CS816: Software Production Engineering
Major-Project : Flight Booking System with Devops

IMT2019034 Gunin Jain, IMT2019066 Pratyush Upadhyay

May 2023

1 Introduction

Flight Booking system is a website built using 3-tier architecture involving React frontend, Spring Boot backend and MySQL database. It allows users to search for flights based on different parameters, book tickets and exchange seats. It allows admins to add airports, aeroplanes and Schedule Flight Trips and remove them.

2 Features implemented

2.1 Admin functionalities

- Add/Delete airports
- Add/Delete airplanes
- Add flight trips based on added airplanes and airports

2.2 User functionalities

- Search for flight trips based on number of seats
- Sort and filter search results of flight trips based on departure/arrival time and price
- Book tickets for multiple passengers with option to specify seats to exchange with

3 Repository Links

- Github Repository: https://github.com/pratyu2364/Flight-Booking-System
- DockerHub Frontend Repository: https://hub.docker.com/repository/docker/guninjain/jahazbooker-frontend
- DockerHub Backend Repository: https://hub.docker.com/repository/docker/guninjain/jahazbooker-backend

4 DevOps

4.1 What is DevOps ?

- DevOps is a set of practices, tools, and a cultural philosophy that automates and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.
- A DevOps team includes developers and IT operations working collaboratively throughout the product life-cycle, in order to increase the speed and quality of software deployment. It's a new way of working, a cultural shift, that has significant implications for teams and the organizations they work for.

- Under a DevOps model, development and operations teams are no longer "siloed." Sometimes, these two teams merge into a single team where the engineers work across the entire application lifecycle — from development and test to deployment and operations — and have a range of multidisciplinary skills.
- DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration.
- DevOps values are sometimes applied to teams other than development. When security teams adopt a DevOps approach, security is an active and integrated part of the development process. This is called DevSecOps.
- The DevOps lifecycle consists of eight phases representing the processes, capabilities, and tools needed for development (on the left side of the loop) and operations (on the right side of the loop). Throughout each phase, teams collaborate and communicate to maintain alignment, velocity, and quality.

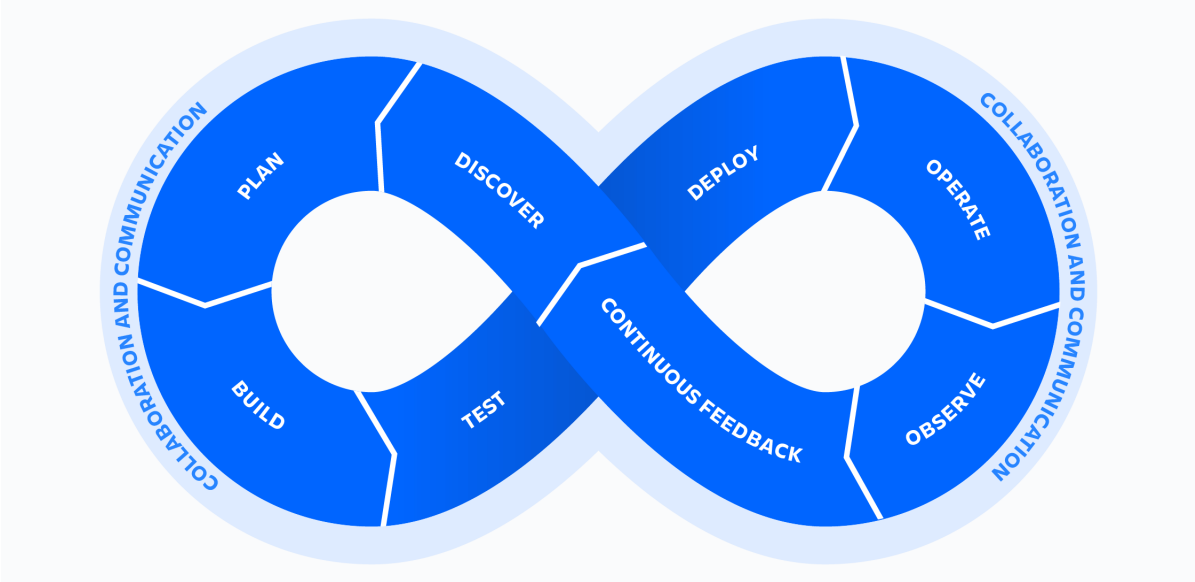

Figure 1: Devops Lifecycle

- DevOps practices
  1. **Continuous Integration** : Continuous integration is the practice of automating the integration of code changes into a software project. It allows developers to frequently merge code changes into a central repository where builds and tests are executed. This helps DevOps teams address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.
  2. **Continuous delivery** : Continuous delivery expands upon continuous integration by automatically deploying code changes to a testing/production environment. It follows a continuous delivery pipeline, where automated builds, tests, and deployments are orchestrated as one release workflow.
  3. **Situational awareness** : It is vital for every member of the organization to have access to the data they need to do their job as effectively and quickly as possible. Team members need to be alerted of failures in the deployment pipeline — whether systemic or due to failed tests — and receive timely updates on the health and performance of applications running in production. Metrics, logs, traces, monitoring, and alerts are all essential sources of feedback teams need to inform their work.
  4. **Automation** : Automation is one of the most important DevOps practices because it enables teams to move much more quickly through the process of developing and deploying high-quality software. With automation the simple act of pushing code changes to a source code repository can trigger a build, test, and deployment process that significantly reduces the time these steps take.

5. **Infrastructure as Code** : Whether your organization has an on-premise data center or is completely in the cloud, having the ability to quickly and consistently provision, configure, and manage infrastructure is key to successful DevOps adoption. Infrastructure as Code (IaC) goes beyond simply scripting infrastructure configuration to treating your infrastructure definitions as actual code: using source control, code reviews, tests, etc.
6. **Microservices** : Microservices is an architectural technique where an application is built as a collection of smaller services that can be deployed and operated independently from each other. Each service has its own processes and communicates with other services through an interface. This separation of concerns and decoupled independent function allows for DevOps practices like continuous delivery and continuous integration.
7. **Monitoring** : DevOps teams monitor the entire development lifecycle — from planning, development, integration and testing, deployment, and operations. This allows teams to respond to any degradation in the customer experience, quickly and automatically. More importantly, it allows teams to "shift left" to earlier stages in development and minimize broken production changes.

4.2 Why Devops ?

1. **Shorter Development Cycles, Faster Innovation**
   When development and operations teams are in separate silos, it's usually difficult to tell if an application is ready for operations. When development teams simply turn over an application, the operations' cycle times are extended needlessly.

   With a combined development and operations team, applications are ready for use much more quickly. This is important, since companies succeed based on their ability to innovate faster than their competitors do.

2. **Reduced Deployment Failures, Rollbacks, and Time to Recover**
   Part of the reason teams experience deployment failures is due to programming defects. The shorter development cycles with DevOps promote more frequent code releases. This, in turn, makes it easier to spot code defects. Therefore, teams can reduce the number of deployment failures using agile programming principles that call for collaboration and modular programming. Rollbacks are similarly easier to manage because, when necessary, only some modules are affected.

   Time to recover is an important issue because some failure has to be expected. But recovery is much faster when the development and operations teams have been working together, exchanging ideas and accounting for both teams' challenges during development.

3. **Improved Communication and Collaboration**
   DevOps improves the software development culture. Combined teams are happier and more productive. The culture becomes focused on performance rather than individual goals. When the teams trust each other, they can experiment and innovate more effectively. The teams can focus on getting the product to market or into production, and their KPIs should be structured accordingly.

   It's no longer a matter of "turning over" the application to operations and waiting to see what happens. Operations doesn't need to wait for a different team to troubleshoot and fix a problem. The process becomes increasingly seamless as all individuals work toward a common goal.

4. **Increased Efficiencies**
   Increased efficiency helps to speed the development process and make it less prone to error. There are ways to automate DevOps tasks. Continuous integration servers automate the process of testing code, reducing the amount of manual work required. This means that software engineers can focus on completing tasks that can't be automated.

   Acceleration tools are another opportunity for increasing efficiency. For example:

   - Scalable infrastructures, such as cloud-based platforms, increase the access the team has to hardware resources. As a result, testing and deployment operations speed up.
   - Build acceleration tools can be used to compile code more quickly.

- Parallel workflows can be embedded into the continuous delivery chain to avoid delays; one team waits for another to complete its work.
- Using one environment avoids the useless task of transferring data between environments. This means you don't have to use one environment for development, a different environment for testing, and a third for deployment.

5. **Reduced Costs and IT Headcount**
   All of the DevOps benefits translate to reduced overall costs and IT headcount requirements.

4.3 Technology Stack

Frontend → React
Backend → Java Spring Boot
Database → MySQL
Testing → Junit
Source Code Management → GitHub
Containerization → Docker
Container image repo → Docker Hub
Automation of Continuous Integration, Continuous Deployment → Jenkins
Continuous delivery and deployment → Ansible
Deployment platform → Minikube
Continuous Monitoring → ELK stack

4.4 Application Architecture and functionalities

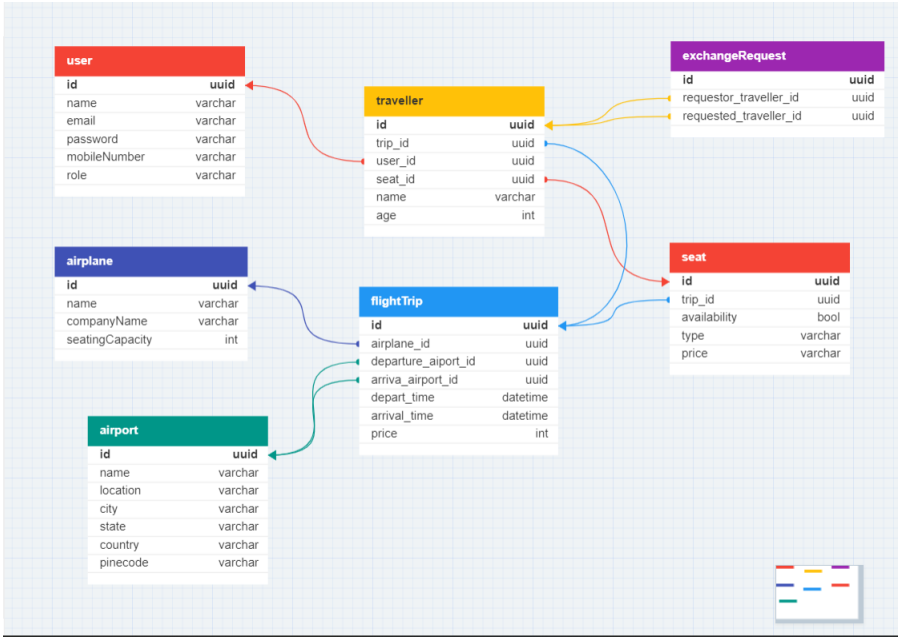

Figure 2: Database Diagram

Actor	Description	Status
Admin		
1	Should be able to Login	Completed
2	Adding and Deleting Airplanes	Completed
3	Adding and Deleting Airports	Completed
4	Scheduling and Cancelling Flight Trips	Completed
User		
1	Search Flights	Completed
2	Sort Flights based on Price, Arrival Time, Departure Time	Completed
3	Filter Flights based on Airlines, Price range	Completed
4	Book Flights for multiple passengers according to their preferred available seats	Completed
5	Request other travellers to exchange seat	Completed

Figure 3: Functionality Requirements

4.5 API Documentation

Endpoint	HTTP Method	Input	Description
/api/v1/auth/register	POST	RegisterRequestDto : RegisterRequest	Registration for User
/api/v1/auth/register-admin	POST	RegisterRequestDto: RegisterRequest	Register for Admin
/api/v1/auth/authenticate	POST	AuthenticationRequestDto: AuthenticationRequest	Login for both
/admin/add-airplane	POST	airplaneDto: Airplane	Admin can add Airplanes
/admin/get-all-airplanes	GET	NIL	Admin can see the list of all Airplanes added
/admin/delete-airplane/	DELETE	airplane_id: String	Admin can delete Airplanes
/admin/add-airport	POST	AirportDto: Airport	Admin can add Airports

Figure 4: API Documentation

/admin/add-airport	POST	AirportDto: Airport	Admin can add Airports
/admin/get-all-airport	GET	NIL	Admin can see the list of all Airports added
/admin/delete-airport/{airport_id}	DELETE	airport_id: String	Admin can delete Airports
/admin/add-flight-trips	POST	flightTripDto: FlightTrip	Admin can add Flight Trips
/admin/get-all-flight-trips	GET	NIL	Admin can see the list of all Flight Trips added
/admin/delete-flight-trip/{flighttrip_id}	DELETE	flightTrip_id	Admin can delete Flight Trips
/user/book	POST	Travellers :	Book tickets for

Figure 5: API Documentation

		TravellerListDto	a list of travellers given user id and flight trip id
/user/{trip_id}/get-all-seats	GET	Trip_id : UUID	Fetch all seats with given trip id
/user/search-all	GET	Dept_city : String, arrival_city : String, date : Date	Search for flight trips based on departure city, arrival city and departure date
/user/search-all-seats	GET	Dept_city : String, arrival_city : String, date : Date, seats_required: int	Search for flight trips based on departure city, arrival city and departure date and number of seats
/user/get-all-airport	GET	NIL	Find all airports
/user/{tripId}/seats/available	GET	tripId : UUID	Get the list of available seats for a flight trip with given tripId
/user/{tripId}/seats/all	GET	tripId:UUID	Get all seats for a flight trip with given id
/user/get-traveller/{email}	GET	email : String	Get all travellers booked by a user with given email id

