Assignment 3 Report

CSCI 5409: Adv. Topics in Cloud Computing

Submitted by:

Aakash Patel (B00807065)

Report [1]:

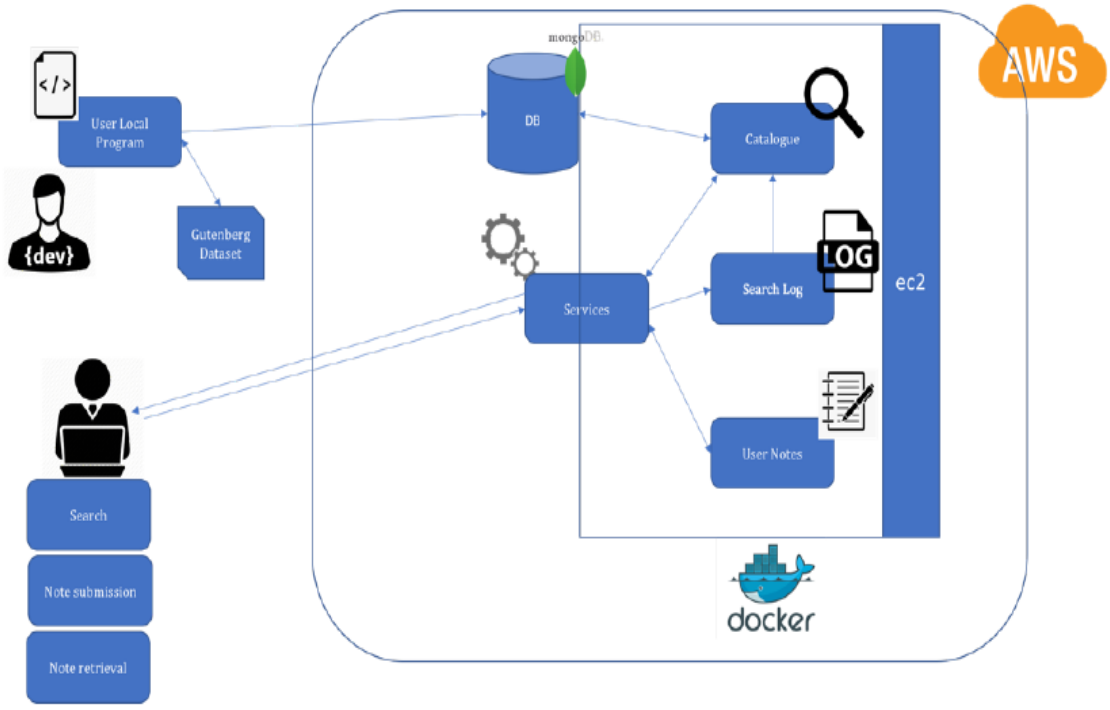

Fig: Book Title, Author Search Application (CSCI 5409 Model Project)

In this figure, cloud based microservice-based server application infrastructure is demonstrated. Here, Amazon Web Services (AWS) is the CSP (Cloud Service Provider), Cloud service owner whereas Customer plays the role of cloud consumer. The server logic of the application is hosted on the EC2 instance of Amazon cloud. There are different micro services which interoperated to process given request. The three services Catalogue, Search log and User Notes are the microservices which take commands from the master named Services. These microservices are containerized docker images of raw business logic segregated into a specific module for independence. These services interact among one another and refer to a common database in Mongo DB collection hosted centrally.

The Catalog microservice searches for the catalog from the application database. The Search log microservice searches for logs under the coordination and control of master. The Search Log looks for the logs under the database when the API call requesting logs is made. User Notes concerns with user notes. User is operating the application remotely through APIs calls made to the server. The microservices are containerized with Docker as it allows to package and run an application in a loosely isolated environment [1]. It also gives Isolation and security to allow run many containers

simultaneously on a given host. User local program is a extraction logic written by a developer to scrap book information from Gutenberg Offline catalogs and create database out of it. This database is then pushed to MongoDB cloud database hosted on AWS EC2 instance.

As a Customer shown in figure at the left bottom end, one has Search, Note Submission and Note Retrieval as business features hosted onto the cloud application. The arrows show the various internal API calls to facilitate the internal operations needed for the application to work.

Extraction Engine

I have created A3_Extraction repo to extract the books information from the Gutenberg offline catalogs. It's a simple Python program which uses BeautifulSoup to extract information from all files from 1996(earlier) to 2020. After processing each file, the book titles and respective author names are inserted into MongoDB[9] database on the AWS EC2 instance. The database is of the name books_db. The books collection stores the extracted information. The files are processed with a delay of 5 minutes. The processing timestamps i.e start time and end time for each file are also stored in process_log collection in the database.

