CSCI 5409: Advanced Topics in Cloud Computing

Assignment 2 (C)

In this assignment, OS level virtualization (containerization) was explored. This assignment used Docker to containerize the simple web application developed with Express.js in assignment 1. The containerized application was deployed on an AWS EC2 instance. The database used was Dal FCS MySQL database. SSH tunnelling was used to connect to it.

Following were the steps followed for this assignment.

Launch and run AWS EC2 instance:

Filter by tags and attributes or search by keyword — 1 to 1 of 1

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IP
Webserver	i-0e319f77382ccc7cf	t2.micro	us-east-1b	running	2/2 checks ...	None	ec2-54-84-51-72.comp...	54.84.51.72	-

Install Docker on the EC2 instance:

3. ec2-54-144-0-5.compute-1.amazona

```
ubuntu@ip-172-31-88-233:~$ sudo apt install docker.io
```

3. ec2-54-144-0-5.compute-1.amazona

```
(Reading database ... 84417 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
Unpacking pigz (2.4-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.5-15ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.5-15ubuntu1) ...
Selecting previously unselected package cgroupfs-mount.
Preparing to unpack .../2-cgroupfs-mount_1.4_all.deb ...
Unpacking cgroupfs-mount (1.4) ...
Selecting previously unselected package runc.
Preparing to unpack .../3-runc_1.0.0~rc10-0ubuntu1~18.04.2_amd64.deb ...
Unpacking runc (1.0.0~rc10-0ubuntu1~18.04.2) ...
Selecting previously unselected package containerd.
Preparing to unpack .../4-containerd_1.3.3-0ubuntu1~18.04.2_amd64.deb ...
Unpacking containerd (1.3.3-0ubuntu1~18.04.2) ...
Selecting previously unselected package docker.io.
Preparing to unpack .../5-docker.io_19.03.6-0ubuntu1~18.04.1_amd64.deb ...
Unpacking docker.io (19.03.6-0ubuntu1~18.04.1) ...
Selecting previously unselected package ubuntu-fan.
Preparing to unpack .../6-ubuntu-fan_0.12.10_all.deb ...
Unpacking ubuntu-fan (0.12.10) ...
Setting up runc (1.0.0~rc10-0ubuntu1~18.04.2) ...
Setting up cgroupfs-mount (1.4) ...
Setting up containerd (1.3.3-0ubuntu1~18.04.2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service →
 /lib/systemd/system/containerd.service.
Setting up bridge-utils (1.5-15ubuntu1) ...
Setting up ubuntu-fan (0.12.10) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service →
 /lib/systemd/system/ubuntu-fan.service.
Setting up pigz (2.4-1) ...
Setting up docker.io (19.03.6-0ubuntu1~18.04.1) ...
Adding group `docker' (GID 115) ...
Done.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/sy
stemd/system/docker.socket.
docker.service is a disabled or a static unit, not starting it.
Processing triggers for systemd (237-3ubuntu10.40) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ureadahead (0.100.0-21) ...
```

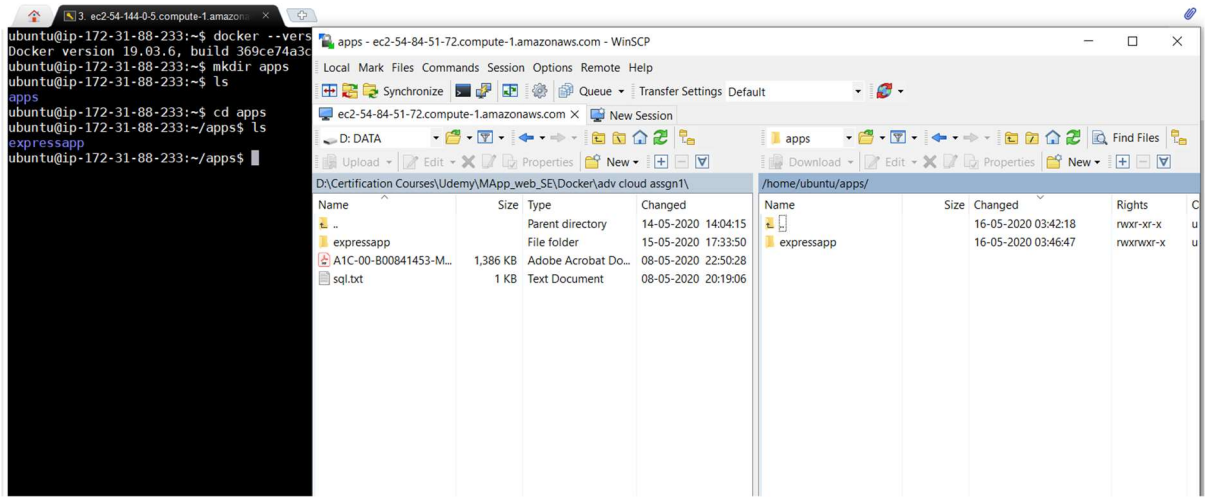
Start the docker engine:

To start the docker engine after installation, enable the docker service and start the docker service daemon (docker engine running as a system service in the background). Enabling docker would automatically start the docker daemon every time we boot the system so that we need not explicitly start it.

```
ubuntu@ip-172-31-88-233:~$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
ubuntu@ip-172-31-88-233:~$ sudo systemctl start docker
ubuntu@ip-172-31-88-233:~$
```

Transfer project files from the local machine to the cloud (EC2):

Used WinSCP tool to transfer the project files to EC2 over SFTP.



The Dockerfile:

```
Dockerfile
#Get base 'node' docker image
FROM node:14

#Set working directory in the image as express_web_app
WORKDIR /express_web_app

#Copy the contents of working directory of host machine to the working directory of the image
COPY ./ /express_web_app

#Get the dependencies for the project
RUN npm install

#Specify the fire up command when the container gets created from this image.
CMD npm start

#Open the port in the conatiner, that is used by the app to listen on.
EXPOSE 3000
```

Build the docker image of the app using the Dockerfile:

```
ubuntu@ip-172-31-88-233:~/apps/expressapp$ sudo docker build -t abhinav7896/expressapp_ubuntu:1.0 .
Sending build context to Docker daemon   6.68MB
Step 1/6 : FROM node:14
14: Pulling from library/node
1c6172af85ee: Pull complete
b194b0e3c928: Pull complete
1f5ec00f35d5: Pull complete
93b1353672b6: Pull complete
3d7f38db3cca: Pull complete
21e102f9fe89: Pull complete
e63dac87cb8e: Pull complete
2e15f38664d9: Pull complete
7639f95d3d43: Pull complete
Digest: sha256:31f2b2a55583b906129f2b38e4128f0a5247fedc80ab6dc134c0a8676ae3dfe3
Status: Downloaded newer image for node:14
 ---> e00884c768d7
Step 2/6 : WORKDIR /express_web_app
 ---> Running in 0908ca509aca
Removing intermediate container 0908ca509aca
 ---> de6536e35b23
Step 3/6 : COPY ./ /express_web_app
 ---> 9a5e6fdb1598
Step 4/6 : RUN npm install
 ---> Running in a29db8902839
npm WARN expressapp@1.0.0 No description
npm WARN expressapp@1.0.0 No repository field.

audited 87 packages in 0.948s
found 0 vulnerabilities

Removing intermediate container a29db8902839
 ---> 90dc0e649e8b
Step 5/6 : CMD npm start
 ---> Running in bcd7f41343a6
Removing intermediate container bcd7f41343a6
 ---> 353687d4eee8
Step 6/6 : EXPOSE 3000
 ---> Running in 2a608eb71a0a
Removing intermediate container 2a608eb71a0a
 ---> e4937fa2e55d
Successfully built e4937fa2e55d
Successfully tagged abhinav7896/expressapp_ubuntu:1.0
ubuntu@ip-172-31-88-233:~/apps/expressapp$
```

A docker image is a filesystem snapshot of our app's files which include the OS binaries plus the required base software, app's source code files, dependencies, and all other files required for the app. All these are specified in the Dockerfile based on which the image is created.

