Lab 5 - Building a DockerFile

Introduction

Dockerfile defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to “copy in” to that environment.

In this lab, you will learn how to create a **Dockerfile** and you can expect that the build of your app defined in this **Dockerfile** behaves exactly the same wherever it runs.

1. Building Image from a Dockerfile

Dockerfile is used for automation of work by specifying all step that we want on docker image. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using Docker build users can create an automated build that executes several command-line instructions in succession.

1.1 Login as **“root”** user on **aio110** host:

Copy

```
ssh root@aio110
```

1.2 Create a directory and a Dockerfile.

Copy

```
mkdir example

cd example
```

1.3 Each instruction creates a new layer of the image.

Copy

```
cat > Dockerfile <<EOF

# This is a comment
```

```
FROM ubuntu

RUN apt-get update && apt-get install -y ruby ruby-dev

EOF
```

Note: The first instruction is, **FROM** tells the source of our image.
RUN instruction executes a command inside the image.

1.4 Now let's take Dockerfile and use the docker build command to build an image.

Copy

```
 docker build -t example .
```

Output:

```
Sending build context to Docker daemon 2.048 kB

Step 1 : FROM ubuntu

 ---> bd3d4369aebc

Step 2 : RUN apt-get update && apt-get install -y ruby ruby-dev

 ---> Using cache

 ---> 9f65bddd0425

Successfully built 9f65bddd0425
```

Note: Location of Dockerfile using the . to indicate a Dockerfile in the current directory.

1.5 Add a tag to an existing image after commit or build it. We can do this using the docker tag command. Now, add a new tag to example image.

Syntax :

```
docker tag docker-image-id repo-name:tag
```

Copy

```
docker tag example example:ver1
```

1.6 Check the image we have built on host.

Copy

```
docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE

example             latest              f429d692e383        About a
minute ago   192MB

example             ver1                f429d692e383        About a
minute ago   192MB

ubuntu              latest              0458a4468cbc        4 weeks
ago          112MB
```

2. Some more activity on Dockerfile

2.1 Create a dockerfile.

Copy

```
cat > test <<EOF

############################################################

# Dockerfile to build MongoDB container images

# Based on Ubuntu

############################################################

# Use the Trusty (14.04) Ubuntu base image
```

```
FROM ubuntu:14.04

# File Author / Maintainer

MAINTAINER Example Test

# Following installation instructions from
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/

################## BEGIN INSTALLATION ######################

# Import the MongoDB public GPG key

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
7F0CEB10

# Create a list file for MongoDB

RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart
dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb.list

# Update the local package database

RUN apt-get update

# Install the latest stable version of MongoDB (note -y flag for non-
interactive)

RUN apt-get install -y mongodb-org

# Create MongoDB's default data directory (-p creates parent
directories - in this case, /data)

RUN mkdir -p /data/db

# Because this database will run in a Docker container,

# we must configure it to accept connctions from foreign hosts

RUN echo "bind_ip = 0.0.0.0" >> /etc/mongodb.conf

# Expose the MongoDB port
```

```
EXPOSE 27017

# Set the default command for this image

CMD ["mongod"]

EOF
```

2.2 Take Dockerfile and use the docker build command to build an image.

Copy

```
docker build -f /root/example/test -t mongo_db .
```

2.3 Check the image that has built.

Copy

```
docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE

mongo_db            latest              d9130e44201d        About a
minute ago   378MB

example             latest              f429d692e383        4 minutes
ago        192MB

example             ver1                f429d692e383        4 minutes
ago        192MB

ubuntu              latest              0458a4468cbc        4 weeks
ago          112MB
```

2.4 Come out of the directory “example”.

Copy

```
cd
```

3. Launching a newly built Container

3.1 In this step, verify the new images and then run new image. Open a command line terminal type docker images. This command lists the images we have locally.

Copy

```
docker images
```

3.2 Specify tag_name it will automatically run image with 'latest' tag. Instead of image_name we can also specify Image Id (no tag_name).

Run newly built image,

Syntax :

```
docker run image-name:tag
```

Copy

```
docker run --name custom_container -dit mongo_db:latest
```

Output:

```
238a403ff22a28765316b4eb36441b4f882d5bdc24cd597c4c673cfd80e6f272
```

Note: You may also notice that Docker didn't have to download anything. That is because the image was built locally and is already available.

4. Deploy Static HTML Website as Container using Docker file

Docker Images start from a base image. The base image should include the platform dependencies required by your application, for example, having the JVM or CLR installed. This base image is defined as an instruction in the **Dockerfile**. Docker Images are built based on the contents of a Dockerfile. The Dockerfile is a list of instructions describing how to deploy your application.

We have base image is the **Alpine version of Nginx**. This provides the configured web server on the Linux Alpine distribution.

