Major Project Report
CabPool

GitHub: https://github.com/Bambo0st/CabPool

Aditya Nagaraja IMT2021054
Devendra Rishi Nelapati IMT2021076

December 10, 2024

Contents

Abstract

CabPool is a peer-to-peer ride-sharing platform aimed at reducing travel costs and promoting environmental sustainability. By implementing cutting-edge technologies, including ReactJS, NodeJS, ExpressJS, MongoDB and container orchestration with Kubernetes, configuration management with Ansible and Pipeline Automation with Jenkins this project showcases an end-to-end DevOps pipeline for efficient application deployment and monitoring.

Chapter 1

Introduction

1.1 Problem Statement

CabPool enables users to share cabs by connecting them with suitable ride matches. It aims to:

- Reduce individual travel costs.
- Minimize carbon emissions.
- Provide a seamless ride-sharing experience.

1.2 Objective

The objective of this project is to design, implement and deploy a robust ride-sharing platform that adheres to modern software engineering principles, focusing on scalability and user experience.

1.3 Implementation Links

- **GitHub:** https://github.com/Bambo0st/CabPool
- **DockerHub Backend:** https://hub.docker.com/repository/docker/bambo0st/backend/general
- **DockerHub Frontend:** https://hub.docker.com/repository/docker/bambo0st/frontend/genera

