RX-M

Docker Images

Lab – Dockerfiles I

Docker can build images automatically by reading the instructions in a Dockerfile. A Dockerfile is a text file that contains all of the commands you would normally execute manually in order to build a Docker image from a container with “docker commit”. By calling “docker build” from your terminal, you can have Docker build the image for you, sequentially executing the instructions in your Dockerfile.

In this lab you will create a Dockerfile to specify a simple service image which we will use to host a web server. We will name the repository “websvr”.

1. Setup a Dockerfile context directory

Dockerfiles are usually placed at the root of a folder hierarchy containing everything you need to build one or more Docker images. This folder is called the build context. The entire build context will be packaged and sent to the Docker Engine for building. In simple cases the build context will have nothing in it but a Dockerfile.

To create a build context (directory) on your lab system, simply create a new directory. We will create a directory called websvr to host our Docker websvr project.

```
user@ubuntu:~$ mkdir websvr

user@ubuntu:~$ cd websvr
user@ubuntu:~/websvr$
```

2. Create a Dockerfile

A Dockerfile contains a list of instructions for the Docker build command to perform when creating the target image. We will build the following Dockerfile:

```
FROM ubuntu:12.04
MAINTAINER docker lab
RUN apt-get update && \
    apt-get install -y apache2 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

Dockerfile statements can be loosely organized into three groups:

- **File System Setup –** Steps that copy or install files into the image’s file system (programs, libraries, etc)
- **Image Metadata –** Descriptive image data outside of the filesystem (exposed ports, labels, etc.)
- **Execution Metadata –** Commands and environment settings that tell Docker what to do when users run the image

Our Dockerfile has three statements in the **Image Metadata** category, FROM, MAINTAINER and EXPOSE. Dockerfiles begin with a FROM statement which defines the parent image for the new image. We will base our web server on the “Ubuntu 12.04” image. You can create images with no parent by using the reserved name “scratch” as the parent image (e.g. FROM scratch). Dockerfiles also usually contain a MAINTAINER statement defining the author and/or collaborators. The EXPOSE instruction describes ports that the container’s service(s) typically listen on.

The **File System Setup** section of our Dockerfile will need to run apt-get commands to update the system and install apache2.

The **Execution Metadata** section of our Dockerfile will need to setup environment variables for the user and group for the Apache Web Server to run under. This section will also need to define the port to run the Apache web server.

Create a Dockerfile for your Apache Web Server in the websvr directory as per below:

```
user@ubuntu:~/websvr$ vi Dockerfile

user@ubuntu:~/websvr$ cat Dockerfile
FROM ubuntu:12.04
MAINTAINER docker lab

RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]

user@ubuntu:~/websvr$
```

The apache2 binary runs the Apache Web Server (in Debian packaging), the -D switch passes parameters and the FOREGROUND parameter tells the web server not to call setsid(), keeping it from creating a new shell session (important if you want the console output to appear in the Docker logs.)

Use the Dockerfile reference to look up any commands you are not familiar with from class:
https://docs.docker.com/reference/builder

3. Build the image

The `docker build` subcommand reads Dockerfiles and creates images from them. The build command provides a --tag switch which you can use to give your new image a repository and tag name.

Build your websvr image as follows:

```
user@ubuntu:~/websvr$ docker build --tag="lab/websvr:0.2.0" .
...

user@ubuntu:~/websvr$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRT SIZE
lab/websvr              0.2.0               f8afca4ea32d        6 minutes ago       184.3 MB
localhost:5000/devenv   latest              a0deaea7302f        35 minutes ago      226.1 MB
...
```

Note the size of your new image. Not bad for an Apache web server with a complete Linux distro and configuration. Use the `docker history` subcommand to display the various image layers in your application’s image ancestry.

