Experiment No.9
To learn dockerfile instructions build an image for a sample web application using dockerfile
Date of Performance:
Date of Submission:

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

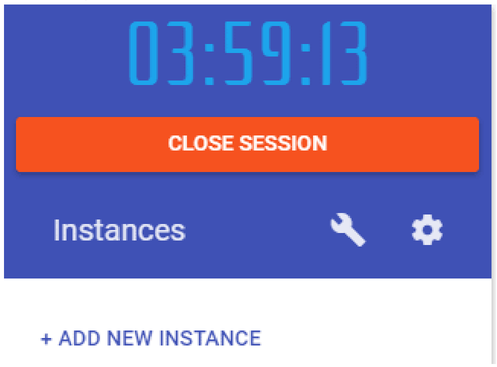
```
# The line below states we will base our new image on the Latest Official Ubun
FROM ubuntu:latest

#
# 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.

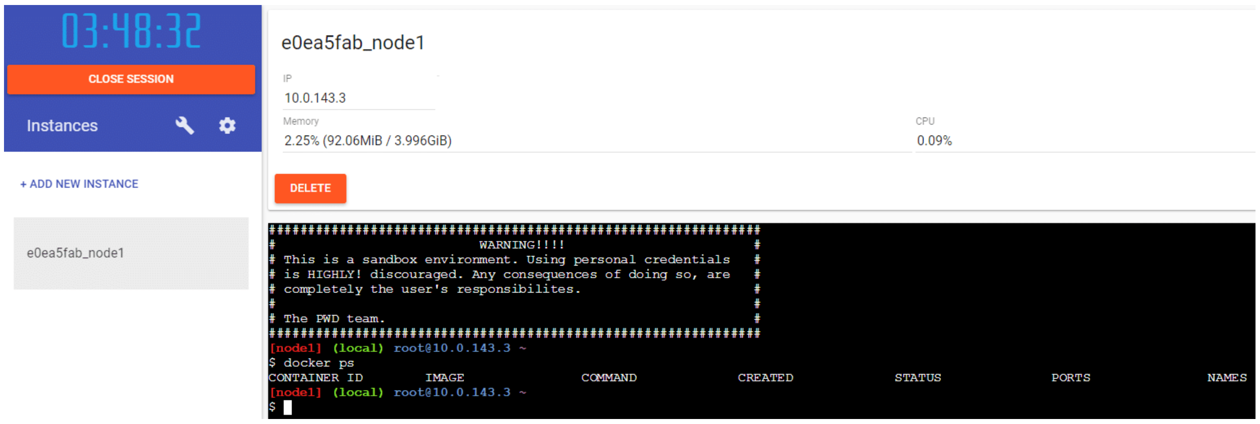
Now Building and Testing Dockerfiles

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":

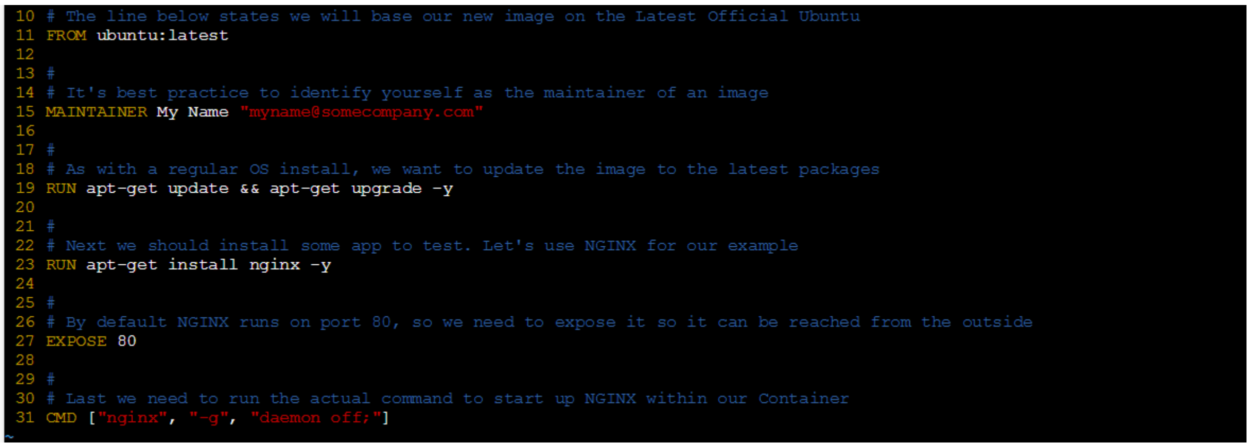


3. A new instance will start with a Docker Engine ready to accept commands



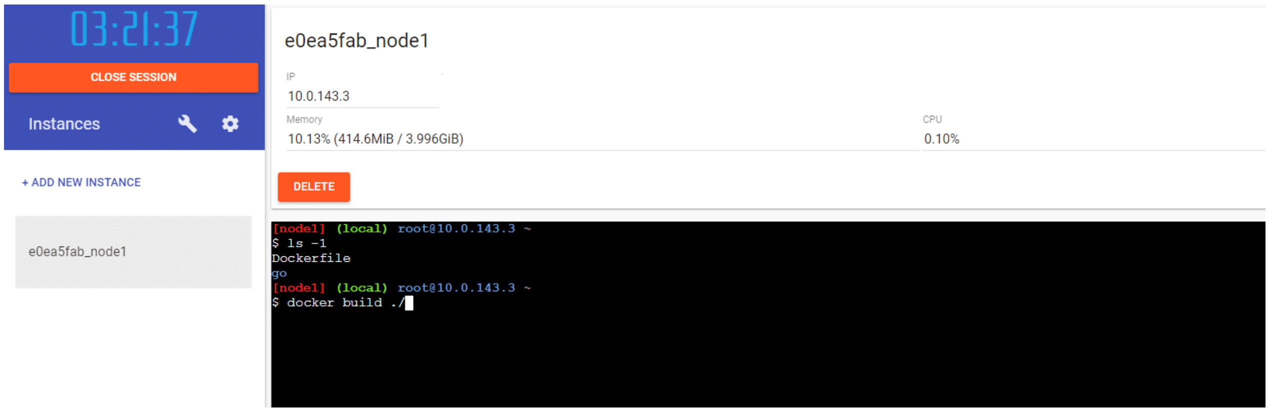
```
###############################################################
