Aim: To learn dockerfile instructions build an image for a sample web application using dockerfile

Objective: The objective of learning Dockerfile instructions to build an image for a sample web application is to acquire the knowledge and skills necessary to define the environment and dependencies required to containerize and deploy the application effectively.

Theory:

Dockerfile:

A Dockerfile is a text configuration file written using a special syntax. It describes step-by-step instructions of all the commands you need to run to assemble a Docker Image. The docker build command processes this file generating a Docker Image in your Local Image Cache, which you can then start-up using the docker run command, or push to a permanent Image Repository. Creating a Dockerfile is as easy as creating a new file named "Dockerfile" with your text editor of choice and defining some instructions. The name of the file does not really matter.

Docker Image:

A Docker image is a lightweight, standalone, executable package that contains everything needed to run an application or service, including the application code, libraries, dependencies, and operating system. It is a template that can be used to create Docker containers, which are instances of an image that can be run in isolation.

Here are some common Dockerfile instructions:

1. FROM - Specifies the base image to use for the build.
2. RUN - Executes a command inside the container during the build process. This is often used to install software or dependencies.
3. COPY - Copies files from the host system into the container.
4. ADD - Similar to COPY, but can also download files from a URL and extract compressed files.
5. WORKDIR - Sets the working directory inside the container for subsequent commands.
6. ENV - Sets environment variables inside the container.
7. EXPOSE - Informs Docker that the container listens on the specified network ports at runtime.

8. CMD - Specifies the command to run when the container is started.
9. ENTRYPOINT - Specifies the command to run when the container is started and allows arguments to be passed to the command.

Steps:

1. Create a new directory for your Dockerfile and application files.

2. Create a new file named Dockerfile in this directory.

3.Open the Dockerfile and write the instructions to build your Docker image.

Here's an example

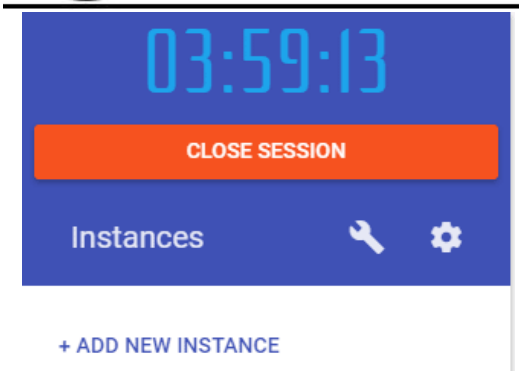
```
# The line below states we will base our new image on the Latest Official Ubun
FROM ubuntu:latest

#
# Identify the maintainer of an image
LABEL maintainer="myname@somecompany.com"

#
# Update the image to the latest packages
RUN apt-get update && apt-get upgrade -y

#
# Install NGINX to test.
RUN apt-get install nginx -y

#
# Expose port 80
EXPOSE 80

#
# Last is the actual command to start up NGINX within our Container
CMD ["nginx", "-g", "daemon off;"]
```

