Cambridge Medium (2020), **37**, 1–42

SPE Final Project - NewsGenie

Sivani Channamsetty, R Prasannavenkatesh, Nandadeep Medicharla

IMT2019020, IMT2019063, IMT2019049

Abstract

NewsGenie is a revolutionary news aggregation application that puts the power in the hands of the user. With NewsGenie, users can customize their news feed according to their interests, ensuring that they only receive updates on the topics that matter most to them. The app is powered by natural language processing transformers like t5 that can analyze news sources.

The project also integrates the application with a DevOps toolchain. The toolchain will be designed using various DevOps tools such as Git for version control, Jenkins for continuous integration, Ansible for configuration management and deployment, etc. The pipeline will consist of testing the code using Jest and pytest, building the code using tools such as uvicorn and node, containerizing the code using Docker, and deploying it using configuration management tools. Once the code is deployed, The ELK stack and Sentry will be used for monitoring the code, and logs will be generated to pass through the stack.

By integrating a newsgenie into the DevOps toolchain, developers will have a powerful tool for easy configuration and deployment. The pipeline will streamline the development process, ensuring that the code is thoroughly tested, built, containerized, and deployed efficiently. The ELK stack and sentry application will provide real-time monitoring of the code, enabling developers to identify and fix any issues quickly. Overall, this project will improve the efficiency of the development process and provide with ease of deployment access for the development process.

NewsGenie (What's the news?)

NewsGenie is a revolutionary news aggregation application that puts the power in the hands of the user. With NewsGenie, users can customize their news feed according to their interests, ensuring that they only receive updates on the topics that matter most to them. The app is powered by natural language processing transformers like t5 that can analyze news sources.

In addition to delivering personalized news content, NewsGenie also provides a range of features that allow users to interact with the news in meaningful ways. Users can read detailed news summaries to quickly understand the key points of a story, leave comments to share their opinions and insights with others, and save articles for later viewing. With the ability to view their news history and visualize it in various ways, users can gain valuable insights into their reading habits and stay informed on the issues that matter most to them. With its advanced summarization models, NewsGenie is the ultimate news companion for anyone seeking to stay informed in today's fast-paced world. The following are some of the major features of the NewsGenie application:

- user authentication
- news summarization
- article creation
- commenting on articles
- article saving
- article history
- article filtering
- article rating

Devops: What is Devops and Why Devops?

DevOps is a set of practices that combines software development and IT operations to improve the quality and speed of software delivery. In other words, DevOps is a culture that brings together developers and operations teams to work collaboratively, share responsibility and automate the software delivery process. DevOps focuses on continuous integration and delivery, automation, and monitoring to improve the efficiency and reliability of software development and deployment.

DevOps is widely used in the commercial world as it helps organizations to develop and deploy software faster and with higher quality. In the past, software development and operations were considered separate entities, but with the advent of DevOps, these two areas are now integrated into a seamless process. DevOps has become an essential tool for many organizations in today's fast-paced digital world, where software development and deployment are critical for business success. By using DevOps, organizations can achieve higher efficiency, reduce costs, and deliver software faster with fewer defects.

One of the advantages of DevOps is that it helps to reduce the time-to-market for software products. By automating the software delivery process, organizations can deliver software updates more frequently and efficiently, allowing them to stay ahead of their competition. DevOps also encourages collaboration between developers and operations teams, resulting in faster issue resolution and better quality software. Additionally, by automating the deployment process, organizations can reduce the risk of errors and minimize the time and resources required for manual deployment. These advantages make DevOps an essential tool for modern organizations looking to stay ahead in the highly competitive digital market.Some major advantages of using devops are:

- Faster software delivery
- Increased frequency of software releases
- Reduced time-to-market
- Better collaboration between development and operations teams
- Faster issue resolution
- Reduced errors and defects
- Increased efficiency and productivity
- More reliable software deployments
- Improved security and compliance
- Better monitoring and feedback loops
- Stable operations

Tools Used

The tools used to create the application & devops chain and for implementing the same are as follows:

- **Frontend:** React (Node.js)
- **Backend:** FastAPI (Python)
- **Database:** MongoDB
- **Transformer Models:** Pytorch (Python)
- **Authentication:** Supabase
- **IDE:** VSCode
- **Dependency and Package Management:** npm, pip
- **Testing:** Jest, PyTest
- **Version Control System:** Git
- **Continuous Integration:** Jenkins
- **Containerisation:** Docker
- **Configuration Management:** Ansible
- **Monitoring:** ELK stack, Sentry

Installation Process

Node.js installation

- sudo apt-get update
- curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh
- curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
- source /.bashrc
- nvm install lts/hydrogen
- node –version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ nvm --version
0.39.3
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ node --version
v18.16.0
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$
```

Figure 1. Node.js installation: Checking for version

Here the long term support version of Node.js is used as the de-facto version of the language. If some other version of Node.js has to be installed it can be changed.

- nvm list (List all available Node.js versions)
- nvm install <version>
- nvm use <version>

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ nvm list
->     v18.16.0
default -> lts/hydrogen (-> v18.16.0)
iojs -> N/A (default)
unstable -> N/A (default)
node -> stable (-> v18.16.0) (default)
stable -> 18.16 (-> v18.16.0) (default)
lts/* -> lts/hydrogen (-> v18.16.0)
lts/argon -> v4.9.1 (-> N/A)
lts/boron -> v6.17.1 (-> N/A)
lts/carbon -> v8.17.0 (-> N/A)
lts/dubnium -> v10.24.1 (-> N/A)
lts/erbium -> v12.22.12 (-> N/A)
lts/fermium -> v14.21.3 (-> N/A)
lts/gallium -> v16.20.0 (-> N/A)
lts/hydrogen -> v18.16.0
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$
```

Figure 2. Node.js installation: Updating to correct version

Python Installation

- sudo apt-get update
- sudo apt install python3.8
- python --version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ python3 --version
Python 3.8.10
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$
```

Figure 3. Python installation: Checking for version

MongoDB Installation

- sudo apt-get update
- curl -fsSL https://pgp.mongodb.com/server-6.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg –dearmor
- echo "deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list
- sudo apt-get install -y mongodb-org
- mongod --version

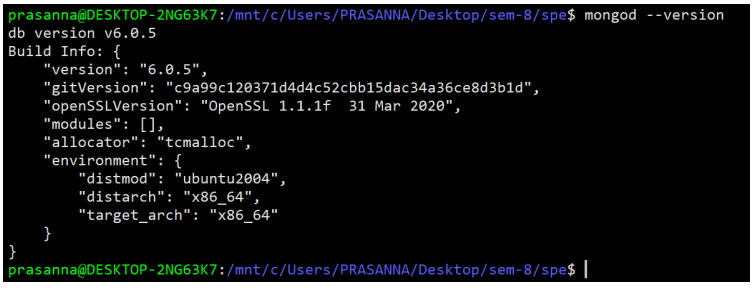


Figure 4. Mongo installation: Checking for version

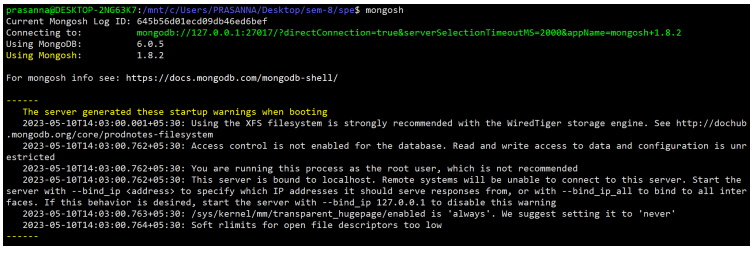


Figure 5. Mongo installation: Running the Mongo shell

FastAPI Installation

- pip install "fastapi[all]"
- pip freeze | findstr fastapi

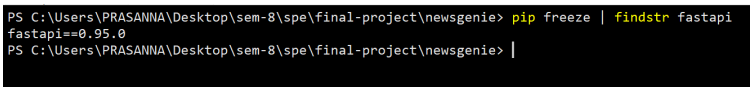


Figure 6. FastAPI installation: Checking for version

Git Installation

- sudo apt-get update
- sudo apt-get install git
- git --version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop$ git --version
git version 2.25.1
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop$
```

Figure 7. Git installation: Checking for version

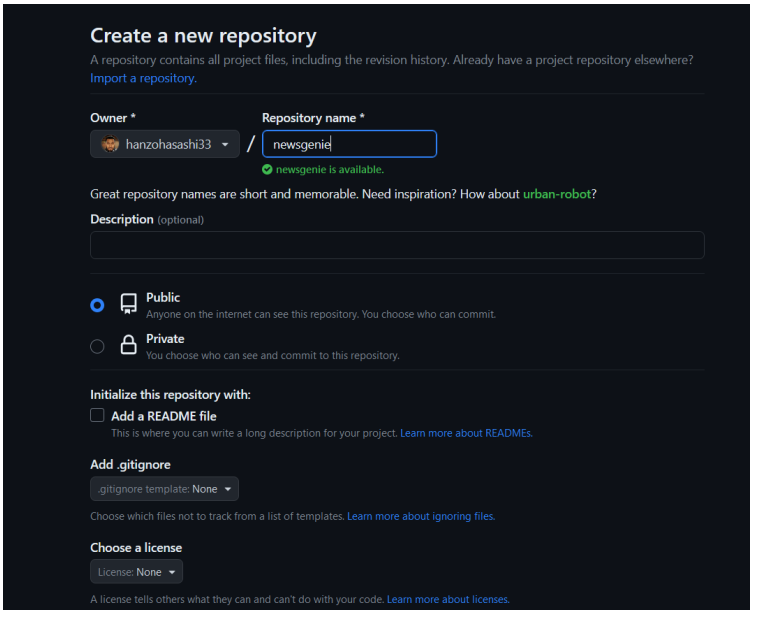


Figure 8. Github: Create new Repository

Jenkins Installation

- wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
- sudo sh -c 'echo deb https://pkg.jenkins.io/debian binary/etc/apt/sources.list.d/jenkins.list'
- sudo apt install ca-certificates
- sudo apt-get update
- sudo apt-get install jenkins
- sudo service jenkins status
- jenkins --version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$ sudo service jenkins start
[sudo] password for prasanna:
 * Starting Jenkins Automation Server jenkins
Correct java version found
Setting up max open files limit to 8192
                                                                                                    [ OK ]
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$
```

Figure 9. Jenkins installation: start the service

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$ jenkins --version
2.375.2
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$
```

Figure 10. Jenkins installation: Checking for version

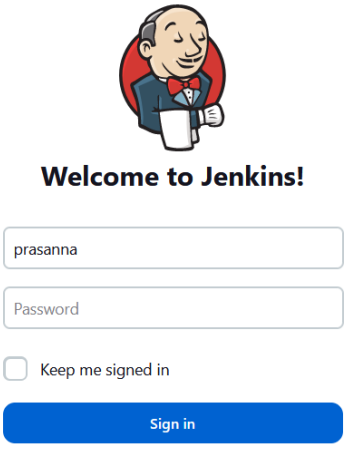


Figure 11. Jenkins installation: Home page

Docker Installation

- sudo apt-get update
- sudo apt-get install docker.io
- sudo dockerd (To start the docker service)
- docker --version
- sudo docker images (To list all local docker images)

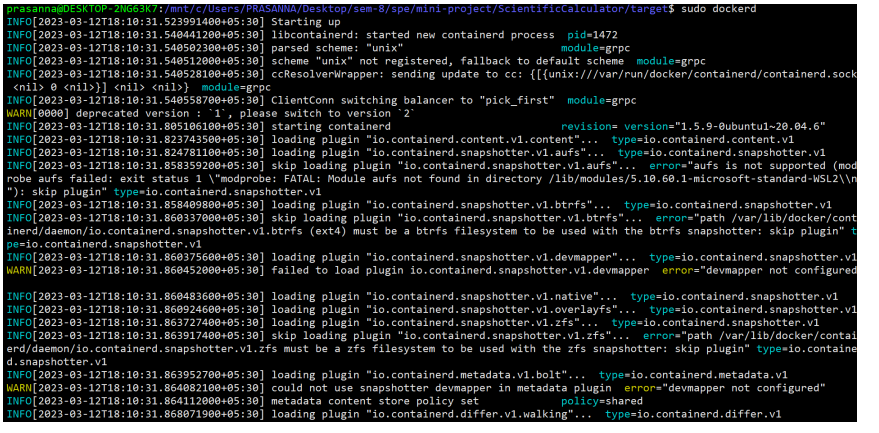


Figure 12. Docker installation: Starting the docker service

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$
```

Figure 13. Docker installation: Checking for version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$ sudo docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
kangarzoo1                  latest    77330093f729   15 minutes ago   654MB
<none>                      <none>    672f6496119f   18 minutes ago   654MB
ubuntu                      latest    58db3edaf2be   6 weeks ago      77.8MB
openjdk                     11        47a932d998b7   7 months ago     654MB
thangaraju/webserver_test   latest    bb72c20d14f6   7 years ago      794MB
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe$
```

Figure 14. Docker installation: Listing all local docker images

Ansible Installation

- sudo apt-get install openssh-server
- sudo apt-get update
- sudo apt-get install ansible
- pip install docker
- ansible --version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$ ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/prasanna/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/prasanna/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Jun 22 2022, 20:18:18) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$
```

Figure 15. Ansible installation: Checking for version

Logstash Installation

- sudo apt-get update
- sudo apt-get install apt-transport-https
- wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | apt-key add -
- echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | tee -a /etc/apt/sources.list.d/elastic-7.x.list
- sudo apt-get install logstash
- sudo systemctl start logstash
- /usr/share/logstash/bin/logstash --version

