**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:**

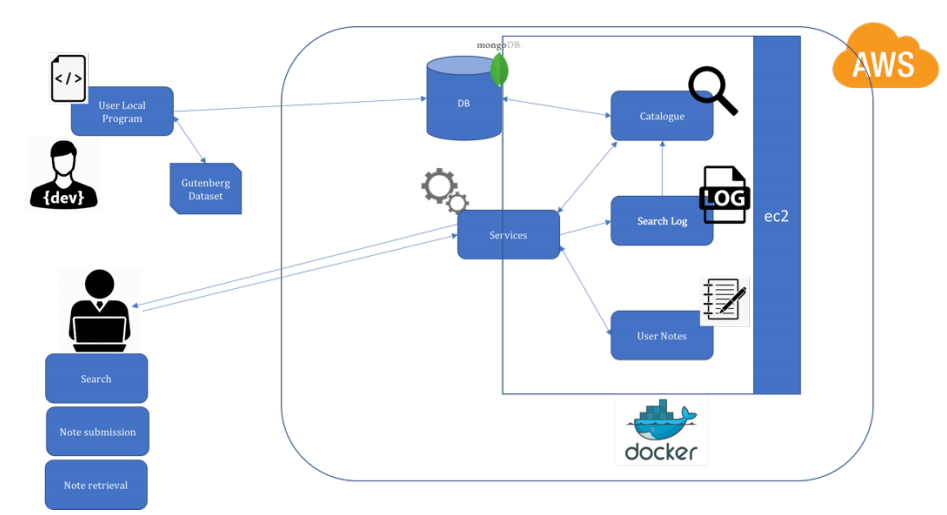


Fig 1. The architecture of the web-application Source: CSCI 5409 assignment-3 (2020)

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.