Figure 6: API Documentation

4.6 Snapshots of working Site

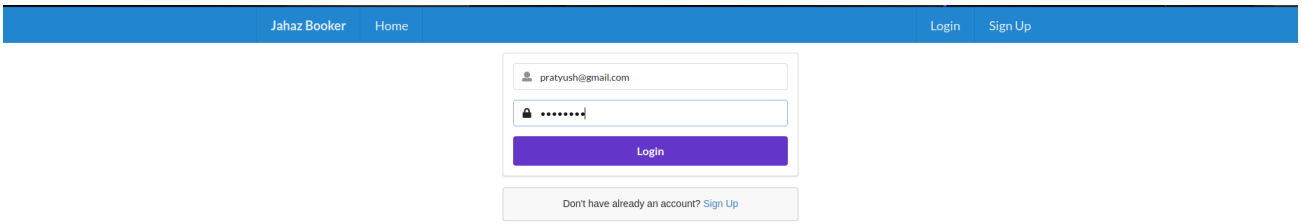

Figure 7: login page

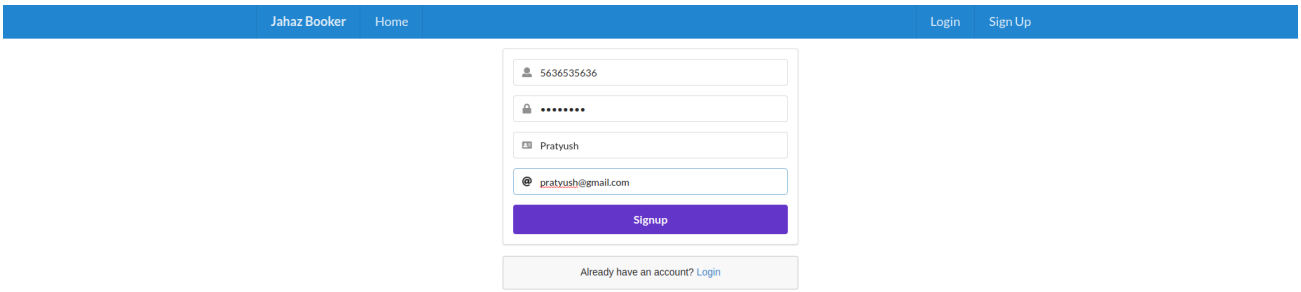

Figure 8: Signup Page

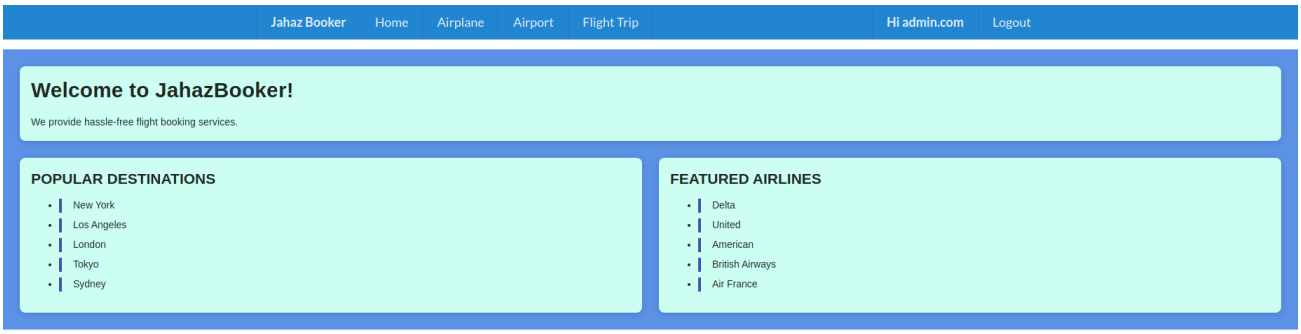

Figure 9: Home Page

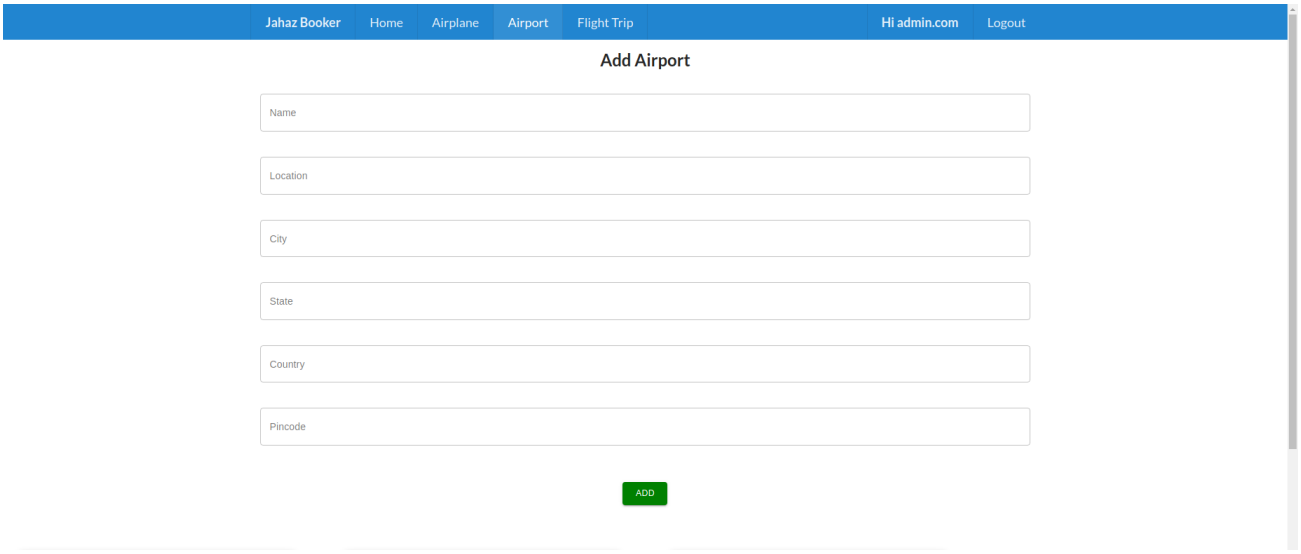

Figure 10: Add Airports Page

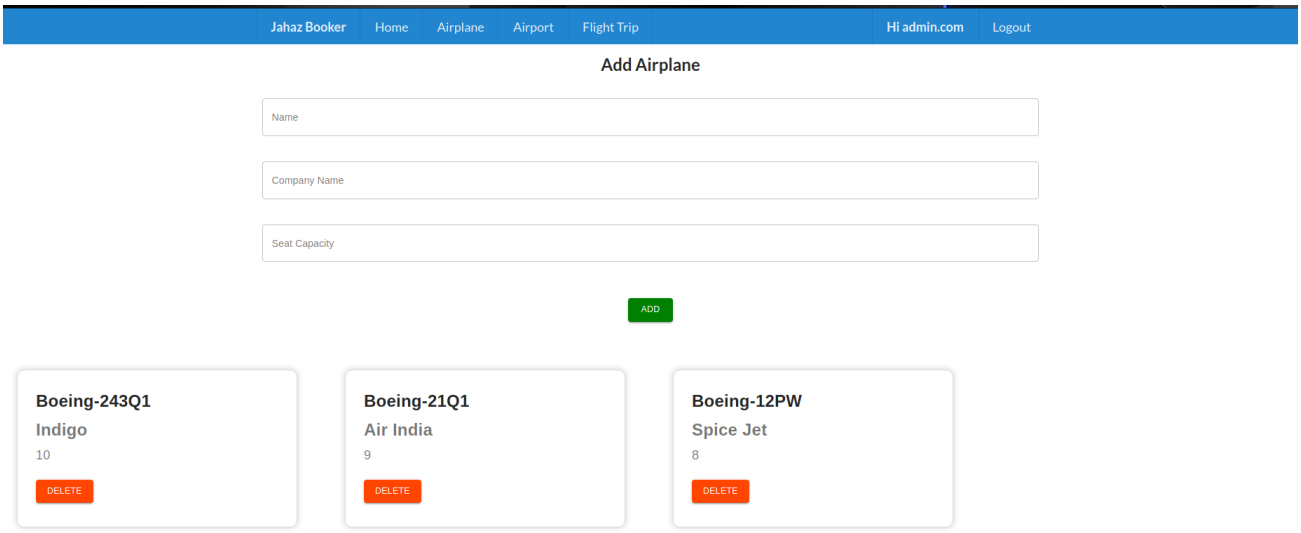

Figure 11: Add Airplane Page

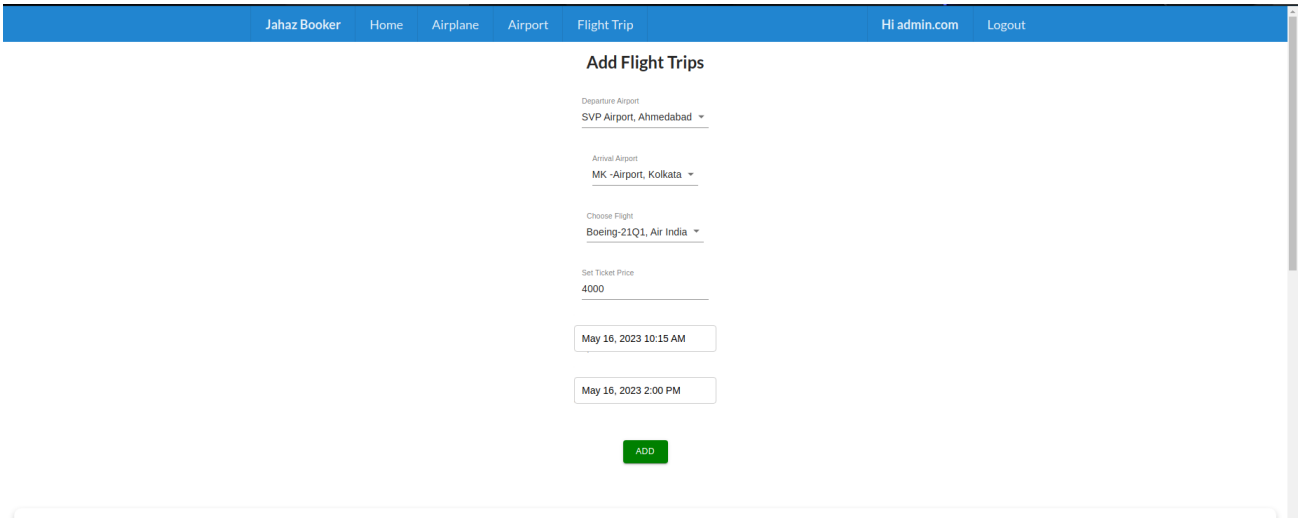

Figure 12: Add Flight Trips Page

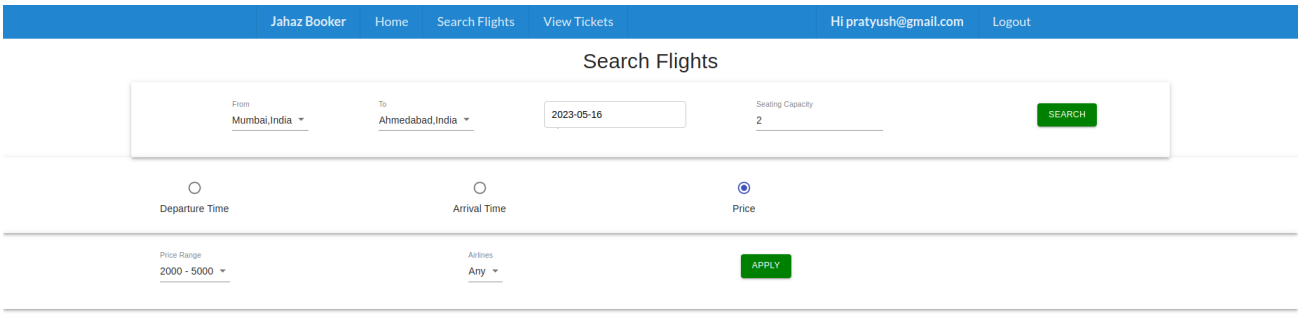

Figure 13: Search FLight Trips Page

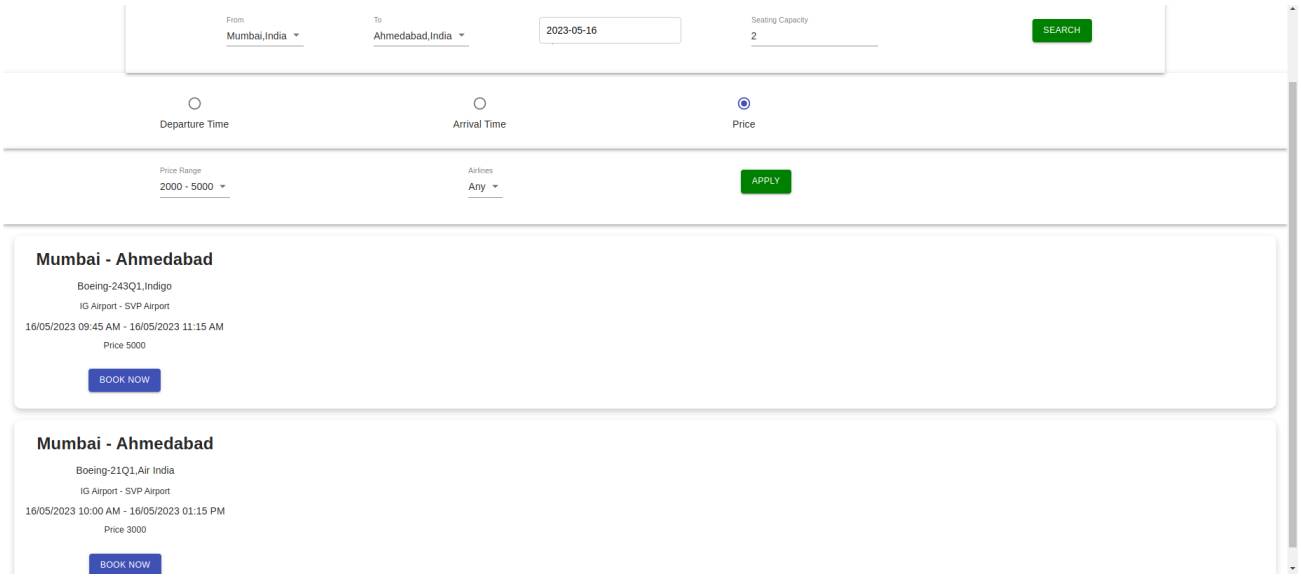

Figure 14: Search Flight Trips Page(continued)

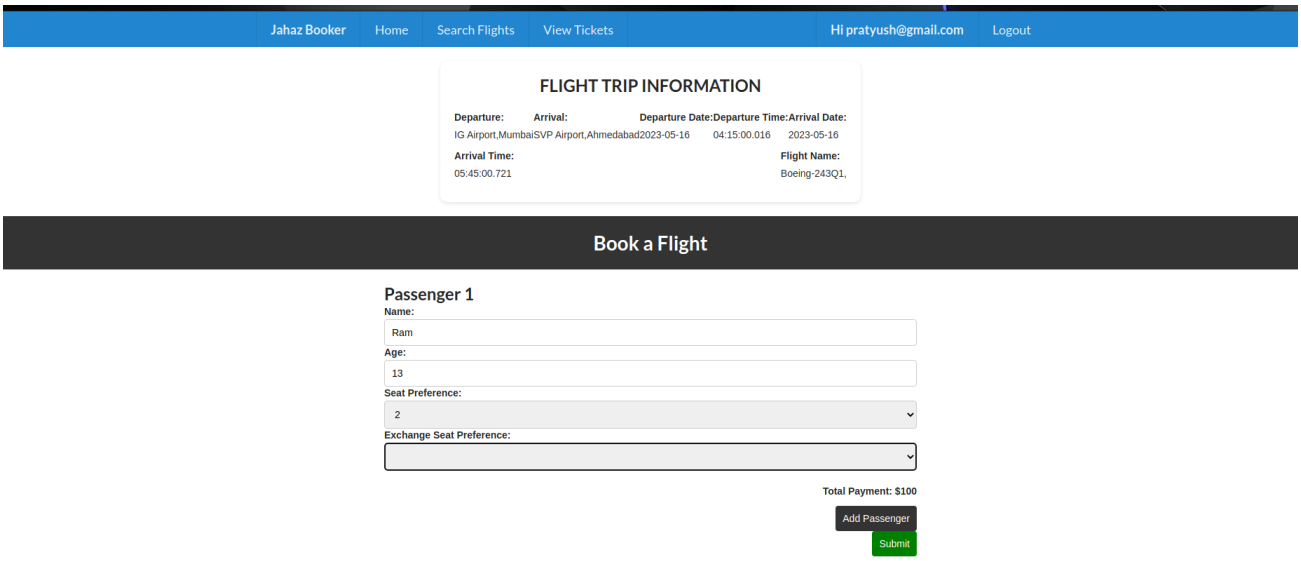

Figure 15: Book Flight Trip Page

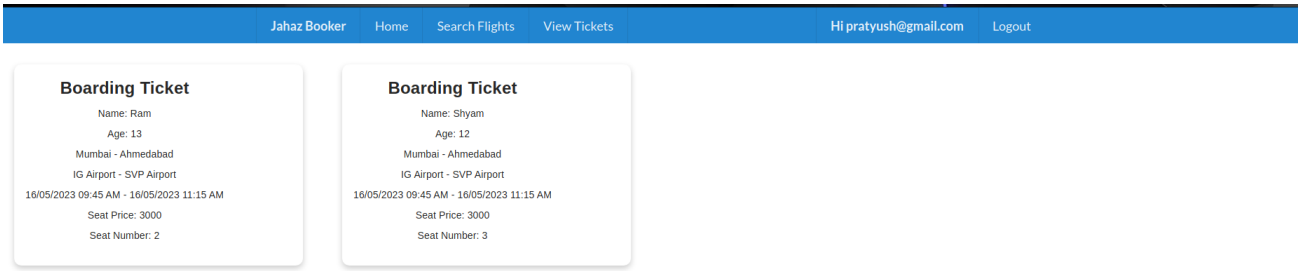

Figure 16: View Ticket Page

5 SOFTWARE DEVELOPMENT LIFE CYCLE

5.1 Spring Boot Backend

Spring boot is a framework based on Java that allows building MVC applications quickly and easily. Benefit of using spring boot was we get support of an embedded tomcat server which makes the developed code production-ready, opinionated starter dependencies that simplify the build process. It can be easily integrated with JPA-compliant persistent databases without any complicated XML configurations. Moreover, spring boot is a maven project (unless you choose a gradle version) and hence it is easy to run and test using maven commands.

