Exercise – Dockerfiles

Objective

The objective of this exercise is to create some docker containers from Dockerfiles and upload them to the hub.

Overview

Sign up to Docker Hub

We will be creating some docker images of our own and pushing them to the hub. To do this create an account with the Docker hub at **https://hub.docker.com/account/signup/**

Feel free to use any email address you like (as long as it's yours!)

Create a simple Docker container

To **create a new container** we need to create a **Dockerfile** to build from. The **Dockerfile** contains all the steps required to setup and create a new container. It is assumed that it will be in the current directory and it will be called **Dockerfile**, but you can call it anything you like and directly refer to it.

We are going to create a simple **Dockerfile** for a **Hello World** java program.

1. Create a new directory and go into it (it can be called anything)

```
$ mkdir example
```

2. Create a file in that directory called **hello.java** containing the following

```
public class hello {
  public static void main(String[] args) {
        System.out.println("Hello World!");
  }
}
```

3. Create another file in the same directory called **Dockerfile** containing

```
FROM centos:centos7
RUN yum update -y
RUN yum install -y java-1.8.0-openjdk
RUN yum install -y java-1.8.0-openjdk-devel
COPY hello.java /
RUN javac hello.java
ENTRYPOINT java hello
```

4. When you have created both files go back to the terminal

```
$ sudo docker build -t [yourusername]/centos-hello-java .
```

5. building your container, run the following inside it

```
$ sudo docker run [yourusername]/centos-hello-java java hello
```

The output should look like:

```
Hello World!
```

6. Push the container to the Docker repository entering your details. You will need to login first to setup the credentials file.

```
$ sudo docker login
$ sudo docker push [yourusername]/centos-hello-java
```

Creating a more complex Docker container

7. We are going to create a new docker file that will compile and run a hello-scalatra project. This project is hosted in a git repository and requires java and maven to compile and run.

   Clone the following repository:

```
$ cd ~
$ git clone https://gitlab.com/qatraining/qadockerint-exercises
$ cd qadockerint-exercises
```

   In the start/Ex02 directory there is the skeleton of a Dockerfile available. Your task is to fill in the gaps between the comments to setup a small scalatra project. You need to ensure that you have installed all the correct software, pulled the git repository into the container, built the project and then setup the external ports and entry point.

   HINTS:

   - Read the comments carefully, they tell you what to do at each step
   - Ubuntu uses apt-get to update and install packages
   - You can install maven from the apt repositories directly, same with git
   - The Java version you want is: openjdk-7-jdk
   - You will need to be in the correct directory with the pom.xml file after cloning your repository to run "mvn package"

SOLUTION:

```
FROM ubuntu:14.04
MAINTAINER Lina
RUN apt-get update && apt-get clean
RUN sudo apt-get install -y maven
RUN sudo apt-get install -y git
RUN apt-get install -y openjdk-7-jdk
RUN git clone https://gitlab.com/qatraining/qadockerint-hello-
scalatra hello-scalatra
WORKDIR /hello-scalatra
RUN mvn package
EXPOSE 8080


CMD mvn jetty:run
```

8. Build your Docker container with

```
$ docker build -t hub-name/any-name:any-number .
```

You can dip into your container by running bash within it and seeing what has happened as the docker build has happened.

```
$ docker run -ti hub-name/any-name:any-number bash
```

Eg.:

```
$ docker run -ti qa-hub/exercise10:2 bash
```

If you are having issues then try creating the container manually through bash and translating each command you use in the terminal into a line for the Dockerfile.

You should run your container with the following:

```
$ docker run -d -P hub-name/any-name:version mvn jetty:run
```

Work out what port number your docker container is using with `docker ps` and then point a browser at that location to see your hello message!

http://54.154.168.2:32770/

SOLUTION:

```
$ docker ps
CONTAINER ID    IMAGE            COMMAND          CREATED          STATUS          PORTS                     NAMES
2f664c7d7b6f  qa-hub/img1:1  "mvn jetty:run"  7 seconds ago  Up 6 seconds 0.0.0.0:32770->8080/tcp sharp_w
```

Using the tomcat container

9. In reality we probably don't want to be compiling our code in the container we are going to deploy to a server (especially since we have Jenkins to do that for us).

   The **hello-scalatra** project produces a war file that can be served by **Tomcat**. Search the **Docker** hub for a tomcat image. There should be one without a username attached - this is the official tomcat project.

   a. Clone a copy of your **hello-scalatra** project and go into the directory.

```
$ git clone https://<user>@bitbucket.org/<user>/hello-scalatra.git
```

   b. Build the project using maven (you will have to install maven to do this!)

```
$ mvn package
```

   c. Create a **Dockerfile** in the base directory and add the following:

```
FROM tomcat
MAINTAINER Lina


RUN apt-get update && apt-get clean
COPY  target/scalatra-maven-prototype.war
                  /usr/local/tomcat/webapps/hello-scalatra.war
EXPOSE 8080
CMD ["catalina.sh", "run"]
```

   Build your docker container and give it a name.

```
$ sudo docker build -t hub-name/hello-scalatra:version .
```

   Then, run this docker container with

```
$ sudo docker run -d -p 8888:8080 hub-name/ hello-scalatra
```

   Point your browser at

```
http://<Docker Public IP>:8888/hello-scalatra
```

   SOLUTION:

```
http://54.154.168.2:8888/hello-scalatra
```

   There is often little point in re-inventing the wheel. If an official (or a trusted) container exists we can use it!

Mounting Volume

To do this we will create a container with a volume on the host machine. Volumes would normally get set up in the Dockerfile with the "volume" command, but we can create on demand as well.

1. Create a new directory on your host machine

```
[on host]$ cd ~
[on host]$ sudo mkdir /dockerdata
```

2. Create a container which links to that directory with an interactive terminal

```
[on host]$ sudo docker run -it -v /dockerdata:/insidecontainer
ubuntu bash
```

3. Create a file in the linked directory inside the container

```
[insidecontainer]$ echo "hello outside world!" >
/insidecontainer/msg
```

4. Check it exists on the host

```
[insidecontainer]$ exit
[on host]$ cat /dockerdata/msg
```

You should see your message. Any data that is written into this directory can be accessed via the host or other containers. You can then arrange for the host to back this data up.