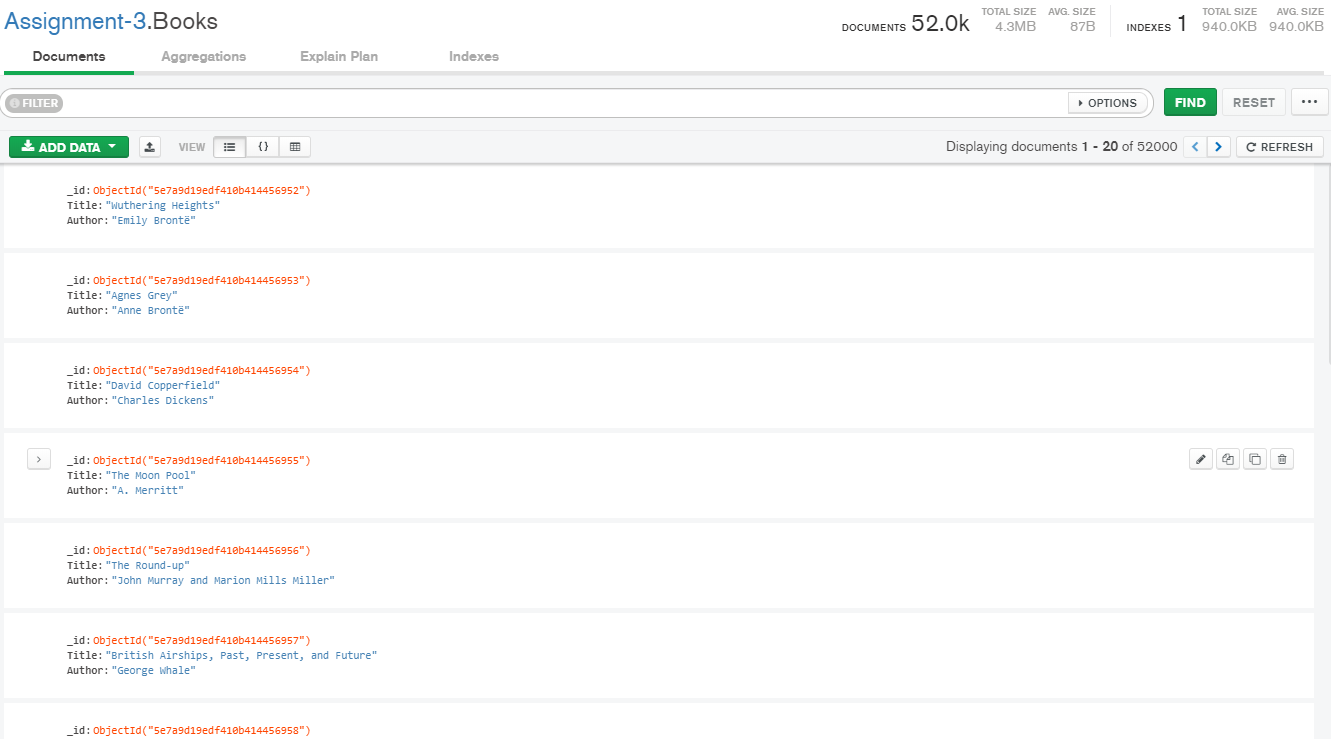


Figure 2.1 Books data collection on local MongoDB

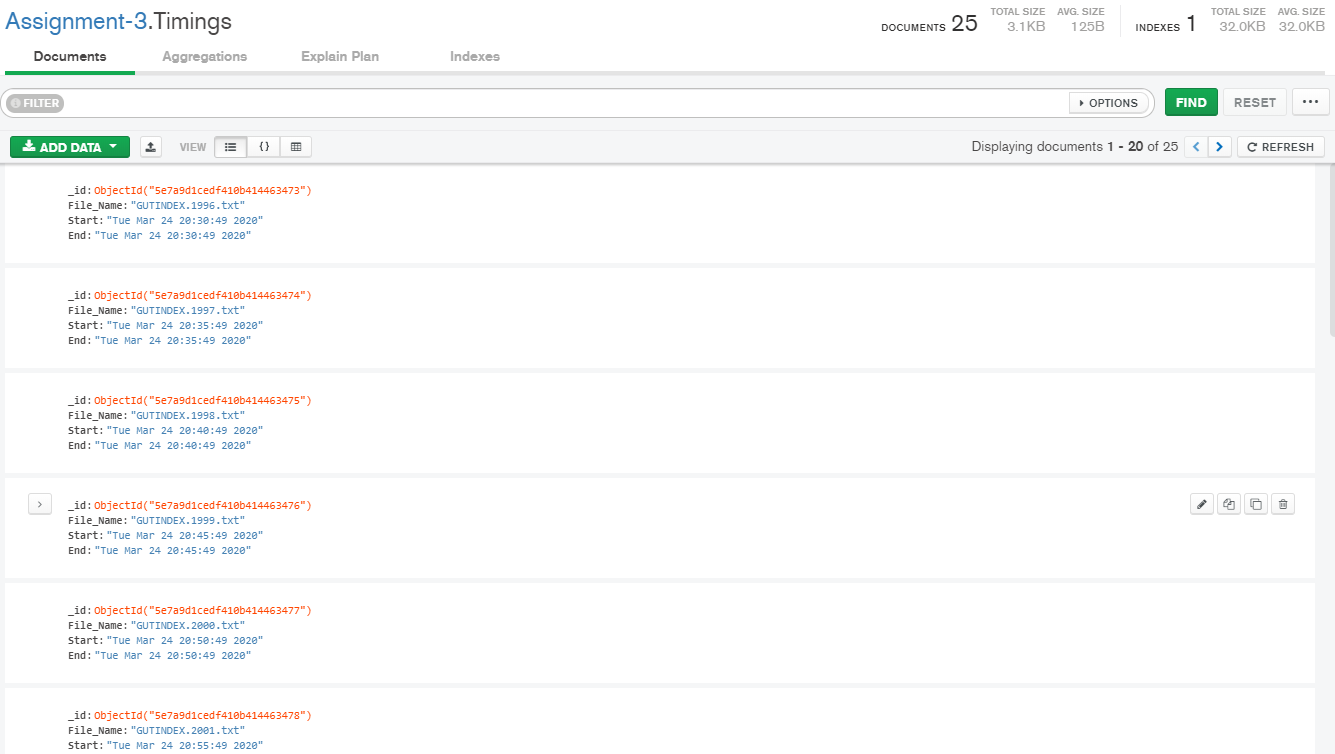


Fig 2.2 Timings data collection on local machine