1.4 Frontend webpage

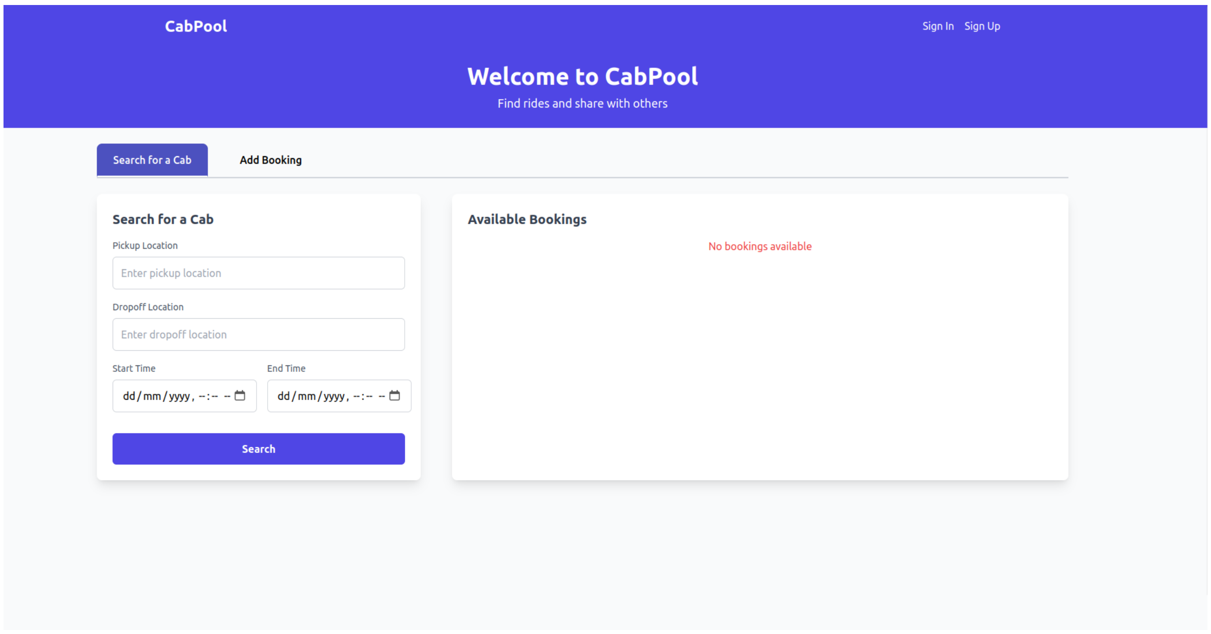


Figure 1.1: MainPage