- First we need to set up a spring boot project. For this we can either install appropriate spring extensions in

our IDE or directly download the project from the Spring initializer https://start.spring.io/

- In the initializer we can add dependencies and java version → We chose java 17 and used Spring Web, Lombok, MySQL driver, Spring data JPA, Spring Security dependencies for the development.
- We can set application properties (in src/main/resources/application.yaml file) such as port (default - 8080) that backend is accessible, mysql database configurations (database url, type of database etc).
- As it is visible in the image, the src/main/resources/application.yaml are the properties used in deployment. Here we can see we have set the mysql url, username and password as environment variables which are configured at the time of docker container initialization. But this is not suitable for testing the database since these variables are required to be defined at the time of testing. For this we have added src/test/resources/application.yml that override the earlier configurations at the time of testing.

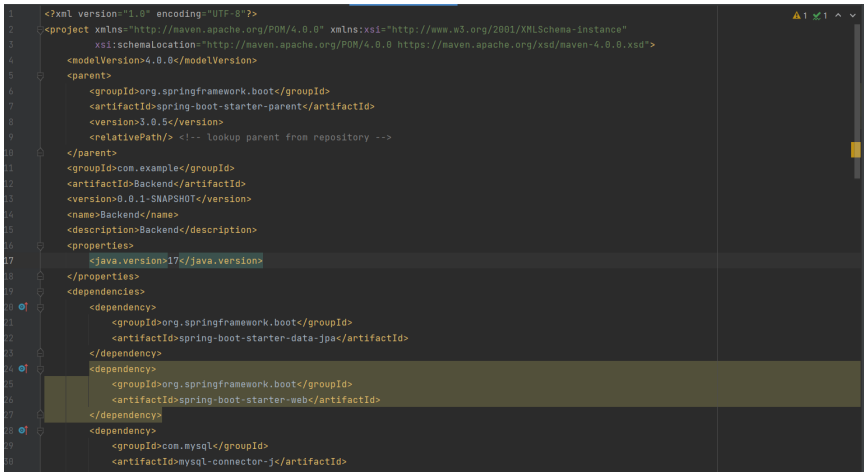

Figure 17: pom.xml

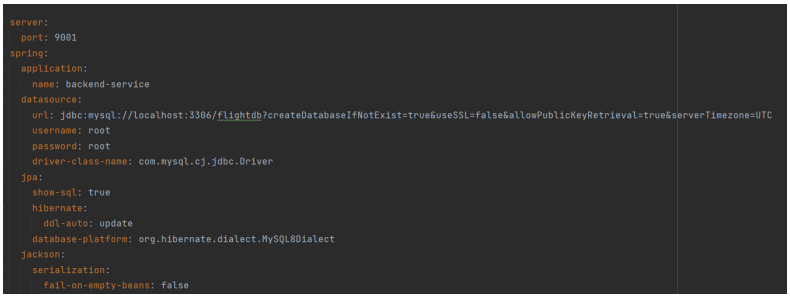

Figure 18: application.yml

- Commands to test, build and package the Backend application in a jar file
  1. Go to the folder $JB_Backend$
  2. Run the command **mvn clean install** to build the jar inside the target folder
  3. To run the backend application, run the command **java -jar target/Backend-0.0.1-SNAPSHOT.jar**

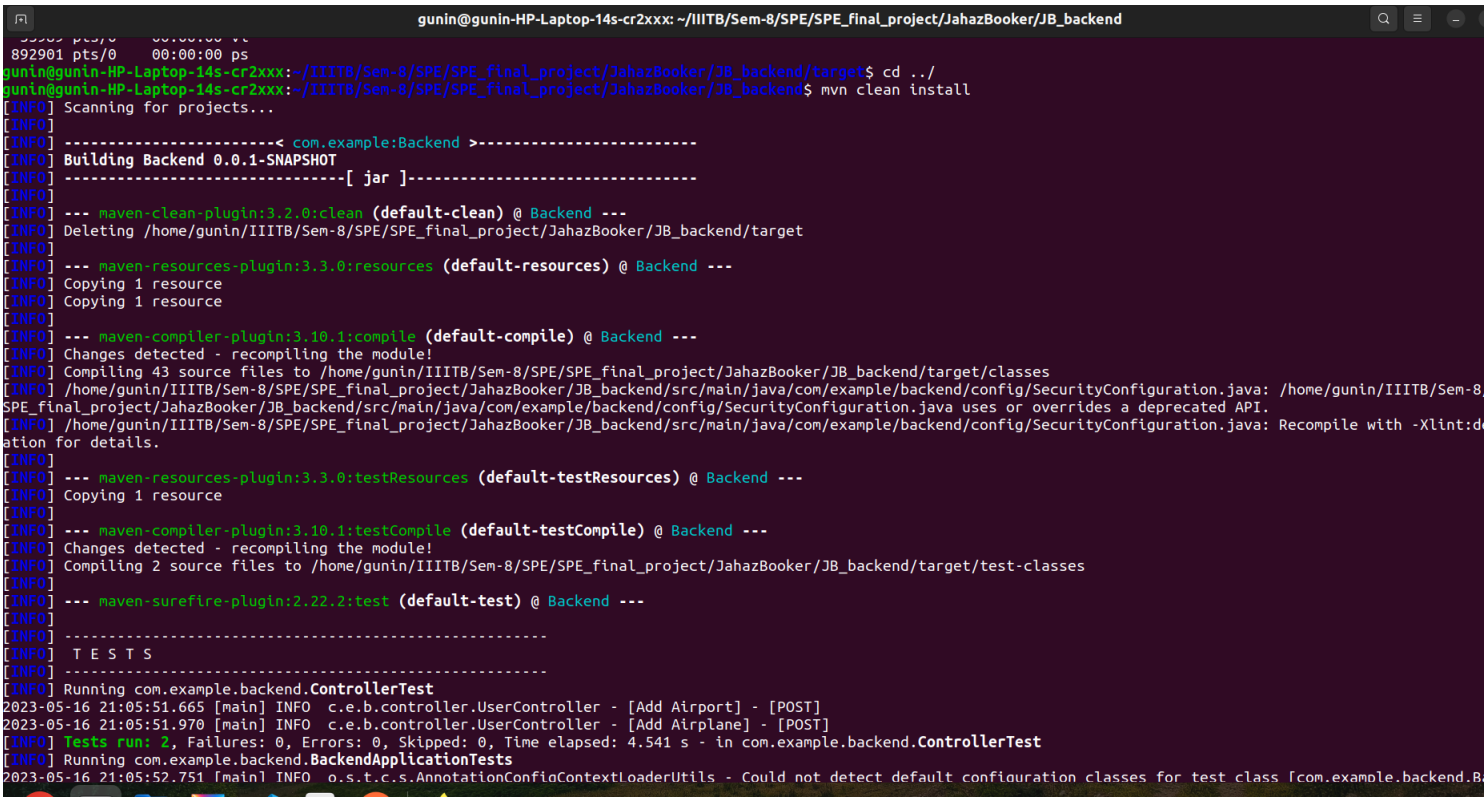

Figure 19: Maven Build and Test

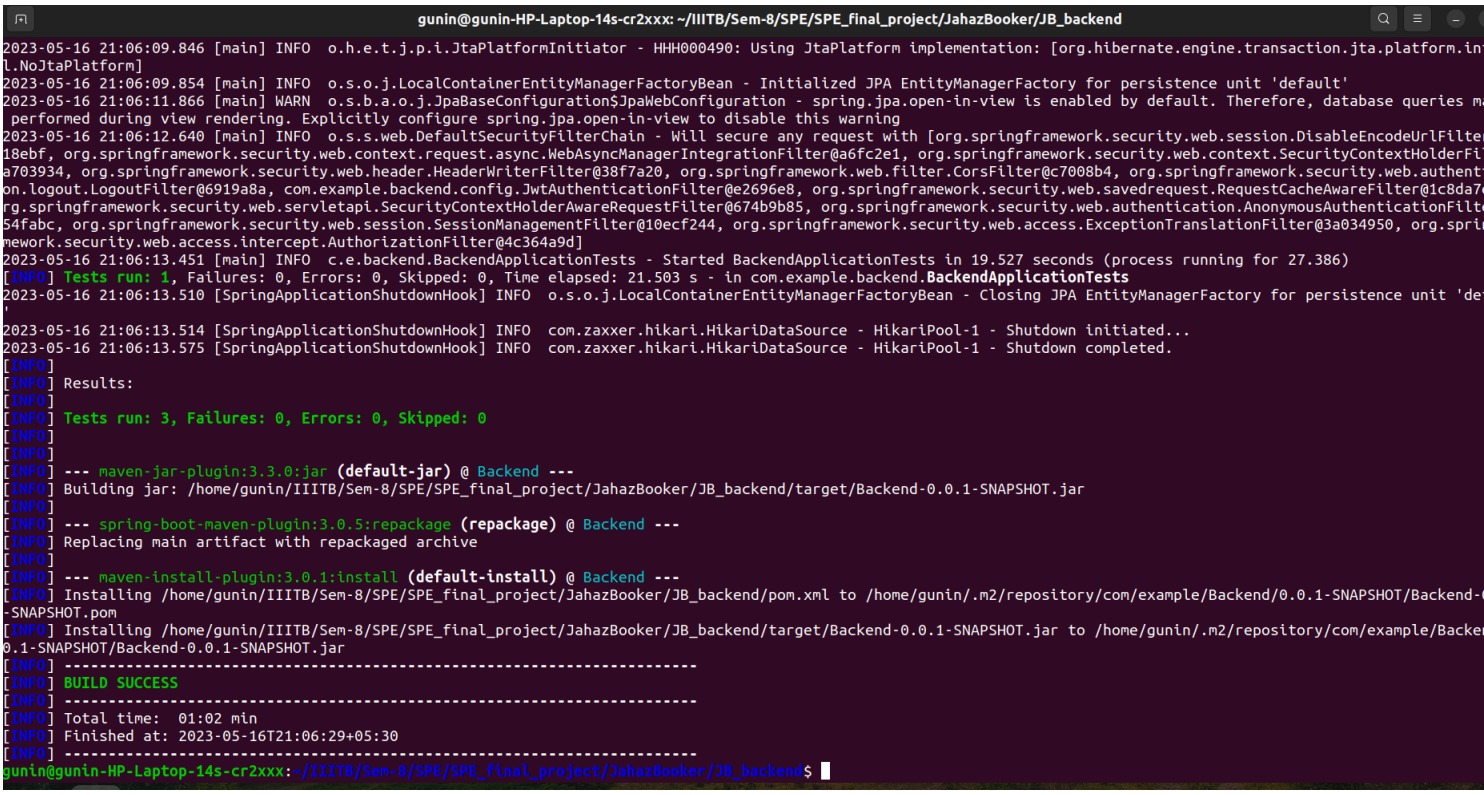

Figure 20: Maven Build and Test

- We develop the application with controllers, services and JPA repository classes using entity class object to communicate with each other.

5.2 React Frontend

React is a Javascript library for building user interfaces. It is declarative making it easier to debug and is component-based helping build modular encapsulated components managing their own state.

React needs a web server tool to be deployed. For this, we have used Nginx open-source web server tool for routing HTTP requests. Axios is another Javascript library used to make http requests from node or XMLHttpRequests from the browser. We are using this to make HTTP requests from the react frontend to backend REST API.

5.2.1 Setup and configuration

1. First we need to install Node.js and Node Package Manager (npm) https://nodejs.org/en/
2. Install react using npm → $npminstall - gcreate - react - appCreatereactappworkspace$ß create-react-app ¡project Name¿
3. Install axios (required for routing GET and POST calls to backend REST API) → $ npm install axios

5.2.2 Developing the frontend application

1. Create Components for various parts of pages like Navbar, Form, Card, etc.
2. Route requests by specifying Routing information
3. use React hooks to update state of component state variables and share data across components

5.3 Source Code Management

- SCM allows us to maintain multiple versions of our codebase in the form of commit histories. This allows developers to collaborate and keep up to date with others' updates.
- Install Git Version Control system using this.
- Create a Github public repository after creating a Github account
- To be able to push to the Github remote repository without being asked for password every time, set up ssh connection using below steps:

  1. Generate an SSH key pair on local machine using the command

     ```
     ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
     ```

     .This will generate a public key and a private key. The public key will be stored in /.ssh/id_rsa.pub.
  2. Add the public key to your GitHub account. Go to your GitHub account settings, click on "SSH and GPG keys", then click on "New SSH key". Paste the contents of the public key file (id_rsa.pub) into the "Key" field and give the key a descriptive title.
  3. Clone the repository using the SSH URL. In a terminal, navigate to the directory where you want to clone the repository and use the following command:

     ```
     git clone git@github.com:<username>/<repository>.git
     ```

     Replace $< username >$ with your GitHub username and $< repository >$ with the name of the repository you want to clone.

