Assignment-3

Name: Yash Shah
Banner ID: B00841980

Objective:

The primary objective of this project is to understand the working of Docker environment and implement the understanding to deploy a light-weight web-application on AWS using Docker containers.

Job 1:

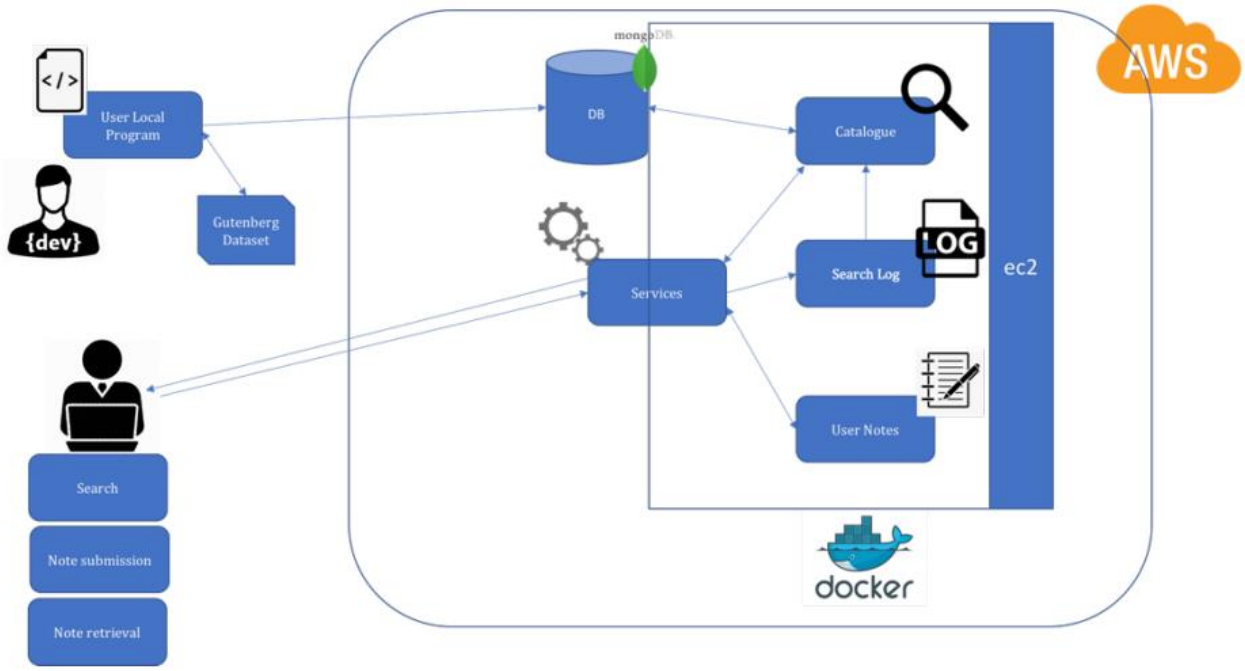

Fig 1. The architecture of the web-application Source: CSCI 5409 assignment-3 (2020)

The given architecture covers the detailed layout of a web-application hosted on cloud. The defined technology for containerizing the services is Docker, that for hosting the web-application is Amazon Web Service (AWS), and that for storing data is MongoDB.

Developers often encounter this problem of an application running on their machine and not running on another developer's machine. This issue was the prime motivation for the creation of

Docker [1]. Docker creates an image of the application and this image is then executed to form a container. A Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application, namely code, runtime, system tools, system libraries and settings. This image becomes a container at runtime. In simple words, Docker containers are an abstraction at the application layer that packages code and dependencies together [2]. Moreover, containers isolate software from its environment and ensure that it works uniformly despite any differences. A developer will have to write a Dockerfile which stores the instructions about how a Docker image is to be built. Once the image is built, the developer can share it with fellow developers and all they will have to do is run this image to create a container on their machines and use the application in the same way.

AWS is an on-demand cloud computing platform where developers can host their applications. One of the most popular service offered by AWS is the Amazon Elastic Compute Cloud (EC2); using this, developers can use a virtual computer (by paying a minimal amount or for free) to deploy their application, which will be available all the time through Internet [3].