```
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$ /usr/share/logstash/bin/logstash --v
ersion
Using bundled JDK: /usr/share/logstash/jdk
logstash 7.17.9
prasanna@DESKTOP-2NG63K7:/mnt/c/Users/PRASANNA/Desktop/sem-8/spe/mini-project/ScientificCalculator$
```

Figure 16. Logstash installation: Checking for version

ngrok Installation

- sudo tar xvzf /Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
- ngrok authtoken <token>
- ngrok --version
- ngrok http 3000

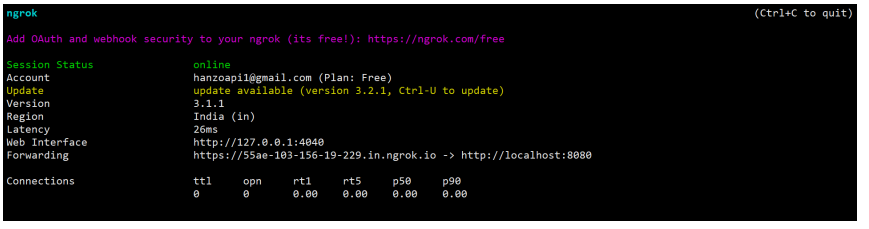


Figure 17. Ngrok installation: running ngrok server

Software Development Phases

Source Code Mangement and Version Control

Git is a popular distributed version control system used to track changes in source code during software development. It allows developers to work collaboratively on the same codebase and provides a way to manage code changes over time. Git works by creating a repository that contains all versions of a project, called commits. Each commit represents a snapshot of the project at a specific point in time, with all the changes made since the previous commit. Git also provides a way to branch off from the main codebase to work on new features or fixes without affecting the main codebase until ready.

Version control systems like Git are essential for software development because they allow developers to work together more efficiently, provide a way to track and manage code changes over time, and offer a safety net in case anything goes wrong. With version control, developers can easily roll back to a previous version of the code, compare different versions of the code, and collaborate with other team members without overwriting each other's work. Git has become the de facto standard for version control in modern software development and is widely used by developers and organizations of all sizes. The following are some of the commands we used to create, track and update the local repositories:

- git init (To initialize the local repository)
- git add <source files> (To track the corresponding files)
- git commit -m "commit message" (To commit the tracked changes to a snapshot commit)
- git status (To check the status of the current repository)
- git log (To check the commit history and logs)

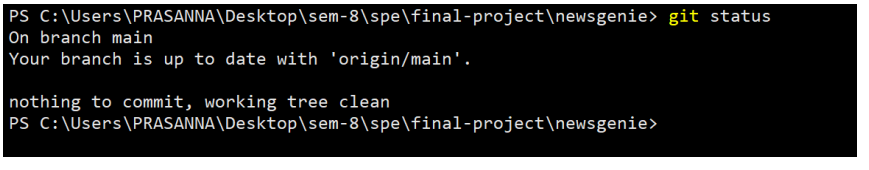


Figure 18. Git: Check status of current repository

```
PS C:\Users\PRASANNA\Desktop\sem-8\spe\final-project\newsgenie> git log --all --oneline
c5661ac (HEAD -> main, origin/main) Added minor changes
c0ac7d7 Added Dockerfile
a70ffd7 Added Dockerfile
a355e63 adding frontend tests
612af3a feat: added sentry monitoring
0d31023 fix: added logout user for logging
eeb95ae fix: repository structure
375ad85 feat: added better logging features
708be02 feat: added mongo design for profile and visits
b1daf55 feat: added mongo design for the comments collection
f81e432 feat: added mongodb and mongo endpoints for articles
ec72c36 feat: added logging services in frontend and backend
e55b47b fix: added new user table and redirected queries to it
```

Figure 19. Git: check history of current frontend repository

Once commits are created in the local repository, a remote link can be established and the commits can be pushed to the user owned remote repository in github using the following commands:

- git remote add origin <remote git link>
- git track -M main
- git push -u origin main

```
PS C:\Users\PRASANNA\Desktop\sem-8\spe\final-project\newsgenie> git push
Everything up-to-date
PS C:\Users\PRASANNA\Desktop\sem-8\spe\final-project\newsgenie>
```

Figure 20. Git: Push commits to remote frontend repository

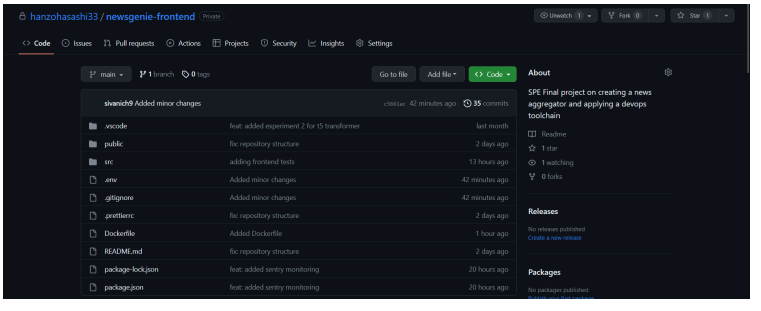


Figure 21. Github: updated frontend repository after pushing commits

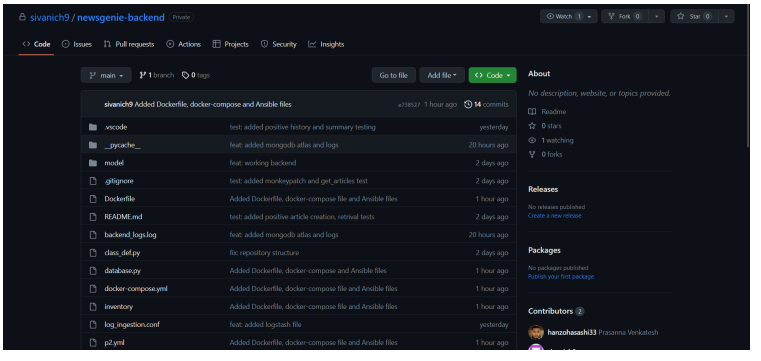


Figure 22. Github: updated backend repository after pushing commits

Implementation

Backend

After the installation phase, the implementation phase of our application backend using FastAPI and MongoDB as the database begins. Our application will be a RESTful API that provides endpoints for creating and managing news articles, handling authentication, and performing CRUD (Create, Read, Update, Delete) operations on data.

To ensure a flexible and modular code-base, we will be following FastAPI's dependency injection system. This will allow us to easily inject dependencies such as database connections and authentication services into our endpoints and other components. We will also be using Pydantic for data validation and serialization, making it easy to handle incoming requests and responses.

During the implementation phase, we will be using popular development tools such as Visual Studio Code as our integrated development environment (IDE), and pip as our package manager. These tools will help us to manage our codebase, collaborate with other developers, and ensure that our code is well-organized, readable, and maintainable.

We will be using MongoDB as our database of choice, which is a NoSQL database that provides high scalability and performance for our application. We will be using PyMongo, a Python library for working with MongoDB, which will allow us to easily interact with the database and perform CRUD operations on our data.

Throughout the implementation phase, we will be following best practices such as writing clean and efficient code, using descriptive variable and function names, and commenting our code to make it understandable to other developers. By following these practices, we will be able to create a high-quality, scalable, and maintainable backend application using FastAPI and MongoDB.

Creating a FastAPI project involves several steps to set up the project and start building the application. First, we need to create a new directory for our project and navigate to it. Then, we will create a virtual environment for our project using a tool like venv or conda to isolate our dependencies. Once the environment is activated, we can use pip to install FastAPI and any other required packages.

Next, we need to create a new Python file that will serve as the entry point for our application. In this file, we will import FastAPI and use it to create a new instance of our application. We can also define our endpoints, which are functions that will handle incoming requests and return responses.

To run our FastAPI application, we can use the uvicorn server, which is a fast and reliable server that supports asynchronous code. We can start the server by running a command that includes the name of our entry point file and the name of our application instance.

To handle the CRUD operations of our newsgenie applications, we create logic for creating, reading, updating, and deleting articles from the database. The file will define the different endpoints for the API, such as /articles to get all articles, /articles/id to get a specific article by ID, and /articles/create to create a new article. The list of all available endpoints and their corresponding definitions are as follows:

- /summary - gets the summary of the news article description sent
- /create_article - create a new article. Here the json request sent must be of the news article data model.
- /get_articles - gets all news articles
- /get_article - gets a specific article with the given id. The json request sent from the frontend should contain the id of the article required.
- /create_comment - creates a new comment for a particular news article. The json request sent must be of the comment data model.
- /get_comments - gets all the comments of the required news article. The json request sent must be of the article id, the comments need to be fetched.

• /post_visit - stores the history of the person in the visits table. The json request sent must be of the visit data model.
• /get_visits - returns the history of the corresponding user. The json request sent must include the identification of the user.
• /docs - returns the swagger API documentation for the present API routes

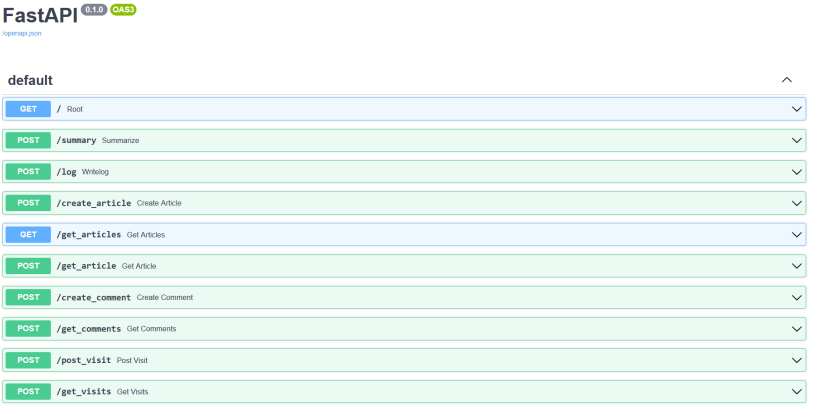


Figure 23. FastAPI: Swagger complient API Documentation

By using pydantic data models, we can make our code modular, easy to read, and maintainable. The database is called from the database.py file, so that the database can be changed as easily as we want. The model experiments are present in the model folder in the form of jupyter notebooks. There is both custom-trained and pre-trained models available in the same. The summarizer_api.py python file contains the API implementation code. By splitting our code into these different files, we can create a well-structured and organized codebase that is easy to maintain and update.

To interpret and run the backend code, we use the following commands:

• uvicorn summarizer_api:app
• http://localhost:8000/ : default host for the routes
• http://localhost:8000/docs : route to read the Swagger API documentation

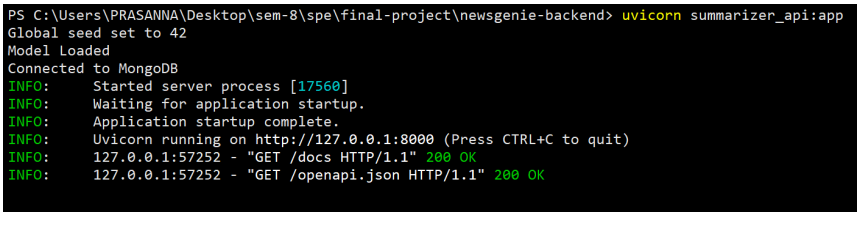


Figure 24. FastAPI: executing the backend code

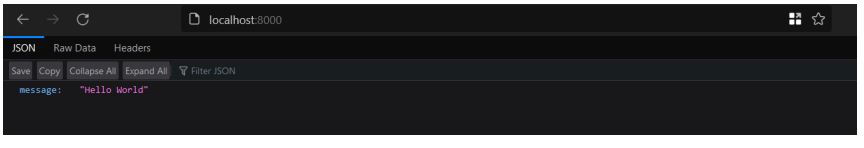


Figure 25. FastAPI: viewing responses in browser

Note : Our summarizer model is 800MB, so it is impossible to push it to github. So download that folder from summarizer model and add that folder to newsgenie-backend.

Frontend

After the backend implementation phase, the implementation phase of a React application begins. For this project, we will be using React as our front-end framework. We will be developing a web-based news article aggregation platform, which will include features listed in the initial section.

To ensure a flexible and modular codebase, we will be following React's component-based architecture. This will allow us to create reusable components that can be easily integrated into different parts of the application. We will also be using React's inbuilt state and context management as our state management library to maintain a centralized store of data that can be accessed by different components. Throughout the implementation phase, we will be following best practices such as writing clean and efficient code, using descriptive variable and function names, and commenting our code to make it understandable to other developers. By following these practices, we will be able to create a high-quality, scalable, and maintainable React application.

Once the implementation is complete, we will use Webpack as our bundler to package our code into a single deployable file. This will allow us to optimize the application's performance by reducing load time and minimizing the number of HTTP requests.

The following are the React project creation steps. Node.js and npm are prerequisites for the same. We will be using the npx create-react-app command to obtain the template for the standard react project and continue from the same. The project structure looks as follows:

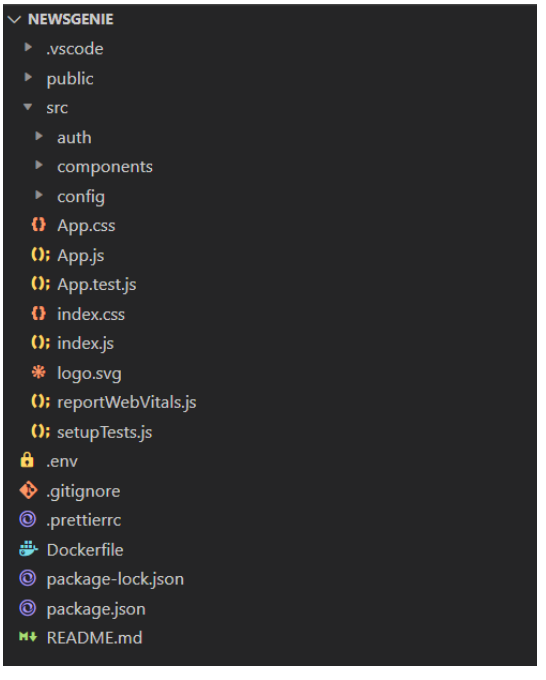


Figure 26. React: project structure

The project structure created by the npx create-react-app command includes the following. Overall, the project structure created by the npx create-react-app command provides a good starting point for building a React application. It includes all the necessary files and directories to get started and provides a well-organized structure for the application's source code.

- node modules: a directory that contains all the dependencies required by the application
- public: a directory that contains the static assets such as HTML, images, and favicon
- src: a directory that contains the application's source code
- index.js: a JavaScript file that is the entry point for the application
- App.js: a React component that defines the main structure of the application
- index.css: a CSS file that contains the global styles for the application
- App.css: a CSS file that contains the styles specific to the App component
- logo.svg: a SVG file that is used as the logo for the application
- package.json: a JSON file that contains the metadata and dependencies for the application
- README.md: a Markdown file that provides instructions for the application
- .gitignore: a file that specifies files and directories to be ignored by Git
- .env: a file that contains environment variables that can be used in the application
- package-lock.json: a JSON file that specifies the exact version of each dependency installed in the application

The src folder contains the main frontend code written in javascript. For easy maintainence, it is split in 3 parts, namely - auth which contains the authentication management pages, components which contain the main article components and config which contain supabase setups and some api call abstraction units. To run the application we perform the following commands: To interpret and run the frontend code, we use the following commands:

- npm install
- npm start
- http://localhost:3000 : link to the application

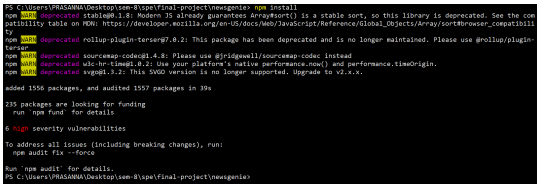


Figure 27. React: installing all the required packages

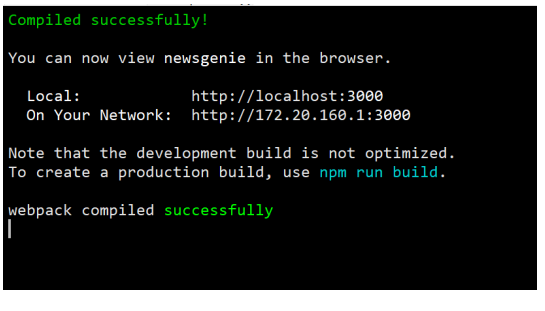
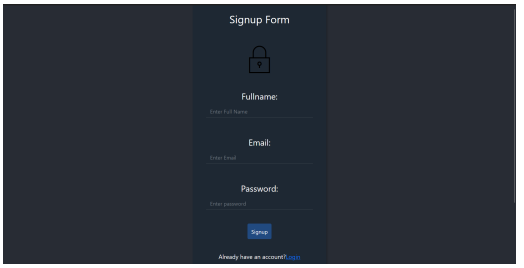
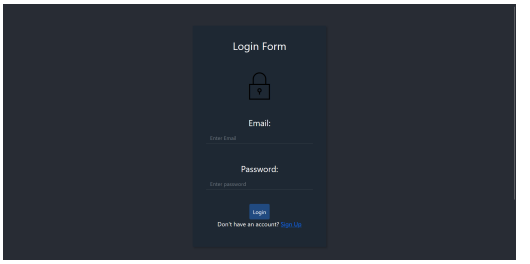


Figure 28. React: running the frontend

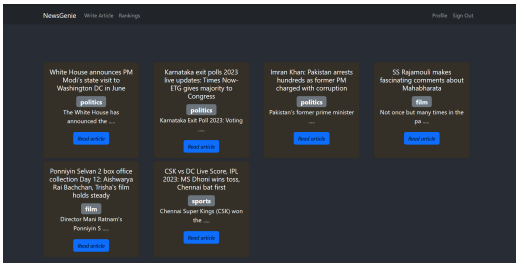
The below images are of the UI pages of the application. It contains the basic UI coverage and conveys the workflow of the application as well.



(a) Frontend: Signup Page

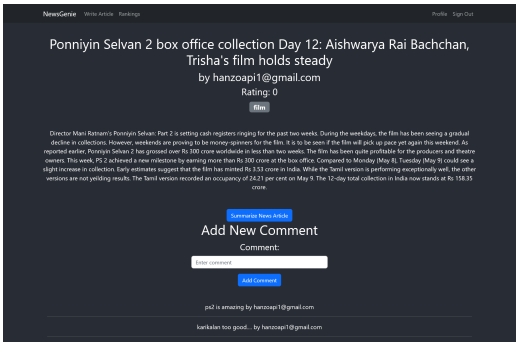


(b) Backend: Login Page

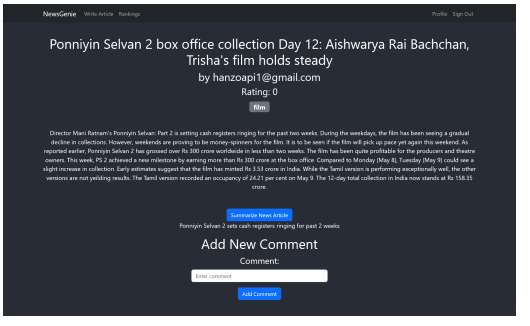


(a) Frontend: NewsGenie Home Page

(b) Frontend: Article Creation Page



(a) Frontend: Article interface page



(b) Frontend: Summary of an article

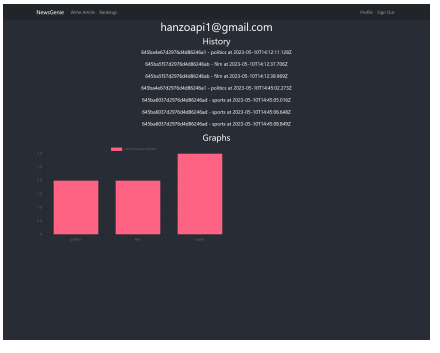


Figure 32. Frontend: Profile Page

Testing

Frontend

Jest is a popular JavaScript testing framework developed by Facebook. Jest is primarily used for testing React based apps. It allows you to write unit tests, integration tests, and snapshot tests to ensure that your code functions correctly.

The following command can be used to install jest globally.This helps us to use Jest commands directly without configuring the package file for npm tests.