4.1 Create a directory for a simple project called **static-compose**:

Copy

```
mkdir static-compose && cd static-compose
```

4.2 Create your **Dockerfile** for building your image by copying the contents below into the editor.

Copy

```
cat > Dockerfile <<EOF

FROM nginx:alpine

COPY . /usr/share/nginx/html

EOF
```

The first line defines our base image.
The second line copies the content of the current directory into a particular location inside the container.

4.3 Build our static HTML image using the build command below.

Copy

```
docker build -t webserver-image:v1 .
```

Output:

```
Sending build context to Docker daemon  2.048kB

Step 1/2 : FROM nginx:alpine

alpine: Pulling from library/nginx
```

```
ff3a5c916c92: Pull complete

b4f3ef22ce5b: Pull complete

8a6541d11dc3: Pull complete

7e869e2dcf68: Pull complete

Digest:
sha256:48947591194ac5a9dce1e110f9198a547debb21630f121081640b87d99ca8b1
1

Status: Downloaded newer image for nginx:alpine

 ---> 537527661905

Step 2/2 : COPY . /usr/share/nginx/html

 ---> 3b9738dd84dd

Removing intermediate container 751e0c338619

Successfully built 3b9738dd84dd

Successfully tagged webserver-image:v1
```

4.4 View a list of all the images on the host :

Copy

```
docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE

webserver-image     v1                  6352c782823c        15 seconds
ago       17.9MB
```

```
mongo_db            latest              6968ce727ff9        About a
minute ago   378MB

example             latest              b932a04f5d28        4 minutes
ago        193MB

example             ver1                b932a04f5d28        4 minutes
ago        193MB

ubuntu              latest              f975c5035748        6 days ago
112MB

nginx               alpine              537527661905        2 weeks
ago          17.9MB
```

The built image will have the name **webserver-image** with a tag of **v1**.

With the built Image it can be launched in a consistent way to other Docker Images. When a container launches, it’s sandboxed from other processes and networks on the host. When starting a container you need to give it permission and access to what it requires.

For example, to open and bind to a network port on the host you need to provide the parameter -p **<host-port>:<container-port>**.

4.5 Launch our newly built image providing the friendly name and tag. As it’s a web server, bind port **80** to our host using the **-p** parameter.

Copy

```
docker run -d -p 80:80 webserver-image:v1
```

Sample output:

```
a970ce156681df2ffa5d8942379db70857199ac47ed2862006f80f0103387469
```

4.6 Verify the containers is created

Copy

```
docker ps -a
```

Output:

```
CONTAINER ID        IMAGE                COMMAND
CREATED              STATUS              PORTS                 NAMES

1ac9d381d550        webserver-image:v1   "nginx -g 'daemon ..."   4
seconds ago        Up 3 seconds        0.0.0.0:80->80/tcp
boring_pasteur

0f24ba66e0c0        mongo_db:latest      "/bin/sh -c usr/bi..."
About a minute ago   Up About a minute   27017/tcp
custom_container
```

4.7 Once started, you'll be able to access the results of port 80 via

Copy

```
curl http://localhost:80
```

Output:

```
<h1>Hello World</h1>

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!

<style>

    body {

        width: 35em;

        margin: 0 auto;

        font-family: Tahoma, Verdana, Arial, sans-serif;
    }

</style>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully
installed and

working. Further configuration is required.</p>



<p>For online documentation and support please refer to

<a href="http://nginx.org/">nginx.org</a>.<br/>

Commercial support is available at

<a href="http://nginx.com/">nginx.com</a>.</p>



<p><em>Thank you for using nginx.</em></p>

</body>

</html>
```

Copy

```
cd
```

5. Cleanup

5.1 To remove all the containers run the below commands.

Copy

```
docker rm `docker ps -a -q` -f
```

Output:

```
88d4efe71fe0

9f9e86f91b0f
```

5.2 To remove all the images run the below commands.

Copy

```
docker rmi `docker images -q` -f
```

Output:

```
,.....

......

Deleted:
sha256:1239c33230909cc231da97b851df65e252dc9811dcee2af0ecf3b225e2805a3
1

Deleted:
sha256:ce4caf69568d9109febd1f5307b62d85ab84e7a947fded041be49c847b412e5
a

Deleted:
sha256:4c711cc0452303f0fb6ce885c84130e32bb649b03f690fd0e4626a874b1cc8c
f

Deleted:
sha256:a375921af0e34b1cb09a35af24265db01b1eb65edabadaf70d56505a60a6de2
b

Deleted:
sha256:c41b9462ea4bbc2f578e42bd915214d54948d960b9b8c6815daf8586811c2d3
8
```

```
Error response from daemon: No such image: 0686ceaf9eea:latest
```

5.3 Verify that containers are removed:

Copy

```
docker ps
```

Output:

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

5.4 Verify that docker images are removed:

Copy

```
docker images
```

Output:

```
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```