According to the architecture given above, a developer (dev) is to write a local program that performs some computation (extraction) on the Gutenberg Dataset. Once it is computed, the local program would load the acquired results to a MongoDB database. Next, a user is allowed to request any three of these functionalities: search, note submission, and note retrieval; these requests will be received from a normal user machine. The user request (API call) will hit the Services component which will redirect the request further to another components (microservices), namely Catalogue, Search Log, and User Notes. Catalogue is a search microservice which receive the API call from Services, and depending on the request connects to the MongoDB database to get an output, and sends it back. Search Log is a log-storing microservice which receives a request from the Services component and then sends a request to Catalogue. User Notes is a comment-storing microservice which receives/sends request/result to the Services component. All of these services: Catalogue, Search Log, User Notes, and Services are inside a container. Another container holds the MongoDB database and both these containers are inside a single Docker. This Docker is deployed on AWS EC2 and the user requests are sent to this EC2 instance.

I have used a python script (Extractor.ipynb) to extract the Titles and Authors from the Gutenberg Database files. These files were downloaded locally and then the extractor script was ran. The data loader script (MongoDB Data Loader.ipynb) is used to load data on EC2 instance after implementation of Dockerfile on EC2. I have used React to create my UI and have used Node to develop my services. These services are containerized and then its Dockerfile (docker compose file) is used to build the application on EC2. The docker compose file consists of the services and the MongoDB image too.

Job2:

All the database files are downloaded on the local machine and are attached with this report, The python script (Extractor.ipynb) is used to extract the Titles and Authors from the Gutenberg Database files, and they are stored in a csv file (Books.csv). A delay of five minutes (300 seconds) is introduced in the script itself. The start time and end time of file processing are stored in a csv file (Timings). Both of these files are attached with this report. Following that, these csv files are loaded to the local MongoDB database as shown in fig. 2.1 and fig, 2.2. Later this database dumps would be uploaded to the MongoDB running on EC2 instance.

