Dockerfile Instructions

What is a Dockerfile?

It is a text file that carries out all the instructions to build a custom image. Or is a text document that contains all the commands a user could call on the command line to assemble a custom image.

Dockerfile Basics

Before we construct our Dockerfile, you need to understand what makes up the file. This will be a text file, named `Dockerfile`, that includes specific keywords that dictate how to build a specific image. The specific keywords you can use in a file are:

- FROM
- ADD
- CMD
- ENTRYPOINT
- ENV
- EXPOSE
- MAINTAINER
- RUN
- USER
- VOLUME
- WORKDIR
- LABEL
- ARG

Ways to Create a Custom Image

They are 2 ways to create custom images

- From a running container (snapshot of a running container at a particular time)
- From Dockerfile (standard way used by companies out there)

Instruction in Dockerfile

1. FROM

The Dockerfile `FROM` command specifies the base image of your Docker images.

A base image is an image that is used to create all of your container images. Your base image can be an official Docker image, such as Centos, or you can modify an official Docker image to suit your needs, or you can create your own base image from scratch.

If you want to start with a bare Linux image, you can use this FROM command:

```
FROM ubuntu:latest
```

1. COPY

The Dockerfile `COPY` command copies one or more files from the Docker host (the computer building the Docker image from the Dockerfile) into the Docker image. The COPY command can copy both a file or a directory from the Docker host to the Docker image. Here are some Dockerfile COPY examples:

```
COPY /myapp/target/myapp.jar /myapp/myapp.jar
```

This example copies a single file call `myapp.jar` from the Docker host at `/myapp/target/myapp.jar` to the Docker image at `/myapp /myapp.jar.` The first argument is the Docker host path (where to copy from) and the second argument is the Docker image path (where to copy to).

```
COPY /myapp/target/* /myapp/
```

This example copies all single files and directories from the Docker host at `/myapp/target/` to the Docker image at `/myapp/`. The first argument is the Docker host path (where to copy from) and the second argument is the Docker image path (where to copy to).

1. ADD

The Dockerfile `ADD` instruction works in the same way as the `COPY` instruction with a few minor differences:

- The `ADD` instruction can copy and extract TAR files from the Docker host to the Docker image.
- The `ADD` instruction can download files via HTTP and copy them into the Docker image.

Here are a few Dockerfile ADD examples:

```
ADD index.html /var/www/html/
ADD /covid19/* /var/www/html/
ADD https://warfiles-for-docker.s3.amazonaws.com/addressbook.war /usr
/local/tomcat/webapps/
```

1. RUN

The Dockerfile `RUN` instruction is executed during the build time of the Docker image, so `RUN` commands are only **executed once**. The RUN command can be used to extract files, download files, install packages, to run all shell command,s or other command-line activities which are necessary to run while building the docker image.

```
RUN apt-get install some-needed-app
RUN yum -y install git
RUN mkdir -p /tmp/file
RUN apt -y update
RUN apt -y install wget
RUN apt -y install unzip

OR

RUN apt -y update && \
    apt -y install wget && \
    apt -y install unzip
RUN cd /usr/local/apache2/htdocs
RUN rm -rf *
RUN wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/covid19.zip
RUN unzip covid19.zip
RUN cp -R covid19/* .
RUN rm -rf covid19.zip
RUN rm -rf covid19

OR

RUN cd /usr/local/apache2/htdocs && \
    rm -rf * && \
    wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/covid19.zip && \
    unzip covid19.zip && \
    cp -R covid19/* .
    rm -rf covid19.zip && \
    rm -rf covid19 && \
```

1. WORKDIR

The `WORKDIR` instruction specifies what the working directory should be inside the Docker image. The working directory will be in effect for all commands following the WORKDIR instruction. Here is an example:

```
WORKDIR /tmp
WORKDIR /var/www/html/
WORKDIR /usr/local/apache2/htdocs
```

1. EXPOSE

The Dockerfile `EXPOSE` instruction opens up network ports in the Docker container to the world. For instance, if your Docker container runs a web server, that web server will probably need port `80` or `8080` open for any client to be able to connect to it. Here is an example of opening a network port using the EXPOSE command:

```
EXPOSE 8080
EXPOSE 80
```

1. VOLUME

The Dockerfile `VOLUME` instruction creates a directory inside the Docker image which you can later mount a volume (directory) to from the Docker host. In other words, you can create a directory inside the docker image, called `/data` which can later be mounted to a directory, called `/container-data/container1` in the Docker host. The mounting is done when the container is started up and it is used to share data between the container and the host. Here is an example of defining a volume (mountable directory) in a Dockerfile using the VOLUME instruction:

```
VOLUME /data
```

8- LABEL and MAINTAINER

The Dockerfile `MAINTAINER` instruction has been deprecated dockerfile and it has been replaced by `LABEL` instruction instead.

