Graded Assignment on Docker

Q1) Pull any image from the docker hub, create its container, and execute it showing the output.
Docker is a software platform to create, test and deploy applications in an isolated environment. Docker uses container to package up an application with all of the parts it needs including, libraries and dependencies. It allows applications to use the kernel and other resources of the host operating system this will boost the performance and reduce the size of the application. Docker Hub is a centralized repository service that allows you to store container images and share them with your team. You can use Pull and Push command to upload and download images to and from the Docker Hub.

*Give the docker version command.

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ docker version
Client:
 Cloud integration: v1.0.29
 Version:           20.10.22
 API version:       1.41
 Go version:        go1.18.9
 Git commit:        3a2c30b
 Built:             Thu Dec 15 22:36:18 2022
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Desktop 4.16.3 (96739)
 Engine:
  Version:          20.10.22
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.9
  Git commit:       42c8b31
  Built:            Thu Dec 15 22:26:14 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.14
  GitCommit:        9ba4b250366a5ddde94bb7c9d1def331423aa323
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Step1:

We can pull the image from the docker hub using the docker pull image_name.

Let us download the image called nginx from the docker hub.

Once the nginx image is downloaded, we get the following output.

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
bb263680fed1: Pulling fs layer
258f176fd226: Pulling fs layer
a0bc35e70773: Pulling fs layer
077b9569ff86: Pulling fs layer
3082a16f3b61: Pulling fs layer
7e9b29976cce: Pulling fs layer
3082a16f3b61: Waiting
7e9b29976cce: Waiting
077b9569ff86: Waiting
a0bc35e70773: Verifying Checksum
a0bc35e70773: Download complete
077b9569ff86: Download complete
3082a16f3b61: Verifying Checksum
3082a16f3b61: Download complete
7e9b29976cce: Download complete
258f176fd226: Verifying Checksum
258f176fd226: Download complete
bb263680fed1: Verifying Checksum
bb263680fed1: Download complete
bb263680fed1: Pull complete
258f176fd226: Pull complete
a0bc35e70773: Pull complete
077b9569ff86: Pull complete
3082a16f3b61: Pull complete
7e9b29976cce: Pull complete
Digest: sha256:6650513efd1d27c1f8a5351cbd33edf85cc7e0d9d0fcb4ffb23d8fa89b601b
a8
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Step2:

Next, create a new nginx container from the downloaded image and expose it on port 80 using the following command.

docker run –name docker-nginx -p 80:80 -d nginx

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ docker run --name docker-nginx -p 80:80 -d nginx
68b1e11f55a790baa936bf21c0d4a48d6c6a8e4f076ca6579595966c4a521e93
```

Step3:

We can also verify the nginx container with the below command.

docker ps

We will get the following output.

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                NAMES
68b1e11f55a7   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp   docker-nginx
```

It will show the container-id.

Step4:

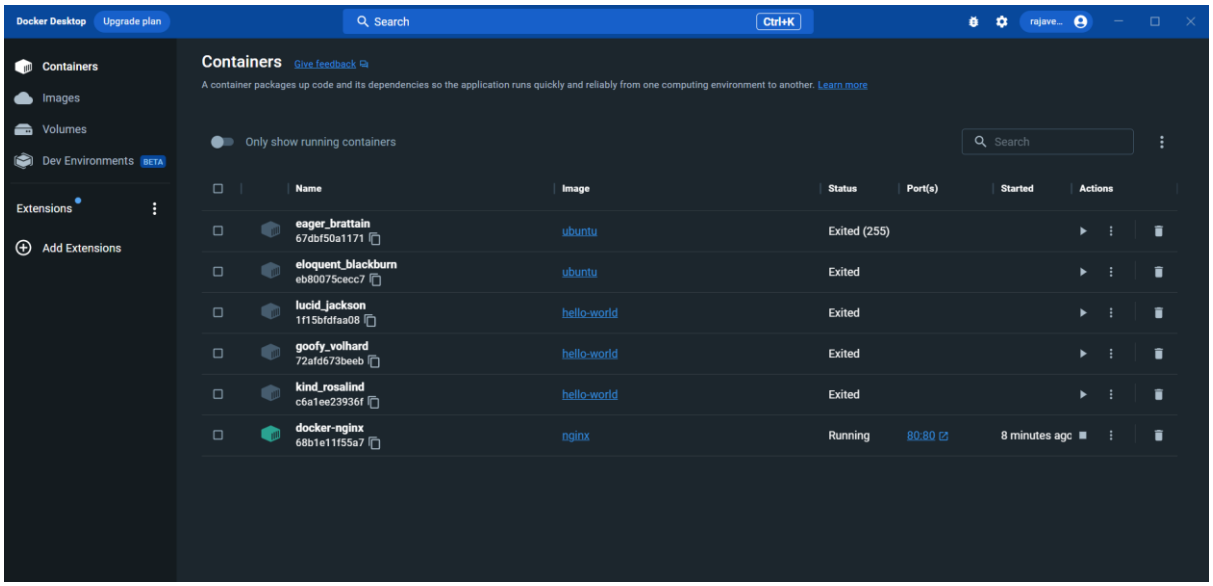
We can connect to the running container with the following command.

```
C:\Users\rajav>docker exec -it docker-nginx /bin/bash
root@68b1e11f55a7:/# apt  update
```