```
$ npm install -g jest
```

The following code is a sample test suit for the NavBar component in a NewsGenie application.

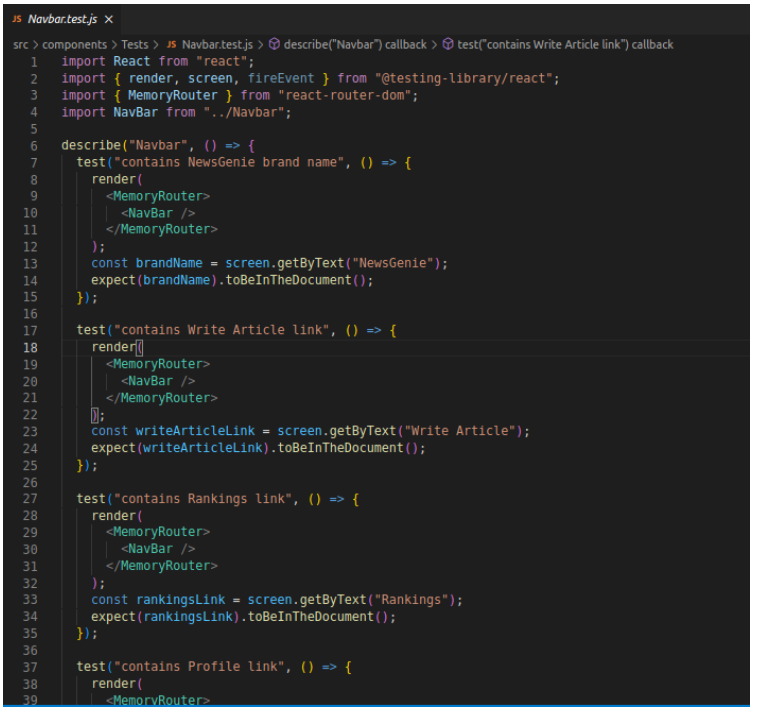


Figure 33. NavBar Jest Test

The above test suit contains multiple individual tests, each testing a specific aspect of the NavBar component. Some of the sample test cases were:

- **Contains NewsGenie brand name**: It verifies that the rendered NavBar component contains the text "NewsGenie" representing the brand name.
- **Check for necessary links**: It verifies that the rendered NavBar component contains the necessary links like Write Article link, Rankings link, Profile link, Signout link etc.

To run these tests, the NavBar component is rendered in a simulated environment provided by the `MemoryRouter` component from the react-router-dom library. This allows testing of the component's behavior related to routing without needing a complete browser environment. The `screen` object from the @testing-library/react library is used to interact with the rendered component.

Similarly jest tests were written to the NewsCard and PostComment components. The following figure shows the output of the `$ npm test` command:

Figure 34. npm test results

Backend

Testing is an important part of software development and ensures that our application works as expected and is bug-free. For the application, backend testing is done using pytest and mongomock for mocking the database.

Pytest is a popular testing framework for Python that allows us to write simple and efficient tests. Pytest supports various types of tests, including unit tests and integration tests. Unit tests are used to test individual components of the application, while integration tests are used to test how different components work together.

To perform CRUD testing, we can use mongomock, a Python library that allows us to mock the MongoDB database. Mongomock provides an in-memory implementation of the MongoDB API, allowing us to test our application without the need for a real database.

To use mongomock in our tests, we can create a mock database instance and use it to perform CRUD operations on our data. For example, we can create a test function that creates a new article, saves it to the mock database, retrieves it, and then verifies that the retrieved article matches the original. A sample positive test unit will look as follows. Here mongomock is the name of the function that sets up the test database that allows all accesses.

```
def test_get_article(mongo_mock):
    response = client.post(
        "/get_article", data=json.dumps({"id": "5e63c3a5e4232e4cd0274ac2"})
    )
    assert response.status_code == 201
    print(response.json())
    article = response.json()["article"]
    assert article["user"]["id"] == "u1"
    assert article["user"]["email"] == "hanzoapi1@gmail.com"
    assert article["headline"] == "test_headline"
    assert article["description"] == "test_desc"
    assert article["genre"] == "test"
```

Figure 35. Pytest: Sample Positive Test Unit

To test the above written tests against the backend, we use the command:

- pytest test_main.py -W ignore::DeprecationWarning

The above command runs the test suite and provides the summary of the testing instance. Depending on the summary, we can figure on how to proceed with the testing procedure.

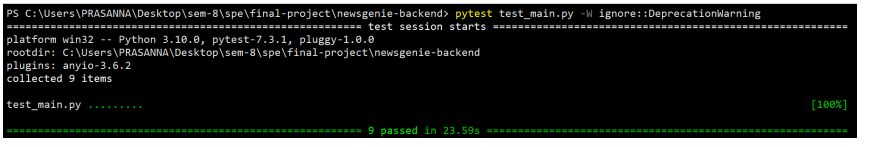


Figure 36. Pytest: Testing Summary

Dockerization

Dockerization refers to the process of packaging an application and its dependencies into a container image that can be easily deployed across multiple environments. Docker is a popular open source tool used for containerization, and it allows developers to create, deploy, and run applications as containers. Docker containers are lightweight, portable, and can run on any machine that supports Docker, which makes them an ideal choice for deploying applications in a variety of environments. Because of the many dependencies and versions of dependencies that must be supported, there will be a number of issues to be resolved when the product is delivered. The clients will find it very difficult to resolve. Docker is helpful in this situation since it enables us to correctly install all of the dependencies and then compress them into an image. As the image was uploaded to DockerHub for a similar reason as github, anyone can now download the image and use it for their own projects. Docker is used to make images through containerization. The benefits of Dockerization include portability, consistency, and scalability.

First we sign in to DockerHub and create two repositories for newsgeniefrontend and newsgeniebackend.

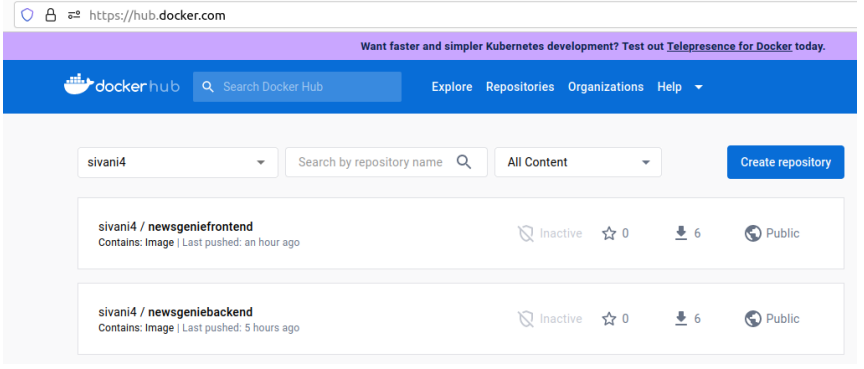


Figure 37. Creating repositories in docker hub

To Dockerize frontend and backend, we use a Dockerfile, which is a text file that contains instructions for building a Docker image. The Dockerfile provided for frontend uses the node:14-alpine image as the base image, sets the working directory, copies json file from the current directory of the Dockerfile into the root directory of the image and runs using the CMD instruction. The Dockerfile provided for backend uses the bitnami/pytorch image as the base image, sets the working directory, copies required files from the current directory of the Dockerfile into the root directory of the image, install the required packages and runs using the CMD instruction. By building the image using this Dockerfile, we can create a containerized version of the frontend and backend that can be easily deployed across different environments without any compatibility issues. We push the images to the created docker hub repositories. The commands to build the image and execute the same are as follows:

- *sudo docker build -t <docker-username/reponame:tagname> <folderpath>.* (builds Docker images from a Dockerfile and a "context")
- *sudo docker push <docker-username/repo-name:tagname>* (to push the image to dockerhub)
- *sudo docker run -d -it -p 3000:3000 –name <container-name> <docker-image-name-or-id>* (create a writeable container layer over the specified image, runs the container in the background, in detached mode and publish port 3000 on the host and map it to port 3000 in the container. This allows us to access the container's service (in this case, a web app) from the host's web browser

by visiting localhost:3000. For backend we use port 8000 to run the container. *sudo docker run -d -it -p 8000:8000 –name <container-name> <docker-image-name-or-id>*)

Dockerfile