**Job 3:**

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

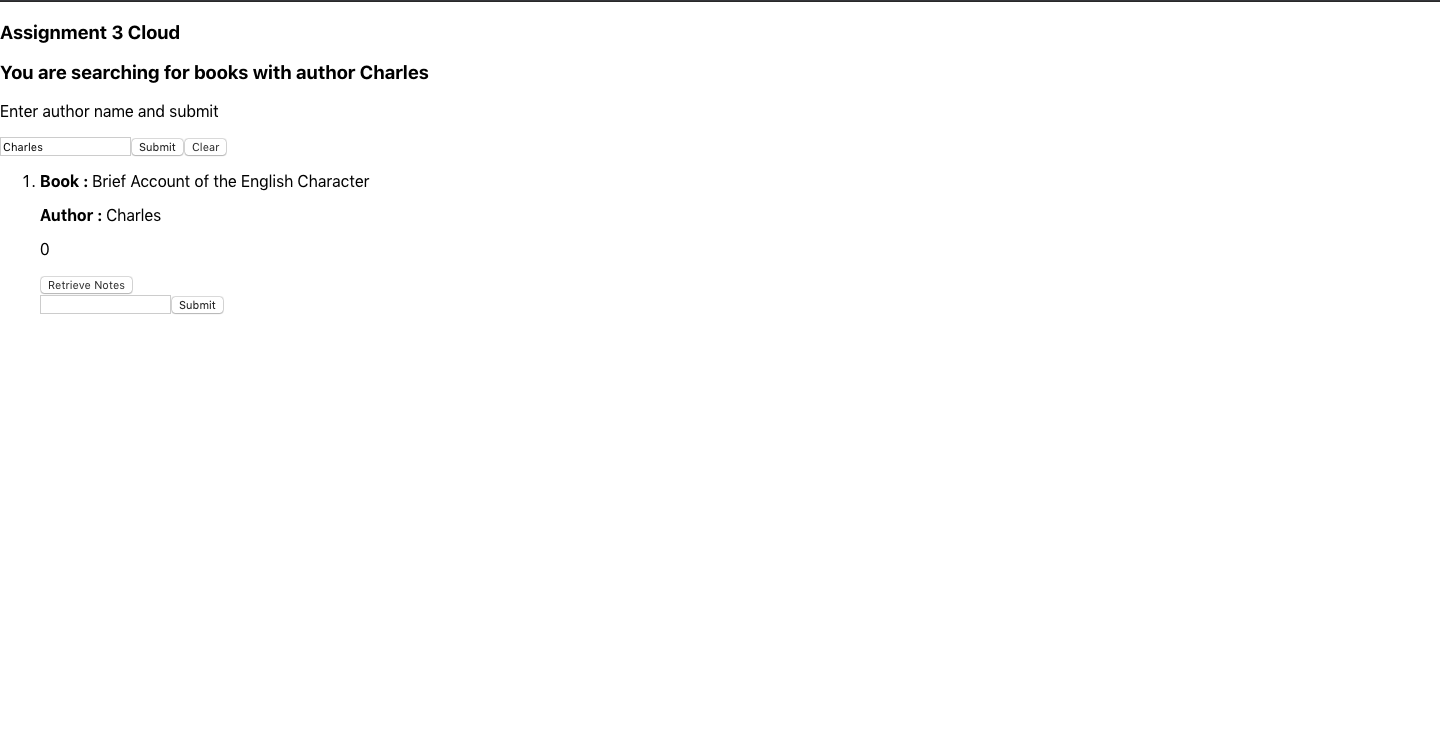


Fig 3.1 UI of the application showing the search result after entering the keyword Charles