Docker command to build an image:

docker build -t <image-name> .

(A good format for image name is *dockerID/image-name:tag* which refers to the corresponding repository on the docker hub. Thus, it enables us to push the image to the hub without requiring us to retag the image to comply to the format.)

Deploy a container from the image:

```
root@ip-172-31-88-233:/home/ubuntu/apps/expressapp# docker run -p 8080:3000 abhinav7896/expressapp_ubuntu:1.0

> expressapp@1.0.0 start /express_web_app
> node app.js

Listening on port 3000
SSH Connection :: READY
Connected to the database successfully!
```

A container is the isolated, running process which actually consumes the resources (CPU, RAM, storage, network etc).

Docker command to deploy a container (with port mapping):

docker run -p <host_port>:<container_port> <image_name>

A container is like a lightweight/mini computer on its own, with just our app, few OS binaries, base software, and dependencies for the project. The container thus has its own pool of network ports like any other computer. To forward the traffic from the host's port to a specific port on the container, we need to specify the port mapping while deploying the container, using the flag: -p.

To verify if the container got all of our intended project files, we can acquire the container's shell and issue *ls* to list the files and folders in our container's working directory. This can be done with:

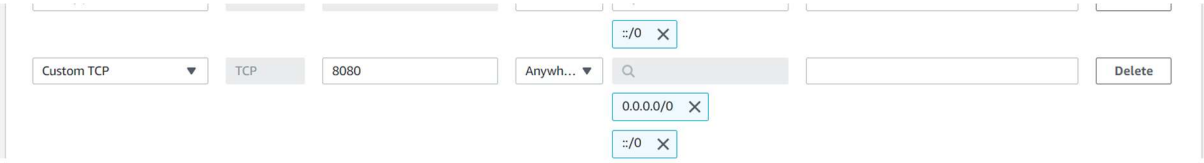
docker exec -it <container_id> sh

```
root@ip-172-31-88-233:/home/ubuntu# docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
83eda3be4763        abhinav7896/expressapp_ubuntu:1.0   "docker-entrypoint.s…"   9 minutes ago       Up 9 minutes        0.0.0.0:8080->3000/tcp   gifted_al
len
root@ip-172-31-88-233:/home/ubuntu# docker exec -it 83e sh
# ls
Dockerfile  app.js  config  errors  models  node_modules  package-lock.json  package.json  routes  test.js  views
#
```

As shown above, we got all of our project structure into the container.

Testing the containerized web service with Postman:

Before making any requests on port 8080, we need to open the port 8080 of the EC2 instance by adding a new custom TCP inbound rule in the security group.



Now that the 8080 port is able to accept inbound traffic, we can start making HTTP requests to that port, which are then forwarded to the container's port: 3000 for processing.

HTTP GET

The jobs table in the database:

Result Grid Filter Rows:

	jobName453	partID453	qty
	i1007	2222	34
	i1008	2298	78
	i1011	2134	62
	NULL	NULL	NULL

Get request to fetch all the jobs:

GET http://ec2-3-85-114-115.compute-1.amazonaws.com:8080/api/jobs453/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 147 ms Size: 367 B Save

Pretty Raw Preview Visualize JSON


```
[
    {
        "jobName453": "j1007",
        "partID453": "2222",
        "qty": 34
    },
    {
        "jobName453": "j1008",
        "partID453": "2298",
        "qty": 78
    },
    {
        "jobName453": "j1011",
        "partID453": "2134",
        "qty": 62
    }
]
```


Get job by jobName:

GET http://ec2-3-85-114-115.compute-1.amazonaws.com:8080/api/jobs453/getJob/j1011

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 137 ms

Pretty Raw Preview Visualize JSON


```
[
    {
        "jobName453": "j1011",
        "partID453": "2134",
        "qty": 62
    }
]
```

Get job by jobName and partID:

GET http://ec2-3-85-114-115.compute-1.amazonaws.com:8080/api/jobs453/getJob/j1008/2298