- Make your changes to the code.
- Commit your changes and push them to GitHub. In a terminal, navigate to the repository directory and use the following commands:

```
git add .
git commit -m "Your commit message"
git push origin main
```

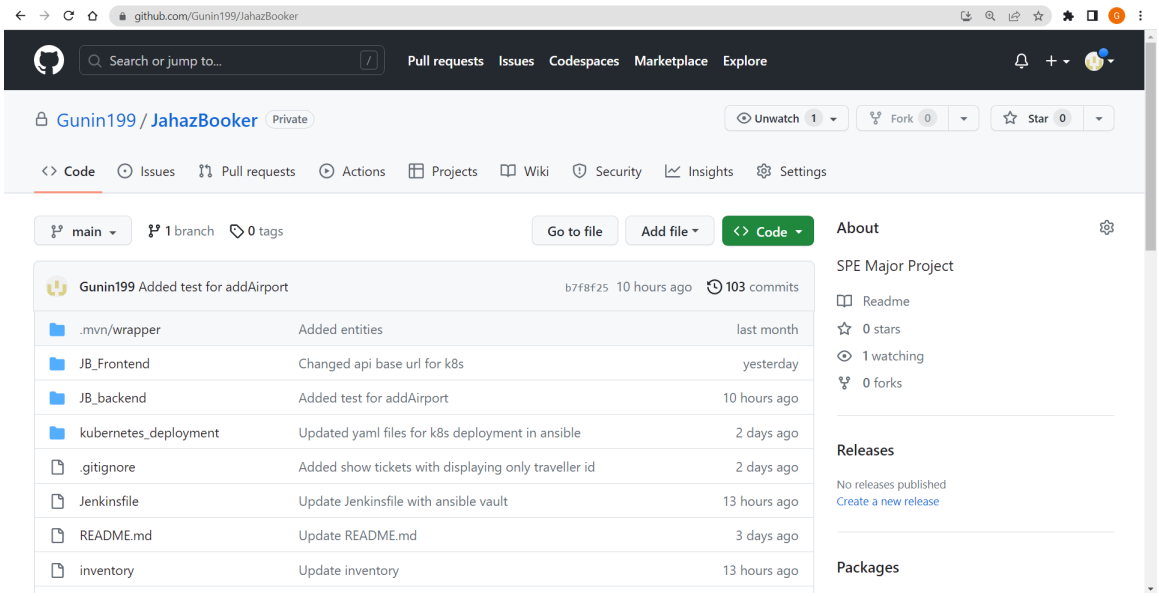

Figure 21: Github Repository

5.4 Containerization using Docker

- Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure.
- Containerization involves building self-sufficient software packages that perform consistently, regardless of the machines they run on. Software developers create and deploy container images—that is, files that contain the necessary information to run a containerized application.
- Developers use containerization tools to build container images based on the Open Container Initiative (OCI) image specification. OCI is an open-source group that provides a standardized format for creating container images. Container images are read-only and cannot be altered by the computer system.
- Container images are the top layer in a containerized system that consists of the following layers :
  - **Infrastructure**
    Infrastructure is the hardware layer of the container model. It refers to the physical computer or bare-metal server that runs the containerized application.
  - **Operating system**
    The second layer of the containerization architecture is the operating system. Linux is a popular operating system for containerization with on-premise computers. In cloud computing, developers use cloud services such as AWS EC2 to run containerized applications.
  - **Container engine**
    The container engine, or container runtime, is a software program that creates containers based on the container images. It acts as an intermediary agent between the containers and the operating system, providing and managing resources that the application needs. For example, container engines can manage multiple containers on the same operating system by keeping them independent of the underlying infrastructure and each other.
  - **Application and dependencies**
    The topmost layer of the containerization architecture is the application code and the other files it needs to run, such as library dependencies and related configuration files. This layer might also contain a light guest operating system that gets installed over the host operating system.
- We will use Docker to create containers. Install docker on your Linux machine using the steps mentioned here.
- Now lets create a Dockerfile. Dockerfile specifies the commands to create an image which can be used to spawn a container(process in linux).
- We create 2 Dockerfiles for our project, one for defining the docker image for the frontend application and other for the backend application.

```
FROM node:alpine

# Create the folder "app"
# for the further COPY, RUN and CMD instructions
WORKDIR /app

# Copy the package.json and package-lock.json files to the "app" folder
COPY package*.json ./


# Install the dependencies mentioned in package.json
RUN npm install --legacy-peer-deps


# Copy the local files to the "app" folder
COPY . .


RUN ls


# Expose port 3000 on the host machine to the container for listening to external connections
EXPOSE 3000

# Start the React applications
CMD ["npm", "start"]
```

Figure 22: Frontend Dockerfile

- FROM: We use the nodejs image based on alpine linux(lightweight linux distribution) as base image on top of which we will build our image
- RUN : runs the shell command inside the container
- COPY: copies files from host machine to container's file system
- WORKDIR : set the present working directory of the container
- CMD : Runs the command only once on start of the container

In brief, the frontend dockerfile takes the nodejs image, copies package-lock.json and package.json, installs all the node modules needed, copies the frontend source code into the container,exposes port 3000 for frntend container and then runs the React application using the command **npm start**.

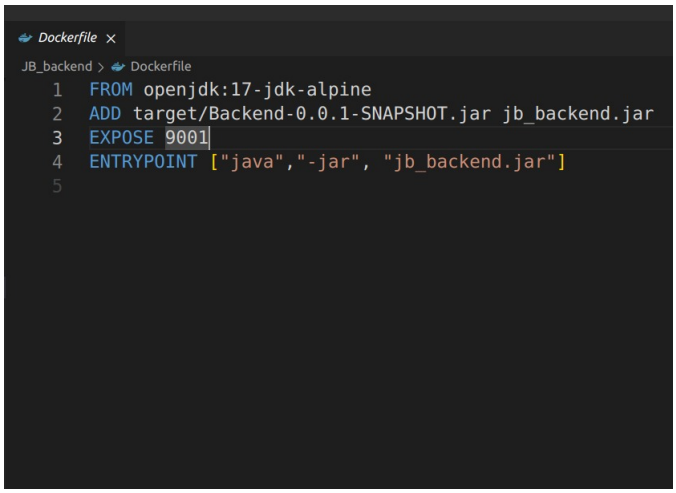

Figure 23: Backend Dockerfile

The backend dockerfile takes the openjdk-17 image as a base image, copies the jar file generated after maven build into the container's file system, exposes port 9001 for backend and finally executes the jar file to start the backend application inside the container.

5.5 Dockerhub

We push the built images to our dockerhub repository so that anyone who wants to run our application can pull those images from dockerhub and spawn a container based on them.

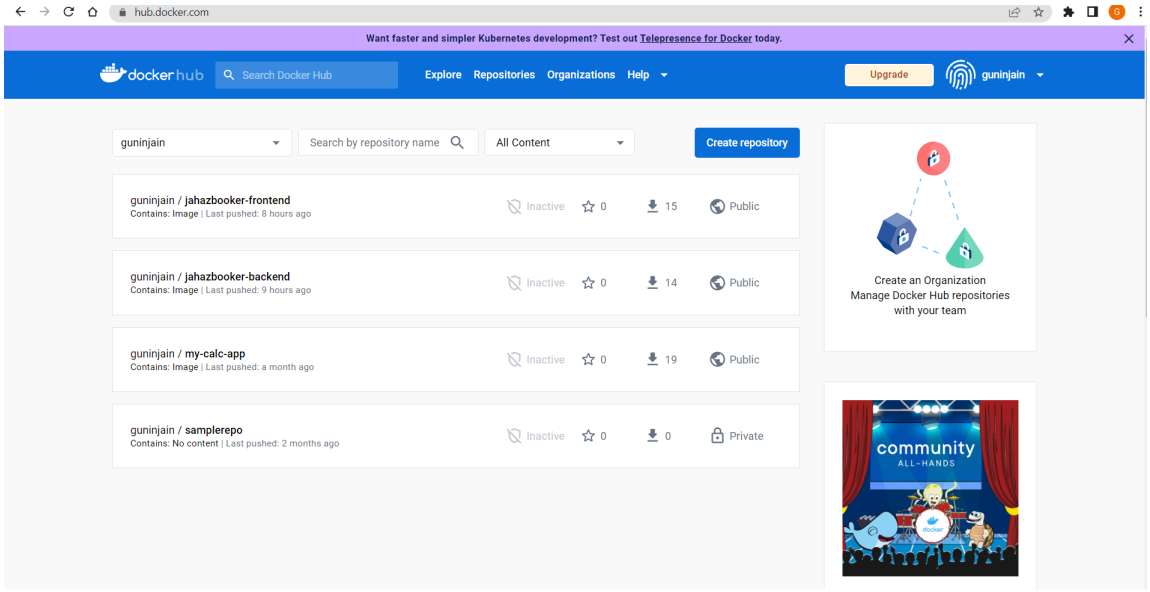

Figure 24: Dockerhub repositories

5.6 Deployment using Docker Compose

Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It uses a YAML file to configure the services, networks, and volumes required for your application's containers. With Docker Compose, you can easily spin up and manage complex application environments with multiple interconnected containers.

We used the following steps to create the docker compose YAML file for our project:

1. Create a Compose file: Create a YAML file called docker-compose.yml (or any other name you prefer) in your project directory. This file will define your application's services, networks, and volumes. The Compose file consists of a series of sections, including version, services, networks, and volumes.

2. Define services: In the services section of the Compose file, you define each containerized service that makes up your application. Specify the image to use for each service, any environment variables, exposed ports, and other configuration options. You can also define dependencies between services.

3. Configure networks and volumes: If your application requires specific networks or volumes, you can define them in the networks and volumes sections of the Compose file. Networks allow containers to communicate with each other, while volumes provide persistent storage for your application data.

4. Run Docker Compose: Open a terminal or command prompt, navigate to your project directory (where the docker-compose.yml file is located), and run the docker-compose up command. This will start all the services defined in your Compose file. You can use the docker-compose up -d command to run the services in detached mode, which keeps them running in the background. Using the detached command is useful when running it inside a Jenkins pipeline so that the docker compose step completes.

In our YAML file, we have defined 3 services, 1 for the DB server, 1 for the Backend and 1 for the frontend.

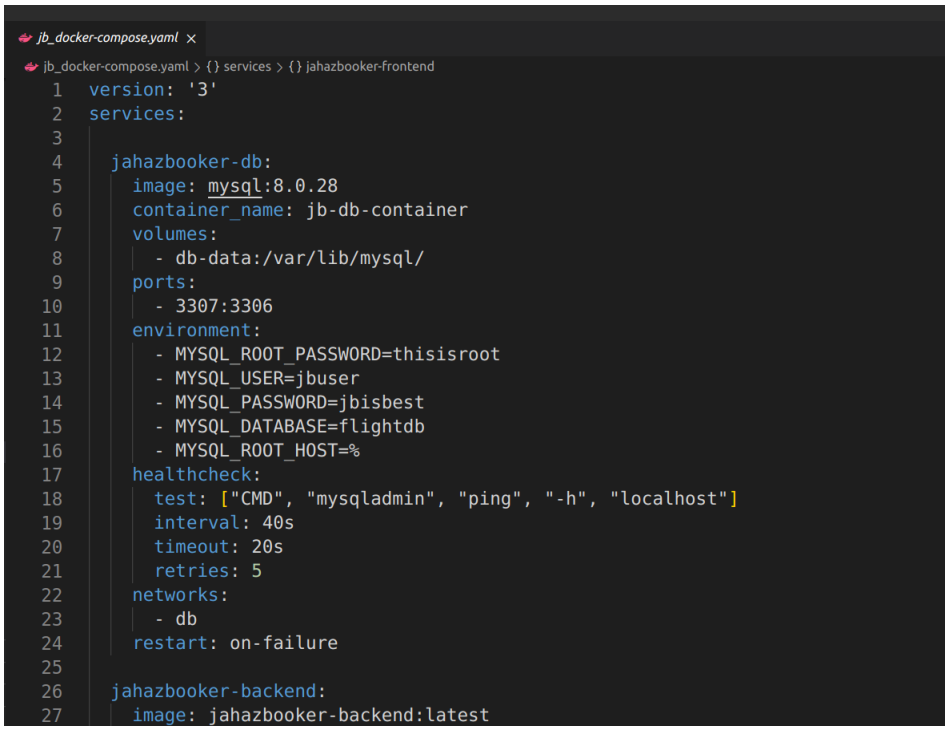

Figure 25: Docker Compose file – DB service

The DB service pulls the mysql v8.0.29 image present on dockerhub. It also specifies a docker volume to the data directory of MYSQL. Environment variables are defined for creating the new user and creating our database flightdb. A healthcheck is also added so that other services dependent on the DB service run only when this healthcheck passes. The DB service is also attached to a docker network named db.

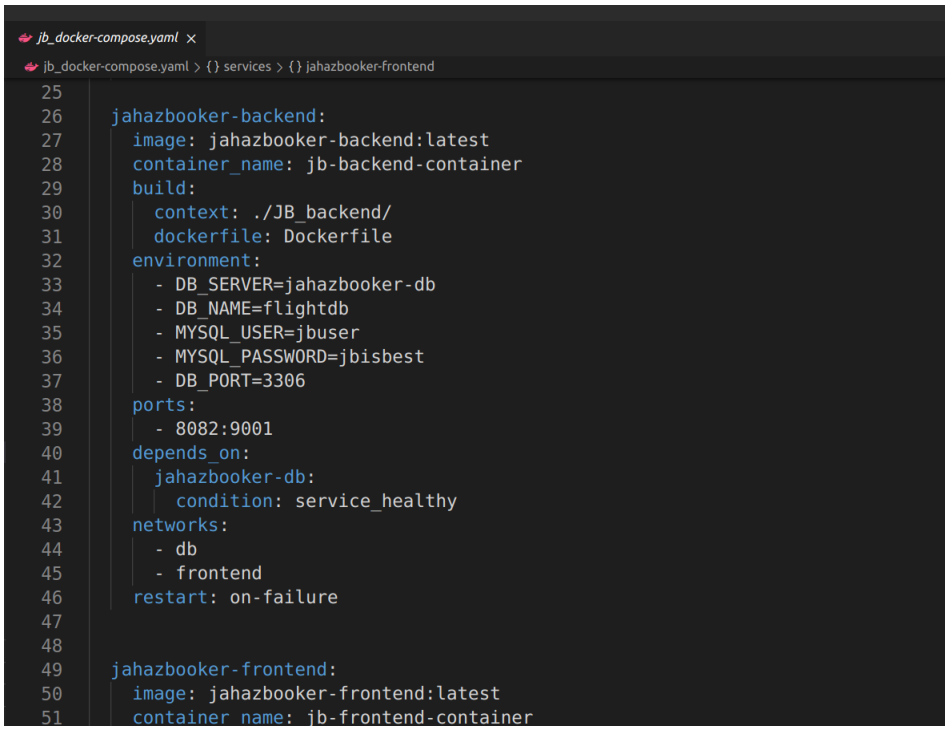

Figure 26: Docker Compose file – Backend service

The Backend service specifies the image to used from our dockerhub repository or it can be built from the Dockerfile as well when given the –build flag when running the command docker compose up. Environments for db connection are specified and ports on host machine are mapped to container.