#                          WARNING!!!!                        #
# This is a sandbox environment. Using personal credentials   #
# is HIGHLY! discouraged. Any consequences of doing so, are   #
# completely the user's responsibilites.                      #
#                                                             #
# The PWD team.                                               #
###############################################################
[node1] (local) root@10.0.143.3 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
[node1] (local) root@10.0.143.3 ~
$
```

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"



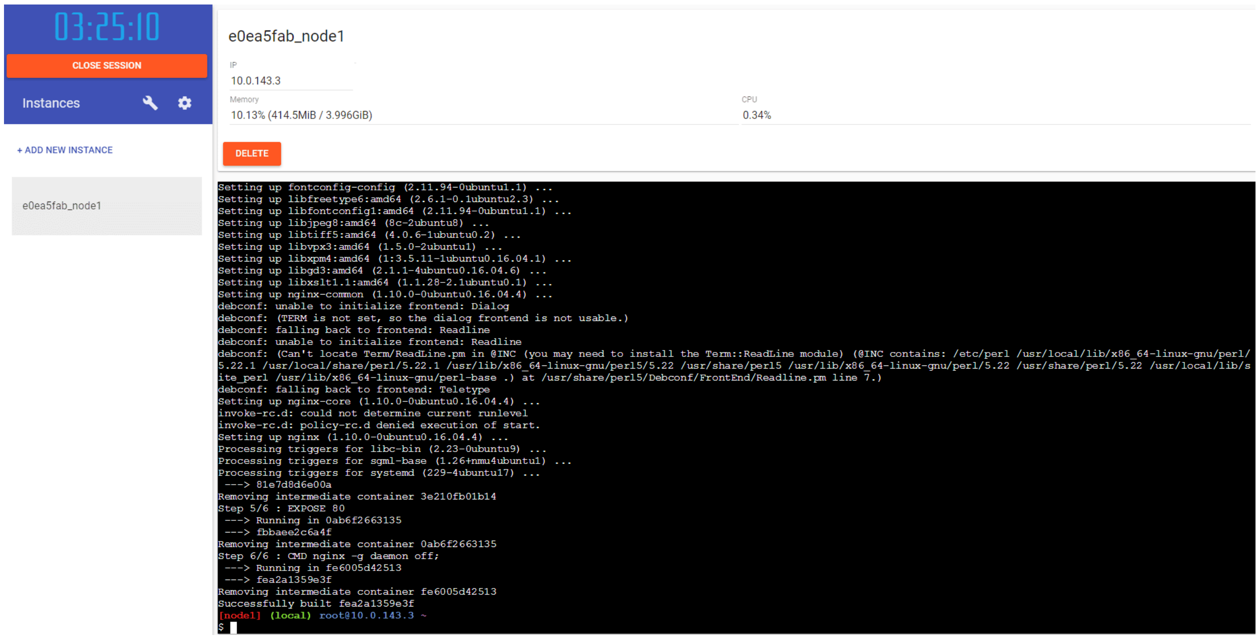
```
# The line below states we will base our new image on the Latest Official Ubuntu
FROM ubuntu:latest