The Dockerfile `LABEL` instruction allows you to add a label or metadata to your docker image.

```
LABEL maintainer="devopseasylearning.com"
LABEL build_date="2017-09-05"
```

1. USER

The Dockerfile `USER` instruction sets the username of the user who is to run the container. By default, containers **run as root**.

```
USER root
```

10. ARG and ENV

dockerfile-jenkins/Dockerfile

The Dockerfile `ENV` instruction lets set you set an environment variable to be used by the system. The is a system variable and does not use defined a variable. The command `env` will list all environment variables in Linux.

The Dockerfile `ARG` instruction lets you define an argument that can be passed to Docker when you build the Docker image from the Dockerfile. `ARG` works the same as a variable. It is like defining a variable in Dockerfile.

```
ARG user=jenkins
ARG group=jenkins
ARG uid=1000
ARG gid=1000
ARG http_port=8080
ARG JENKINS_HOME=/var/jenkins_home

ENV TIA_HOME /var/tia/devops

RUN mkdir -p $JENKINS_HOME \
  && chown ${uid}:${gid} $JENKINS_HOME \
  && groupadd -g ${gid} ${group} \
  && useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -s /bin/bash ${user}

VOLUME $JENKINS_HOME
USER ${user}
EXPOSE ${http_port}
-d = home directory
-u = user ID
-g = group ID
-s = shell
```

11. CMD && ENTRYPOINT

Both are used to execute commands when the container starts or during the creation of the container. We can use `CMD && ENTRYPOINT` to start the process when the container starts or to run a script when the container start.

```
CMD ["script.sh"]
ENTRYPOINT["script.sh"]

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]

CMD ["httpd-foreground"]
ENTRYPOINT ["httpd-foreground"]

CMD ["catalina.sh", "run"]
ENTRYPOINT ["catalina.sh", "run"]
```

What is the difference between CMD and ENTRYPOINT (practical)

You can write Docker `CMD/ENTRYPOINT` instructions in both forms:

```
CMD ["script.sh"]
CMD ["/home/tia/script.sh"]
CMD ["/bin/bash", "-c", "script.sh"]
CMD ["echo", "Hello World"]

ENTRYPOINT ["script.sh"]
ENTRYPOINT ["/home/tia/script.sh"]
ENTRYPOINT ["/bin/bash", "-c", "script.sh"]
ENTRYPOINT ["echo", "Hello World"]
```

CMD

```
vim Dockerfile

FROM ubuntu
MAINTAINER Tia
RUN apt-get update -y
CMD ["/bin/bash", "-c", "uname -a"]
sudo docker build -t cmd .
sudo docker run -it cmd:latest
sudo docker run -it cmd:latest uname -r
sudo docker run -it cmd:latest ls -la
```

ENTRYPOINT

```
vim Dockerfile

FROM ubuntu
MAINTAINER Tia
RUN apt-get update -y
ENTRYPOINT ["/bin/bash", "-c", "uname -a"]
sudo docker build -t entripoint .
sudo docker run -it entripoint:latest
sudo docker run -it entripoint:latest uname -r
sudo docker run -it entripoint:latest ls -la
```

Differences between a Dockerfile, Docker Image, and Docker Container

- **A Dockerfile** is a recipe for creating Docker images
- **A Docker image** gets built by running a Docker command (which uses a Dockerfile)
- **A Docker container** is a running instance of a Docker image

What is the difference between CMD, ENTRYPOINT, AND RUN commands?

The `RUN` command will be executed during the creation of the image while `CMD && ENTRYPOINT` will execute during the creation of the container. `ENTRYPOINT` can not be overwritten at the cli while `CMD` can be overwritten at the CLI.

Dockerfile Example 1

```
vim index.html

<!DOCTYPE html>
<html>
<body style="background-color:rgb(217, 250, 210);">

<h1>Welcome to DevOps Easy Learning</h1>
<h3>This is to test httpd Web server at devopseasylearning.com</h3>

</body>
</html>
vim Dockerfile

FROM centos:latest
ARG port=80
ARG user=root
LABEL maintainer="Tia M"
RUN yum -y update && \
    yum -y install httpd

COPY index.html /var/www/html/
USER ${user}
ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]
EXPOSE ${port}
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:test .
docker run --name test -p 8030:80 -d leonardtia/devops_repo:test
docker login
docker push leonardtia/devops_repo:test
```