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

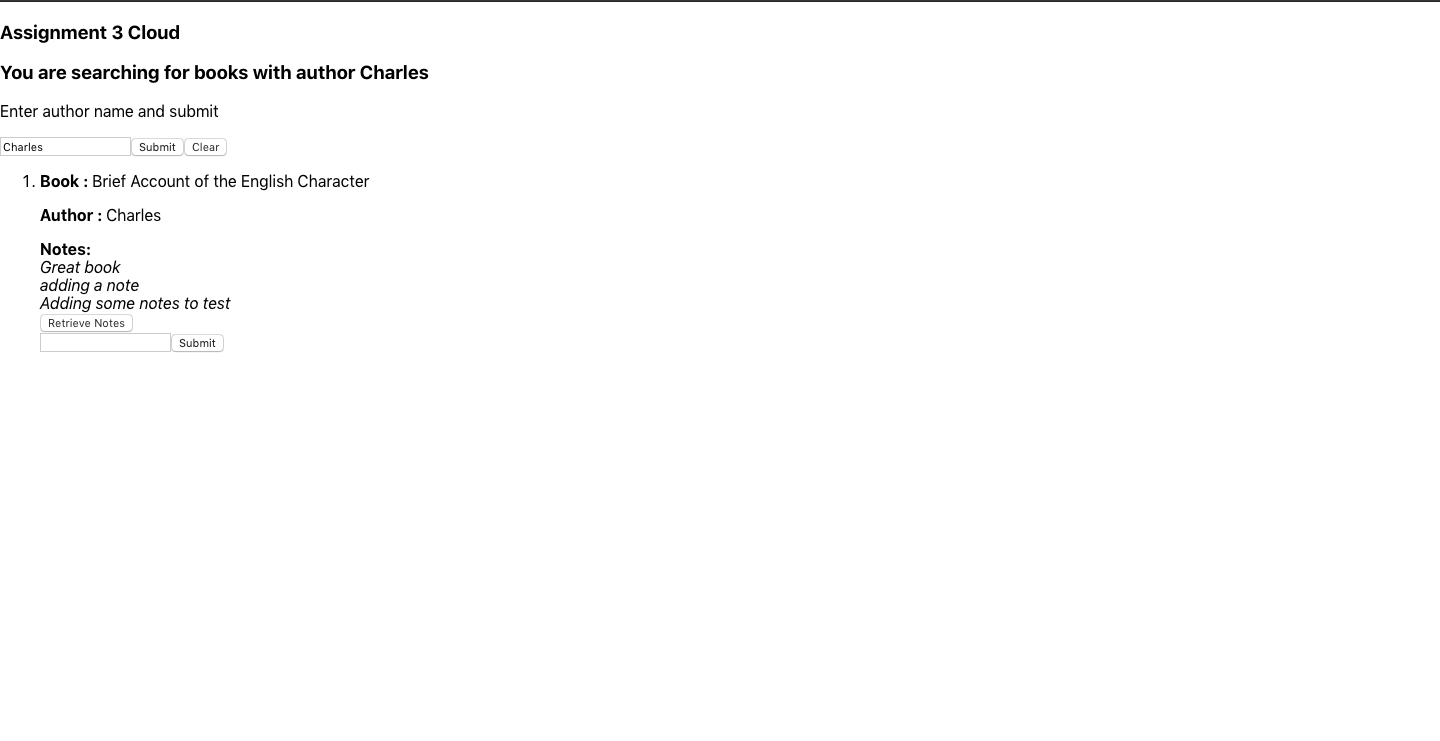


Fig 3.3 UI of the application showing the retrieved notes for a particular book