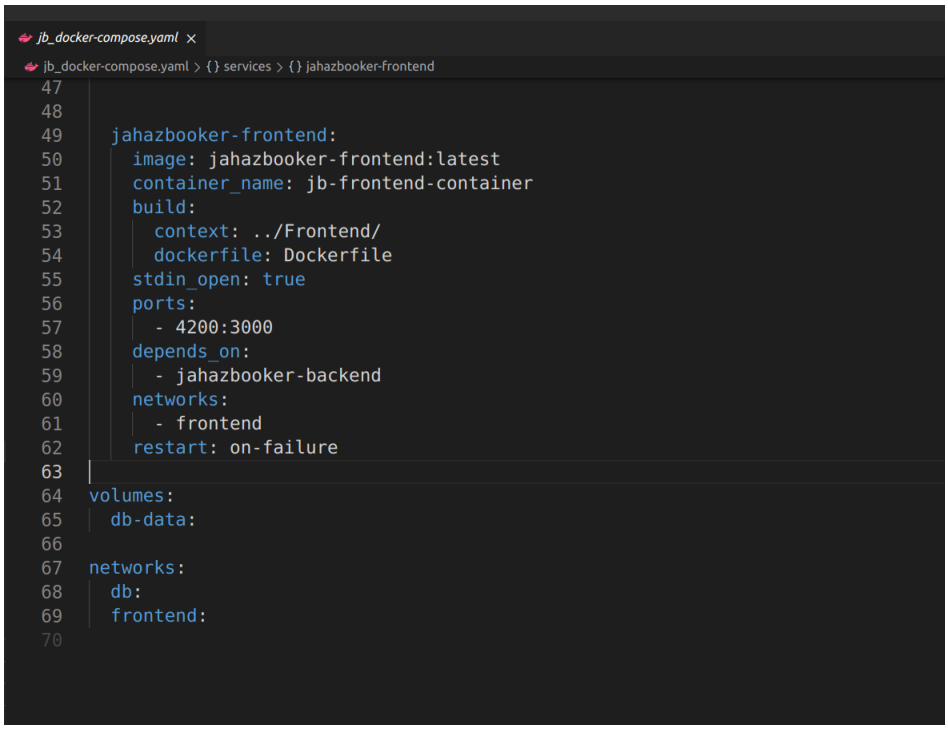

Figure 27: Docker Compose file – Frontend service

Similar to the backend service, we define the frontend service as well.

5.7 Deployment using Kubernetes

Kubernetes is an open-source container orchestration platform that simplifies the deployment, scaling, and management of containerized applications. It allows you to run and manage applications across a cluster of machines without worrying about the underlying infrastructure. Key benefits of Kubernetes:

- **Containerization**: Kubernetes leverages containers to package applications and their dependencies, ensuring consistency and portability.
- **Orchestration**: Kubernetes automates the deployment and scaling of containers, ensuring that the actual state matches the desired state.
- **High Availability**: Kubernetes distributes containers across multiple nodes, automatically rescheduling them if a node fails, and supporting self-healing capabilities.
- **Scalability**: Kubernetes enables horizontal scaling of applications based on demand, handling increased traffic without manual intervention.
- **Service Discovery and Load Balancing**: Kubernetes provides service discovery and built-in load balancing mechanisms for efficient communication between applications.
- **Rolling Updates and Rollbacks**: Kubernetes supports seamless updates and rollbacks of applications, ensuring reliability during deployments.
- **Ecosystem and Extensibility**: Kubernetes has a rich ecosystem of plugins and integrations, making it easy to customize and extend for specific requirements.

In summary, Kubernetes is a powerful platform for managing containerized applications, offering benefits such as scalability, high availability, and simplified management. It allows developers and operators to focus on application development and delivery, accelerating the deployment of cloud-native applications. **Setup and Configuration**:

- Start minikube by running "minikube start".
- We first allocate volume and claim it by applying mysql-pv.yaml as shown below.

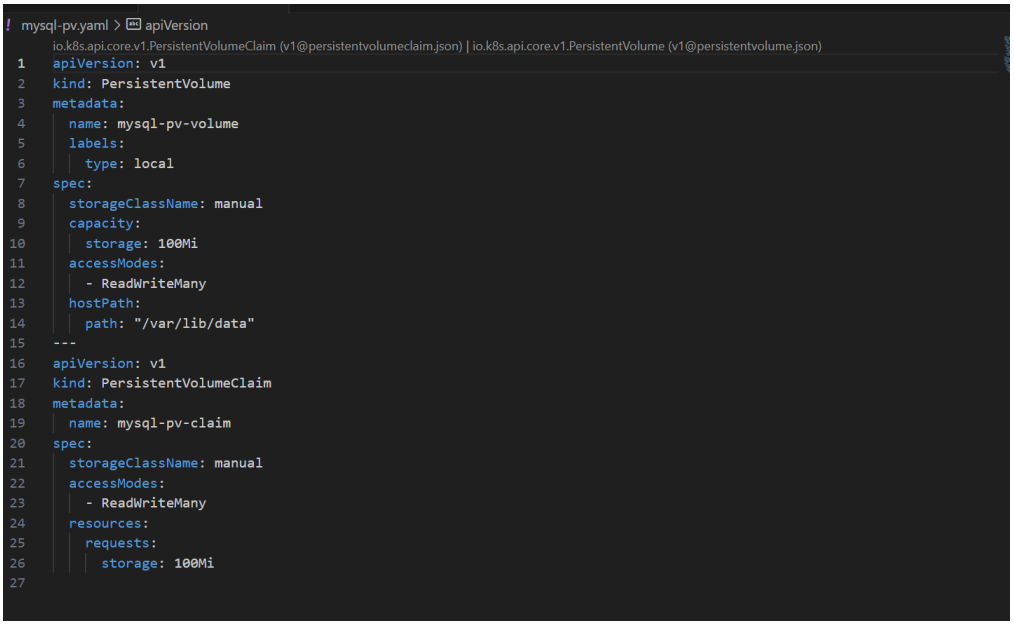

Figure 28: mysql-pv.yaml

- We then create the secrets for mysql user, and configmap for storing configurational settings for the mysql db.

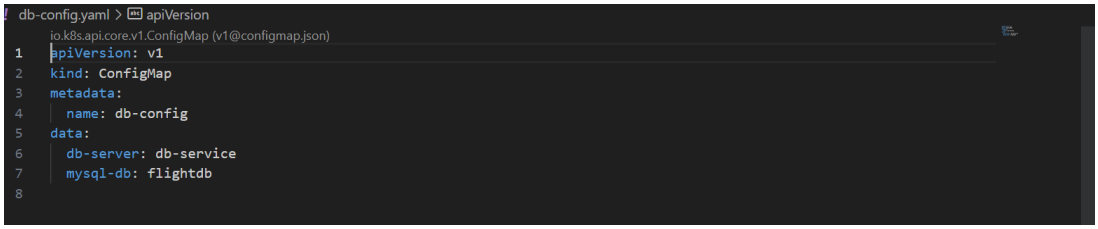

Figure 29: configMap

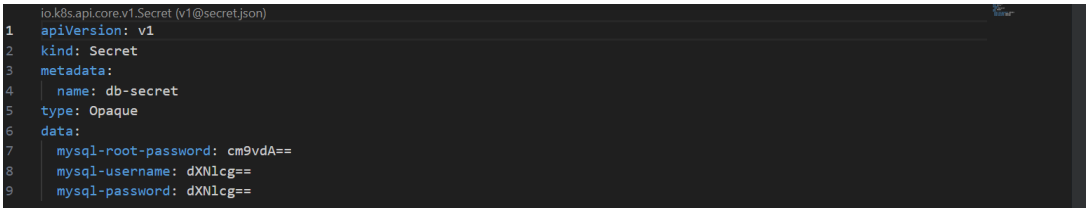

Figure 30: Secrets

- We now create deployment and service for the mysql server by applying the db.yaml

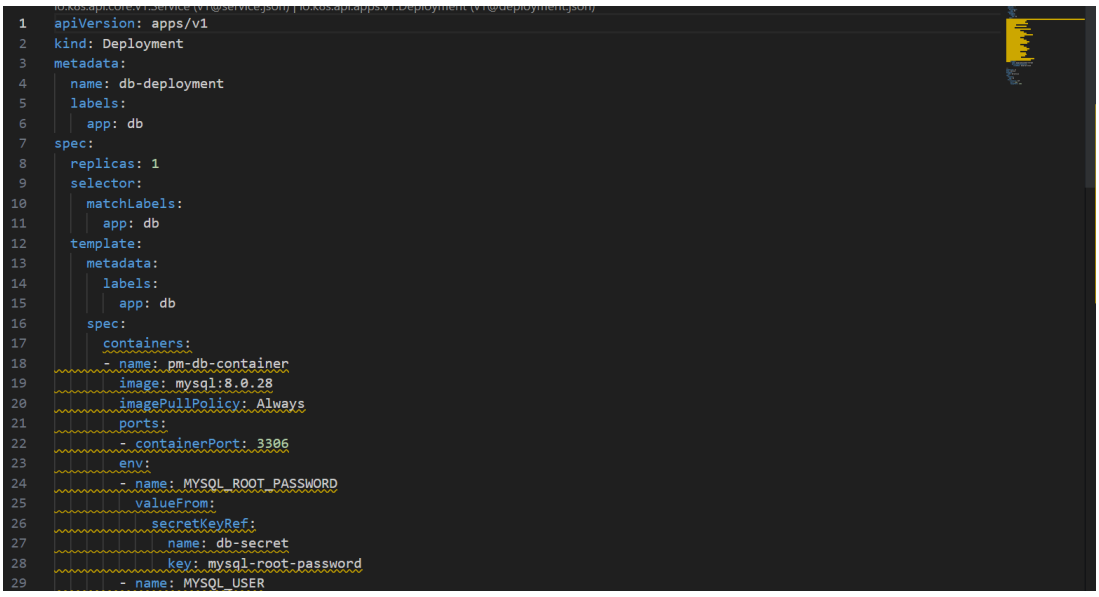

Figure 31: db.yaml

```yaml
            - name: MYSQL_DATABASE
              valueFrom:
                configMapKeyRef:
                  name: db-config
                  key: mysql-db
            volumeMounts:
              - name: mysql-persistent-storage
                mountPath: /var/lib/mysql
          volumes:
            - name: mysql-persistent-storage
              persistentVolumeClaim:
                claimName: mysql-pv-claim

---
apiVersion: v1
kind: Service
metadata:
  name: db-service
spec:
  selector:
    app: db
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

Figure 32: db.yaml(continued)

- Similarly we create services and deployments for the backend and frontend by applying the frontend.yaml and backend.yaml on the same default namespace.

```yaml
io.k8s.api.core.v1.Service (v1@service.json) | io.k8s.api.apps.v1.Deployment (v1@deployment.json)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  labels:
    app: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend-container
          image: guninjain/jahazbooker-frontend:latest
          imagePullPolicy: Always
          ports:
            - name: http
              containerPort: 3000
```

Figure 33: frontend.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
```

Figure 34: frontend.yaml(continued)

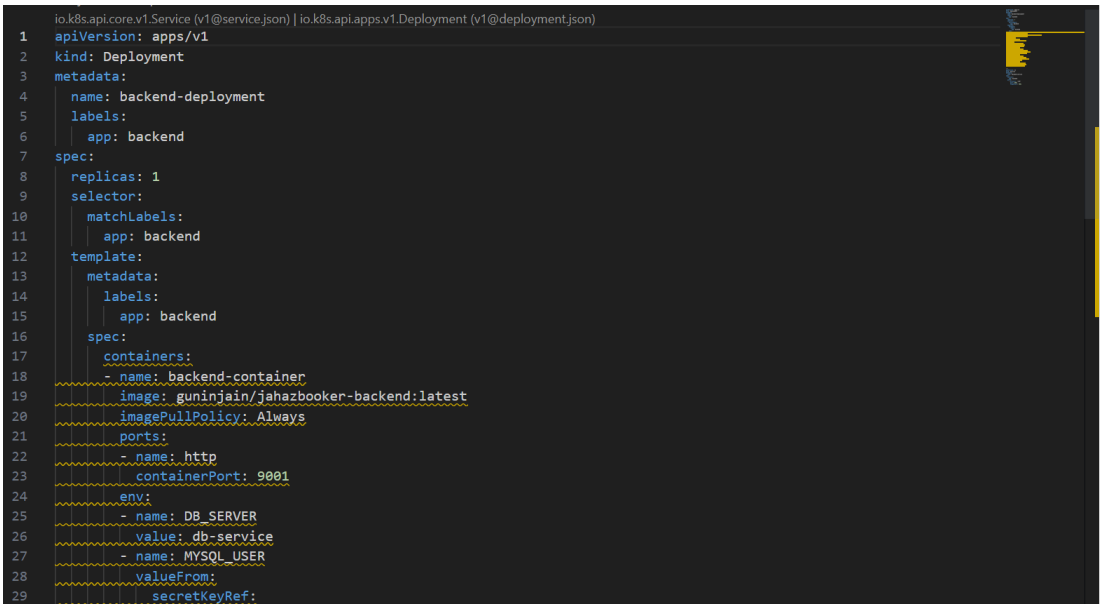

Figure 35: backend.yaml

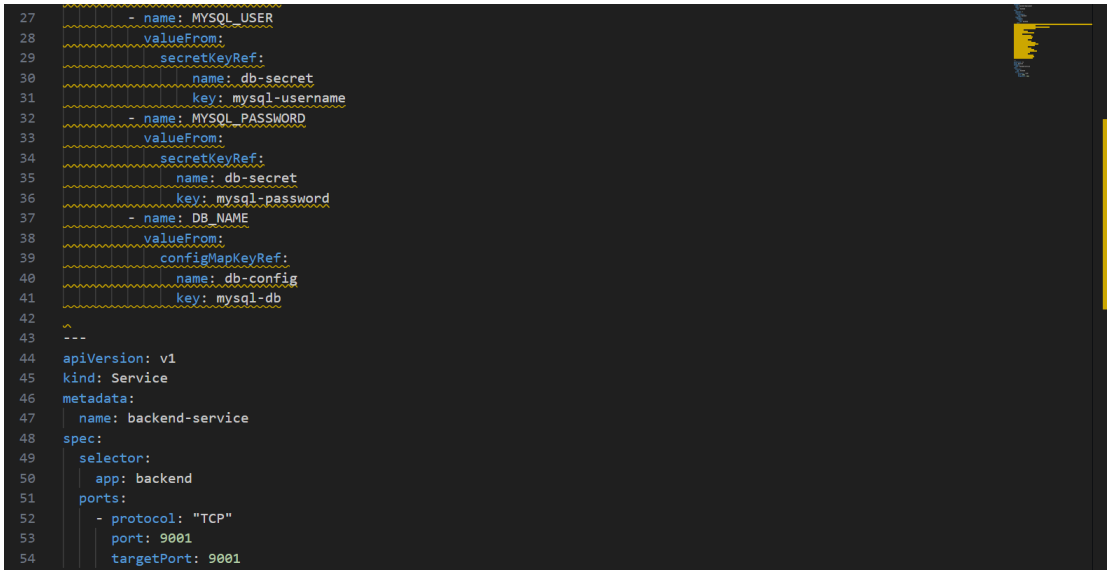

Figure 36: backend.yaml(continued)

- Before creating we need to first add an ingress controller in our minikube controller. This we did using the command "minikube addons ingress". Now we apply ingress.yaml and create Ingress which will act as reverse proxy and load balancer and help in making the cluster accessible to the external environment.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
    - host: api.jahaz-booker-backend.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend-service
                port:
                  number: 9001
    - host: jahaz-booker.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 3000
```

Figure 37: ingress.yaml

- We then check all resources running using kubectl get all.