Access Application locally

```
http://<IP>:8030/
http://10.0.0.94:8030/
```

Dockerfile Example 2

```
vim Dockerfile

FROM httpd
ARG port=80
ARG user=root
LABEL maintainer="Tia M"
RUN apt -y update && \
    apt -y install wget && \
    apt -y install unzip

WORKDIR /usr/local/apache2/htdocs/

RUN rm -rf * && \
    wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/covid19.zip && \
    unzip covid19.zip && \
    cp -R covid19/* . && \
    rm -rf covid19.zip && \
    rm -rf covid19

USER ${user}
CMD ["httpd-foreground"]
EXPOSE ${port}
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:covid19 .
docker run --name covid19 -p 8010:80 -d leonardtia/devops_repo:covid19
docker login
docker push leonardtia/devops_repo:covid19
```

Access Application locally

```
http://<IP>:8010/
http://10.0.0.94:8010/
```

Dockerfile Example 3

```
vim Dockerfile

FROM httpd
LABEL maintainer="Tia M"
RUN apt -y update && \
    apt -y install wget && \
    apt -y install unzip

WORKDIR /usr/local/apache2/htdocs/

RUN rm -rf * && \
    wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/static-website-example.zip && \
    unzip static-website-example.zip && \
    cp -R static-website-example/* . && \
    rm -rf static-website-example.zip && \
    rm -rf static-website-example

USER root
ENTRYPOINT ["httpd-foreground"]
EXPOSE 80-10000
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:static-website-example .
docker run --name static-website-example -p 8020:80 -d leonardtia
/devops_repo:static-website-example
docker login
docker push leonardtia/devops_repo:static-website-example
```

Access Application locally

```
http://<IP>:8020/
http://10.0.0.94:8020/
```

Dockerfile Example 4

```
vim Dockerfile

FROM httpd

ARG WEB_DIRECTORY=static-website-example

LABEL maintainer="Tia M"
RUN apt -y update && \
    apt -y install wget && \
    apt -y install unzip

WORKDIR /usr/local/apache2/htdocs/

RUN rm -rf * && \
    wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/$WEB_DIRECTORY.zip && \
    unzip $WEB_DIRECTORY.zip && \
    cp -R $WEB_DIRECTORY/* . && \
    rm -rf $WEB_DIRECTORY.zip && \
    rm -rf $WEB_DIRECTORY

USER root
ENTRYPOINT ["httpd-foreground"]
EXPOSE 80
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:static-website-example .
docker run --name static-website-example -p 8020:80 -d leonardtia
/devops_repo:static-website-example
docker login
docker push leonardtia/devops_repo:static-website-example
```

Access Application locally

```
http://<IP>:8020/
http://10.0.0.94:8020/
```

Dockerfile Example 5

```
vim Dockerfile

FROM httpd

ARG WEB_DIRECTORY=covid19
ARG port=80
ARG user=root
LABEL maintainer="Tia M"
RUN apt -y update && \
    apt -y install wget && \
    apt -y install unzip

WORKDIR /usr/local/apache2/htdocs/

RUN rm -rf * && \
    wget https://linux-devops-course.s3.amazonaws.com/WEB+SIDE+HTML
/$WEB_DIRECTORY.zip && \
    unzip $WEB_DIRECTORY.zip && \
    cp -R $WEB_DIRECTORY/* . && \
    rm -rf $WEB_DIRECTORY.zip && \
    rm -rf $WEB_DIRECTORY

USER ${user}
ENTRYPOINT ["httpd-foreground"]
EXPOSE ${port}
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:covid19 .
docker run --name covid19 -p 8010:80 -d leonardtia/devops_repo:covid19
docker login
docker push leonardtia/devops_repo:covid19
```

Access Application locally

```
http://<IP>:8010/
http://10.0.0.94:8010/
```

Dockerfile Example 6

```
vim Dockerfile

FROM tomcat
LABEL maintainer="Tia M"
ADD https://warfiles-for-docker.s3.amazonaws.com/addressbook.war /usr
/local/tomcat/webapps/
ENTRYPOINT ["catalina.sh", "run"]
EXPOSE 8080
```

Biuld and test

```
docker build -t <username>/<repository name>:tag
docker build -t leonardtia/devops_repo:addressbook .
docker run --name addressbook -p 8050:8080 -d leonardtia/devops_repo:
addressbook
docker login
docker push leonardtia/devops_repo:addressbook
```

Access Application locally

```
http://<IP>:8050/addressbook
http://10.0.0.94:8050/addressbook
```

Dockerfile Example 7

```
vim Dockerfile

FROM centos
ARG user=jenkins
ARG group=jenkins
ARG uid=1000
ARG gid=1000
ARG JENKINS_HOME=/var/jenkins_home

ENV TIA_HOME /var/tia/devops

RUN mkdir -p $JENKINS_HOME \
  && chown ${uid}:${gid} $JENKINS_HOME \
  && groupadd -g ${gid} ${group} \
  && useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -s /bin/bash ${user}

USER ${user}
-d = home directory
-u = user ID
-g = group ID
-s = shell

docker build -t env .
docker run -it env:latest bash
id
env
echo $JENKINS_HOME
```