```
user@ubuntu:~/websvr$ docker history lab/websvr:0.2.0
IMAGE          CREATED          CREATED BY                                      SIZE
f8afca4ea32d   7 minutes ago    /bin/sh -c #(nop) CMD ["/usr/sbin/apache2" "-   0 B
7c3b509141a6   7 minutes ago    /bin/sh -c #(nop) EXPOSE 80/tcp                 0 B
97aeeedfa019   7 minutes ago    /bin/sh -c #(nop) ENV APACHE_LOG_DIR=/var/log   0 B
a3d570440175   7 minutes ago    /bin/sh -c #(nop) ENV APACHE_RUN_GROUP=www-da   0 B
a2f434aeaded   7 minutes ago    /bin/sh -c #(nop) ENV APACHE_RUN_USER=www-dat   0 B
6d964e563d0c   7 minutes ago    /bin/sh -c apt-get update &&     apt-get inst   46.73 MB
fa3703dbcfb0   9 minutes ago    /bin/sh -c #(nop) MAINTAINER docker lab         0 B
8b1548cecef1   3 weeks ago      /bin/sh -c #(nop) CMD ["/bin/bash"]             0 B
<missing>      3 weeks ago      /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/   1.911 kB
<missing>      3 weeks ago      /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic   156.2 kB
<missing>      3 weeks ago      /bin/sh -c #(nop) ADD file:037585e370463e3c37   137.4 MB
```

Older versions of Docker displayed IDs for all images. Content addressable images no longer use parent images to chain image layers together, rather the image manifest knows all of the metadata and all of the required filesystem layers. Docker v1.10 and later will show “” for image layers without manifests of their own. This was perhaps a poor wording choice, the image data is not missing, just the manifest and its ID.

Now run the `docker images` command to display the images on the Docker host.

- Which images in your application’s ancestry are tagged?
- How can you display the untagged images with the `docker images` command?
- What does the (nop) comment in the build history mean?
- Do all of the images in the application’s ancestry have filesystem layers?
- How can you tell which images have filesystem layers from the history display?
- Which environment variables are defined in the image ancestry?

Use the `docker inspect` subcommand to display the environment variables your image will configure for new containers:

```
user@ubuntu:~/websvr$ docker inspect -f '{{.ContainerConfig.Env}}' lab/websvr:0.2.0
[
    PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    APACHE_RUN_USER=www-data
    APACHE_RUN_GROUP=www-data
    APACHE_LOG_DIR=/var/log/apache2
]
```

- Can you find an image in the build history for each of these definitions?
- Who set the PATH?

The PATH is the one environment variable we did not configure. When a new container is created, Docker will set the following environment variables automatically:

- **PATH** - Includes popular directories (/usr/local/sbin:/usr/local/bin:/usr/sbin, …)
- **HOME** - Set based on the value of USER
- **HOSTNAME** - The hostname associated with the container
- **TERM** - xterm if the container is allocated a pseudo-TTY

The PATH variable is placed in the image’s metadata, however the HOST, HOSTNAME and TERM variables change with each run of the image and are configured by Docker dynamically.

4. Run a Container using the new Image

Like any other image, a Dockerfile generated image can be run with the `docker run` subcommand and we can use the -p switch to map ports from the container to the host interface. Run your image, mapping the container’s port 80 (where

Apache listens by default) to the host's port 8989:

```
user@ubuntu:~/websvr$ docker run -d -p 8989:80 lab/websvr:0.2.0
501ba663cf9205930642db1f508cc4e6639137d5e07bf08ffa1b6560e08418cb
```

Verify that your web server container is running:

```
user@ubuntu:~/websvr$ docker ps
CONTAINER ID  IMAGE            COMMAND                 CREATED  STATUS  PORTS
501ba663cf92  lab/websvr:0.2.0 "/usr/sbin/apache2 -D"  6s ago   Up 5s   0.0.0.0:8989->80/tcp
aceb3baab7e6  registry         "docker-registry"       1h ago   Up 1h   0.0.0.0:5000->5000/tcp
```

Now try to reach the web server on port 80 from the host:

```
user@ubuntu:~/websvr$ wget -O - http://localhost:80
--2015-08-15 14:33:34--  http://localhost/
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... failed: Connection refused.
Connecting to localhost (localhost)|127.0.0.1|:80... failed: Connection refused.

user@ubuntu:~/websvr$
```

This fails because Apache is running on port 80 in the container. The container's ports are not directly accessible from the host. Use the `docker port` subcommand to display the container port mappings for your web server (you will need to substitute the ID of your own webserver container):

```
user@ubuntu:~/websvr$ docker port 501ba663cf92
80/tcp -> 0.0.0.0:8989
```

Now try to reach the web server from the host port 8989:

```
user@ubuntu:~/websvr$ wget -qO- http://localhost:8989
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Success! You now have a running service based on an image you created from a Dockerfile.