```
pratyush@pratyush-virtual-machine:~/Desktop/SPE-Major-Project/JahazBooker/kubernetes_deployment$ kubectl get all
NAME                                       READY   STATUS    RESTARTS   AGE
pod/backend-deployment-7c57464f7f-frcq8    1/1     Running   0          58m
pod/db-deployment-696545dd65-4ww27         1/1     Running   0          58m
pod/frontend-deployment-54bbbdb697-nvb2x   1/1     Running   0          58m

NAME                       TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/backend-service    ClusterIP   10.105.4.223     <none>        9001/TCP   18h
service/db-service         ClusterIP   10.100.204.253   <none>        3306/TCP   18h
service/frontend-service   ClusterIP   10.103.19.235    <none>        3000/TCP   18h
service/kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP    19h

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend-deployment    1/1     1            1           58m
deployment.apps/db-deployment         1/1     1            1           58m
deployment.apps/frontend-deployment   1/1     1            1           58m

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-deployment-7c57464f7f    1         1         1       58m
replicaset.apps/db-deployment-696545dd65         1         1         1       58m
replicaset.apps/frontend-deployment-54bbbdb697   1         1         1       58m
pratyush@pratyush-virtual-machine:~/Desktop/SPE-Major-Project/JahazBooker/kubernetes_deployment$ kubectl get ingress
NAME         CLASS   HOSTS                                              ADDRESS        PORTS   AGE
my-ingress   nginx   api.jahaz-booker-backend.com,jahaz-booker.com   192.168.49.2   80      18h
pratyush@pratyush-virtual-machine:~/Desktop/SPE-Major-Project/JahazBooker/kubernetes_deployment$
```

Figure 38: All resources

- We configure DNS, and add the ip of the cluster corresponding to the domain names of frontend and backend services defined in ingress.yaml. We can now access the site using it's domain name defined for the frontend.

5.8 Continuous Deployment using Ansible

Ansible is an open-source automation tool that simplifies the management and deployment of systems, applications, and infrastructure. It is designed to be simple, agentless, and highly flexible, making it popular for automating tasks in IT operations, configuration management, and application deployment.

Key features and concepts of Ansible include:

1. Declarative Language: Ansible uses a simple, human-readable language called YAML (YAML Ain't Markup Language) to define tasks and configurations. YAML files, known as playbooks, describe the desired state of systems and the steps needed to achieve that state.

2. Agentless Architecture: Ansible does not require any software or agents to be installed on managed nodes. It uses SSH and PowerShell (for Windows systems) to establish connections and execute tasks remotely. This makes Ansible lightweight and easy to set up and manage.

3. Idempotency: Ansible promotes idempotent operations, meaning that running the same playbook multiple times has the same effect as running it once. It only applies changes when necessary, which helps ensure consistency and avoids unnecessary modifications to systems.

4. Inventory: Ansible uses an inventory file to define the managed nodes or hosts. This file can be static or dynamic and can include details such as hostnames, IP addresses, SSH credentials, and groupings of hosts. It allows you to organize and target specific sets of systems for configuration.

5. Modules: Ansible uses modules to perform specific tasks on managed nodes. Modules are small, self-contained units of code that can be executed by Ansible. They cover a wide range of operations, including package installation, file manipulation, service management, and more. Ansible ships with numerous built-in modules, and you can also create custom modules if needed.

6. Playbooks: Playbooks are YAML files that define a set of tasks and configurations to be executed on managed nodes. Playbooks can include variables, conditionals, loops, and other constructs to handle complex scenarios. They provide a way to orchestrate multiple tasks and define the desired state of systems.

7. Roles: Roles are a way to organize and reuse sets of tasks, variables, and files across multiple playbooks. They provide a structured approach to manage and share Ansible configurations. Roles help modularize playbooks and promote code reusability.

To use Ansible, you typically start by defining your inventory of hosts, writing playbooks to describe the desired configurations or tasks, and executing the playbooks using the 'ansible-playbook' command. Ansible will connect to the hosts, execute the tasks defined in the playbooks, and provide you with the results and status of each task.

In our project, we have defined tasks in our playbook.yml file to run the kubectl commands inside the shell module of ansible in order to deploy our application using Kubernetes. We are running the tasks on localhost for this project.

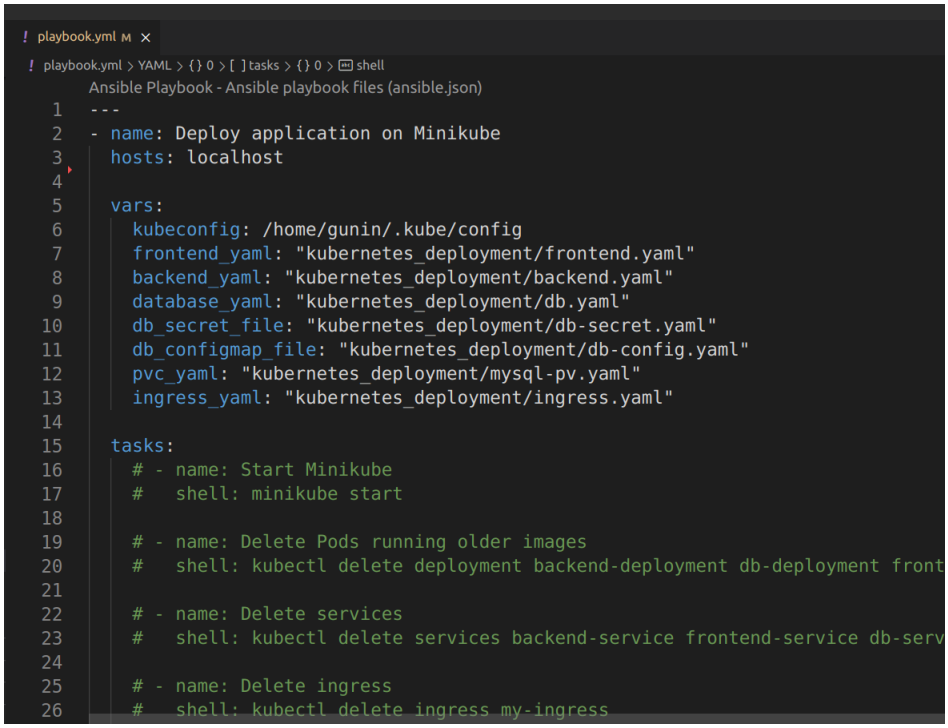

Figure 39: Ansible playbook

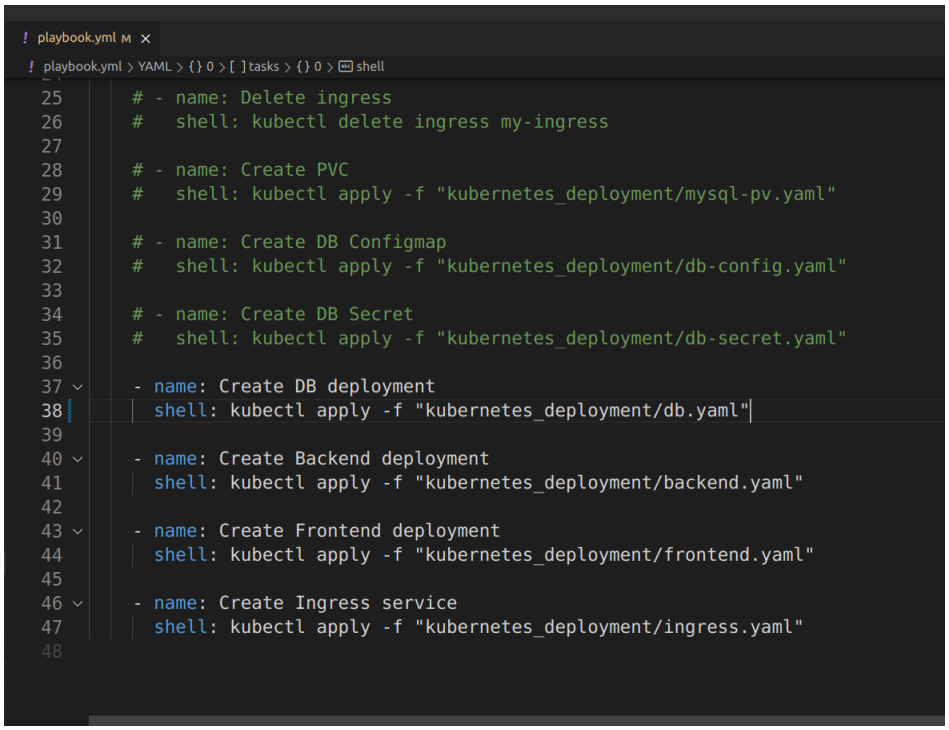

Figure 40: Ansible playbook continued

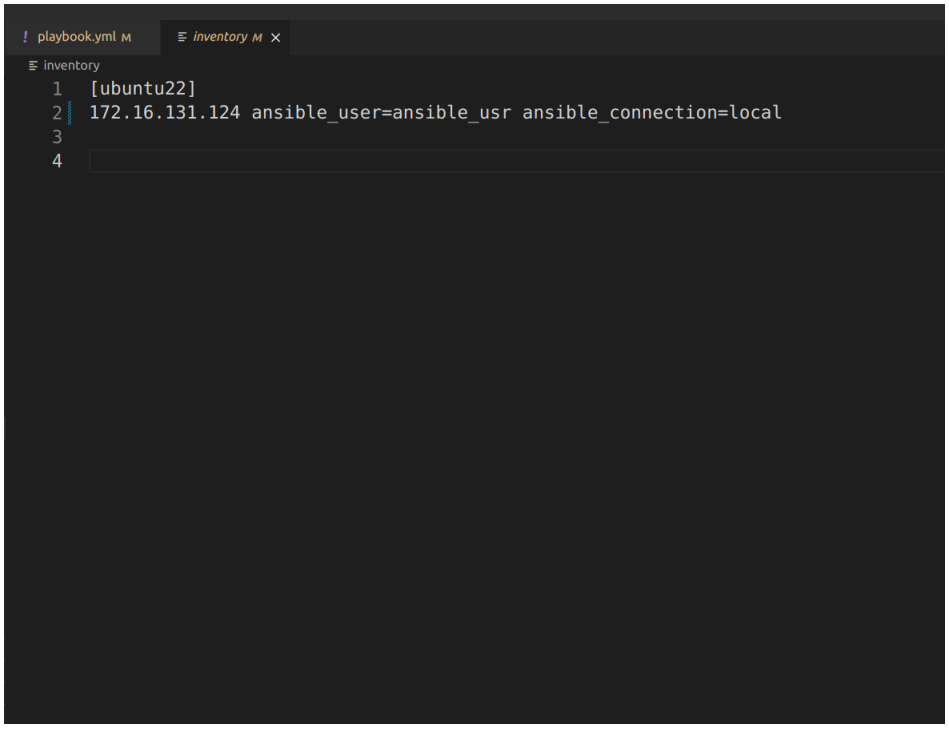

Figure 41: Ansible inventory file

5.9 CI/CD Pipeline using Jenkins

- Jenkins is a Java based open-source automation tool with plugins built for Continuous integration.
- Jenkins is used to enable developers to continuously build,test and deploy their code in a frequent fashion.
- We are using Jenkins pipeline for our project.Jenkins Pipeline is a suite of plugins that allows developers to create and manage continuous delivery pipelines as code. It enables developers to define the entire software delivery process in a single pipeline script.

5.9.1 Jenkins installation and setup

- Install Jenkins using the steps mentioned here.
- Open jenkins UI using localhost:8080 in your browser.
- Install the following plugins by going to Plugin Manager in Manage Jenkins:
  - Git and Github plugins
  - Docker and Docker pipeline plugins
  - Ansible plugin
  - Maven plugin
  - Credentials plugin
- Add dockerhub credentials by going to Manage Jenkins ⟶ Manage Credentials ⟶ System ⟶ Global credentials(unrestricted) ⟶ Add Credentials and then selecting username password. Username is your dockerhub username and password is the auth token generated in Account Settings ⟶ Security in dockerhub.

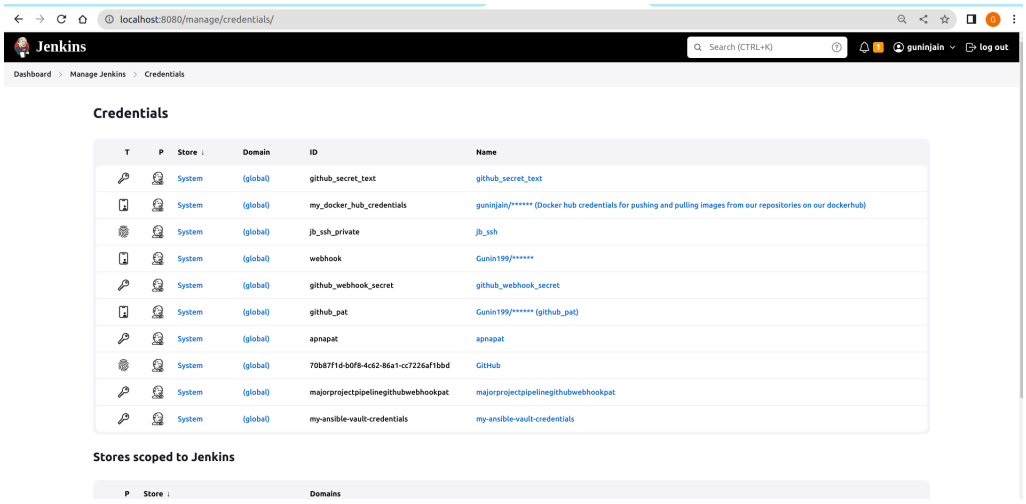

Figure 42: Dockerhub Credentials

5.9.2 Jenkins pipeline project configuration

- Create a pipeline project in Jenkins and specify the Github project url
- In Pipeline, select Pipeline script from SCM and give git repository url with branch to find Jenkinsfile containing pipeline script. Give a PAT as well since we are using a Private Github repository.

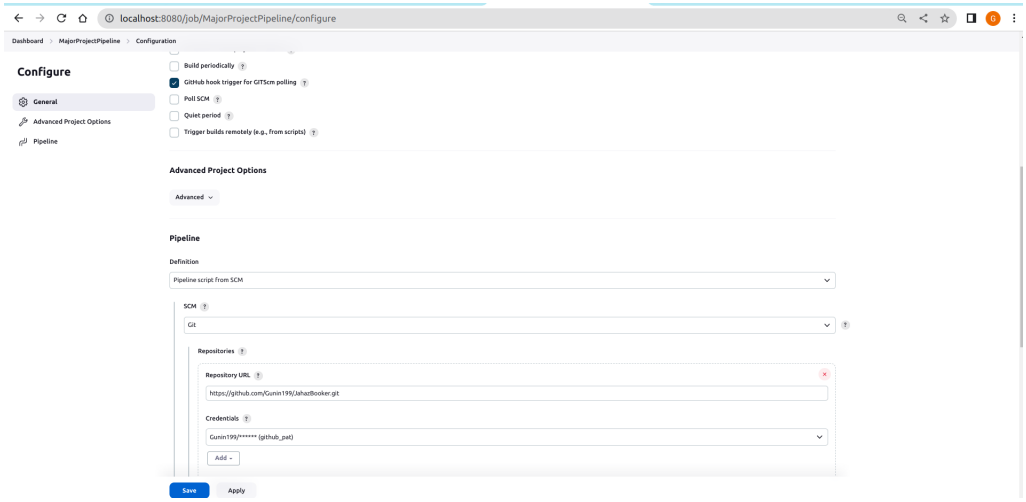

Figure 43: Pipeline configuration

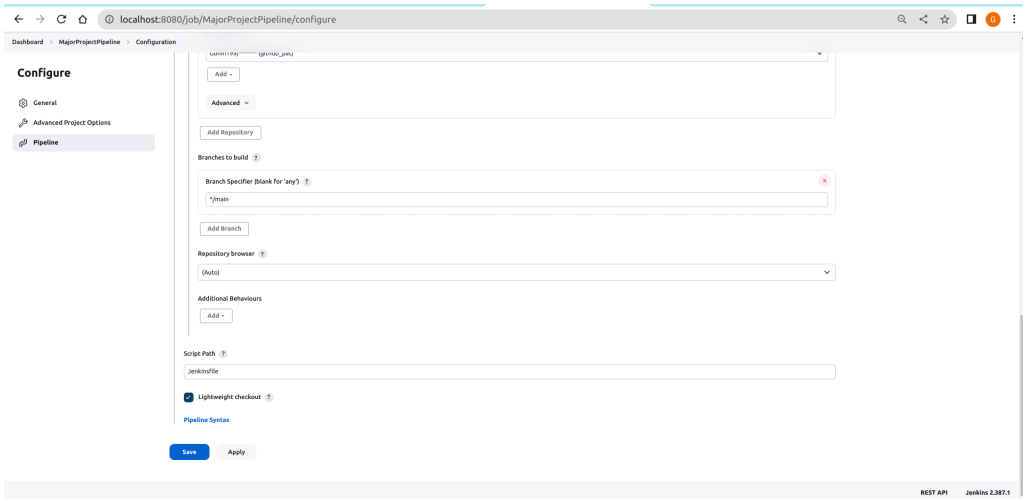

Figure 44: Pipeline configuration continued

5.9.3 Github Webhook

- To send a POST request to the Jenkins CI server, every time a commit is made we need a webhook to be configured on Github.
- This webhook requires the public ip address of the Jenkins CI Server. Since Jenkins is running on localhost:8080, we need Ngrok. Ngrok uses secure tunnels to connect local servers behind NATs (Network Address Translation) and firewalls to the public internet. It has a real-time web interface that allows you to inspect any HTTP traffic passing through your tunnels. It allows you to connect to the internet via a web server running on your local system.
- Follow the steps mentioned here to install ngrok on your linux machine.
- Now start ngrok using