#
# 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.



```
[node1] (local) root@10.0.143.3 ~
$ ls -1
Dockerfile
go
[node1] (local) root@10.0.143.3 ~
$ docker build ./
```

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

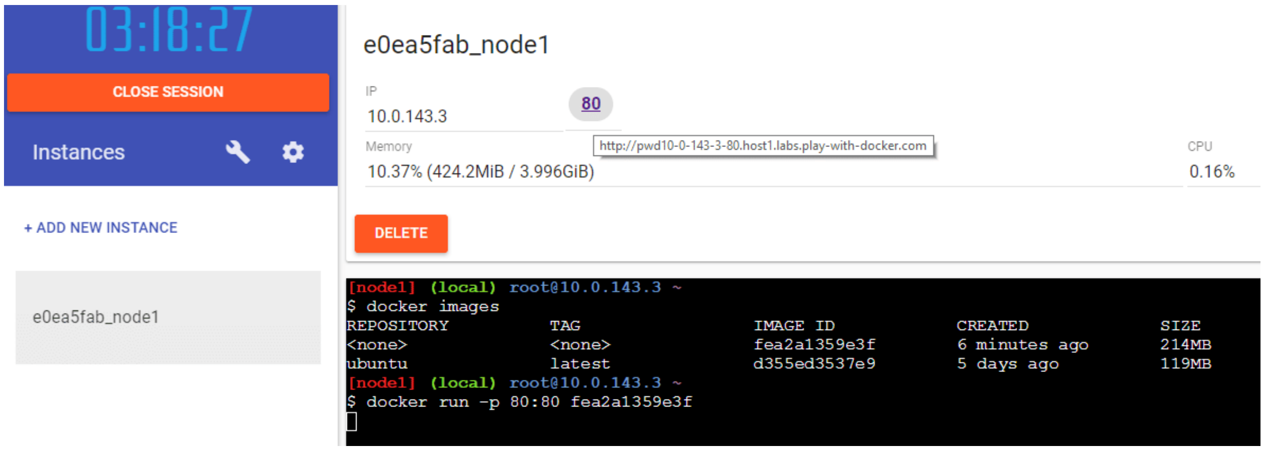


e0ea5fab_node1
IP 10.0.143.3
Memory 10.13% (414.5MiB / 3.996GiB)
CPU 0.34%

```
Setting up fontconfig-config (2.11.94-0ubuntu1.1) ...
Setting up libfreetype6:amd64 (2.6.1-0.1ubuntu2.3) ...
Setting up libfontconfig1:amd64 (2.11.94-0ubuntu1.1) ...
Setting up libjpeg8:amd64 (8c-2ubuntu8) ...
Setting up libtiff5:amd64 (4.0.6-1ubuntu0.2) ...
Setting up libvpx3:amd64 (1.5.0-2ubuntu1) ...
Setting up libxpm4:amd64 (1:3.5.11-1ubuntu0.16.04.1) ...
Setting up libgd3:amd64 (2.1.1-4ubuntu0.16.04.6) ...
Setting up libxslt1.1:amd64 (1.1.28-2.1ubuntu0.1) ...
Setting up nginx-common (1.10.0-0ubuntu0.16.04.4) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC contains: /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/
5.22.1 /usr/local/share/perl/5.22.1 /usr/lib/x86_64-linux-gnu/perl5/5.22 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl/5.22 /usr/share/perl/5.22 /usr/local/lib/s
ite_perl /usr/lib/x86_64-linux-gnu/perl-base .) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Setting up nginx-core (1.10.0-0ubuntu0.16.04.4) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Setting up nginx (1.10.0-0ubuntu0.16.04.4) ...
Processing triggers for libc-bin (2.23-0ubuntu9) ...
Processing triggers for sgml-base (1.26+nmu4ubuntu1) ...
Processing triggers for systemd (229-4ubuntu17) ...
 ---> 81e7d8d6e00a
Removing intermediate container 3e210fb01b14
Step 5/6 : EXPOSE 80
 ---> Running in 0ab6f2663135
 ---> fbbaee2c6a4f
Removing intermediate container 0ab6f2663135
Step 6/6 : CMD nginx -g daemon off;
 ---> Running in fe6005d42513
 ---> fea2a1359e3f
Removing intermediate container fe6005d42513
Successfully built fea2a1359e3f
[node1] (local) root@10.0.143.3 ~
$
```