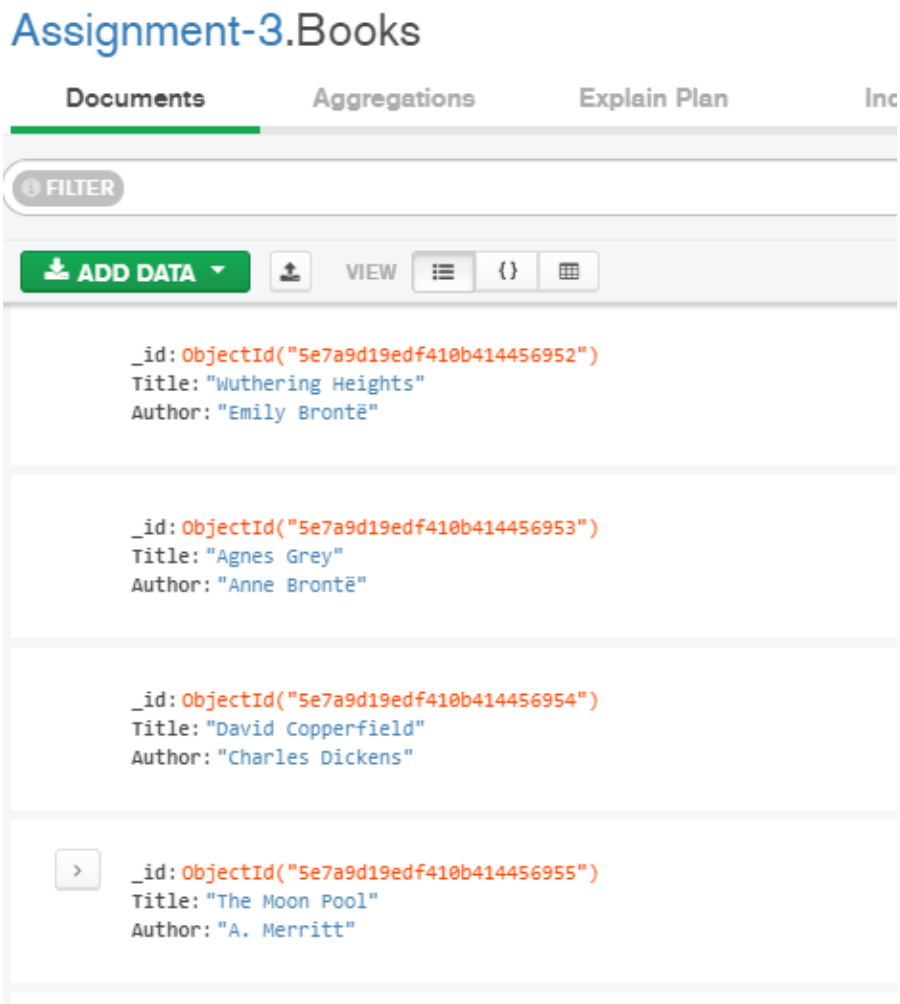

Figure 2.1 Books data collection on local MongoDB

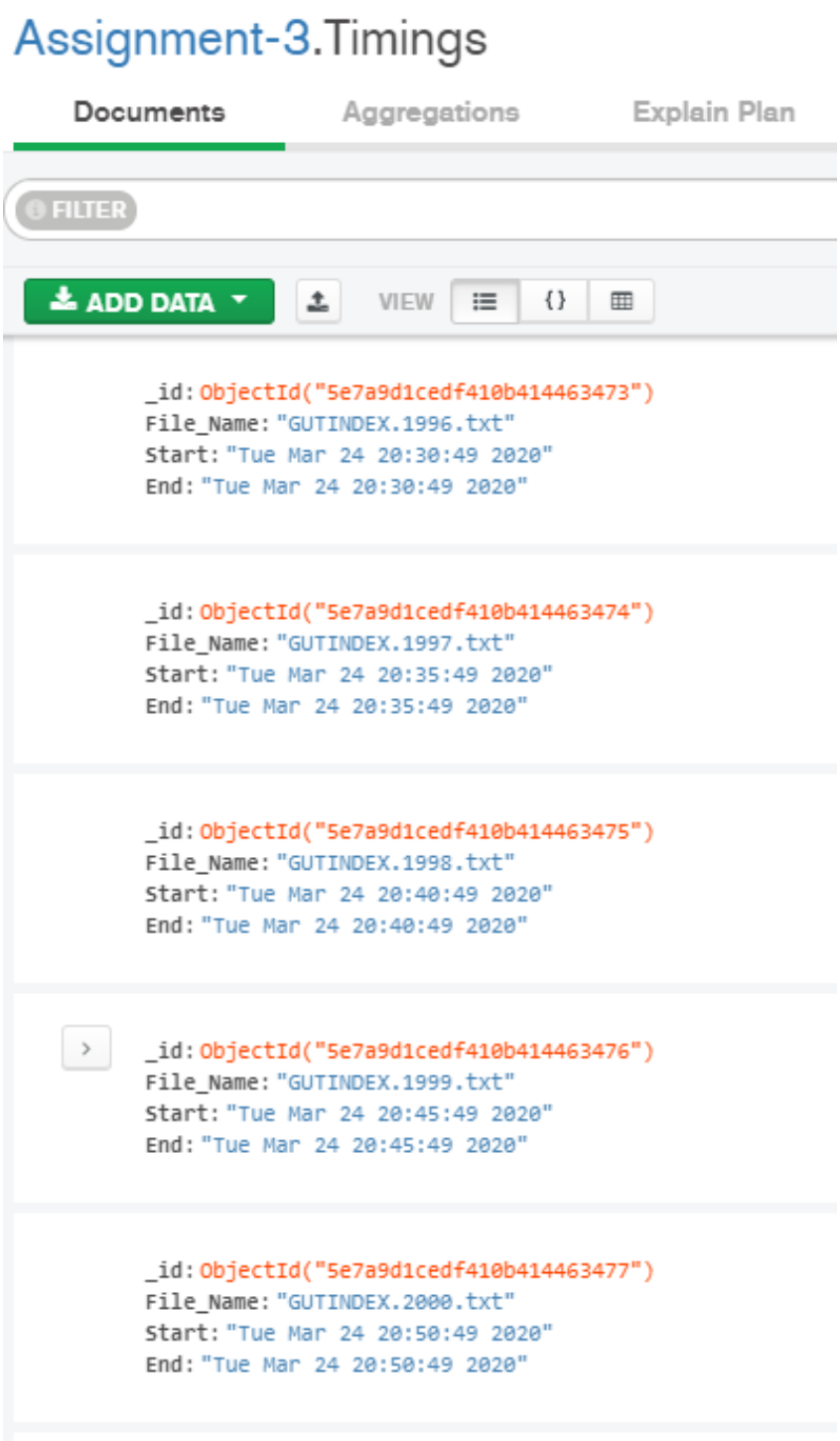

Fig 2.2 Timings data collection on local machine

Job 3:

I have used React and Node to build my web-application. The screenshots of the websites are as shown below:

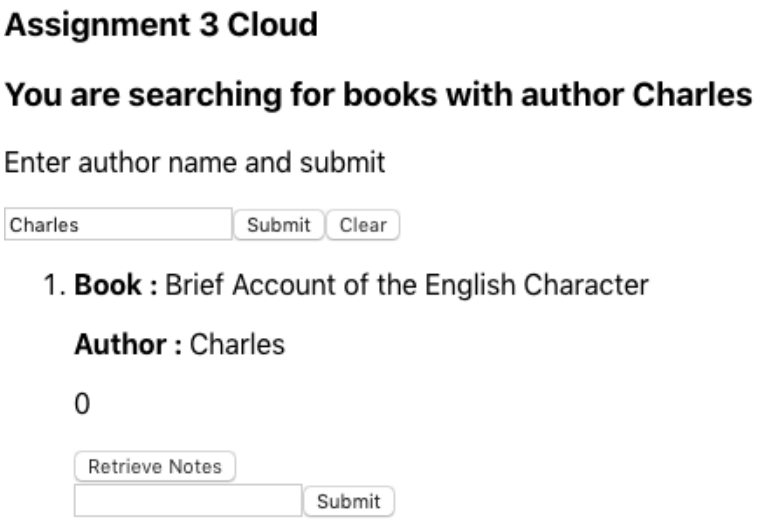

Fig 3.1 UI of the application showing the search result after entering the keyword Charles

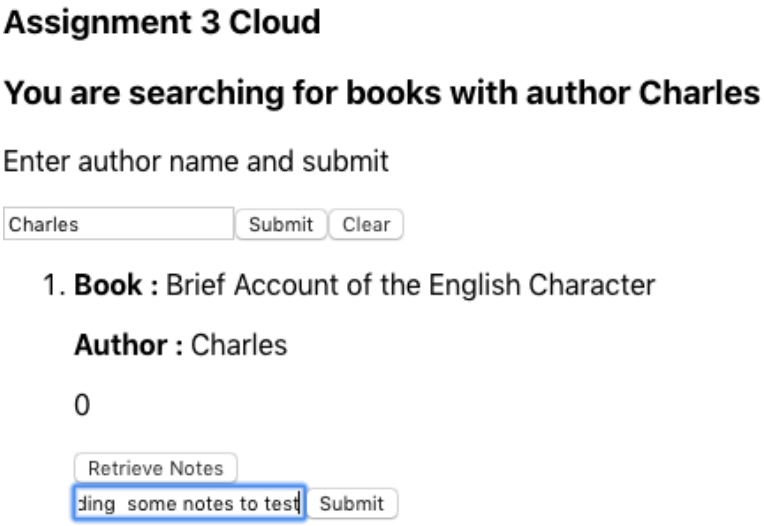

Fig 3.2. UI of the application showing the entering of a note for a particular book

