C – Running your docker in the cloud

44. While Amazon has a new Container Service for running docker instances in EC2, it basically runs a single container per instance, which could work out expensive. In our lab we are going to do the same thing of course, but in reality you could run many docker containers per EC2 instance – even on a micro instance you could run a number of node.js apps for example.

45. Go to the AWS console and start the wizard to launch a new instance.

46. Choose our favourite **Ubuntu 14.04 LTS** image

47. Choose a **t2.micro** instance type

48. Configure the instance details and in the advanced details section put the following into the user data changing it to point to your new docker image:

```
#!/bin/bash
apt-get update
apt-get upgrade -y
apt-get install docker.io -y
docker run -d -p 80:8080 <yrdockerid>/nodeapp:1
```

(A snippet is available here, http://freo.me/oxclo-docker-ud but remember to change the docker image name)

49. Continue, and tag your instance with **oxcloXX-docker**

50. Choose the Security Group **simple-web-server-sg**

51. Launch the instance

52. If you want to follow the startup of the instance you can SSH into it and then **tail –f /var/log/cloud-init-output.log**

53. Otherwise just wait a bit and then browse to the IP address or Public DNS of the instance and you should once again see my little code run.

54. **Now terminate your instance.**

Congratulations! You have completed this lab.

Recap: What we have done is to take the container-ised application we built on our local machine and then deploy it automatically to EC2. While this looks a bit similar to using userdata (which we did in a previous exercise), in fact this is much more replicable because we can test and deploy our docker image on many different systems.

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.