```
FROM node:14-alpine

# Set up the working directory
WORKDIR /app

# Copy the application files
COPY package*.json ./
RUN npm install
COPY . .

# Specify the start command
CMD ["npm", "start"]

```

Figure 38. Dockerfile for frontend

Dockerfile

```
# Use a base image with Python 3.8 pre-installed
FROM bitnami/pytorch

USER root
RUN mkdir /.cache
RUN chmod 777 /.cache

# Set the working directory
WORKDIR /.cache

# Copy the Python files to the container
COPY summarizer_api.py ./
COPY database.py ./
COPY simplet5-epoch-4-train-loss-0.6005-val-loss-1.6554/ ./simplet5-epoch-4-train-loss-0.6005-val-loss-1.6554/

# Install the required packages
RUN pip install simplet5
RUN pip install "fastapi[all]"
RUN pip install uvicorn
RUN pip install loguru
RUN pip install pymongo


# Set the entrypoint command to run the summarizer.py script
#CMD ["python3", "-m", "uvicorn", "summarizer_api:app"]
CMD ["uvicorn", "summarizer_api:app","--host", "0.0.0.0", "--port", "8000"]
```

Figure 39. Dockerfile for backend

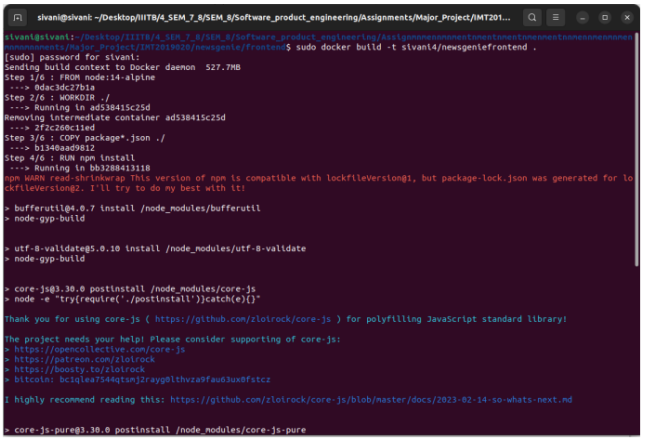


Figure 40. Frontend: Running the Dockerfile to build docker image

Figure 41. Backend: Running the Dockerfile to build docker image

Figure 42. Frontend: Running the docker image

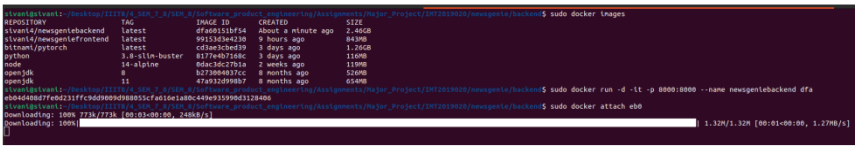
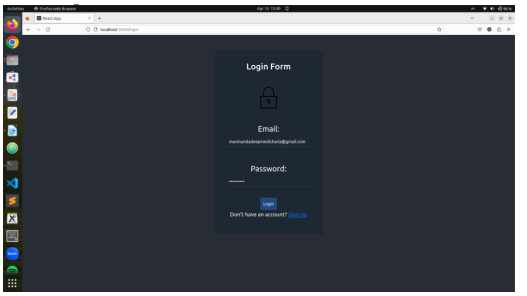
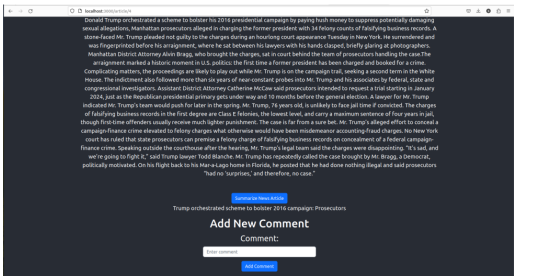


Figure 43. Backend: Running the docker image

After we run the docker image, we go to *localhost:3000/login* to check if frontend is working. To check if backend is working, we can write an article or summarise the article.



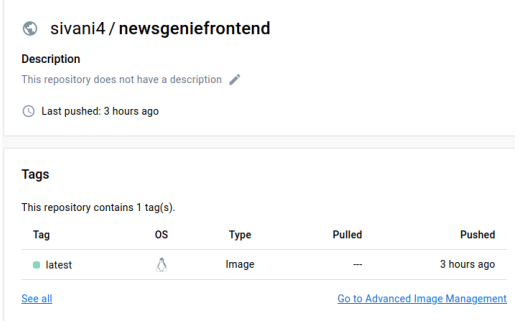
(a) Checking frontend



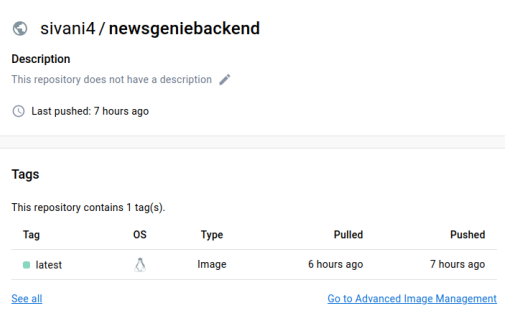
(b) Checking backend

Some docker commands :

- *sudo docker images* - To see docker images
- *sudo docker ps -a* - To see docker containers
- *sudo docker start <container_name or container_id>* - To start the docker container
- *sudo docker attach <container_name or container_id>* - To attach the docker container
- *sudo docker start -a -i <container_name or container_id>* - To start and attach the docker container
- *sudo docker stop <container_name or container_id>* - To stop the docker container



(a) Frontend: After pushing image to docker hub



(b) Backend: After pushing image to docker hub

Docker Compose

Docker Compose is a tool that allows us to define and run multi-container Docker applications. It uses a YAML file usually named docker-compose.yml to define the services that make up your application, as well as any configuration options needed to run those services.

A typical Docker Compose file might define several services, each running a different container image, and specify how those services should be connected to each other and to the outside world. For example, a simple web application might consist of two services: one running a web server container and another running a database container. The Docker Compose file would define these services, specify the images to use, and configure the network connections between them.

With Docker Compose, we can define the configuration of our application in a single file, which makes it easy to manage and deploy our application across multiple environments, such as development, testing, and production. We can also define dependencies between services and specify network settings, making it easy to run complex applications that require multiple containers. The docker-compose file specifies the services, networks, and volumes required to run a complete application stack.

Docker Volumes, on the other hand, provide a way to persist data in Docker containers beyond the container's lifecycle. A Docker volume is a directory on the host system that is mounted into a

container, allowing the container to read and write data to that directory. Volumes can be used to store application data, database files, log files, and other persistent data.

Docker volumes have several advantages over other methods of storing data, such as using host-mounted directories or using a container's writable layer. Volumes are portable, meaning that we can easily move them between hosts or containers, and they can be managed using Docker commands or tools such as Docker Compose. Volumes also support advanced features such as data encryption and backup and restore.

Below is the docker-compose file for our project.

docker-compose.yml

```yaml
version: '3'

services:
  frontend:
    image : sivani4/newsgeniefrontend:latest
    build : ../newsgenie-frontend
    container_name : newsgeniefrontend
    volumes:
      - frontend-volume:/app/node_modules
    ports:
      - "3000:3000"
    depends_on:
      - backend


  backend:
    image : sivani4/newsgeniebackend:latest
    build : .
    container_name : newsgeniebackend
    volumes:
      - backend-volume:/.cache
    ports:
      - "8000:8000"
    environment:
      - MONGO_HOST=mongodb
      - MONGO_PORT=27017
    depends_on:
      - mongodb

  mongodb:
    container_name: mongodb
    image: mongo
    volumes:
      - mongodb-volume:/data/db
    ports:
      - "27017:27017"

volumes:
  frontend-volume:
  backend-volume:
  mongodb-volume:
```

Figure 46. docker-compose file

docker-compose file explanation :

In this docker-compose file, we have three services defined: frontend, backend, and mongodb.

The frontend service uses an image sivani4/newsgeniefrontend:latest. It creates a container with the name newsgeniefrontend. It maps the local node modules directory to a volume named

frontend-volume, and exposes the container's port 3000 to the host machine's port 3000. It depends on the backend service.

The backend service uses an image sivani4/newsgeniebackend:latest. It creates a container with the name newsgeniebackend. It maps a volume named backend-volume to /.cache directory of the container. It exposes the container's port 8000 to the host machine's port 8000, and sets environment variables for MONGO_HOST and MONGO_PORT. It depends on the mongodb service.

The mongodb service uses the official mongo image, creates a container with the name mongodb, maps a volume named mongodb-volume to /data/db directory of the container, and exposes the container's port 27017 to the host machine's port 27017.

Finally, we have defined three named volumes frontend-volume, backend-volume, and mongodb-volume that are used by the respective services to persist data across container restarts.

Note : If the image specified in the *image* field is not already present on the host, Docker Compose will try to pull it from the configured docker registry (by default, Docker Hub). If the image is not found, docker Compose will return an error. The *build* keyword can be used instead of the *image* keyword, and it can be used to instruct docker Compose to build an image from a Dockerfile. For example, *build: .* in the docker Compose file tells docker Compose to look for the Dockerfile in the current directory (.), and use it to build the image for the service. If the image is already built and available locally or in a registry, you can use the image keyword instead to specify the name of the image to use. In this file I used build after image keyword. So basically it first checks image in the host, if not it pulls the image from docker hub. If image is not there in dockerhub then using build, it will build the image using Dockerfile which present in the mentioned directory.

Command to run the docker-compose file : *sudo docker-compose up*
Run the command from the directory of the docker-compose file

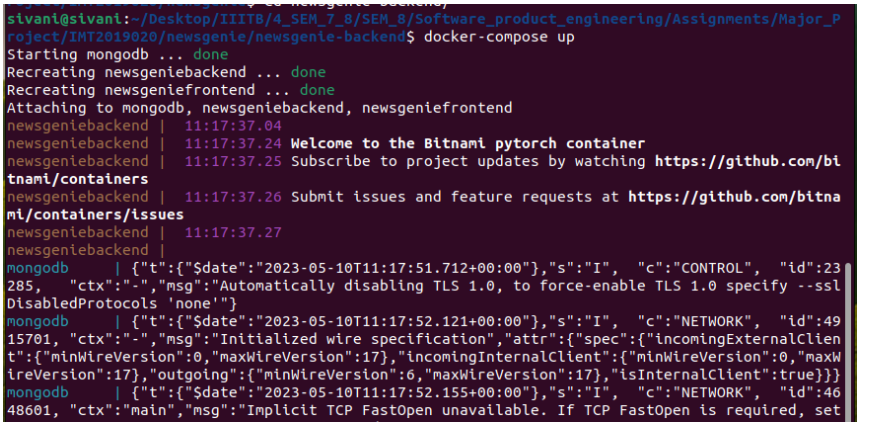


Figure 47. Running docker-compose file

To see docker volumes do *sudo docker volume ls*

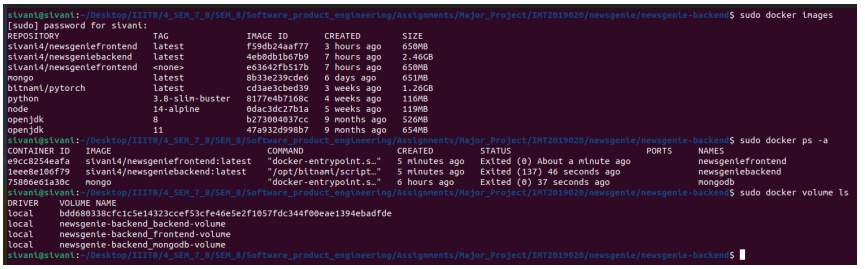


Figure 48. Created docker images, containers and volumes after running docker-compose file

Configuration Management

Configuration management is the process of automating the management of system configurations to ensure that they are consistent, reliable, and up-to-date. It involves managing infrastructure as code, which means that configurations are defined in code and can be version-controlled, tested, and automated.

Ansible is a popular configuration management tool that automates the deployment and management of applications, services, and systems. It uses a simple syntax called YAML to define configuration files, which are called playbooks. Playbooks define a set of tasks that need to be performed on remote hosts, and Ansible uses SSH to connect to these hosts and execute the tasks.

One of the key advantages of Ansible is its inventory management system, which allows you to define and group hosts based on different criteria, such as environment, application, or function. The inventory file is a simple text file that contains a list of hostnames or IP addresses, and Ansible uses this file to know which hosts to target with its tasks.

To use Ansible for configuration management, you can create a playbook that defines the tasks that need to be performed on the remote hosts. For example, you can define a playbook to install and configure the scientific calculator on a set of remote hosts. The playbook can contain tasks such as installing the required dependencies, copying the files to the remote hosts, and configuring the firewall rules to allow access to the application.

Once the playbook is defined, you can use the ansible-playbook command to execute the playbook on the remote hosts. Ansible uses the inventory file to know which hosts to target with the tasks defined in the playbook. Ansible also provides various modules that can be used to perform tasks such as managing users, installing packages, and configuring network settings. By using Ansible for configuration management, you can automate the deployment and management of applications and systems, reduce manual errors, and ensure consistency and reliability across your infrastructure.