  ```
  ngrok http <port-num>
  ```

  . Specify the port on which your web server is listening to ngrok in $< port - num >$.

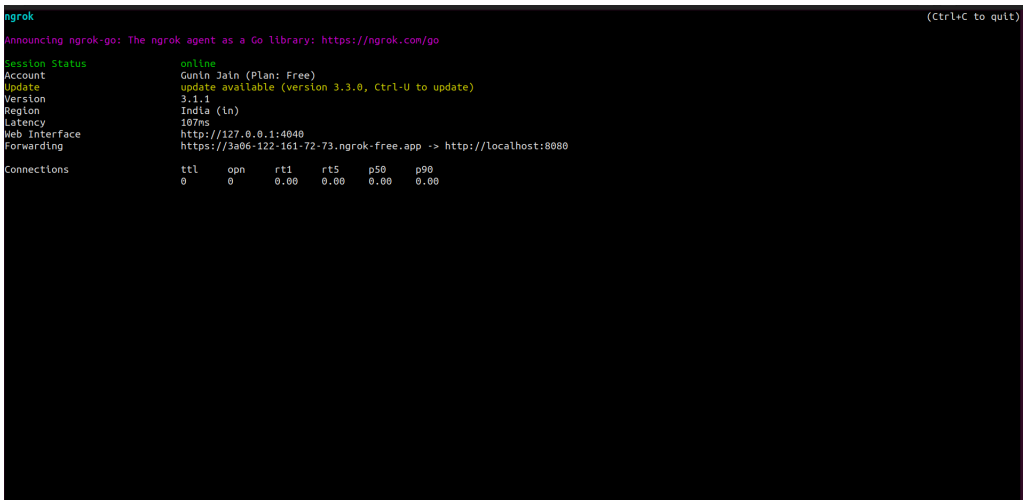

Figure 45: Starting ngrok

- Now go to Settings ⟶ Webhooks ⟶ Add webhook and put the Ngrok Forwarding URL appended with /github-webhook/ endpoint in the Payload URL section.
- Add a Personal Access token(Generate using Github ⟶ Settings ⟶ Developer Tools ⟶ Personal access tokens) as secret field in the webhook setup.

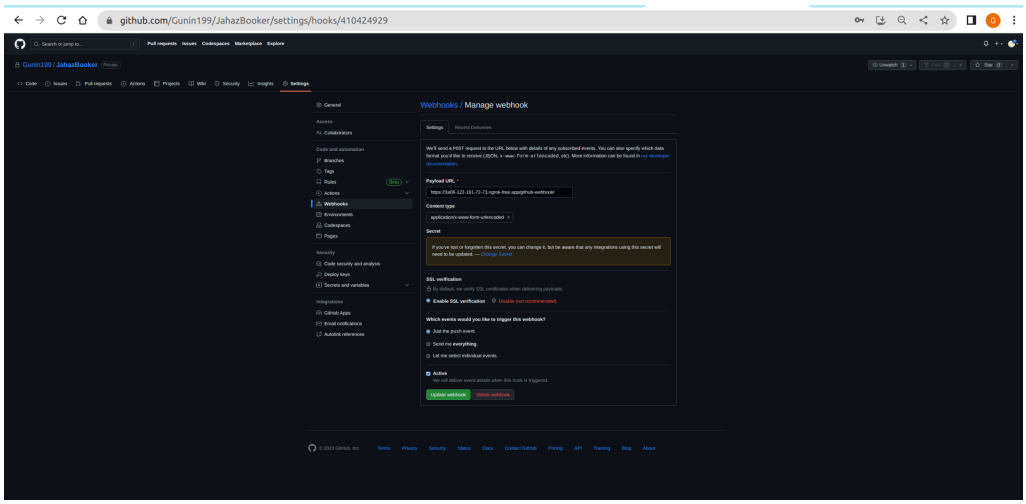

Figure 46: Github Webhook Configuration

- Now go to Manage Jenkins ⟶ Configure System and add the Ngrok Forwarding URL to the Jenkins URL.

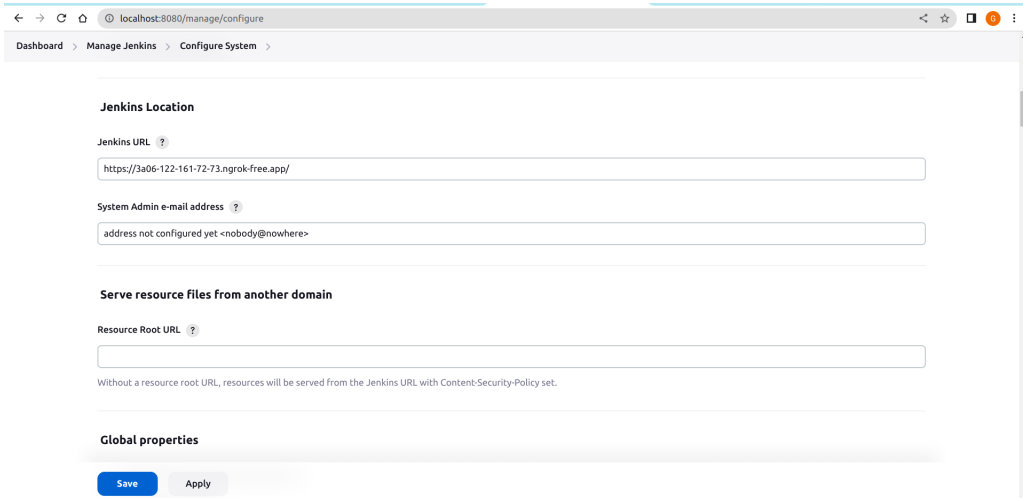

Figure 47: Jenkins URL setup

- Now go to Github Servers section in the above section and add your previously generated Personal Access token here.

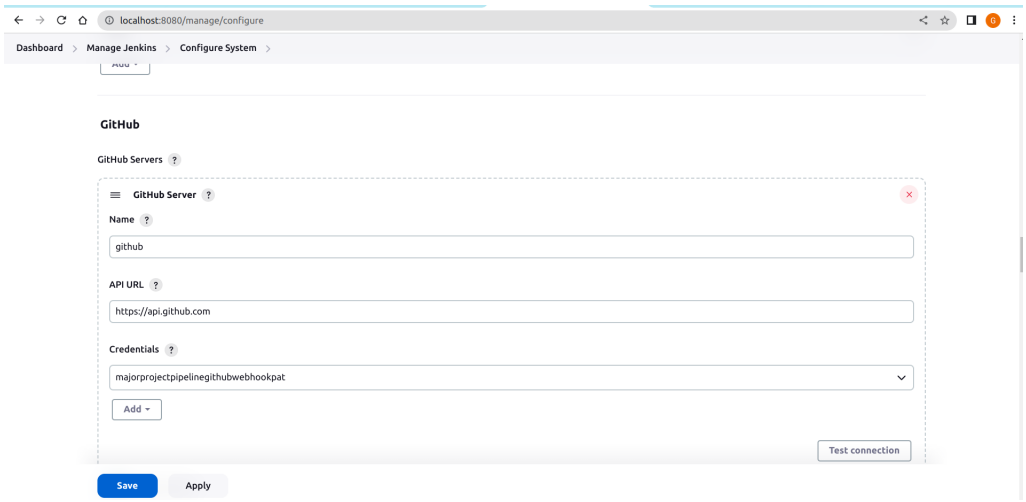

Figure 48: Add Secret text to Github server configuration in Jenkins

- In your jenkins pipeline project, ensure that you configure the Gitscm hook trigger option in the Build Triggers section.

5.9.4 Jenkins pipeline script

- For running docker commands via jenkins pipeline script, we need to add jenkins user(since this is the user using which jenkins software executes the pipeline) to the docker group which has access permissions to access the docker daemon unix socket.
- Thus, we will use this command *sudo usermod -aG docker jenkins*
- The above command followed by system boot once, will allow docker daemon to be accessed by jenkins user while running pipeline script and no docker.sock permission denied error will come.

- Refs:

    1. Solving-docker-permission-denied-while-trying-to-connect-to-the-docker-daemon-socket
    2. What-does-sudo-usermod-a-g-group-user-do-on-linux

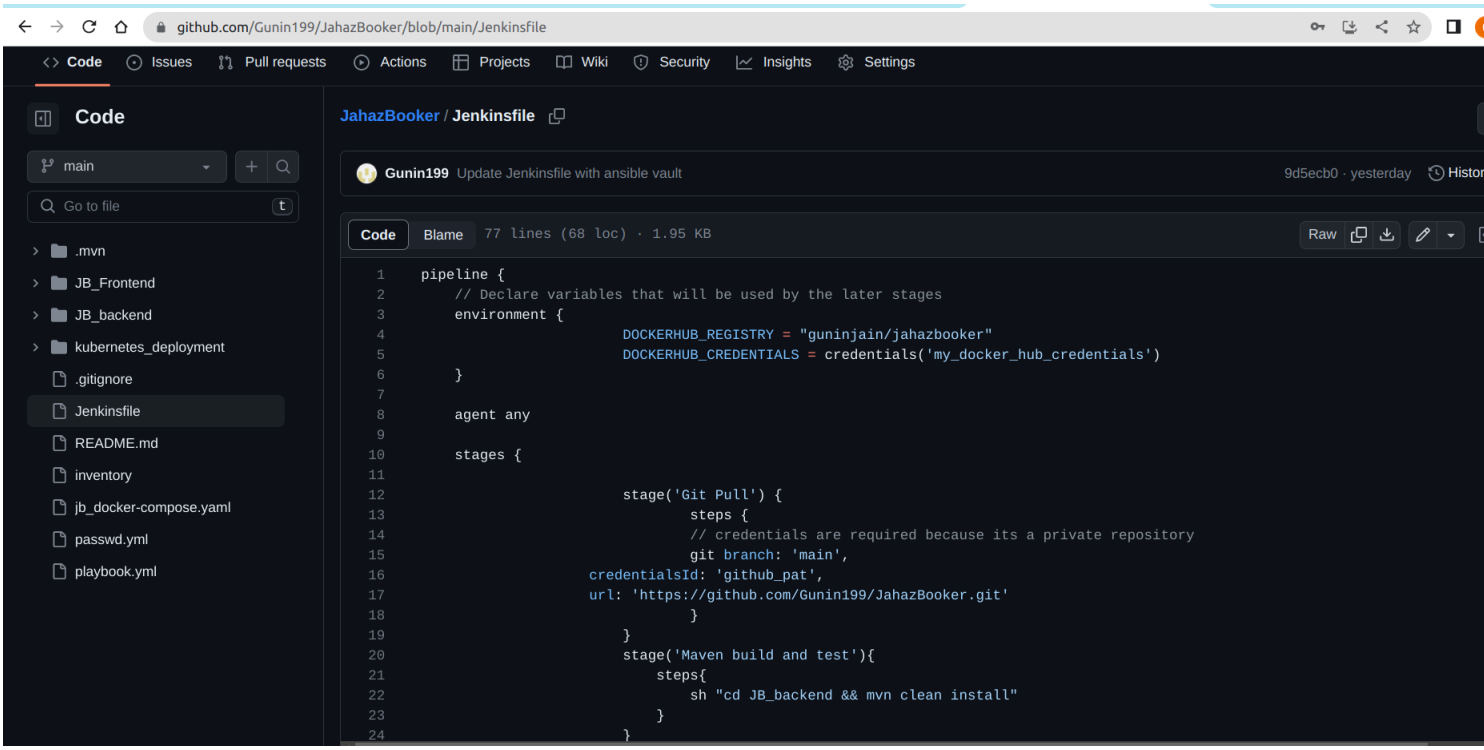

Figure 49: Jenkins pipeline script

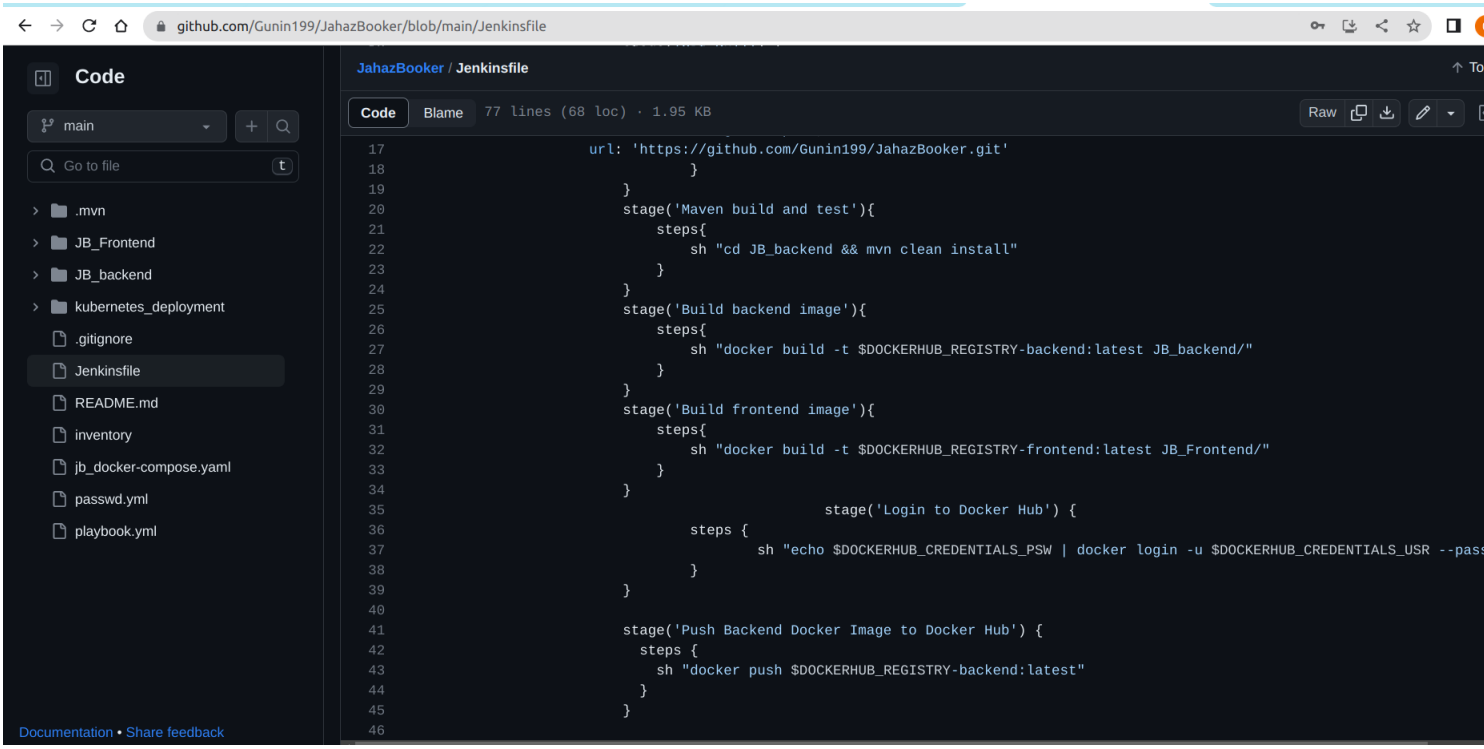

Figure 50: Jenkins pipeline script continued

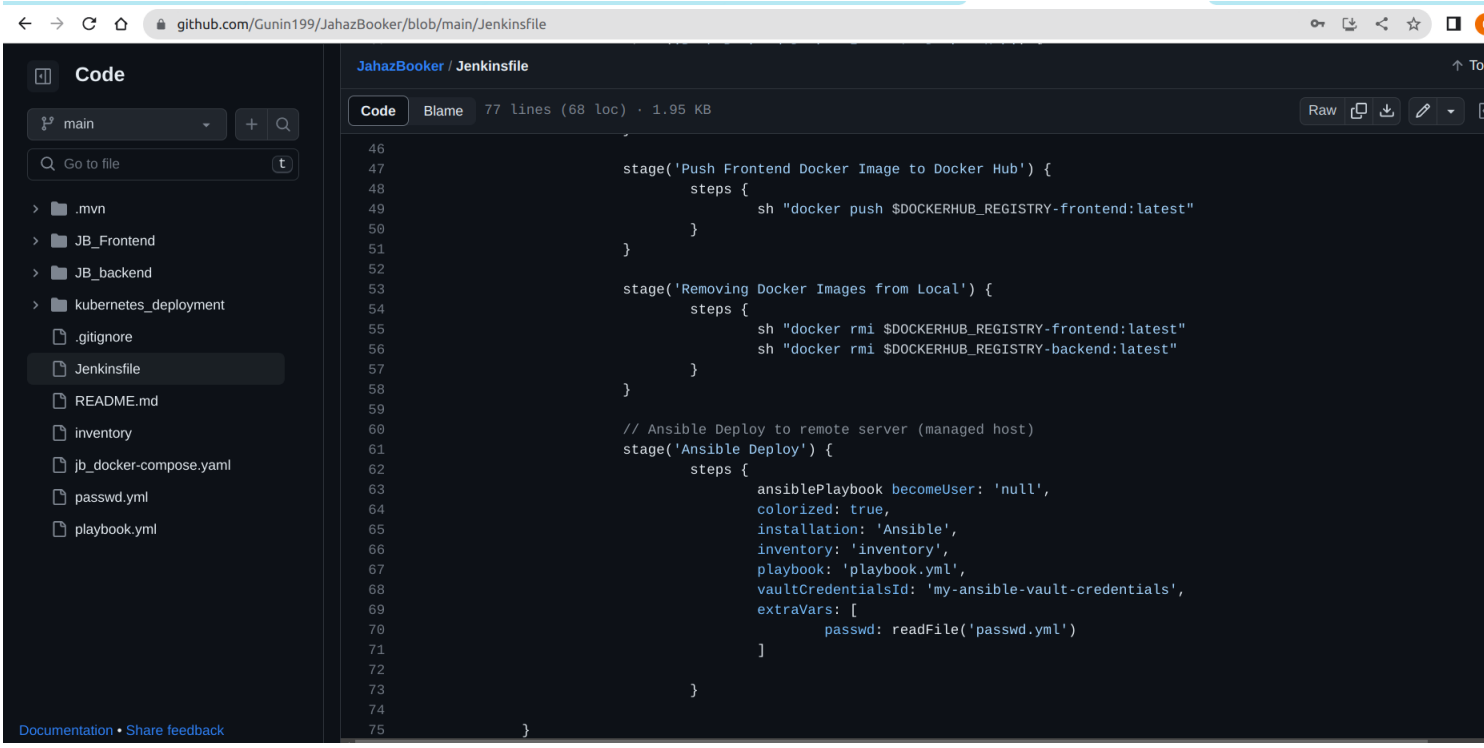

Figure 51: Jenkins pipeline script continued

In the pipeline script, we define environment variables for dockerhub repository and dockerhub login credentials. We first pull the code from our private github repository using the credentials id of the credentials given to the jenkins manager. Then we build and test the backend. After that we build the docker image of our frontend and backend using the Dockerfiles in Frontend and Backend respectively. We then push these images to dockerhub post login using stored dockerhub credentials in the credentials manager. Then we remove previous versions of our docker images and run the ansible playbook. We can specify the password for sudo user using ansible vault encrypted yaml file.

5.10 Monitoring logs using ELK Stack

ELK consists of three open source softwares:
Elasticsearch: It is a search and analytics engine.
Logstash: It is a server-side data processing pipeline that ingests data from multiple sources simultaneously transforms it, and then sends it to a "Stash" like Elasticsearch.
Kibana: It lets users visualize data with charts and graphs made from Elasticsearch.
This is used for continuous monitoring of the deployment and creating visualizations for better understanding of the working and business profits from the deployment.

- For logging we have used the slf4j logger and configured the logback.xml as shown below. In this way we define the format of logs that will be stored in the file and displayed on the console.