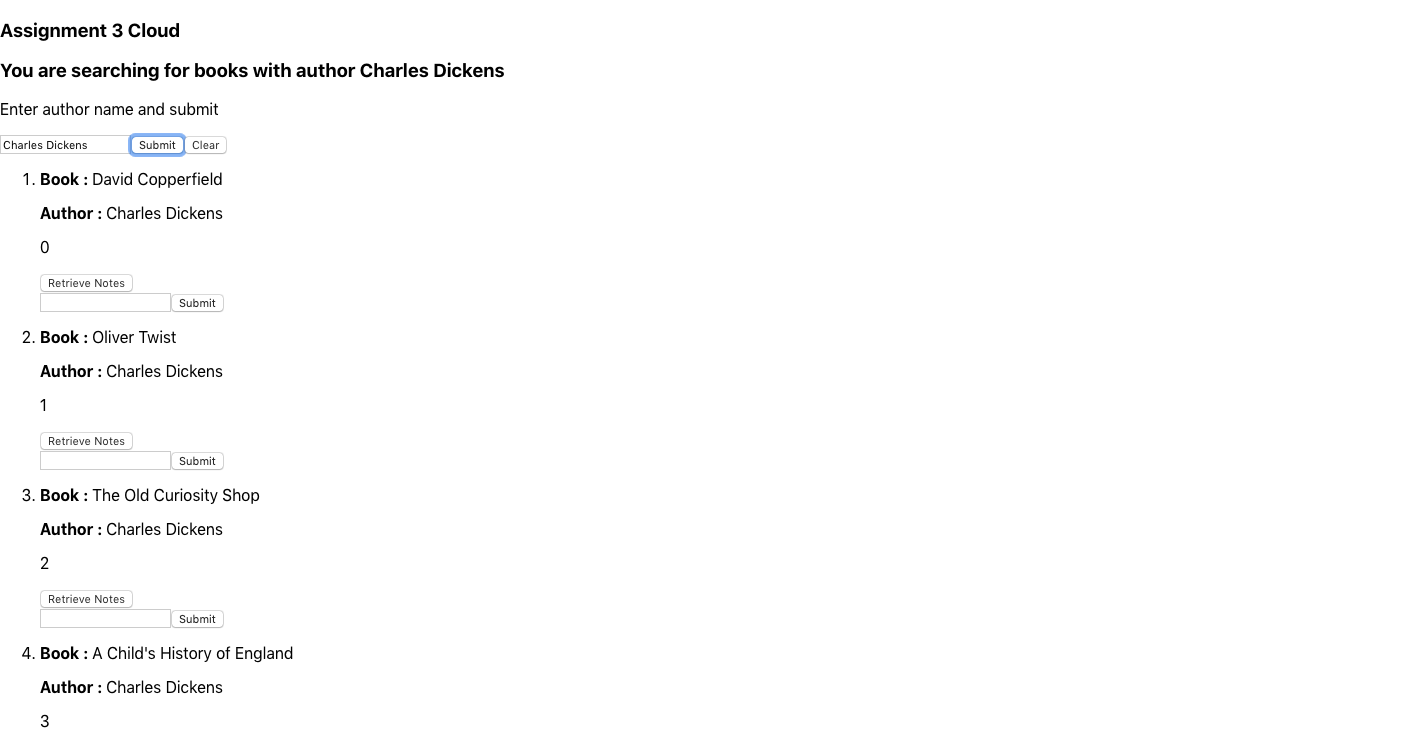


Fig 3.4 UI of the application showing multiple results for a particular keyword Charles Dickens