4. Save and close the docker file.

Now Building and Testing Dockerfiles

1. First of all, head over to http://play-with-docker.com and start a new session. You need to create an account first.

2. Once your session is active click on "Add New Instance":



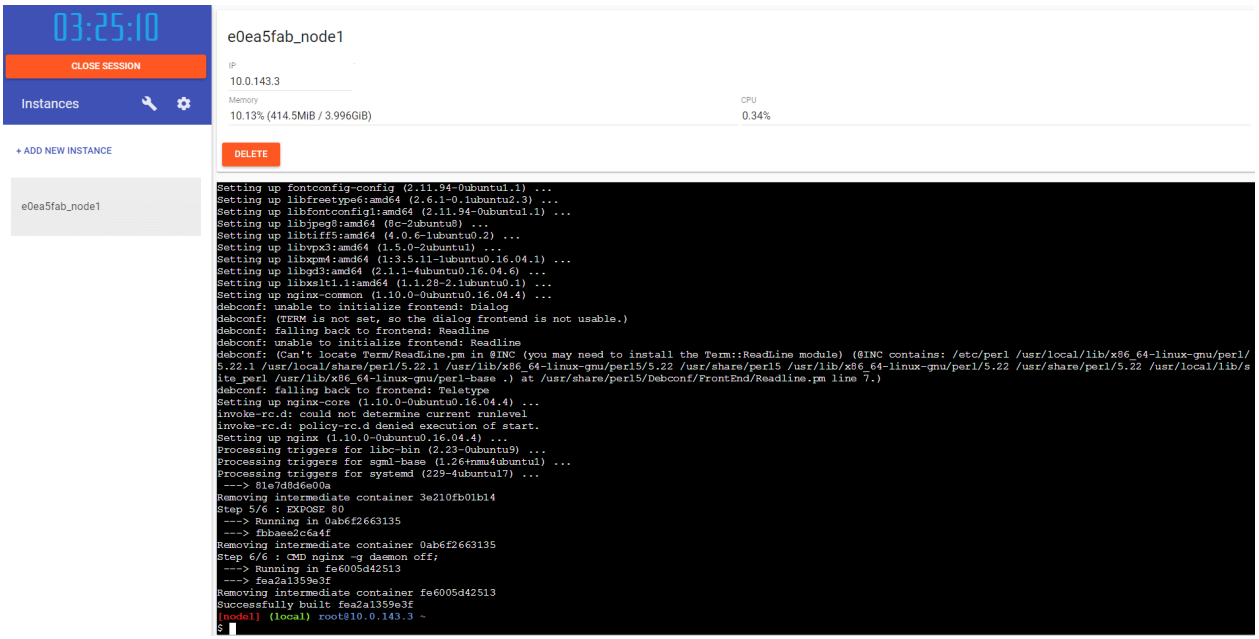
3. A new instance will start with a Docker Engine ready to accept commands

4. Next create/edit the Dockerfile. Run "vi Dockerfile", press "i" to switch to "Insert Mode", copy/paste the contents of our Dockerfile, press "Esc" to exit "Insert Mode", and save+exit by typing ":x"

```
# The line below states we will base our new image on the Latest Official Ubuntu
FROM ubuntu:latest

#
# It's best practice to identify yourself as the maintainer of an image
MAINTAINER My Name "myname@somecompany.com"

#
# As with a regular OS install, we want to update the image to the latest packages
RUN apt-get update && apt-get upgrade -y

#
# Next we should install some app to test. Let's use NGINX for our example
RUN apt-get install nginx -y

#
# By default NGINX runs on port 80, so we need to expose it so it can be reached from the outside
EXPOSE 80

#
# Last we need to run the actual command to start up NGINX within our Container
CMD ["nginx", "-g", "daemon off;"]
```

5. Build the new image using the command docker build <path>. Path refers to the directory containing the Dockerfile.

6. At the end of the process you should see the message "Successfully built <image ID>"



7. Start the new image and test connectivity to NGINX. Run the command

docker run -p 80:80 <image ID>. The option -p 80:80 exposes the Container port 80 as the Host port 80 to the world

8. As a result a port 80 link should have become active next to the IP. Click on it to access your NGINX service.

9. That's it !!! We have created a docker image and run it in our local machine.

Output:

Conclusion:

Q1. What is Dockerfile?

A Dockerfile is a text-based script used to define the configuration and steps required to build a Docker image. It contains instructions that specify how to assemble the image layer by layer, including commands to install dependencies, copy files, set environment variables, and configure the runtime environment. Dockerfiles provide a reproducible and version-controlled way to automate the creation of Docker images, ensuring consistency and reliability across different environments. By defining the image's contents and behavior in a Dockerfile, developers can easily share and distribute their applications as Docker images, streamlining the deployment process and facilitating the adoption of containerization technologies.

Q2. What is Docker Image?

A Docker image is a lightweight, standalone, executable package that contains everything needed to run a piece of software, including the code, runtime, libraries, dependencies, and configuration files. It serves as a snapshot of a specific environment at a particular moment in time, encapsulating the application and its dependencies in a portable and reproducible format. Docker images are created from Dockerfiles using the Docker build command, and they can be stored and distributed via container registries like Docker Hub. Images are used as the basis for running Docker containers, which are instances of those images executing in isolated environments. This architecture enables developers to easily package, deploy, and scale applications across different infrastructure environments, making Docker images a fundamental building block of containerized workflows.