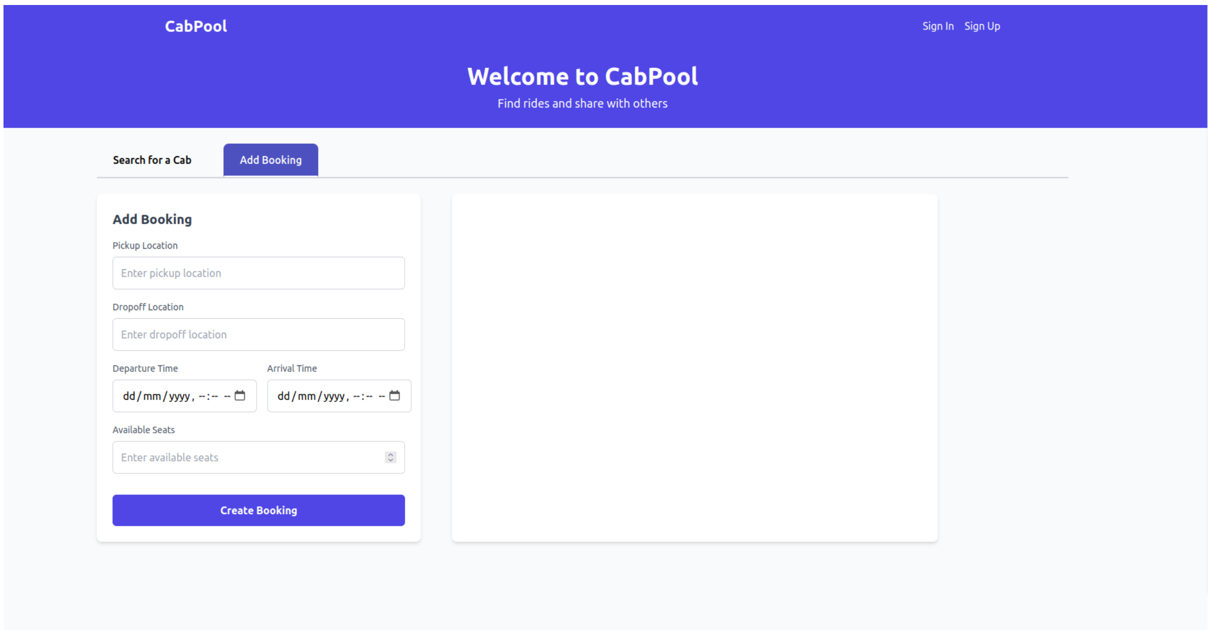


Figure 1.2: AddBooking

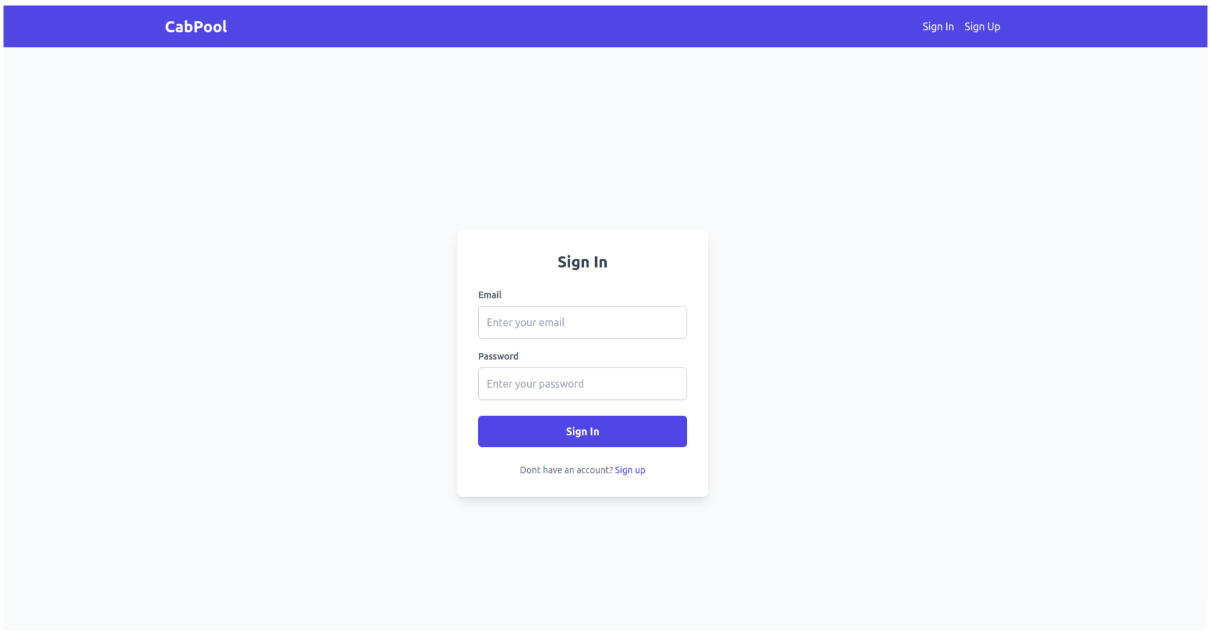