```
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>logs/application.log</file>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>
    <root level="INFO">
        <appender-ref ref="CONSOLE" />
        <appender-ref ref="FILE" />
    </root>
```

Figure 52: logback.xml

- The logs we obtain are of this form.

```
2023-05-16 11:05:55.962 [main] INFO  o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 54 ms. Found 7 JPA
2023-05-16 11:05:56.364 [main] INFO  o.s.b.w.e.tomcat.TomcatWebServer - Tomcat initialized with port(s): 9001 (http)
2023-05-16 11:05:56.369 [main] INFO  o.a.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler ["http-nio-9001"]
2023-05-16 11:05:56.370 [main] INFO  o.a.catalina.core.StandardService - Starting service [Tomcat]
2023-05-16 11:05:56.370 [main] INFO  o.a.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/10.1.7]
2023-05-16 11:05:56.434 [main] INFO  o.a.c.c.C.[Tomcat].[localhost].[/] - Initializing Spring embedded WebApplicationContext
2023-05-16 11:05:56.434 [main] INFO  o.s.b.w.s.c.ServletWebServerApplicationContext - Root WebApplicationContext: initialization completed in 99
2023-05-16 11:05:56.786 [main] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
2023-05-16 11:05:56.891 [main] INFO  com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@5c5732
2023-05-16 11:05:56.894 [main] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
2023-05-16 11:05:56.909 [main] INFO  SQL dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2023-05-16 11:05:57.616 [main] INFO  o.s.o.j.LocalContainerEntityManagerFactoryBean - Initialized JPA EntityManagerFactory for persistence unit
2023-05-16 11:05:58.226 [main] WARN  o.s.b.a.o.j.JpaBaseConfiguration$JpaWebConfiguration - spring.jpa.open-in-view is enabled by default. There
2023-05-16 11:05:58.426 [main] INFO  o.s.s.web.DefaultSecurityFilterChain - Will secure any request with [org.springframework.security.web.sessi
2023-05-16 11:05:58.582 [main] INFO  o.a.coyote.http11.Http11NioProtocol - Starting ProtocolHandler ["http-nio-9001"]
2023-05-16 11:05:58.595 [main] INFO  o.s.b.w.e.tomcat.TomcatWebServer - Tomcat started on port(s): 9001 (http) with context path ''
2023-05-16 11:05:58.602 [main] INFO  c.example.backend.BackendApplication - Started BackendApplication in 3.506 seconds (process running for 4.0
2023-05-16 11:15:14.345 [SpringApplicationShutdownHook] INFO  o.s.o.j.LocalContainerEntityManagerFactoryBean - Closing JPA EntityManagerFactory
```

Figure 53: application.log

- Once the log file is generated we upload the file on our web deployment and define the grok pattern as follows.

```
%{TIMESTAMP_ISO8601:timestamp} \[%{DATA:thread}\] %{LOGLEVEL:log_level} %{DATA:logger} - %{GREEDYDATA:message}
```

Figure 54: Grok Pattern

Since the log file will be generated in the backend pod we first copy the log from the pod to our current directory and then upload it. We can analyze the message and visualize using Kibana as shown below.

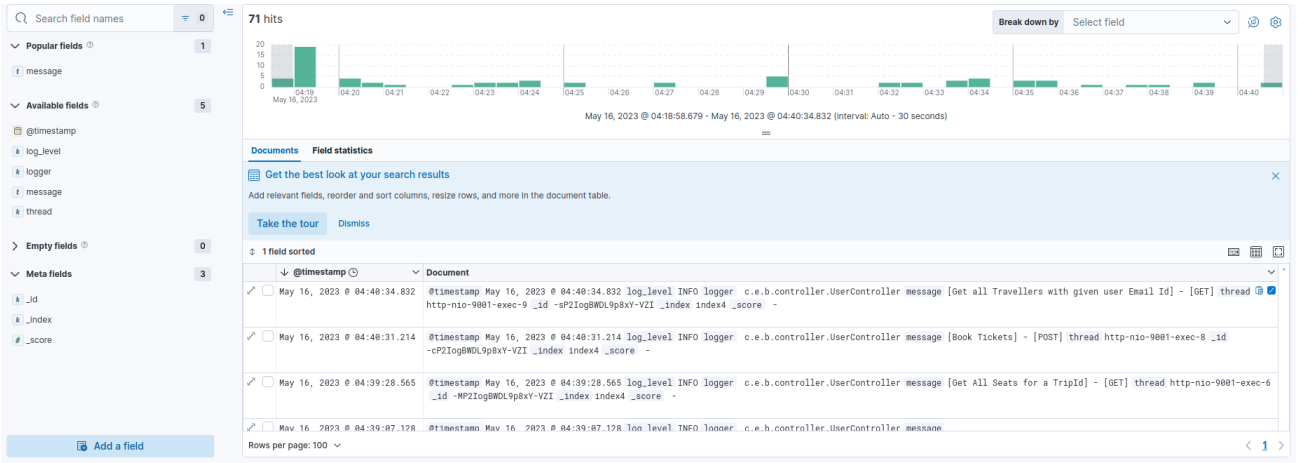

Figure 55: visualization

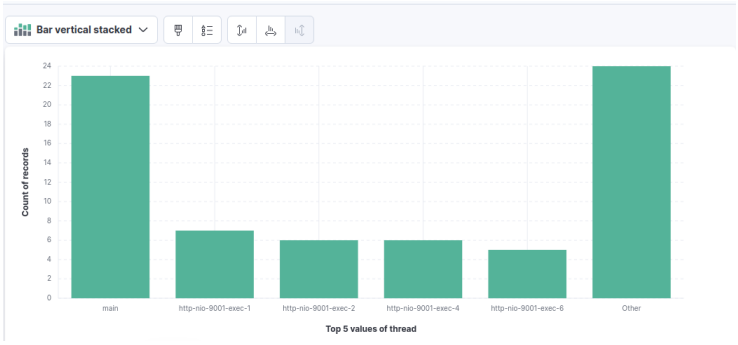

Figure 56: Kibana visualisation

Figure 57: visualization

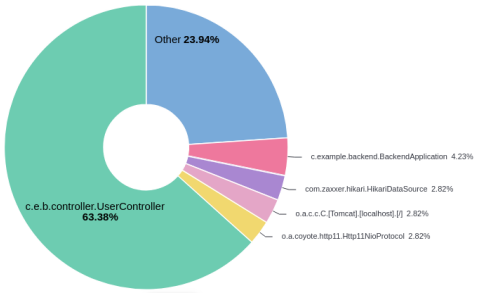

Figure 58: Visualization