**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).

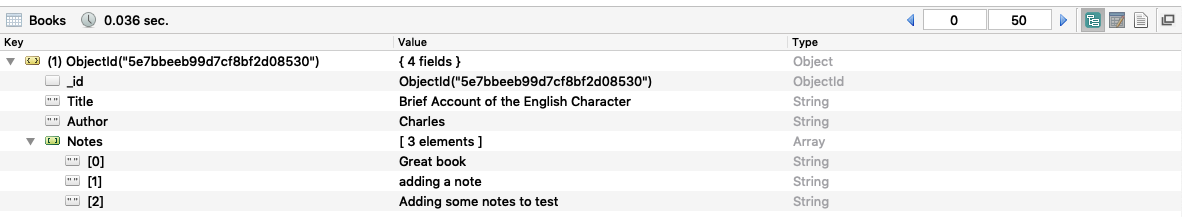


Fig 4.1. The notes for a particular book in a database

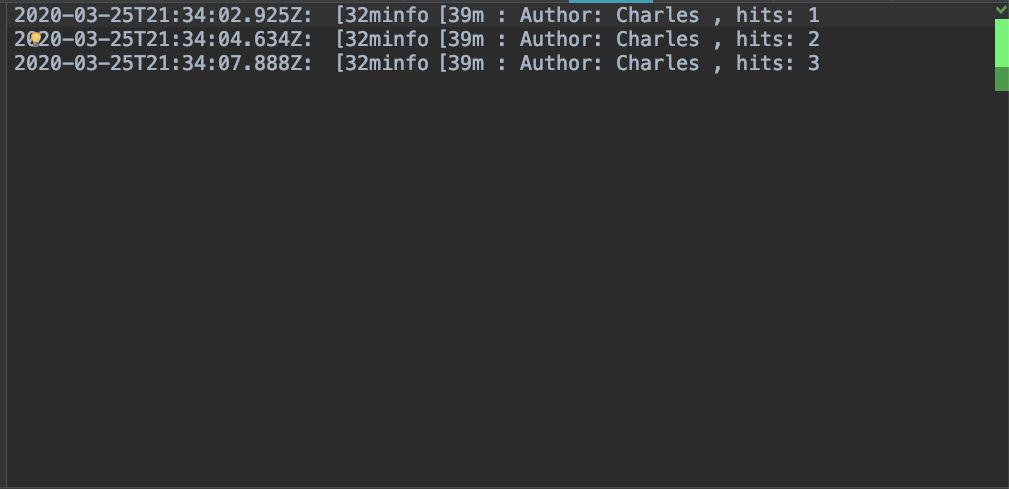


Fig 4.2. The log file storing the keyword and the hits

**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.

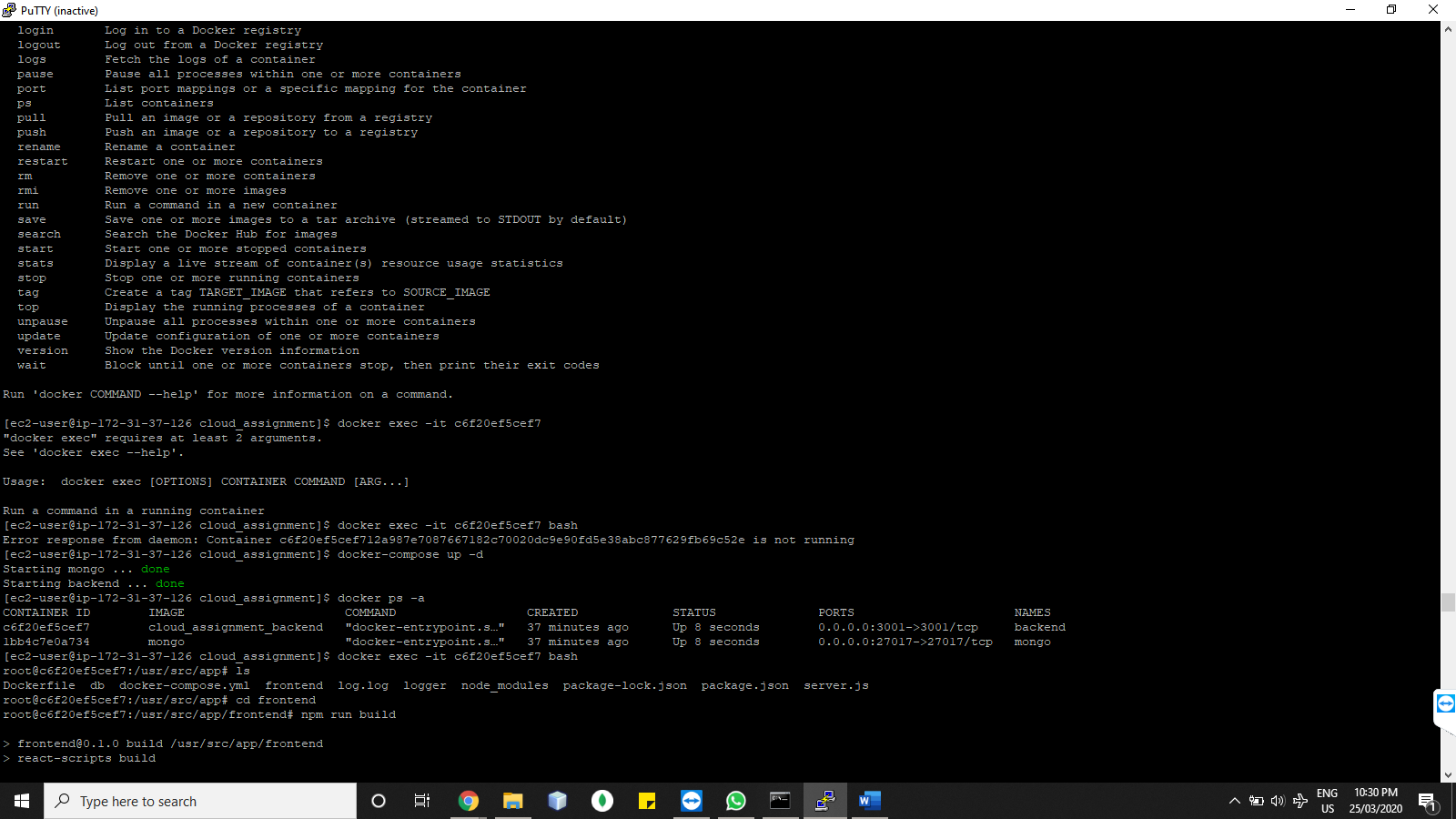


Fig 5.1. MongoDB and my application (backend) running on the EC2 instance after the execution of docker file