```
1 #192.168.225.84 ansible_user=maninandadeep ansible_password=
2 localhost ansible_user=sivani ansible_connection=local
```

Figure 49. Ansible: inventory

We deployed in our local machines and also other host.

Figure 50. Ansible: playbook

Above is an ansible playbook that deploys software using docker Compose.

ansible-playbook explanation :

name: "Deploying software" : The name of the playbook, which is "Deploying software".
hosts: "all": The list of hosts on which the playbook will be executed. In this case, it's "all" which means it will be executed on all hosts.
tasks: The list of tasks to execute on each host.
name: "copy docker-compose": The name of the task, which is "copy docker-compose". This task copies the docker-compose.yml file from the local machine to the remote host.
copy: This is an Ansible module used to copy files from the local machine to the remote host.
src: docker-compose.yml: The source file to copy, which is docker-compose.yml.
dest: docker-compose.yml: The destination file on the remote host, which is docker-compose.yml.
mode: '0644': The permissions of the file on the remote host, which is 0644.
name: "Pull docker images": The name of the task, which is "Pull docker images". This task pulls the Docker images specified in the docker-compose.yml file.
command: "docker-compose -f docker-compose.yml pull": This is a shell command that executes the docker-compose pull command to pull the Docker images.
name: "Start docker compose": The name of the task, which is "Start docker compose". This task starts the Docker containers specified in the docker-compose.yml file.
command: "docker-compose -f docker-compose.yml up -d": This is a shell command that executes the docker-compose up -d command to start the Docker containers in the background.

The commands to run the ansible playbook locally are as follows:

- ssh-keygen -t rsa
- ssh-copy-id uname@ip
- ansible-playbook p2.yml -i inventory

After running playbook, images created in localhost and other host are as follows

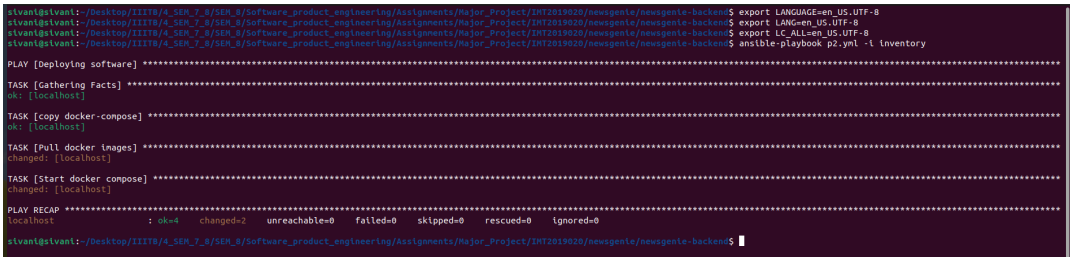


Figure 51. Ansible: Running the playbook in control node

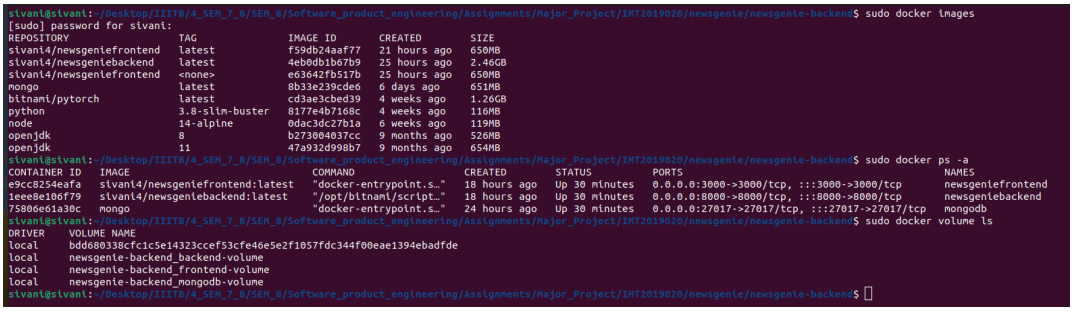


Figure 52. Ansible: After running the playbook - local host

To deploy in another managed host other than local host, enable and start ssh server by using commands *sudo systemctl enable ssh* and *sudo systemctl start ssh*. Copy the ip address(See by using command *ifconfig*) of managed host and connect to it using *ssh-copy-id user@ip_address* and enter password of managed host. After deploying in managed host, docker images, container and volumes are created.

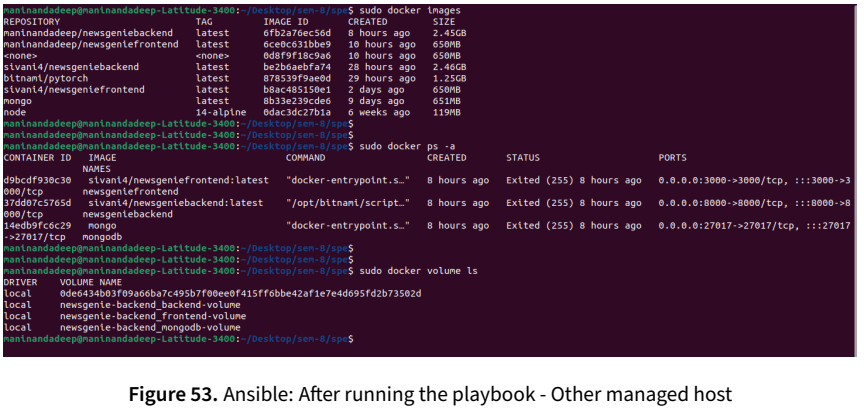
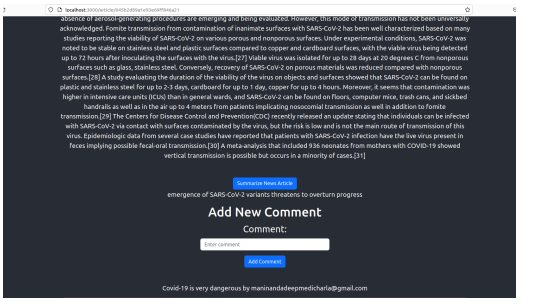


Figure 53. Ansible: After running the playbook - Other managed host

(a) Ansible: Checking if web app running after ansible deploy



(b) Ansible: Checking if web app running after ansible deploy

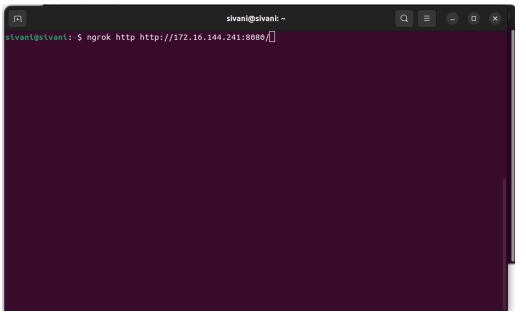
Ngrok and Github Webhooks

Whenever the developer builds the code and commit it and push that to Github, webhooks automatically build the code in Jenkinks. To perform a webhook, the private IP address of the local machine needs to be converted to a public IP address. Automated messages known as "webhooks" are triggered when changes occur. Specifically, in our situation, updates made to the GitHub repository will activate the Jenkins pipeline through the webhook.

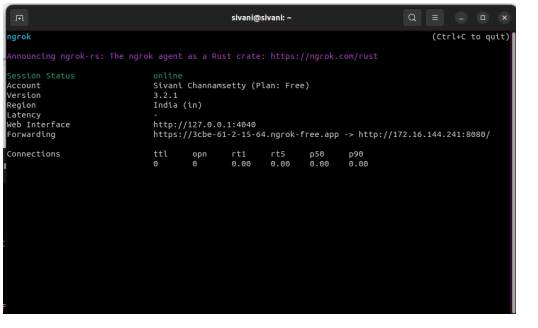
Jenkins can use webhooks for triggering builds. A webhook (or web callback) is a way for an app to provide other applications with real-time information. This can ensure that GitHub will send a webhook to our Jenkins server and our Jenkins application will build and test the changes and send the build status for the commit that was made. However the ip-address must be a public one, hence we use ngrok to create a public address and use that to configure the webhook. Once the webhook is set up, all commits to the remote repository, sends a POST request to the payload ngrok URL, which in turn tells the jenkins server to trigger the build.

Ngrok is a tool that establishes secure tunnels to connect local servers located behind Network Address Translation (NAT) and firewalls to the internet. It features a web interface that provides real-time monitoring of any HTTP traffic that passes through the tunnels. This tool enables users to access the internet via a web server running on their local system, with the port on which the web server is listening to being specified to ngrok.

To start ngrok, use command : ngrok <http of jenkins server> 8080



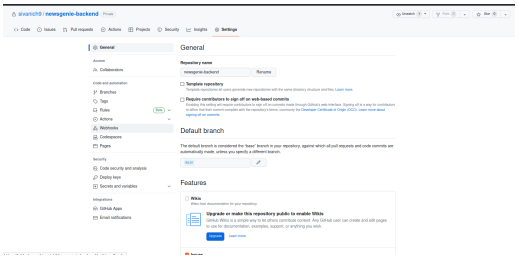
(a) Ngrok: Starting ngrok



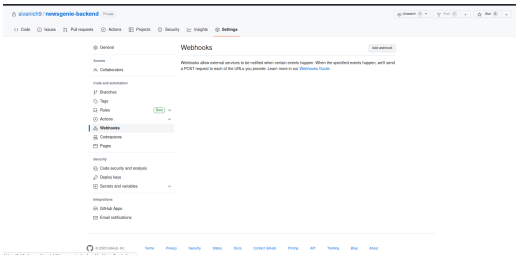
(b) Ngrok: Starting ngrok

Setting Up WebHooks :

Navigate to the settings section of the Github repository, then proceed to the Webhooks tab and select Add webhook. Add the ngrok address in payload URL and the personal access token in secret



(a) Webhook: Setting up webhook



(b) Webhook: Setting up webhook

Run ngrok in terminal and copy the ngrok public ip address

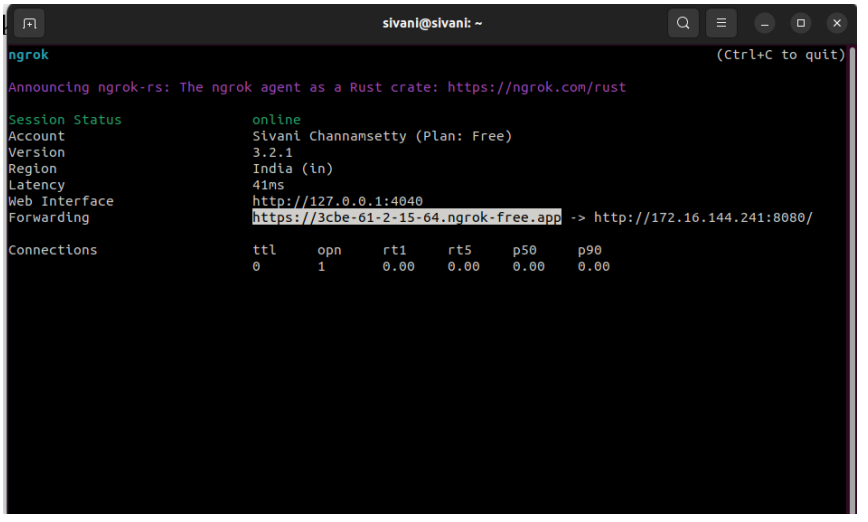


Figure 57. Ngrok: Copying ngrok public ip address

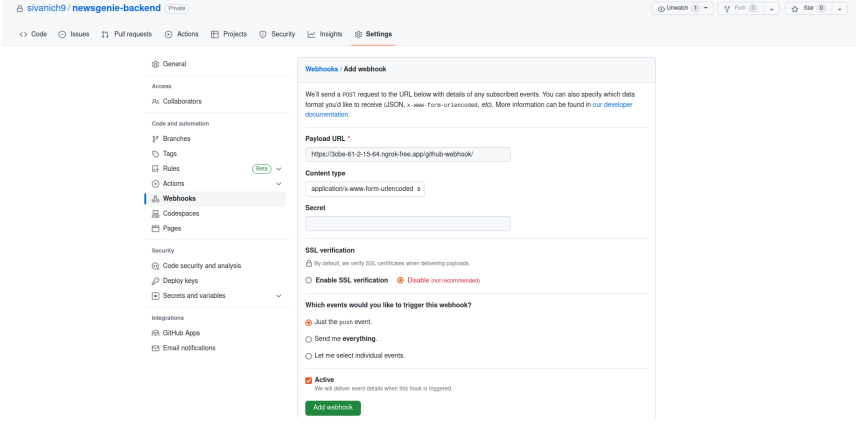


Figure 58. Webhook: Copying ngrok public ip address to github webhooks

If webhook is already set up then just edit it and copy paste the new ngrok public ip address.

Continuous Integration

Continuous integration (CI) is the practice of integrating code changes from multiple developers into a shared repository frequently. It is a crucial component of modern software development, and Jenkins is one of the most popular tools for automating CI pipelines.

Jenkins is an open-source automation server that allows developers to automate their software development processes. It supports a wide range of plugins and integrations, making it an ideal tool for automating the entire software development lifecycle. One way to automate CI with Jenkins is by using a Jenkins pipeline script.

A Jenkins pipeline is a set of steps that define the entire build process, from compiling the code to testing, building, and deploying the application. The pipeline script is written in the Groovy scripting language and is executed on the Jenkins master node. The pipeline can be defined in a Jenkinsfile, which is stored in the source code repository alongside the application code.

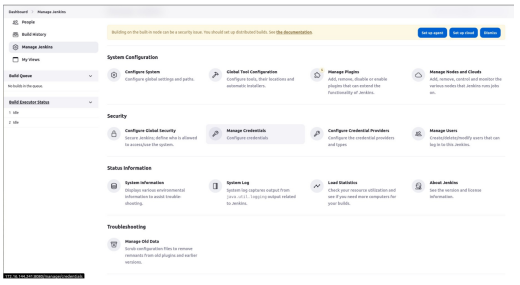
Jenkins pipeline scripts can be used to automate the entire CI/CD pipeline, including the building, testing, and deployment of applications. They can be easily customized and adapted to the specific needs of a project, making it possible to integrate different tools and technologies into the pipeline. By automating the entire pipeline, developers can reduce the time and effort required for testing and deployment, leading to faster release cycles and improved software quality.

Install the plugins git, docker and ansible. Go to Dashboard and the Manage Jenkins and then Manage Plugins and install the required plugins.

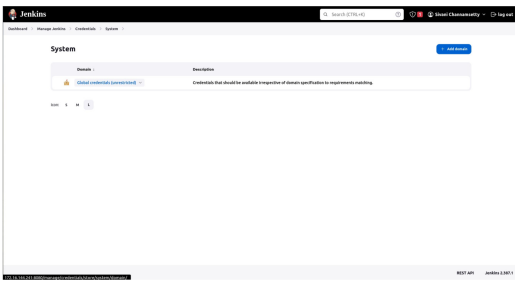
Adding dockerhub and github credentials in jenkins

Go to Manage Jenkins > Manage Credentials > System > Global credentials (unrestricted) > Add Credentials
Adding dockerhub credentials : Choose Kind as Username with password and Scope as Global; provide your docker username and password with a unique ID of your own. This ID is used for referring the username with a password later in your pipelines.

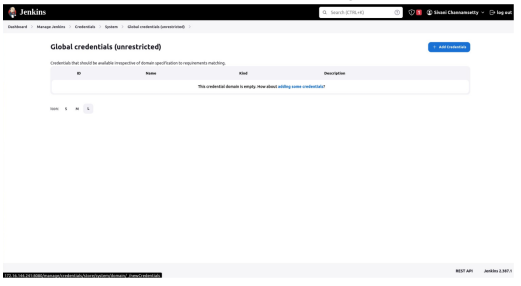
Adding github credentials : In password, I added personal access token of Github(that I have already). To generate new token go to Github and then settings/Developer settings/Personal access tokens/tokens(classic)/Generate new token. But I have it already so I am pasting that in jenkins.



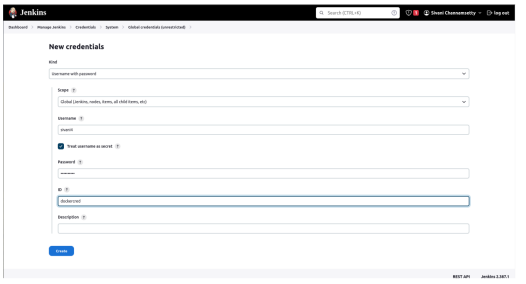
(a) Jenkins: Steps to add dockerhub and github credentials



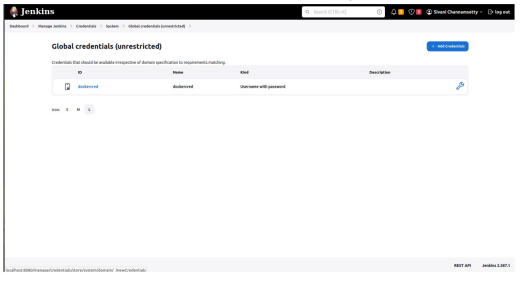
(b) Jenkins: Steps to add dockerhub and github credentials



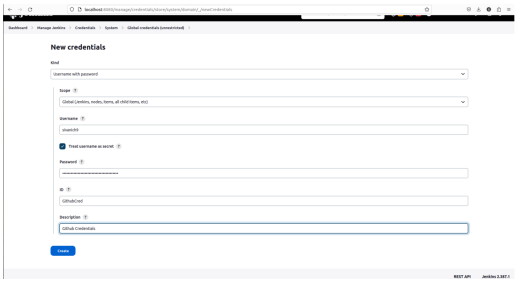
(c) Jenkins: Steps to add dockerhub and github credentials



(d) Jenkins: Steps to add dockerhub and github credentials

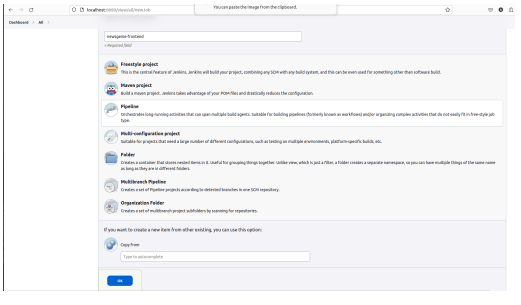


(e) Jenkins: Steps to add dockerhub and github credentials

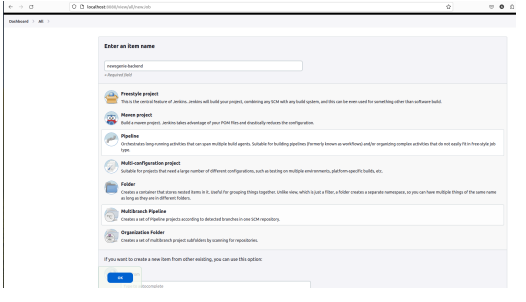


(f) Jenkins: Steps to add dockerhub and github credentials

Now we have to create two jenkins projects for frontend and backend and select pipeline.



(a) Jenkins: Created project for frontend



(b) Jenkins: Created project for backend

Frontend stages : The jenkins pipeline consists of the stages: cloning repository from github, installing react project, testing frontend using *npm test*, building the docker image, pushing the docker image to dockerhub.

Backend stages : The jenkins pipeline consists of the stages: cloning repository from github, installing dependencies, testing backend using *pytest*, building the docker image, pushing the docker

image to dockerhub and deploying the whole project using ansible.

Ansible deploy stage for webapp is included in backend jenkins file.

Frontend

The jenkins pipeline script will be as follows:

The first stage is cloning the code repository from github. This uses the github plugin, takes in the appropriate credentials, url and branch name to download the repository from github.

