Docker Custom Images

Creating Docker Custom Images on the Raspberry Pi

In this tutorial we are expanding on the previous video where we looked at using Docker and the Docker engine on the Raspberry Pi. Here are still working with the basics of Docker at an overview level but we will gain a better understanding of how and why we use Docker by building Docker custom images. We will stick with the Raspberry Pi 2 has the Docker host but you may be using any host Docker system. As I am using ARM hardware I will use the armbuild/debian image as my base but you may just use Debian if you are using standard Intel hardware. At the end of this module you will be able to create Docker custom image from a Dockerfile.

Select Your Base Image

In Docker images are Read-Only templates that can be used to provision Containers. Containers and vaguely comparable to Virtual Machines in other technologies, but very vaguely . One major way that Docker Containers differ from traditional virtual machines is that they are designed to run one process only. This may be your web server or your database server etc. Containers have a thin read-write layer that overlays the underlying Image that it was provisioned from. In the scenario we want to deploy and Apache HTTPD server with PHP. We will use a Debian base image for this. Later we will add the required packages to the base image to create the Docker Custom image.

```
$ docker pull armbuild/debian:8.0 #I am using ARM hardware just debian:8.
0 for Intel
```

Create the Dockerfile

A Docker file is a text file that contains instructions on how to build the new image. It has to be called **Dockerfile** in the exact case. In a perfect world you will create this in its own empty directory as contents from the directory that the Dockerfile is located in can be added to the image you are

building.

For the purpose of this we will create a new test directory in or HOME directory:

```
$ mkdir $HOME/test ; cd $HOME/test
```

From with the new directory we can create the **$HOME/test/Dockerfile** with the editor of choice:

```
FROM armbuild/debian:8.0  RUN apt-get update && apt-get install php5 && a
pt-get clean  EXPOSE 80  CMD ["/usr/sbin/apache2ctl","D","FOREGROUND"]
```

FROM

This instruction defines the image that we base the image from

RUN

These are instructions to run inside the temporary container during image creation. We have combined the commands together to reduce the amount of layers that are created in the resulting image. Installing PHP5 on the debian image will install the Apache HTTPD package as a dependency.

EXPOSE

We tell the resulting container run from this image to listen on port 80 or to open port 80. We need this to talk to the wen server. We will later map port 80 on the host to port 80 on the Docker container to allow access from external hosts.

CMD

This defines the command that will be PID 1 or Process ID 1 when the container start. We can use CMD or ENTRYPOINT but CMD allows us to overwrite the command from the command line whereas ENTRYPOINT does not. This is useful sometime in faulting a container in that we can stat it with a bash shell when ENTRYPOINT is not defined.

Create the Docker Custom Image

Now we have the Dockerfile we can build the new image. From the test directory:

```
$ cd $HOME/test  $ docker build -t debian/web .
```

We use the **-t** option to set the tag or name of the image. As it is based on **debian** I use that and then **/web** as it is a web server image. These names are fine so long as you do not intend to upload them to Docker hub where they will need to named after your userid. The dot or period at the end denotes that we look for the Dockerfile in the current directory. When we run the command it may take a few minutes installing the software

```
$ docker images
```

Using the above command we should see the new image once it is created.

Building Containers

We can run a test container to see that it works

```
$ docker run -d -p 80:80 --name test debian/web
```

We should now be able to browse to the Docker host on port 80 and see the standard Debian welcome page. To add our own content we need to ensure the website is available in a directory, such as **$HOME/www**. We will first stop the test container and then start a new one with the **$HOME/www** directory on the host mapped to **/var/www/html/** on the container. Remember we must create the website in on the Dockerhost and mount it to the container at runtime

```
$ docker stop test  $ docker rm test  $ docker run -d -p 80:80 -v /home/p
i/www:/var/www/html --name test debian/web
```

Now when browsing to the site we should see the content of the website we created in your brand new container.

The video follows please take a look....