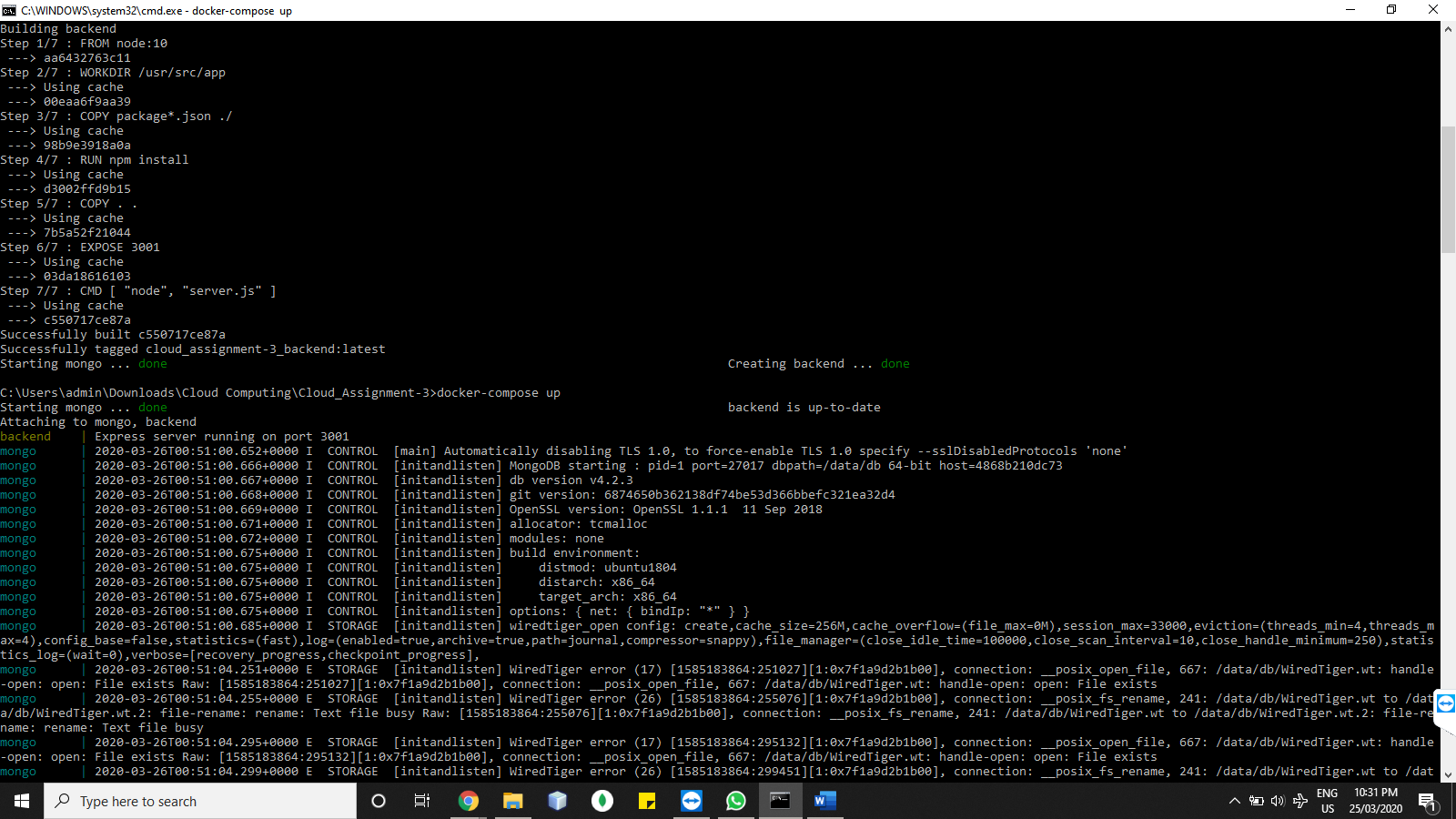


Fig 5.2 MongoDB and my application (backend) beginning to start on the EC2 instance and under docker file build