```
stage('Git Pull stage') {
    steps {
        git url: 'https://github.com/hanzohasashi33/newsgenie-frontend',
        branch: 'main',
        credentialsId : 'GithubCred'
    }
}
```

Figure 61. Jenkins: Cloning stage - Frontend

The second stage is installing node modules using *npm install*. The stage performs the appropriate shell command on the downloaded repository.

```
stage('Install react project'){
    steps{
        script{
            sh 'npm install'
        }
    }
}
```

Figure 62. Jenkins: Installing stage - Frontend

The third stage is testing the code using *npm test*. The stage performs the appropriate shell command on the downloaded repository.

```
stage('Frontend react test'){
    steps{
        script{
            sh 'npm test'
        }
    }
}
```

Figure 63. Jenkins: Testing stage - Frontend

The fourth stage builds a docker image using the presented docker file. It uses the appropriate docker plugins and builds the docker image with the latest tag.

```
stage('Docker build image'){
    steps{
        script{
            sh 'docker build -t sivani4/newsgeniefrontend:latest .'
        }
    }
}
```

Figure 64. Jenkins: Building Docker image stage - Frontend

The fifth stage pushes the built docker image to dockerhub using the appropriate docker jenkins plugins.

```
stage('Push docker image'){
    steps{
        script{
            withDockerRegistry([ credentialsId: "dockercred", url: "" ]){
                sh 'docker push sivani4/newsgeniefrontend:latest'
            }
        }
    }
}
```

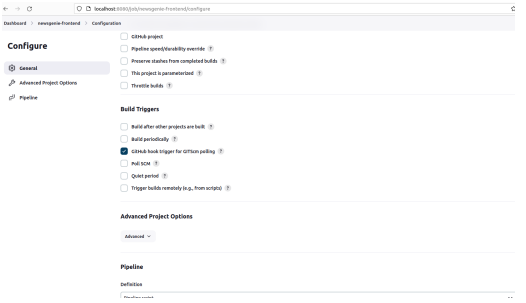
Figure 65. Jenkins: Pushing image stage - Frontend

Jenkinsfile

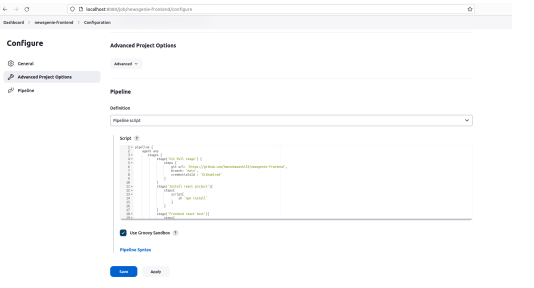
```
pipeline {
    agent any
        stages {
            stage('Git Pull stage') {
                steps {
                    git url: 'https://github.com/hanzohasashi33/newsgenie-frontend',
                    branch: 'main',
                    credentialsId : 'GithubCred'
                }
            }
            stage('Install react project'){
                steps{
                    script{
                        sh 'npm install'
                    }
                }
            }
            stage('Frontend react test'){
                steps{
                    script{
                        sh 'npm test'
                    }
                }
            }
            stage('Docker build image'){
                steps{
                    script{
                        sh 'docker build -t sivani4/newsgeniefrontend:latest .'
                    }
                }
            }
            stage('Push docker image'){
                steps{
                    script{
                        withDockerRegistry([ credentialsId: "dockercred", url: "" ]){
                            sh 'docker push sivani4/newsgeniefrontend:latest'
                        }
                    }
                }
            }
        }
}
```

Figure 66. Jenkins: Jenkins file - Frontend

As we set Github WebHooks, jenkins automatically build the code. Note that we have to run ngrok in our terminal. If we come out of ngrok, it doesn't build. We have to again run ngrok in our terminal then the address and accordingly we have to update the webhook in Github.



(a) Jenkins: Configuring - Frontend



(b) Jenkins: Jenkins pipeline - Frontend

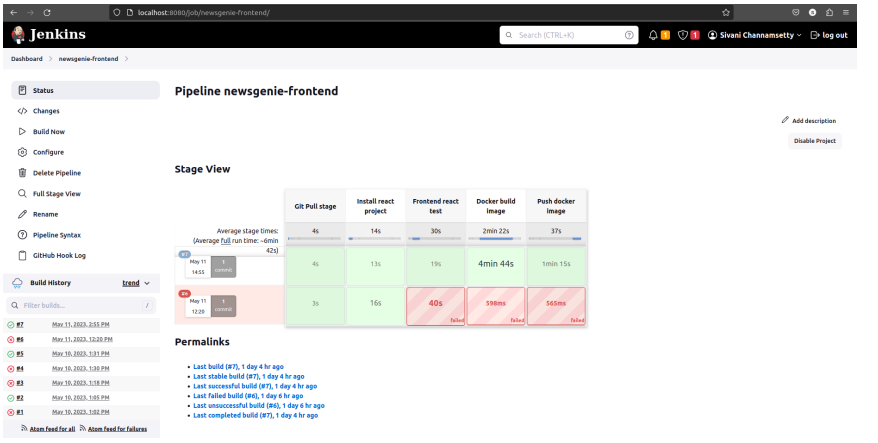


Figure 68. Jenkins: Jenkins stage view - Frontend

Backend

The jenkins pipeline script will be as follows:

The first stage is cloning the code repository from github. This uses the github plugin, takes in the appropriate credentials, url and branch name to download the repository from github.

```
stage('Git Pull stage') {
    steps {
        git url: 'https://github.com/sivanich9/newsgenie-backend',
        branch: 'main',
        credentialsId : 'GithubCred'
    }
}
```

Figure 69. Jenkins: Cloning stage - Backend

The second stage is copying the summarizing model folder from */var/lib/jenkins* to current jenkins workspace. As model is 800 MB, we are unable to push that to github. In newsgenie-backend add folder from summarizer model For jenkins add that folder to */var/lib/jenkins*.

```
stage('Copy summarizing model') {
    steps {
        script{
            sh 'cp -r /var/lib/jenkins/simplet5-epoch-4-train-loss-0.6005-val-loss-1.6554 ./'
        }
    }
}
```

Figure 70. Jenkins: Copying summarizing model - Backend

The third stage is installing node dependencies. The stage performs the appropriate shell command on the downloaded repository.

```
stage('Install dependencies'){
    steps{
        script{
            sh '''
                pip3 install simplet5
                pip3 install "fastapi[all]"
                pip3 install uvicorn
                pip3 install loguru
                pip3 install pymongo
                pip3 install pytest
                pip3 install mongomock
            '''
        }
    }
}
```

Figure 71. Jenkins: Installing dependencies - Backend

The fourth stage is testing the code using *pytest*. The stage performs the appropriate shell command on the downloaded repository.