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 133 ms

Pretty Raw Preview Visualize JSON


```
[
    {
        "jobName453": "j1008",
        "partID453": "2298",
        "qty": 78
    }
]
```


HTTP POST

The jobs table before the POST request:

Result Grid Filter Rows:

jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
NULL	NULL	NULL

POST http://ec2-3-85-114-115.compute-1.amazonaws.com:8080/api/jobs453/addJob

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON


```
{
    "jobName": "j1999",
    "partID": 4444,
    "qty": 22
}
```


Body Cookies Headers (6) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON


```
{
    "jobName453": "j1999",
    "partID453": 4444,
    "qty453": 22
}
```

The jobs table after the POST request. The tuple {“j1999”, 4444, 22} is added.

Result Grid | Filter Rows:

jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
i1999	4444	22
NULL	NULL	NULL

HTTP PUT

The jobs table before the PUT request:

Result Grid | Filter Rows:

jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
i1999	4444	22
NULL	NULL	NULL

PUT request to update {“j1999”, 4444, 22} to {“j1999”, 4444, 6}

PUT http://ec2-3-85-114-115.compute-1.amazonaws.com:8080/api/jobs453/updateJob

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "jobName": "j1999",
    "partID": 4444,
    "qty": 6
}
```

Body | Cookies | Headers (6) | Test Results | Status: 200 OK

Pretty | Raw | Preview | Visualize | JSON

```
{
    "jobName": "j1999",
    "partID": 4444,
    "qty": 6
}
```

The jobs table after the update:

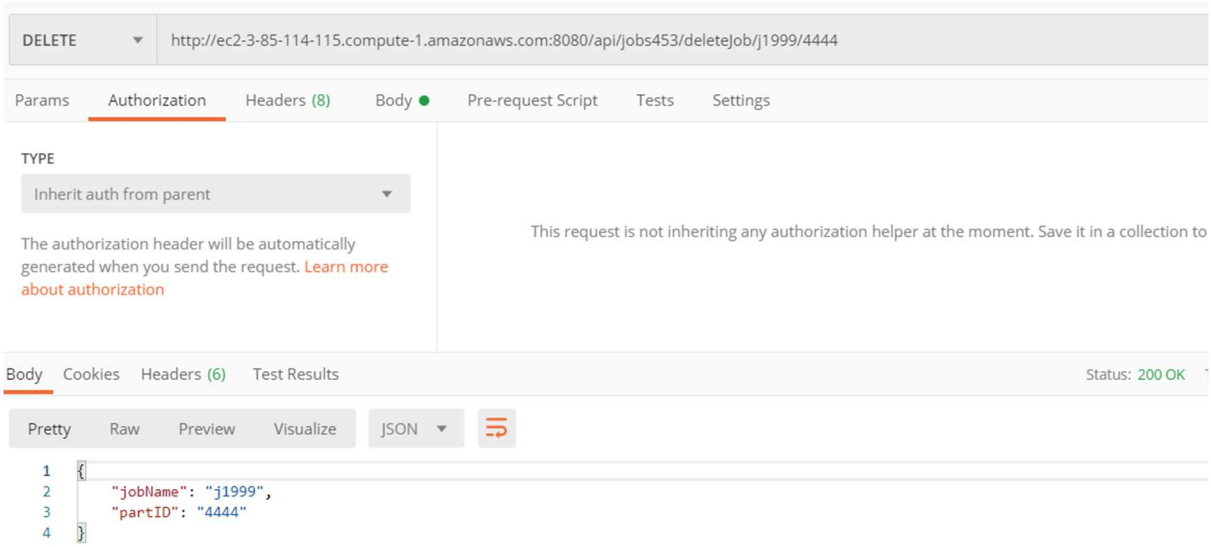
jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
i1999	4444	6
NULL	NULL	NULL

HTTP DELETE

The jobs table before the DELETE request:

jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
i1999	4444	6
NULL	NULL	NULL

DELETE request to remove the job with the primary key: (“j1999”, 4444). Returns the deleted record’s primary key in the response.



The jobs table after deleting {“j1999”, 4444, 6}:

jobName453	partID453	qty
i1007	2222	34
i1008	2298	78
i1011	2134	62
NULL	NULL	NULL

A Better Approach with Docker Hub

Here, we have transferred our project files from the local machine to the EC2 instance and then containerized the application on EC2. However, the whole purpose of docker is making the portability of applications/software across different machines easy, without the hassle of running into environment related issues. So, in this section, a docker image of the web application is created on my local machine and then pushed to my repository on the docker hub. The image is then pulled from the repository and deployed as a container on the cloud (EC2).

Building a docker image locally:

```
Abhinav@LAPTOP-DLH7D2J0 MINGW64 /d/Certification Courses/Udemy/MApp_web_SE/Docker/adv cloud assgn1/expressapp
$ docker build -t abhinav7896/expressapp_windows:1.0 .
Sending build context to Docker daemon   6.68MB
Step 1/6 : FROM node:14
 ---> e00884c768d7
Step 2/6 : WORKDIR /express_web_app
 ---> Running in fa3fa29a418d
Removing intermediate container fa3fa29a418d
 ---> 0167b19bebef
Step 3/6 : COPY ./ /express_web_app
 ---> 771f57db6cc3
Step 4/6 : RUN npm install
 ---> Running in 27505d180f24
npm WARN expressapp@1.0.0 No description
npm WARN expressapp@1.0.0 No repository field.

audited 87 packages in 1.542s
found 0 vulnerabilities

Removing intermediate container 27505d180f24
 ---> d789e511dffb
Step 5/6 : CMD npm start
 ---> Running in eb5a03218b63
Removing intermediate container eb5a03218b63
 ---> 4d763845ddb3
Step 6/6 : EXPOSE 3000
 ---> Running in 4af49b6c2778
Removing intermediate container 4af49b6c2778
 ---> 41d72a787aa5
Successfully built 41d72a787aa5
Successfully tagged abhinav7896/expressapp_windows:1.0
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double
check and reset permissions for sensitive files and directories.

Abhinav@LAPTOP-DLH7D2J0 MINGW64 /d/Certification Courses/Udemy/MApp_web_SE/Docker/adv cloud assgn1/expressapp
$
```

The local image is named as *abhinav7896/expressapp_windows:1.0*

Pushing the image to my docker repository:

Login to the docker hub:

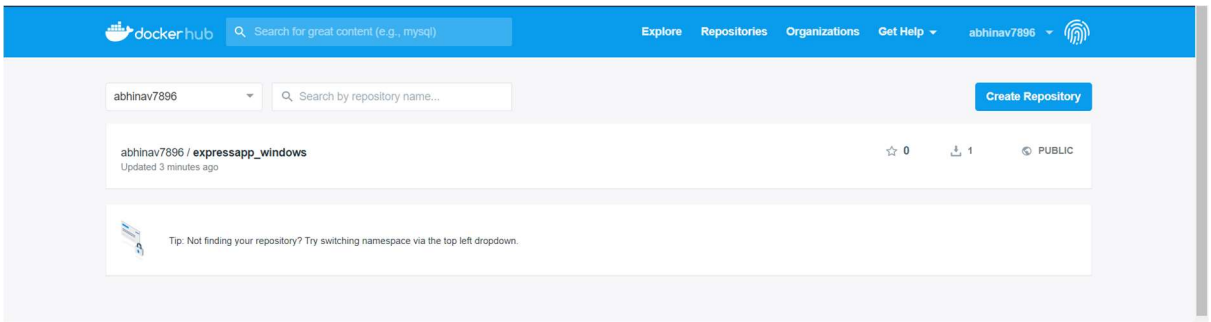
```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: abhinav7896
Password:
WARNING! Your password will be stored unencrypted in C:\Users\Abhinav\.docker\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