Figure 1.3: Sign in

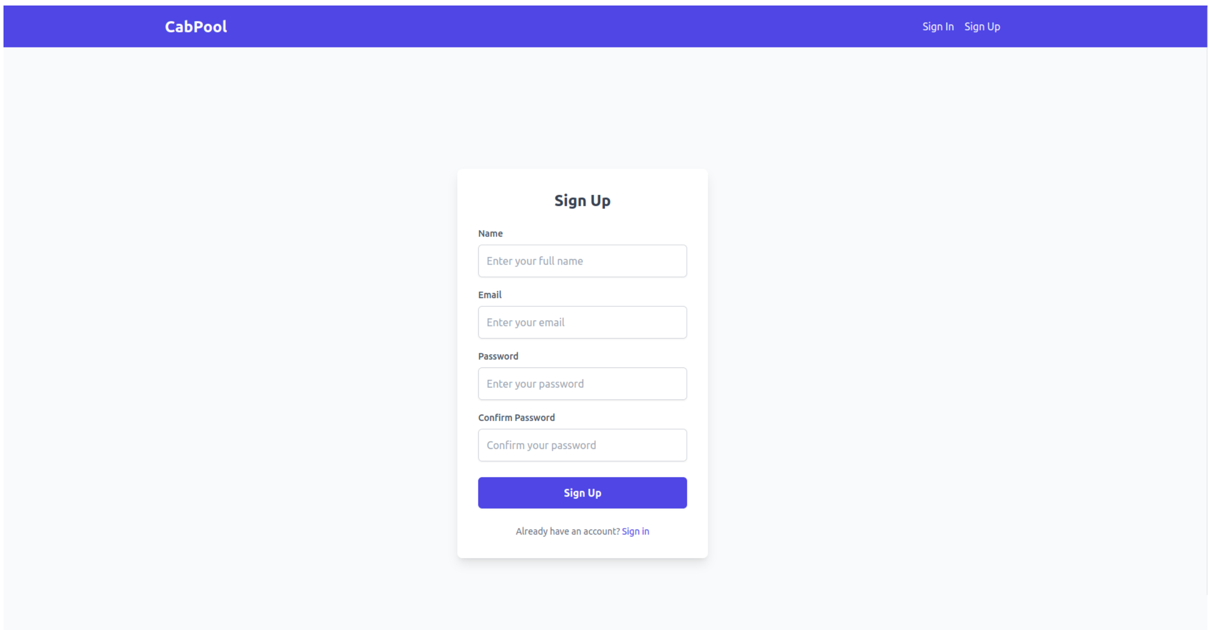
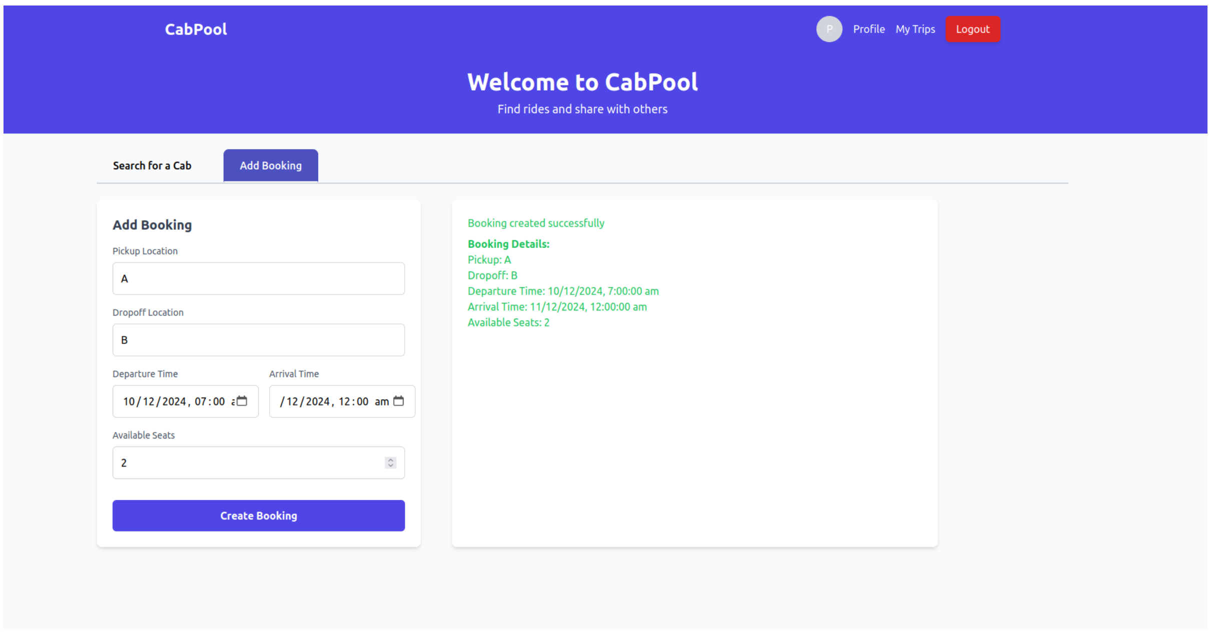