```
stage('Backend test'){
    steps{
        script{
            sh 'python3 -m pytest test_main.py --disable-warnings'
        }
    }
}
```

Figure 72. Jenkins: Testing stage - Backend

The fifth stage builds a docker image using the presented docker file. It uses the appropriate docker plugins and builds the docker image with the latest tag.

```
stage('Docker build image'){
    steps{
        script{
            sh 'docker build -t sivani4/newsgeniebackend:latest .'
        }
    }
}
```

Figure 73. Jenkins: Building Docker image stage - Backend

The sixth stage pushes the built docker image to dockerhub using the appropriate docker jenkins plugins.

```
stage('Push docker image'){
    steps{
        script{
            withDockerRegistry([ credentialsId: "dockercred", url: "" ]){
                sh 'docker push sivani4/newsgeniebackend:latest'
            }
        }
    }
}
```

Figure 74. Jenkins: Pushing image stage - Backend

The seventh and final stage using the configuration management tool ansible and its appropriate jenkins-ansible plugins to execute the given playbook across the hosts given in the inventory file.

```
stage('Ansible deploy'){
    steps{
        script{
            sh "/usr/bin/pip3 install docker"
            sh "ansible-playbook p2.yml -i inventory"
        }
    }
}
```

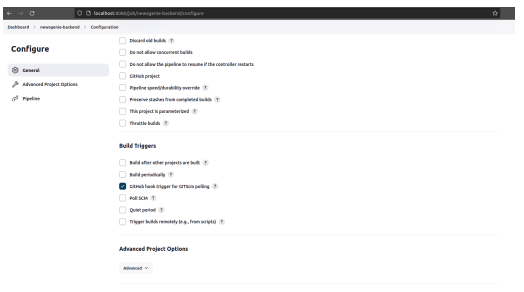
Figure 75. Jenkins: Ansible deploy stage

Jenkinsfile

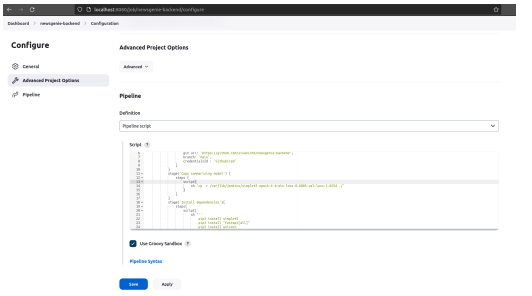
```
pipeline {
    agent any
      stages {
          stage('Git Pull stage') {
              steps {
                  git url: 'https://github.com/sivanich9/newsgenie-backend',
                  branch: 'main',
                  credentialsId : 'GithubCred'
              }
          }
          stage('Copy summarizing model') {
              steps {
                  script{
                      sh 'cp -r /var/lib/jenkins/simplet5-epoch-4-train-loss-0.6005-val-loss-1.6554 ./'
                  }
              }
          }
          stage('Install dependencies'){
              steps{
                  script{
                      sh '''
                          pip3 install simplet5
                          pip3 install "fastapi[all]"
                          pip3 install uvicorn
                          pip3 install loguru
                          pip3 install pymongo
                          pip3 install pytest
                          pip3 install mongomock
                      '''
                  }
              }
          }
          stage('Backend test'){
              steps{
                  script{
                      sh 'python3 -m pytest test_main.py --disable-warnings'
                  }
              }
          }
          stage('Docker build image'){
              steps{
                  script{
                      sh 'docker build -t sivani4/newsgeniebackend:latest .'
                  }
              }
          }
          stage('Push docker image'){
              steps{
                  script{
                      withDockerRegistry([ credentialsId: "dockercred", url: "" ]){
                          sh 'docker push sivani4/newsgeniebackend:latest'
                      }
                  }
              }
          }
          stage('Ansible deploy'){
              steps{
                  script{
                      sh "/usr/bin/pip3 install docker"
                      sh "ansible-playbook p2.yml -i inventory"
                  }
              }
          }
      }
}
```

Figure 76. Jenkins: Jenkins file - Backend

As we set Github WebHooks, jenkins automatically build the code. Note that we have to run ngrok in our terminal. If we come out of ngrok, it doesn't build. We have to again run ngrok in our terminal then the address and accordingly we have to update the webhook in Github.



(a) Jenkins: Configuring - Backend



(b) Jenkins: Jenkins pipeline - Backend

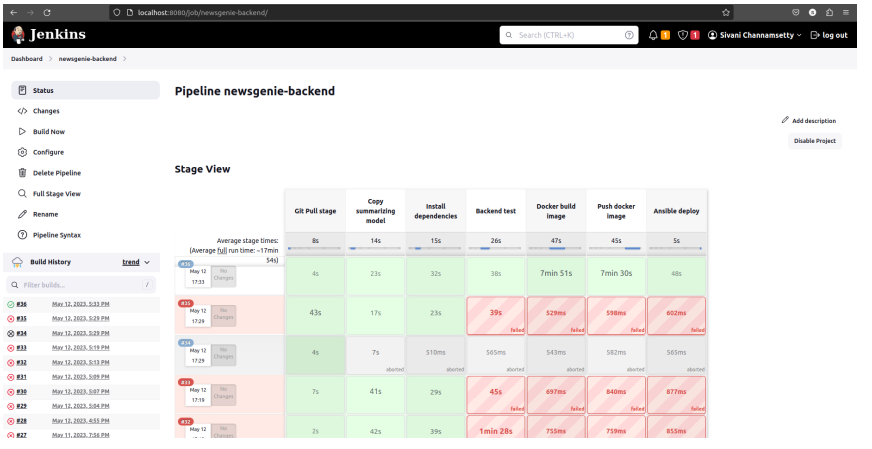
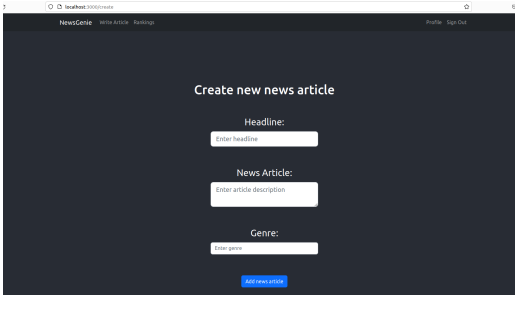
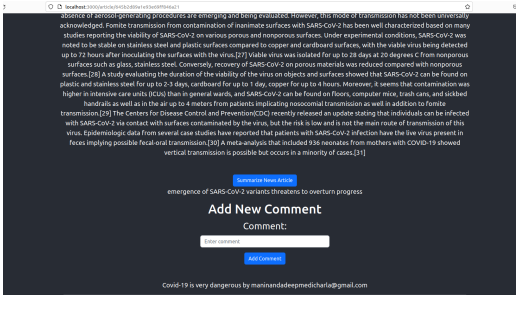


Figure 78. Jenkins: Jenkins stage view - Backend

Once building of jenkins pipeline stage is done, we can check in our managed hosts if project deploying is done successfully.



(a) Jenkins: Checking for successful deployment of project



(b) Jenkins: Checking for successful deployment of project

The above CI pipeline goes from downloading the code to pushing it to the corresponding hosts and executing them appropriately. Performing this automates the entire development life cycle and reduces the margins of man-made errors.

Monitoring

ELK Stack

Monitoring is a critical aspect of modern software development and is essential for ensuring that applications are performing as expected. The ELK stack is one of the most popular tools for monitoring applications and analyzing logs.

The ELK stack is a combination of three open-source tools: Elasticsearch, Logstash, and Kibana. Elasticsearch is a distributed search and analytics engine that is used to store and search large volumes of data. Logstash is a tool for processing and parsing log data, and Kibana is a web-based interface for visualizing and analyzing data.

To monitor an application using the ELK stack, developers can configure the application to generate logs using a logging framework like Log4j, loguru, etc. The logs generated by the application can then be processed by Logstash and indexed in Elasticsearch. Once the logs are indexed, they can be visualized and analyzed using Kibana.

loguru is a popular logging framework for python applications and can be used to generate logs that can be analyzed using the ELK stack. By configuring loguru to generate logs in a specific format, developers can ensure that the logs are easily processed and indexed by Logstash. We download loguru using pip and add logging formats to the file. The log file generated from the above looks similar to the below log file image.

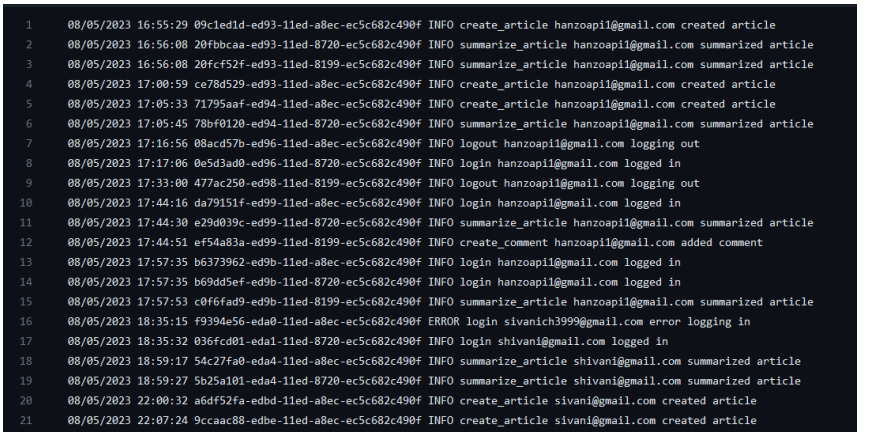


Figure 80. Logging: NewsGenie Log files

Uploading the log file to monitor using the ELK stack can be done in 2 ways. One is by using logstash and the second is directly uploading the log file and giving the grok pattern.

The uploading file way is done in the following manner. We login to elastic cloud home and create a deployment for the same. Once the deployment is created, we upload the calculator's log file to the deployment, and create an index and a data view by supplying a corresponding a grok pattern as well.