Assignment 3 Cloud

You are searching for books with author Charles

Enter author name and submit

Charles Submit Clear

1. **Book :** Brief Account of the English Character

   Author : Charles

   Notes:
   Great book
   adding a note
   Adding some notes to test
   Retrieve Notes
   Submit

Fig 3.3 UI of the application showing the retrieved notes for a particular book

Assignment 3 Cloud

You are searching for books with author Charles Dickens

Enter author name and submit

Charles Dickens Submit Clear

1. **Book :** David Copperfield

   Author : Charles Dickens

   0

   Retrieve Notes
   Submit

2. **Book :** Oliver Twist

   Author : Charles Dickens

   1

   Retrieve Notes
   Submit

3. **Book :** The Old Curiosity Shop

   Author : Charles Dickens

   2

   Retrieve Notes
   Submit

4. **Book :** A Child's History of England

   Author : Charles Dickens

Fig 3.4 UI of the application showing multiple results for a particular keyword Charles Dickens

Job 4:

All the files are containerised and the Docker file for them is attached with this report. The notes submitted for a particular book-author pair is stored in JSON format to the Books data collection itself (fig 4.1). Also, whenever a query is hit and the Catalogue service returns a particular result to the user, a log file is updated to store the number of times a particular keyword is searched (fig. 4.2).

Key	Value	Type
(1) ObjectId("5e7bbeeb99d7cf8bf2d08530")	{ 4 fields }	Object
_id	ObjectId("5e7bbeeb99d7cf8bf2d08530")	ObjectId
Title	Brief Account of the English Character	String
Author	Charles	String
Notes	[3 elements]	Array
[0]	Great book	String
[1]	adding a note	String
[2]	Adding some notes to test	String

Fig 4.1. The notes for a particular book in a database

```
2020-03-25T21:34:02.925Z:  [32minfo [39m : Author: Charles , hits: 1
2020-03-25T21:34:04.634Z:  [32minfo [39m : Author: Charles , hits: 2
2020-03-25T21:34:07.888Z:  [32minfo [39m : Author: Charles , hits: 3
```

Fig 4.2. The log file storing the keyword and the hits

Job 5:

An AWS EC2 instance made on Linux 2 AMI is used to deploy the application. Software of Docker compose, Git and Node are installed on the instance as the first step. Later using Git, the docker file in my local machine is uploaded to my Git repository and then using Git again, that same file is pulled in the EC2 instance. Later, the docker file is executed and then the application and Mongo starts running on the instance (fig 5.1 and 5.2). Once it is running, the application starts running on the instance IP address as shown in fig 5.3.

```
Run a command in a running container
[ec2-user@ip-172-31-37-126 cloud_assignment]$ docker exec -it c6f20ef5cef7 bash
Error response from daemon: Container c6f20ef5cef712a987e7087667182c70020dc9e90fd5e38abc877629fb69c52e is not running
[ec2-user@ip-172-31-37-126 cloud_assignment]$ docker-compose up -d
Starting mongo ... done
Starting backend ... done
[ec2-user@ip-172-31-37-126 cloud_assignment]$ docker ps -a
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS                      NAMES
c6f20ef5cef7        cloud_assignment_backend   "docker-entrypoint.s…"   37 minutes ago      Up 8 seconds        0.0.0.0:3001->3001/tcp     backend
1bb4c7e0a734        mongo                      "docker-entrypoint.s…"   37 minutes ago      Up 8 seconds        0.0.0.0:27017->27017/tcp   mongo
[ec2-user@ip-172-31-37-126 cloud_assignment]$ docker exec -it c6f20ef5cef7 bash
root@c6f20ef5cef7:/usr/src/app# ls
Dockerfile  db  docker-compose.yml  frontend  log.log  logger  node_modules  package-lock.json  package.json  server.js
root@c6f20ef5cef7:/usr/src/app# cd frontend
root@c6f20ef5cef7:/usr/src/app/frontend# npm run build

> frontend@0.1.0 build /usr/src/app/frontend
> react-scripts build
```