Web Page

The web component part is just an .html file running on an external stylesheet and internal javascript functions. I used an online web template[7] [10] and pruned and adapted it to make the UI interface for the application. It makes HttpRequest to the Flask Server Backend to retrieve and display data appropriately. User just need to submit the search keywords in the search bar by hitting Enter key. The results are displayed just below the search bar and features allowing user to enter/view records are available. If the user wants to add a note he may do so by clicking 'Add Notes' button and providing the required note. On pressing the 'Add Notes' button, notes already available for the keyword are retrieved into the notes section. The newly added note is updated to both the frontend and the database. The search keyword with their frequency count are maintained in the search_log.txt as well. It's updated for every search keyword progressively with frequency and search timestamp.

Docker Containers and Deployment

The cloud application logic is containerized into various docker services each running on different ports. The docker-compose file specifies the main Services logic by providing internal service details like build, command, volumes, ports and expose details. These services are thus a part of multi containerized application infrastructure. These services are exposed on EC2 instance IP on various ports. They interact among each other using API calls.

Test Cases

Following Test Cases are used to validate the application's workability:

TC_1: Verify that index.html is rendered properly in a browser.

TC_2: Verify that user can search book records with the help of book keyword.

TC_3: Verify that user can search book records with the help of Author keyword.

TC_4: Verify that user can search book records with the help of search string.

TC_5: Verify that user is only able to Add/View Notes once a successful search is performed.

TC_6: Verify that search_logs are updated with search keywords with their frequency and timestamp for every search request made.

TC_7: Verify that on successful search, if user clicks on Add Notes button, the past user notes are displayed in the Notes section.

TC_8: Verify that data.json file is updated with search records each time a successful search is performed.

TC_9: Verify that note is added successfully to notes collection in the mongodb database.

TC_10: Verify that blank notes are not getting entered to the database.

TC_11: Verify multiple notes can be added for the single search keyword.

TC_12: Verify Notes can't be added for a blank search keyword.

TC_13: Verify newly added note is appended in the Notes section.

TC_14: Verify user can see notes for the search keyword on clicking View Notes button.

TC_15: Verify Notes are displayed in the order of their creation timestamps.

TC_16: Verify upon pressing Enter again with same search keyword, the books details are not added again in the frontend.

TC_17: Verify multiple notes for the same keyword are stored in JSON format in notes collection for the given keyword.

TC_18: Verify the note is not added to the database, if the user clicks cancel to the note prompt.

TC_19: Verify the note is not added to the notes section in frontend, if the user clicks cancel to the note prompt.

References

[1]"Docker overview", Docker Documentation, 2020. [Online]. Available: https://docs.docker.com/engine/docker-overview/. [Accessed: 10- Mar- 2020].8 IEEE 11th International Conference on Cloud Computing
(CLOUD), San Francisco, CA, 2018, pp. 434-441.

[2] H. Castilho and M. Belkamel, "MongoDB loads but breaks, returning status=14", Ask Ubuntu, 2020. [Online]. Available: https://askubuntu.com/questions/823288/mongodb-loads-but-breaks-returning-status-14. [Accessed: 26- Mar- 2020].

[3] "MongoDB status failed after editing mongo.conf (code-exited, status=2)", Stack Overflow, 2020. [Online]. Available: https://stackoverflow.com/questions/57380491/mongodb-status-failed-after-editing-mongo-conf-code-exited-status-2. [Accessed: 26- Mar- 2020].

[4] "Install MongoDB Community Edition on Ubuntu — MongoDB Manual", Docs.mongodb.com, 2020. [Online]. Available: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/. [Accessed: 26- Mar- 2020].

[5] I. London, "How to connect to your remote MongoDB server", Ian London's Blog, 2020. [Online]. Available: https://ianlondon.github.io/blog/mongodb-auth/. [Accessed: 26- Mar- 2020].

[6]"How To Slide Down a Bar on Scroll", W3schools.com, 2020. [Online]. Available: https://www.w3schools.com/howto/howto_js_navbar_slide.asp. [Accessed: 26- Mar- 2020].

[7] M. Hossain, "20 Best Free Bootstrap Search Bar Templates 2019 - Colorlib", Colorlib, 2020. [Online]. Available: https://colorlib.com/wp/bootstrap-search-bar/. [Accessed: 26- Mar- 2020].

[8] "How to Install Python 3.7 on Ubuntu 18.04", Linuxize.com, 2020. [Online]. Available: https://linuxize.com/post/how-to-install-python-3-7-on-ubuntu-18-04/. [Accessed: 26- Mar- 2020].

[9] "Collection Methods — MongoDB Manual", Docs.mongodb.com, 2020. [Online]. Available: https://docs.mongodb.com/manual/reference/method/js-collection/. [Accessed: 26- Mar- 2020].

[10] "Thank you for downloading! - Colorlib", Colorlib, 2020. [Online]. Available: https://colorlib.com/thank-you-for-downloading/?dlm-dp-dl=1825. [Accessed: 26- Mar- 2020].