Now we are connected to the running container.

```
C:\Users\rajav>docker exec -it docker-nginx /bin/bash
root@68b1e11f55a7:/# apt  update
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8183 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages [226 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [14.6 kB]
Fetched 8632 kB in 10s (877 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
1 package can be upgraded. Run 'apt list --upgradable' to see it.
root@68b1e11f55a7:/#
```

*Open docker desktop to view whether the container is created or not.



Q2) Create the basic java application, generate its image with necessary files, and execute it with docker.

Creating the basic java application.

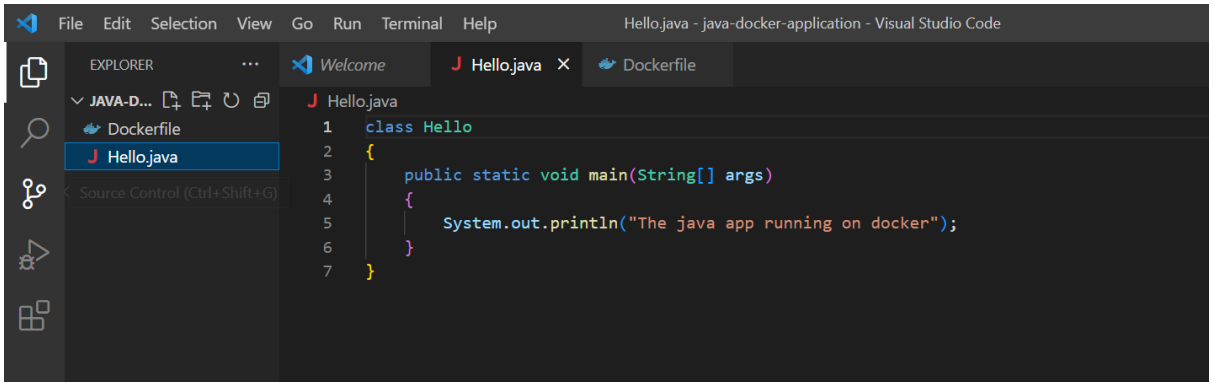
Step1: Create a directory, it is used to store the files.

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ mkdir java-docker-application
```

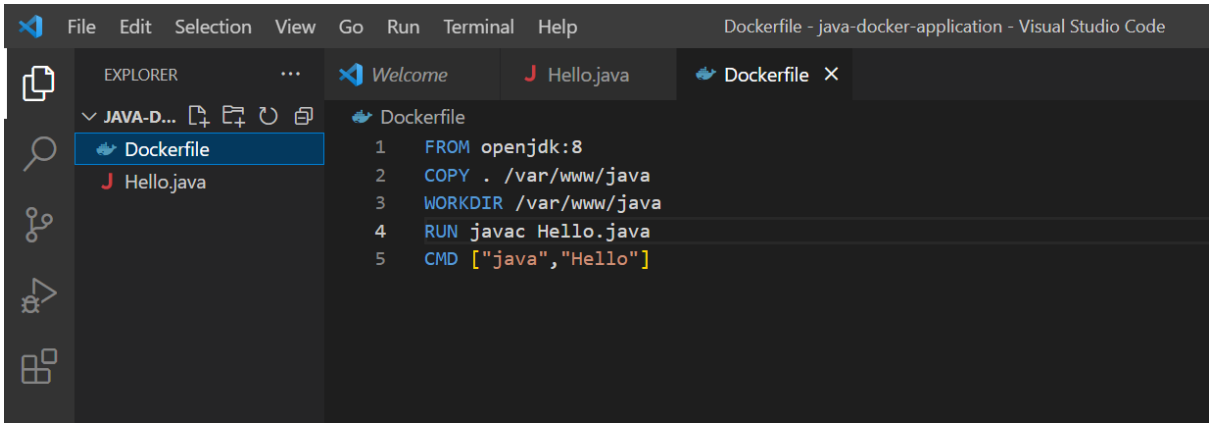
Step2: Go to the directory that you have created.