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

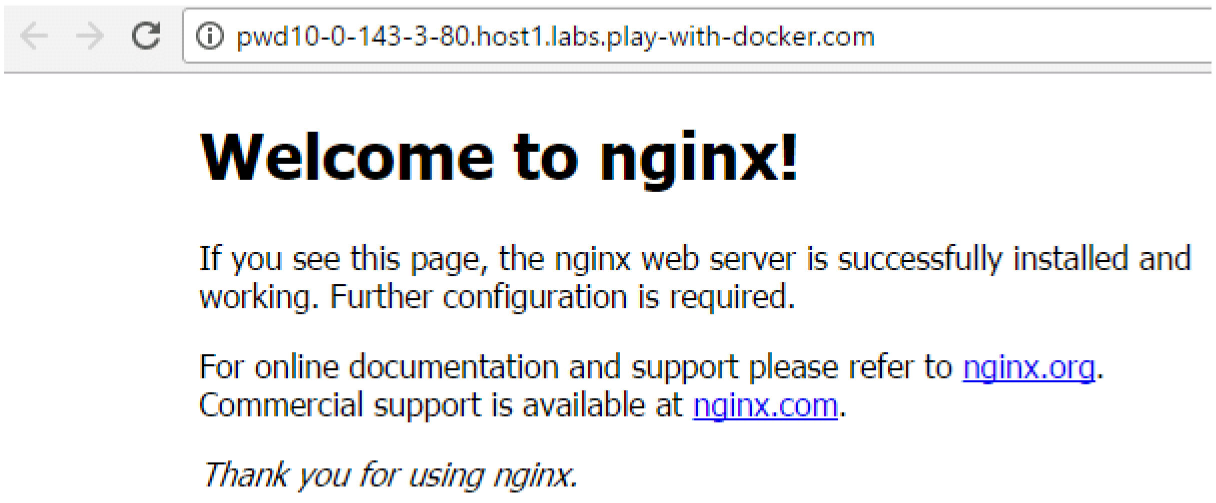


e0ea5fab_node1
IP 10.0.143.3
80
http://pwd10-0-143-3-80.host1.labs.play-with-docker.com
Memory 10.37% (424.2MiB / 3.996GiB)
CPU 0.16%

```
[node1] (local) root@10.0.143.3 ~
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              fea2a1359e3f        6 minutes ago       214MB
ubuntu              latest              d355ed3537e9        5 days ago          119MB
[node1] (local) root@10.0.143.3 ~
$ docker run -p 80:80 fea2a1359e3f
```

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:


pwd10-0-143-3-80.host1.labs.play-with-docker.com

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Conclusion:

Q1. What is Dockerfile?

A Dockerfile is a text document that contains a set of instructions or commands used to assemble a Docker image. It serves as a blueprint for defining what goes into the image, including the base operating system, application dependencies, environment variables, and other configurations. These instructions are written in a simple syntax and allow developers to automate the creation of lightweight, portable, and reproducible containerized applications. When the Dockerfile is executed with the `docker build` command, it produces a Docker image that can then be used to create Docker containers, providing a consistent environment for running applications across different systems.

Q2. What is Docker Image?

A Docker image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and dependencies. It's built from a Dockerfile, which contains instructions on how the image should be constructed. Images are the building blocks of Docker containers, allowing developers to easily package applications and services so they can be run reliably and consistently across different environments, from development to production, without worrying about differences in system configurations.