Fig 5.1. MongoDB and my application (backend) running on the EC2 instance after the execution of docker file

```
Building backend
Step 1/7 : FROM node:10
 ---> aa6432763c11
Step 2/7 : WORKDIR /usr/src/app
 ---> Using cache
 ---> 00eaa6f9aa39
Step 3/7 : COPY package*.json ./
 ---> Using cache
 ---> 98b9e3918a0a
Step 4/7 : RUN npm install
 ---> Using cache
 ---> d3002ffd9b15
Step 5/7 : COPY . .
 ---> Using cache
 ---> 7b5a52f21044
Step 6/7 : EXPOSE 3001
 ---> Using cache
 ---> 03da18616103
Step 7/7 : CMD [ "node", "server.js" ]
 ---> Using cache
 ---> c550717ce87a
Successfully built c550717ce87a
Successfully tagged cloud_assignment-3_backend:latest
Starting mongo ... done                                                    Creating backend ... done

C:\Users\admin\Downloads\Cloud Computing\Cloud_Assignment-3>docker-compose up
Starting mongo ... done                                                    backend is up-to-date
Attaching to mongo, backend
backend    | Express server running on port 3001
mongo      | 2020-03-26T00:51:00.652+0000 I  CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
mongo      | 2020-03-26T00:51:00.666+0000 I  CONTROL  [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=4868b210dc73
mongo      | 2020-03-26T00:51:00.667+0000 I  CONTROL  [initandlisten] db version v4.2.3
mongo      | 2020-03-26T00:51:00.668+0000 I  CONTROL  [initandlisten] git version: 6874650b362138df74be53d366bbefc321ea32d4
mongo      | 2020-03-26T00:51:00.669+0000 I  CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.1.1  11 Sep 2018
mongo      | 2020-03-26T00:51:00.671+0000 I  CONTROL  [initandlisten] allocator: tcmalloc
mongo      | 2020-03-26T00:51:00.672+0000 I  CONTROL  [initandlisten] modules: none
mongo      | 2020-03-26T00:51:00.675+0000 I  CONTROL  [initandlisten] build environment:
mongo      | 2020-03-26T00:51:00.675+0000 I  CONTROL  [initandlisten]     distmod: ubuntu1804
mongo      | 2020-03-26T00:51:00.675+0000 I  CONTROL  [initandlisten]     distarch: x86_64
mongo      | 2020-03-26T00:51:00.675+0000 I  CONTROL  [initandlisten]     target_arch: x86_64
mongo      | 2020-03-26T00:51:00.675+0000 I  CONTROL  [initandlisten] options: { net: { bindIp: "*" } }
mongo      | 2020-03-26T00:51:00.685+0000 I  STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=256M,cache_overflow=(file_max=0M),session_ma
ax=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000,close_scan_int
tics_log=(wait=0),verbose=[recovery_progress,checkpoint_progress],
mongo      | 2020-03-26T00:51:04.251+0000 E  STORAGE  [initandlisten] WiredTiger error (17) [1585183864:251027][1:0x7f1a9d2b1b00], connection: __posix_open_
-open: open: File exists Raw: [1585183864:251027][1:0x7f1a9d2b1b00], connection: __posix_open_file, 667: /data/db/WiredTiger.wt: handle-open: open: File exi
mongo      | 2020-03-26T00:51:04.255+0000 E  STORAGE  [initandlisten] WiredTiger error (26) [1585183864:255076][1:0x7f1a9d2b1b00], connection: __posix_fs_re
a/db/WiredTiger.wt.2: file-rename: rename: Text file busy Raw: [1585183864:255076][1:0x7f1a9d2b1b00], connection: __posix_fs_rename, 241: /data/db/WiredTige
name: rename: Text file busy
mongo      | 2020-03-26T00:51:04.295+0000 E  STORAGE  [initandlisten] WiredTiger error (17) [1585183864:295132][1:0x7f1a9d2b1b00], connection: __posix_open_
-open: open: File exists Raw: [1585183864:295132][1:0x7f1a9d2b1b00], connection: __posix_open_file, 667: /data/db/WiredTiger.wt: handle-open: open: File exi
mongo      | 2020-03-26T00:51:04.299+0000 E  STORAGE  [initandlisten] WiredTiger error (26) [1585183864:299451][1:0x7f1a9d2b1b00], connection:   posix_fs_re
```