```
raja vemana@Raja-PC MINGW64 ~ (master)
$ cd java-docker-application

raja vemana@Raja-PC MINGW64 ~/java-docker-application (master)
$ code .
```

Step3: Create a java file, save it as Hello.java



Step4: Create a docker file.



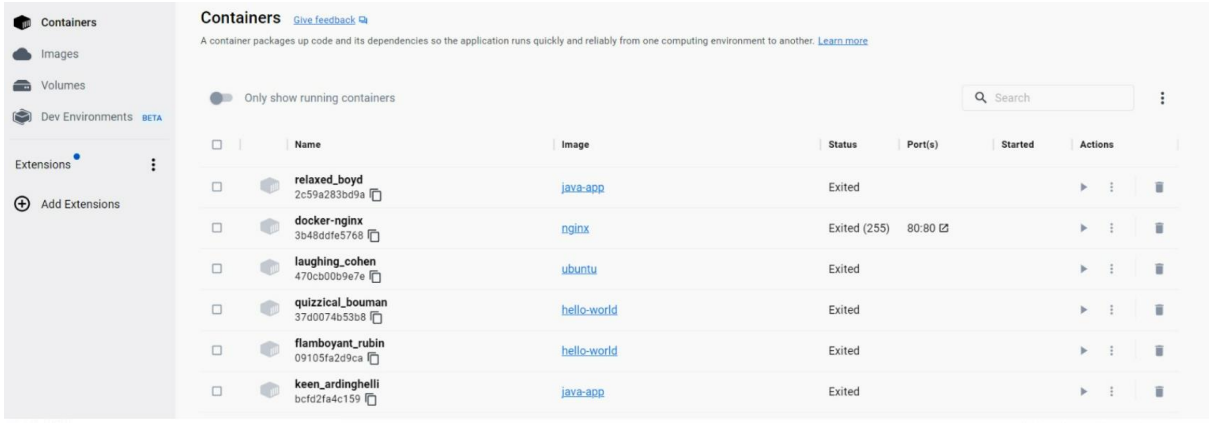
Step5: Now create an image by following below command. We must login as root in order to create a image. In the following command, java-app is name of the image. We can have any name for our docker image.

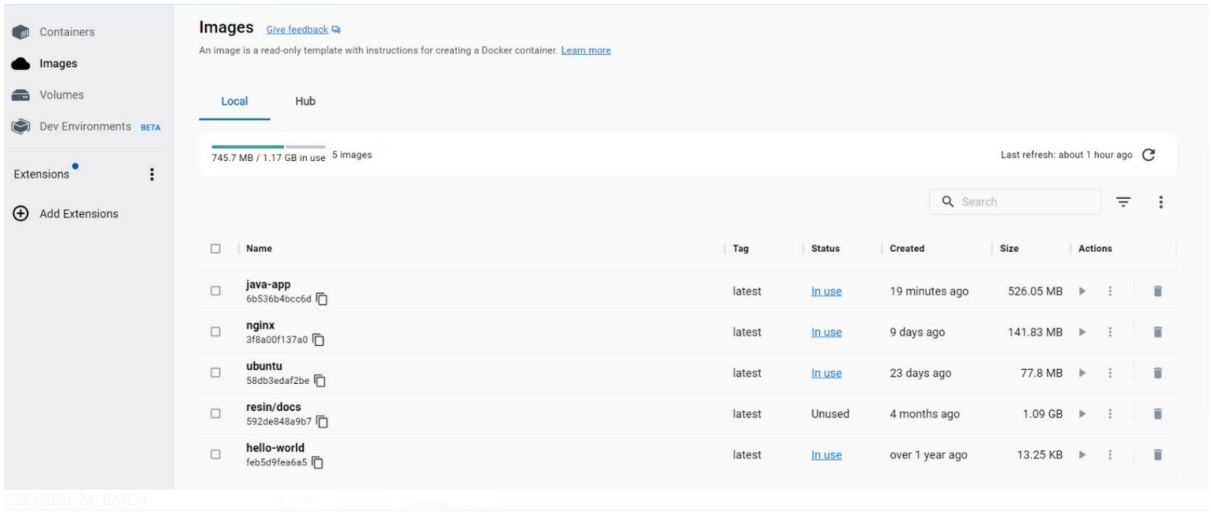
```
raja vemana@Raja-PC MINGW64 ~/java-docker-application (main)
$ docker build -t java-app .
#1 [internal] load build definition from Dockerfile
#1 sha256:579c8aac9534ec849cfe70249d8c1e5f797fe7ae519e516bf6e499eebcdca5e4
#1 transferring dockerfile: 31B done
#1 DONE 0.0s