5. Rebuild a Dockerfile/Context

The Docker build cache avoids the long waits associated with building the same image successively. If the Docker Engine sees a Dockerfile instruction with a string and parent image ID that it has already built, it will use the preexisting image and skip the build step. To demonstrate we will add a LABEL instruction to our existing Dockerfile and rebuild it.

LABELs allow you to add arbitrary metadata to a Docker image. Make the following change to your Dockerfile to add the "release-notes" label:

```
user@ubuntu:~/websvr$ vi Dockerfile

user@ubuntu:~/websvr$ cat Dockerfile
FROM ubuntu:12.04
```

```
MAINTAINER docker lab
RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
LABEL release_notes="this is a test release"
EXPOSE 80
CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]

user@ubuntu:~/websvr$
```

Be sure to add only the LABEL instruction without tampering with any of the other lines in the file. Now rebuild your image stack with the new 0.3.0 tag:

```
docker build --tag="lab/websvr:0.3.0" .
```

```
user@ubuntu:~/websvr$ docker build --tag="lab/websvr:0.3.0" .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM ubuntu:12.04
 ---> d0e008c6cf02
Step 2 : MAINTAINER docker lab
 ---> Using cache
 ---> e492d8a1894f
Step 3 : RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf
/var/lib/apt/lists/*
 ---> Using cache
 ---> a96e4384e405
Step 4 : ENV APACHE_RUN_USER www-data
 ---> Using cache
 ---> cead72b5611c
Step 5 : ENV APACHE_RUN_GROUP www-data
 ---> Using cache
 ---> ae5988bf05bc
Step 6 : ENV APACHE_LOG_DIR /var/log/apache2
 ---> Using cache
 ---> d791b3e86ee4
Step 7 : LABEL release_notes "this is a test release"
 ---> Running in 96925e50327c
 ---> 74f33d05fb7a
Removing intermediate container 96925e50327c
Step 8 : EXPOSE 80
 ---> Running in c09ead176000
 ---> 2f9968fa4ef6
Removing intermediate container c09ead176000
Step 9 : CMD /usr/sbin/apache2 -D FOREGROUND
 ---> Running in ab02f0365703
 ---> 52a85053cef2
Removing intermediate container ab02f0365703
Successfully built 52a85053cef2
```

- Why did the first five steps use the cache?
- Why did the unchanged EXPOSE and CMD lines rebuild?

Use the inspect command to view your new LABEL metadata:

```
$ docker inspect -f '{{.ContainerConfig.Labels.release_notes}}' lab/websvr:0.3.0
this is a test release
```

Examine the new image history:

```
user@ubuntu:~/websvr$ docker history lab/websvr:0.3.0
IMAGE          CREATED      CREATED BY                                      SIZE
52a85053cef2   5 mins ago   /bin/sh -c #(nop) CMD ["/usr/sbin/apache2" "-     0 B
2f9968fa4ef6   5 mins ago   /bin/sh -c #(nop) EXPOSE 80/tcp                   0 B
74f33d05fb7a   5 mins ago   /bin/sh -c #(nop) LABEL release_notes=this is     0 B
d791b3e86ee4   1 hour ago   /bin/sh -c #(nop) ENV APACHE_LOG_DIR=/var/log     0 B
ae5988bf05bc   1 hour ago   /bin/sh -c #(nop) ENV APACHE_RUN_GROUP=www-da     0 B
cead72b5611c   1 hour ago   /bin/sh -c #(nop) ENV APACHE_RUN_USER=www-dat     0 B
a96e4384e405   1 hour ago   /bin/sh -c apt-get update && apt-get install  46.73 MB
e492d8a1894f   1 hour ago   /bin/sh -c #(nop) MAINTAINER docker lab           0 B
d0e008c6cf02   11 days ago  /bin/sh -c #(nop) CMD ["/bin/bash"]               0 B
3c8e79a3b1eb   11 days ago  /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/ 1.911 kB
bc99d1f906ec   11 days ago  /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic 156.2 kB
a69483e55b68   11 days ago  /bin/sh -c #(nop) ADD file:9a07e8c7e9e6c71e5c 134.6 MB
```

Compare the IDs from this build to the prior build.

- Which images are the same between build 0.2.0 and build 0.3.0?
- How can you infer from the build history which images were built together?
- Run the “docker images -a” command, what order does it list images in?

Congratulations, you have completed the Dockerfile lab!