Abhinav@LAPTOP-DLH7D2J0 MINGW64 /d/Certification Courses/Udemy/MApp_web_SE/Docker/adv cloud assgn1/expressapp
$
```

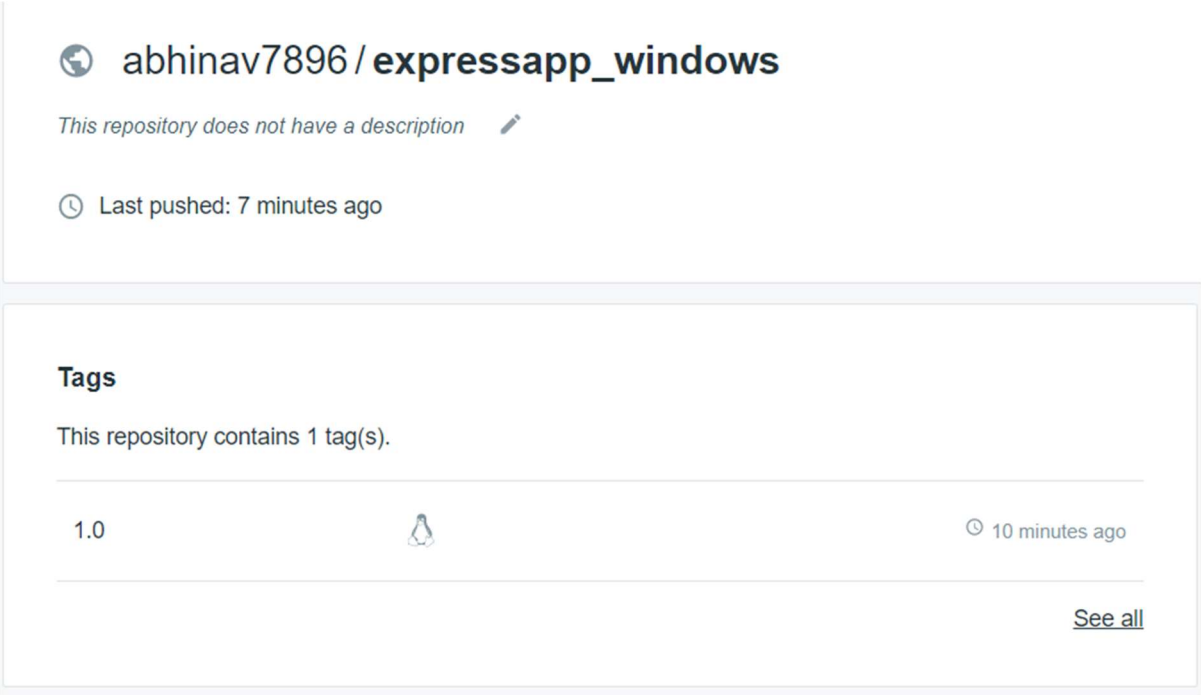
Push the image with *docker push <image-name>* :