To create a deployment, we click the press deployment button in elastic cloud home and then give the corresponding details for the deployment like name, region etc. This will create a deployment and provide the credentials and a cloud id which we can use for logstash and other tools.

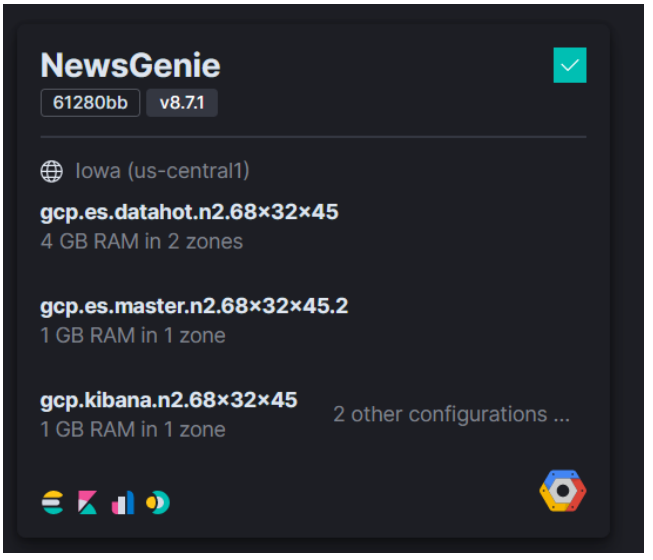


Figure 81. ELK: deployments

Once the deployment is created we go to the stack management, upload a file and create a corresponding index and dataview. We also supply a corresponding grok pattern to the generated logs based on the pattern we supplied in the log file.

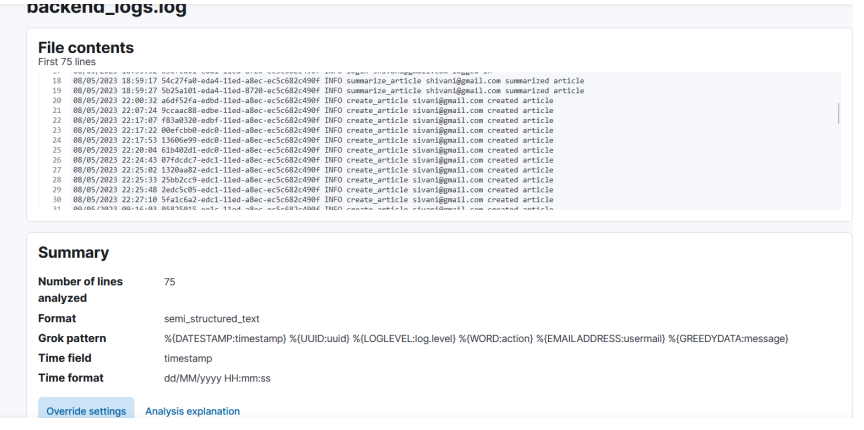


Figure 82. ELK: File Upload

Once the grok pattern is matched and accepted with the logs, the data file stats is created and imported accordingly.

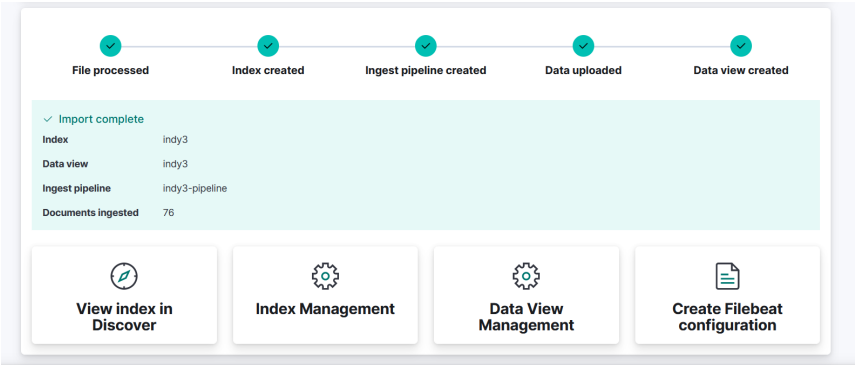


Figure 83. ELK: File Stats

Once the data is ingested we can create an index and dataview, and can go to kibana to create dashboards and visualizations for the uploaded data.

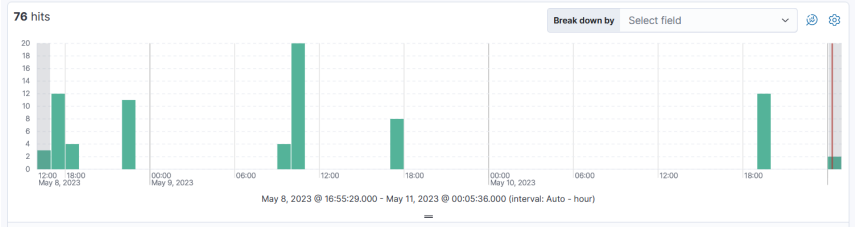


Figure 84. ELK: Discover Index

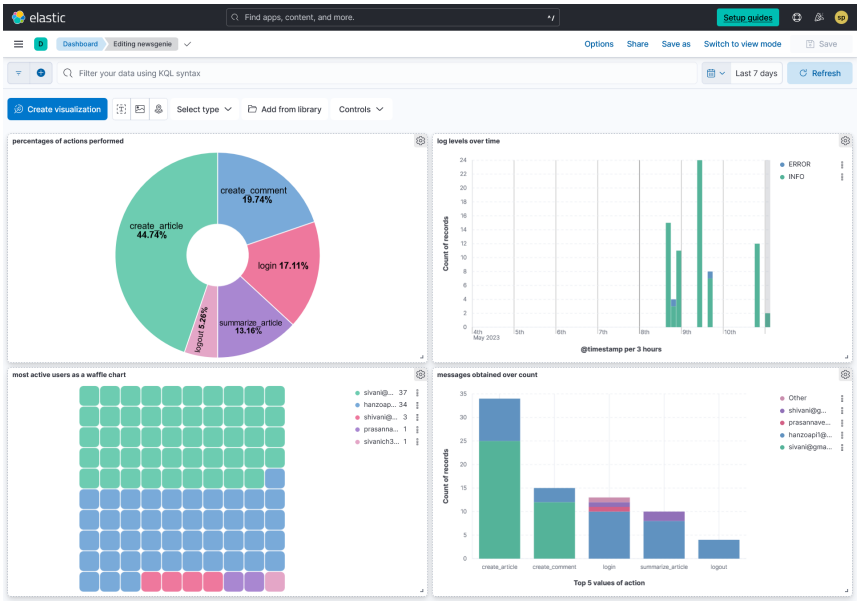


Figure 85. ELK: Dashboard

In summary, the ELK stack is a powerful tool for monitoring applications and analyzing logs.

By generating logs using a logging framework like loguru and processing them using Logstash and Elasticsearch, developers can gain valuable insights into the performance and behavior of their applications. The visualizations provided by Kibana make it easy to identify and troubleshoot issues, leading to faster resolution times and improved software quality.

Log automatic processing

We have already looked at installing Logstash in the Installations section. Once we have installed the logstash, to automatically process the log files follow the below steps:

- Open the .bashrc file by running the command `sudo nano ~/.bashrc`.
- Add the following lines to the .bashrc file:
  `export LOGSTASH_HOME=/usr/share/logstash`
  `export PATH=$PATH:$LOGSTASH_HOME/bin`
- Save the changes and exit the editor. Reload the .bashrc file to apply the changes by running the command `source ~/.bashrc`.
- Navigate to the Logstash directory by running the command cd /usr/share/logstash.
- Start Logstash with the configuration file by running the command: `sudo bin/logstash -f /path/to/log_ingestion.conf`
- Once Logstash is running, it will continuously monitor the backend_logs.log file. Any modifications made to this file will be updated in the running terminal and will be sent to the ELK stack for indexing.
- Keep the terminal running Logstash active and connected to the ngrok service to ensure continuous updates to the logs.

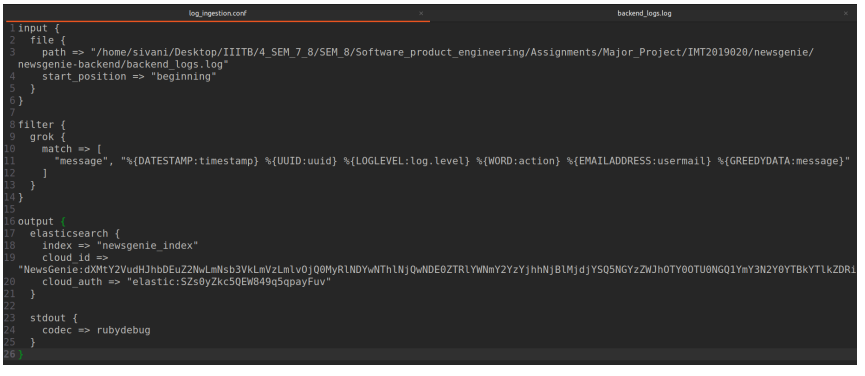


Figure 86. Logstash: Logstash configuration file

Sentry

Sentry is an open-source error monitoring and reporting platform that helps you monitor and fix crashes in real time. Here are the steps to using Sentry to monitor your application:

- Install the Sentry SDK in your application. The Sentry SDK is available for most programming languages and frameworks. The SDK will capture any errors or exceptions in your application and report them to your Sentry project.
- Create a Sentry project. Go to sentry.io and sign up for a free account. Click "Create Project" and give your project a name. You will get a project DSN (Domain Secret Name) that you will use to configure the Sentry SDK.
- Configure the Sentry SDK with your project DSN. Pass the DSN to the SDK initialization to link it to your Sentry project. The SDK will then start sending error reports to that project.

- Review errors in Sentry. When errors occur in your application, you will see them show up in the Sentry interface. You can filter, search, and group errors to identify trends and issues.

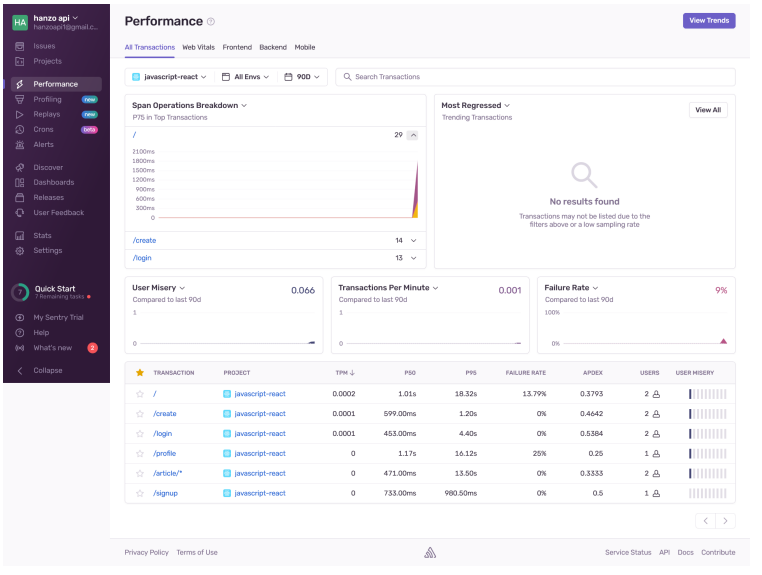


Figure 87. Sentry: All Transactions performance

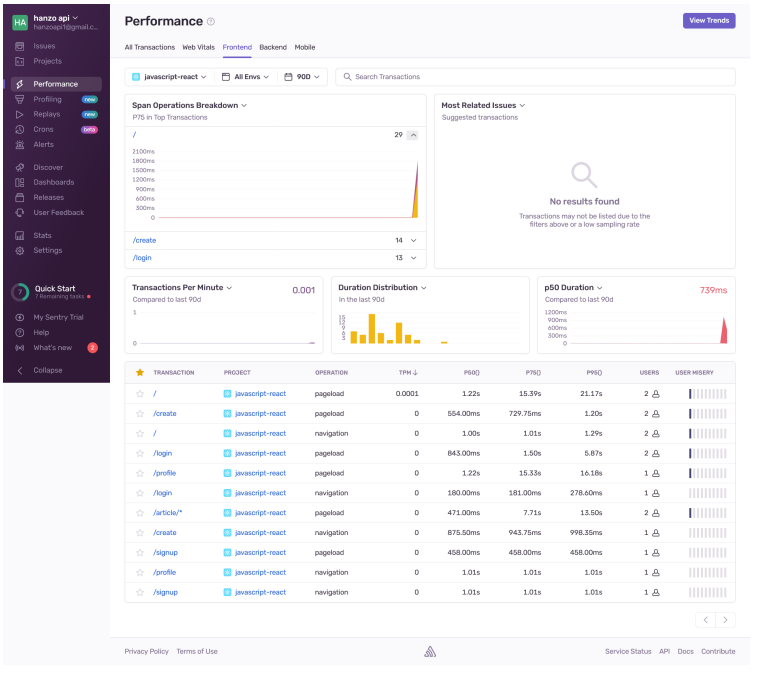


Figure 88. Sentry: Frontend Performance

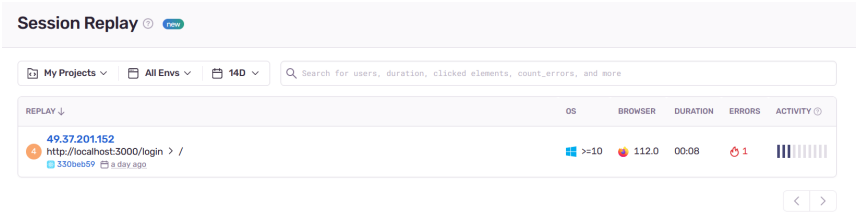


Figure 89. Sentry: Error Report

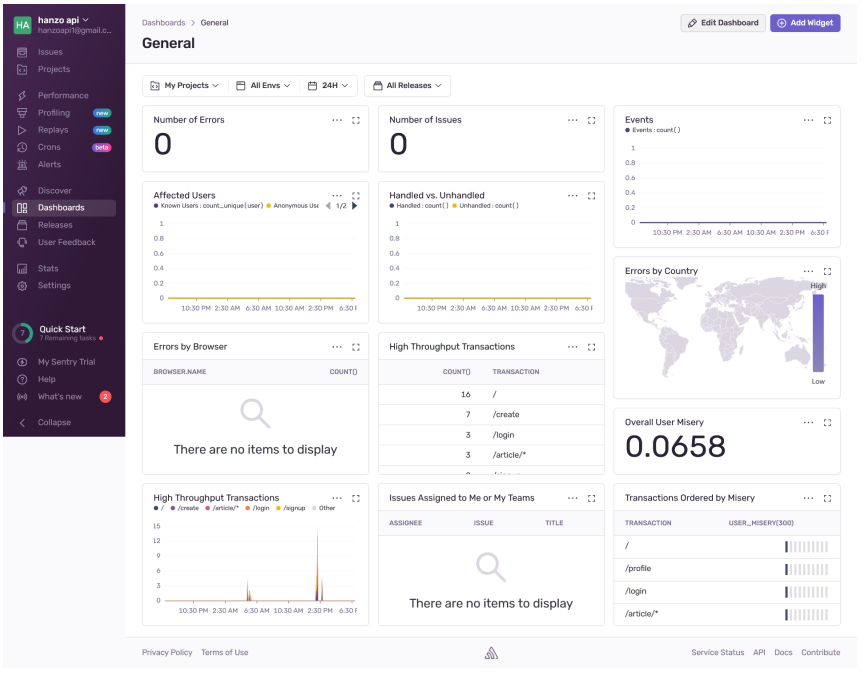


Figure 90. Sentry: Dashboard

Conclusion

In conclusion, this project has successfully integrated a scientific calculator with a DevOps toolchain using various tools such as GitHub, Jenkins, Ansible, JUnit, Maven, Docker, and the ELK stack. The pipeline has been designed to ensure that the code is thoroughly tested, built, containerized, and deployed efficiently. The ELK stack has been utilized for monitoring of the code logs, enabling developers to identify and fix any issues quickly. The integration of a scientific calculator with the DevOps toolchain has streamlined the development process, improving the efficiency of the development process. The pipeline has ensured that the code is of high quality, and any issues are identified and fixed in real-time.

Overall, this project has demonstrated the importance of integrating a scientific calculator with the DevOps toolchain. The pipeline designed can be replicated for future projects, and it will provide valuable insights into the efficiency of the development process. The project has showcased how DevOps tools can be utilized to streamline the development process, ultimately providing developers with valuable tools for high quality software development.

Links

Github

- newsgenie-frontend : newsgenie-frontend
- newsgenie-backend : newsgenie-backend

Dockerhub

- newsgenie-frontend : newsgenie-frontend
- newsgenie-backend : newsgenie-backend

Summarizer model

- newsgenie-frontend : Summarizer model

References

Docker. https://www.docker.com/. Accessed: 2023-03-10.

Dockerize maven project. https://stackoverflow.com/questions/27767264/how-to-dockerize-maven-project-and-how-many-ways-to-accomplish-it. Accessed: 2023-03-15.

Elastic data view. https://www.elastic.co/guide/en/kibana/current/data-views.html. Accessed: 2023-03-17.

How to install node.js on ubuntu 20.04. https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-20-04. Accessed: 2023-05-10.

Install mongodb community edition on ubuntu. https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/. Accessed: 2023-05-10.

Jenkins docker permissions. https://stackoverflow.com/questions/47854463/docker-got-permission-denied-while-trying-to-connect-to-the-docker-daemon-socke. Accessed: 2023-03-16.