Kelly Technologies

Docker - Playing with Dockerfile

Docker images are created either by pulling them online or by creating on our local machine.

Containers can be created by using the docker run command. Once the container is started, we have to manually login to the container and perform the actions that we want the container to work with. Docker provides another way to create images by using the Dockerfile.

A *Dockerfile* is a text file with instructions written in a format understood by the Docker. To create an image we need to write our own docker instructions in the Dockerfile. There are only a handful of docker instructions available that goes into the Dockerfile. In this article we will see how we can create a basic container image using the Docker instructions in the DockerFile. We will cover most of the Instructions available in this example

Here is the example Dockerfile that we are going to use in this demo

```
[puppet@root$:/test/docker]$  cat Dockerfile
FROM ubuntu
MAINTAINER jagadish "jagadish.manchala@gmail.com"
RUN useradd -p redhat123 demoDock
USER demoDock
RUN id
USER root
WORKDIR /tmp
COPY test.txt /tmp
ADD http://unec.edu.az/application/uploads/2014/12/pdf-sample.pdf  /tmp
VOLUME /work
ONBUILD RUN mkdir /tmp/tempo
RUN apt-get update
RUN apt-get install -y nginx
ENV CONTAINER_NAME DemoContainer
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
EXPOSE 80
```

1. From sets the Base Image for subsequent instructions. In the above example we are building a image that is based on ubuntu image. Once we try to build the image , the ubuntu image is downloaded first or if available on the local repository it will use that.

2. MAINTAINER sets the AUthor field of the generated image

3. RUN executes any commands given in a new layer on the top of the current image and commits the result

4. ENV sets environment variables

5. USER sets the user name for following RUN / CMD and ENTRYPOINT commands

6. WORKDIR sets the working directory. This gets reflected when we login to the Container

7. COPY copies new files or directories to the container. The files are copied to a location specified in the instruction

8. ADD copies new files , directories. This works much like COPY instruction but with additional features. This can ADD files from a remote location. So if we want to download a package from online when a container is started, we can use the ADD instruction

9. CMD sets default commands which can be overwritten from command line when docker container runs.

10. VOLUME creates a mount point for externally mounted volumes or other containers.

11. LABEL apply key value metadata to the image or container

12. ONBUILD adds a trigger instruction when the above created image is used as a base image for building another image

13. STOPSIGNAL sets the system call signal that will be sent to the container to exit

14. EXPOSE informs docker that the container listens on the specified network ports at runtime. This actually make the ports accessible from the host machine

When we build the image using the Dockerfile,

```
[puppet@root$:/test/docker]$  docker build -t sample-1 .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM ubuntu
 ---> f49eec89601e
Step 2 : MAINTAINER jagadish "jagadish.manchala@gmail.com"
 ---> Using cache
 ---> 94fd532ca66b
Step 3 : RUN useradd -p redhat123 demoDock
 ---> Using cache
 ---> e357022cfa3d
Step 4 : USER demoDock
 ---> Using cache
 ---> 044aff8c76a6
Step 5 : RUN id
 ---> Using cache
 ---> e6ea35026fb1
Step 6 : USER root
```

```
 ---> Using cache
 ---> 6362a86140fe
Step 7 : WORKDIR /tmp
 ---> Using cache
 ---> eec500226e04
Step 8 : COPY test.txt /tmp
 ---> Using cache
 ---> 597eb11573fa
Step 9 : ADD http://unec.edu.az/application/uploads/2014/12/pdf-sample.pdf /tmp
Downloading [==================================================>]
7.945 kB/7.945 kB
 ---> 1e4e9c10e01b
Removing intermediate container 30e8c9b3d9ce
Step 10 : VOLUME /work
 ---> Running in d76ab01c6951
 ---> c94b506c1080
Removing intermediate container d76ab01c6951
Step 11 : ONBUILD run mkdir /tmp/tempo
 ---> Running in 4bfc6be733fb
 ---> 6bf428e22280
Removing intermediate container 4bfc6be733fb
Step 12 : RUN apt-get update
 ---> Running in 71e68b270971
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial/main Sources [1103 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial/restricted Sources [5179 B]
Get:6 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:9 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial-updates/main Sources [291 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial-updates/restricted Sources [2815 B]
Get:12 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [162 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [604 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [12.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [502 kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-security/main Sources [72.1 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-security/restricted Sources [2392 B]
Get:18 http://archive.ubuntu.com/ubuntu xenial-security/universe Sources [22.4 kB]
Get:19 http://archive.ubuntu.com/ubuntu xenial-security/main amd64 Packages [268 kB]
Get:20 http://archive.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.0 kB]
Get:21 http://archive.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [91.7 kB]
Fetched 24.8 MB in 1min 14s (331 kB/s)
Reading package lists...
 ---> 0a40de0d78a4
Removing intermediate container 71e68b270971
Step 13 : RUN apt-get install -y nginx
 ---> Running in 8b6c131cab89
```

```
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
********************
Removing intermediate container 8b6c131cab89
Step 14 : ENV CONTAINER_NAME DemoContainer
 ---> Running in d1c3f7769b6f
 ---> ab78532293fa
Removing intermediate container d1c3f7769b6f
Step 15 : RUN service nginx restart
 ---> Running in 02170bfea4cb
 * Restarting nginx nginx
   ...done.
 ---> 2a6838821ad5
Removing intermediate container 02170bfea4cb
Step 16 : EXPOSE 80
 ---> Running in 566d56d32142
 ---> 8c35bcbe88be
Removing intermediate container 566d56d32142
Successfully built 8c35bcbe88be
```

The build is success and if we check the "docker images" command we can see the "sample-1" image in our local repository.

Now lets run the container using

```
[puppet@root$:/test/docker]$  docker run -d -p 8011:80 --name my2 sample-1
bf854781d85ddf6f5141435aa0dc9f05f7d7c7f2cbd7925806cbb89e3a7ecc45
```

Now we can test the nginx from our local machine using localhost:8011 URL

More article to comes , Happy learning