#2 [internal] load .dockerignore
#2 sha256:e6da1a084f3939d416aacd4806d00912a26607290267ffc79f91e138e6bb4249
#2 transferring context: 2B done
#2 DONE 0.0s

#3 [internal] load metadata for docker.io/library/openjdk:8
#3 sha256:c1006613b124ab347fbb29ae49e2ab62add271baf34bdfe7a7a4c383ac159b76
#3 ...

#4 [auth] library/openjdk:pull token for registry-1.docker.io
#4 sha256:deeda5a36343acdf15ada35db30e08d8171c1c3b25d5903495b16aef4f39dceb
#4 DONE 0.0s

#3 [internal] load metadata for docker.io/library/openjdk:8
#3 sha256:c1006613b124ab347fbb29ae49e2ab62add271baf34bdfe7a7a4c383ac159b76
#3 DONE 2.6s

#5 [1/4] FROM docker.io/library/openjdk:8@sha256:86e863cc57215cfb181bd319736d0ba
f625fe8f150577f9eb58bd937f5452cb8
#5 sha256:1e7d9e224eeb34ef733a8ab6274c72dbf6d09407a6df09d7e6001657f4d7ee92
#5 DONE 0.0s

#6 [internal] load build context
#6 sha256:4bf05193abbf5e0c4099a05df143ffe763f83e2da1c6a2472f7eab3637909ad0
#6 transferring context: 61B done
#6 DONE 0.0s

#8 [3/4] WORKDIR /var/www/java
#8 sha256:254bbcc66f4115bf5c8a6b4e8ba627bec5846eaf3d8f1e568fe722da7bcfae51
#8 CACHED

#7 [2/4] COPY . /var/www/java
#7 sha256:63d803f994c98e6e212873285901234a0eb248fdfeaebd1ef69f0dc5473d8a05
#7 CACHED

#9 [4/4] RUN javac Hello.java
#9 sha256:fd8750f31575137cf40d63a1a8a6e15d05631743d381076876a821bd7d64bb31
#9 CACHED

#10 exporting to image
#10 sha256:e8c613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00
```

Step6: After successfully building the image, now we can run docker by using run command.

```
raja vemana@Raja-PC MINGW64 ~/java-docker-application (main)
$ docker run java-app
The java app running on docker
```

*Open docker desktop and you can see that the java application is running.


Containers
Images
Volumes
Dev Environments BETA
Extensions
Add Extensions
Containers
Give feedback
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more
Only show running containers
Search
Name
Image
Status
Port(s)
Started
Actions
relaxed_boyd
2c59a283bd9a
java-app
Exited
docker-nginx
3b48ddfe5768
nginx
Exited (255)
80:80
laughing_cohen
470cb00b9e7e
ubuntu
Exited
quizzical_bouman
37d0074b53b8
hello-world
Exited
flamboyant_rubin
09105fa2d9ca
hello-world
Exited
keen_ardinghelli
bcfd2fa4c159
java-app
Exited


Containers
Images
Volumes
Dev Environments BETA
Extensions
Add Extensions
Images
Give feedback
An image is a read-only template with instructions for creating a Docker container. Learn more
Local
Hub
745.7 MB / 1.17 GB in use
5 images
Last refresh: about 1 hour ago
Search
Name
Tag
Status
Created
Size
Actions
java-app
6b536b4bcc6d
latest
In use
19 minutes ago
526.05 MB
nginx
3f8a00f137a0
latest
In use
9 days ago
141.83 MB
ubuntu
58db3edaf2be
latest
In use
23 days ago
77.8 MB
resin/docs
592de848a9b7
latest
Unused
4 months ago
1.09 GB
hello-world
feb5d9fea6a5
latest
In use
over 1 year ago
13.25 KB