Fig 5.2 MongoDB and my application (backend) beginning to start on the EC2 instance and under docker file build

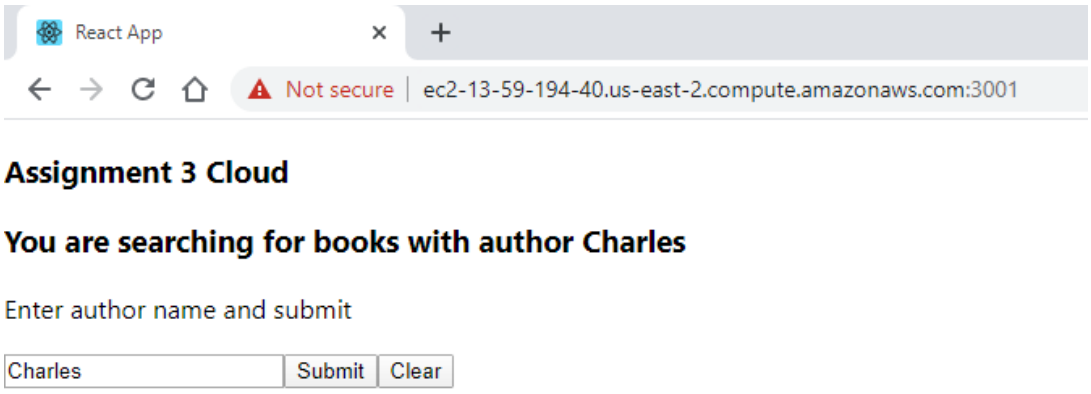

Fig 5.3 Web application running on the AWS IP address and on port 3001

Job 6:

The evidence of the test cases can be seen from the screenshot of the application displayed above. The test cases are as follows:

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Result	Actual Result	Pass/Fail & Comments
T01	Check launching of the application	1. Enter the URL in a broswer	No input	Application should launch with two button and a text field, with some header text	Result as expected	Pass
T02	Check the submit option	1. Enter the URL 2. Enter a name in the text field 3. Press submit	Charles	Application should display the result of the query; no output for invalid name and list of output for valid name	Result as expected	Pass
T03	Check the display of add note functionality	1. Enter the URL 2. Enter a name in the text field 3. Press submit	Charles	Application should display the result of the query, with a note addition option on successful query match	Result as expected	Pass
T04	Check the display of retrieve note functionality	1. Enter the URL 2. Enter a name in the text field 3. Press submit	Charles	Application should display the result of the query, with a note retrieval button on successful query match	Result as expected	Pass
T05	Check the add note button	1. Enter the URL 2. Enter a name in the text field 3. Press submit 4. Enter the note 5. Press Submit	Name: Charles Note: Great Book	Application should display the result of the query, with a note submission option. On pressing submit the note should be added the database	Result as expected	Pass
T06	Check the add note button for 2	1. Enter the URL 2. Enter a	Name: Charles	Application should display the result of the query, with a note	Result as expected	Pass

	or more notes addition	name in the text field 3. Press submit 4. Enter the note 5. Press Submit 6. Enter another note 7. Press submit	Note: adding a note Note: adding some note to test	submission option. On pressing submit the notes should be added the database		
T07	Check the note retrieval button	1. Enter the URL 2. Enter a name in the text field 3. Press submit 4. Press Retrieve note	Name: Charles	Application should display the result of the query, with all the notes submitted for that particular pair	Result as expected	Pass

References:

[1] “Why Docker? | Docker”, Docker 2020 [Online]. Available: https://www.docker.com/why-docker. Accessed on: 25 March, 2020.

[2] “What is a container? A standardized unit of software”, Docker 2020 [Online] .Available: https://www.docker.com/resources/what-container. Accessed on: 25 March, 2020.

[3] “Amazon Web Services”, Wikipedia | The free Encyclopedia 2020 [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Web_Services. Accessed on: 25 March, 2020.