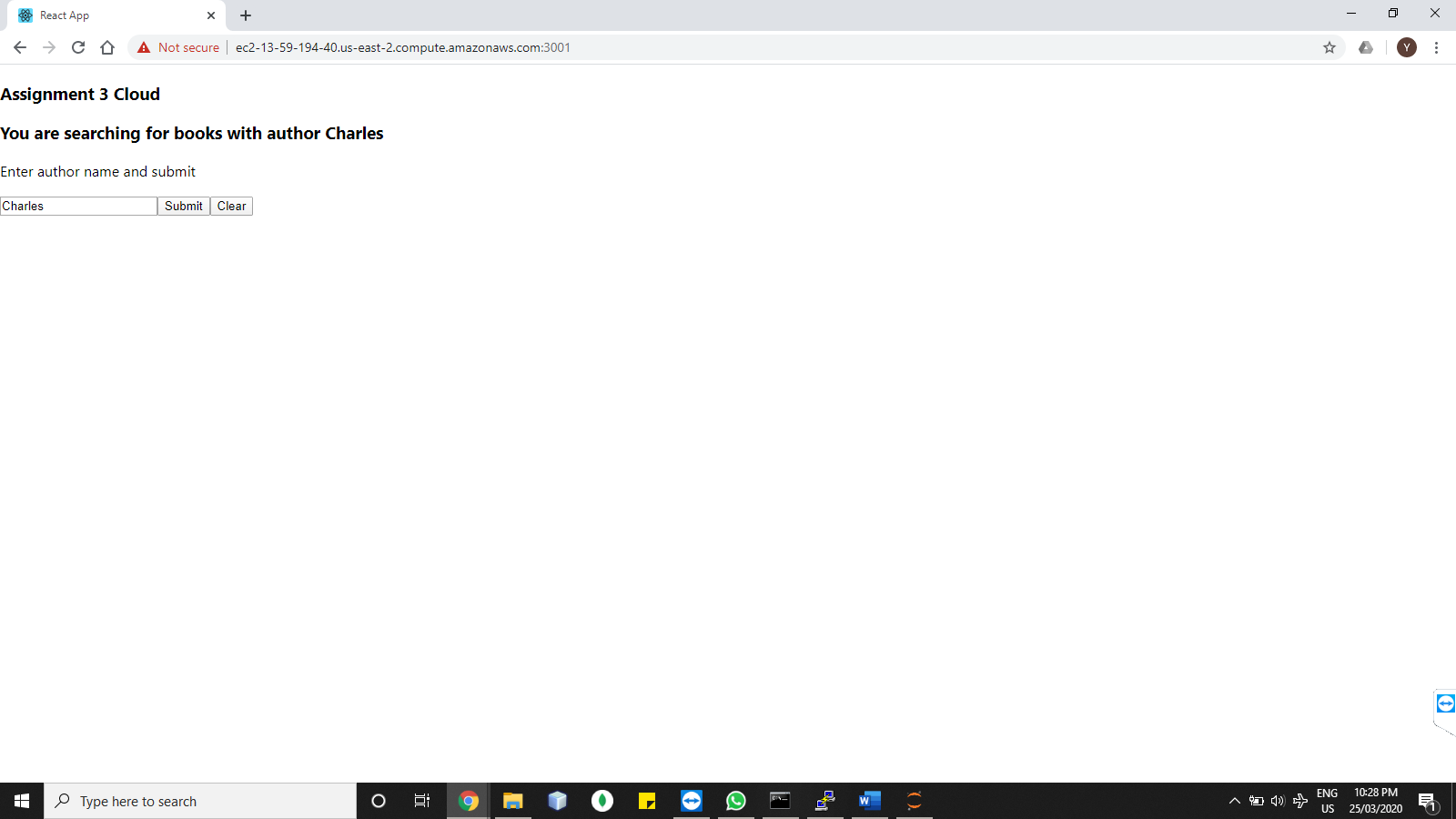


Fig 5.3 Web application running on the AWS IP address and on port 3001

**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 or more notes addition | 1. Enter the URL  2. Enter a name in the text field 3. Press submit  4. Enter the note  5. Press Submit  6. Enter another note  7. Press submit | Name: Charles  Note: adding a note  Note: adding some note to test | Application should display the result of the query, with a note submission option. On pressing submit the notes should be added the database | Result as expected | Pass |
| 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://ww w.docker.com/resources/what-container. Accessed on: 25 March, 2020.

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