```
Abhinav@LAPTOP-DLH7D2J0 MINGW64 /d/Software/Docker Toolbox
$ docker push abhinav7896/expressapp_windows:1.0
The push refers to repository [docker.io/abhinav7896/expressapp_windows]
6fa794206871: Pushed
59193bb2ab25: Pushed
471bdd9f238c: Pushed
5460b9303c9c: Mounted from library/node
8929fcddb211: Mounted from library/node
9d5f185a5034: Mounted from library/node
5aea01ea0a0f: Mounted from library/node
05f4935ad90a: Mounted from library/node
c96f2308ab16: Mounted from library/node
38c2f9ead82d: Mounted from library/node
0dabcc98eeef: Mounted from library/node
6885f9305c0a: Mounted from library/node
1.0: digest: sha256:7f38d10e69a847e9946a269ee43b7cdab71e808959004822569f4ed18f62c629 size: 2840
```

(Note the format of image name before pushing: *dockerID/image_name:tag)*

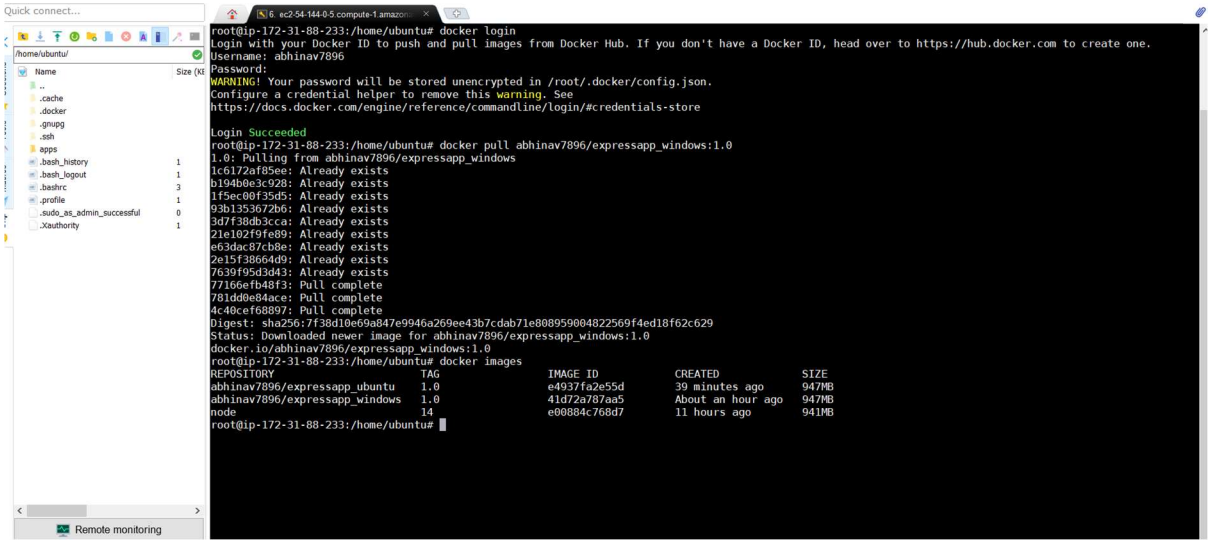
Image added to the repository in the docker hub:





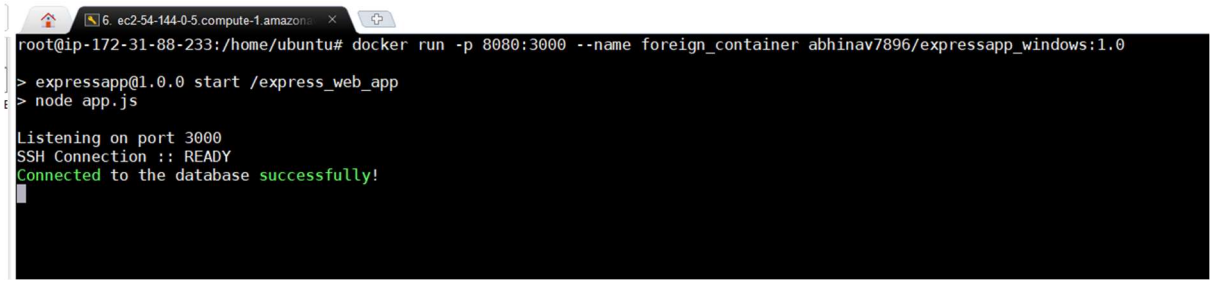
Pull the image from the repo, on EC2:

Login and use *docker pull <image-name>* to pull the image.



As shown in the figure, the image *abhinav7896/expressapp_windows:1.0* is now present in EC2

Deploy a container from the pulled image:



Thus, the web application is developed, containerized and deployed on cloud infrastructure. Also, the docker image of the app can be pulled from the repository by other authorized developers and worked upon without going through the hassle of environment setup, configuration issues, and other undesirable blockers.

References:

[1]"NodeJS / Express: Using SSH to access MySQL remotely", *Medium*, 2020. [Online]. Available: https://medium.com/@devontem/nodejs-express-using-ssh-to-access-mysql-remotely-60372832dd08. [Accessed: 16- May- 2020].

[2]"Orientation and setup", *Docker Documentation*, 2020. [Online]. Available: https://docs.docker.com/get-started/. [Accessed: 16- May- 2020].

[3]"Docker and Kubernetes: The Complete Guide", Udemy, 2020. [Online]. Available: https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide/. [Accessed: 16- May- 2020].