Figure 1.4: Sign Up



Welcome to CabPool

Find rides and share with others

Add Booking

Pickup Location: A

Dropoff Location: B

Departure Time: 10/12/2024, 07:00 a

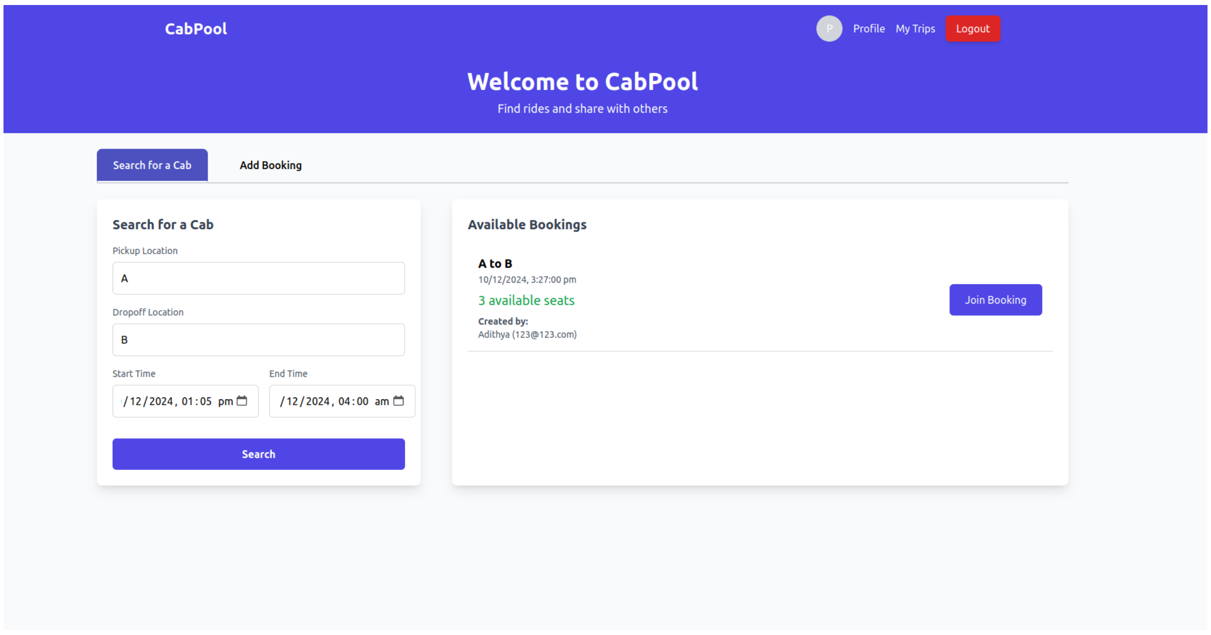
Arrival Time: /12/2024, 12:00 am

Available Seats: 2

Booking created successfully

Booking Details:
Pickup: A
Dropoff: B
Departure Time: 10/12/2024, 7:00:00 am
Arrival Time: 11/12/2024, 12:00:00 am
Available Seats: 2

Figure 1.5: Adding a Booking



Welcome to CabPool

Find rides and share with others

Search for a Cab

Pickup Location: A

Dropoff Location: B

Start Time: /12/2024, 01:05 pm

End Time: /12/2024, 04:00 am

Available Bookings

A to B
10/12/2024, 3:27:00 pm
3 available seats
Created by:
Adithya (123@123.com)

Figure 1.6: Searching for a Booking

Chapter 2

Architecture Overview and Getting Started

Backend

The backend employs **Node.js** with **Express.js**, a lightweight and flexible web application framework.

Database

MongoDB is used for data storage, with **Mongoose** for managing schemas and queries.

Frontend

The frontend is developed using **React.js**, providing a dynamic and responsive user interface.

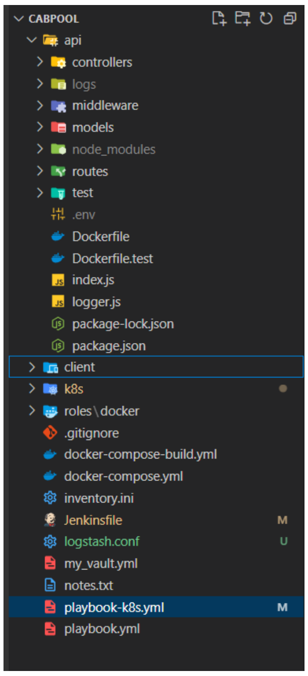
Authentication

JWT (JSON Web Tokens) is used for user-safe authentication. Passwords are hashed using **bcrypt.js**.

Logging and Monitoring

Winston is utilized for logging server activities, while system metrics are monitored using ELK Stack : elasticsearch, logisticsearch and kibana

2.1 Codebase Structure



```
CABPOOL
  api
    controllers
    logs
    middleware
    models
    node_modules
    routes
    test
    .env
    Dockerfile
    Dockerfile.test
    index.js
    logger.js
    package-lock.json
    package.json
  client
  k8s
  roles\docker
  .gitignore
  docker-compose-build.yml
  docker-compose.yml
  inventory.ini
  Jenkinsfile                  M
  logstash.conf                U
  my_vault.yml
  notes.txt
  playbook-k8s.yml             M
  playbook.yml
```

2.2 Tools Used

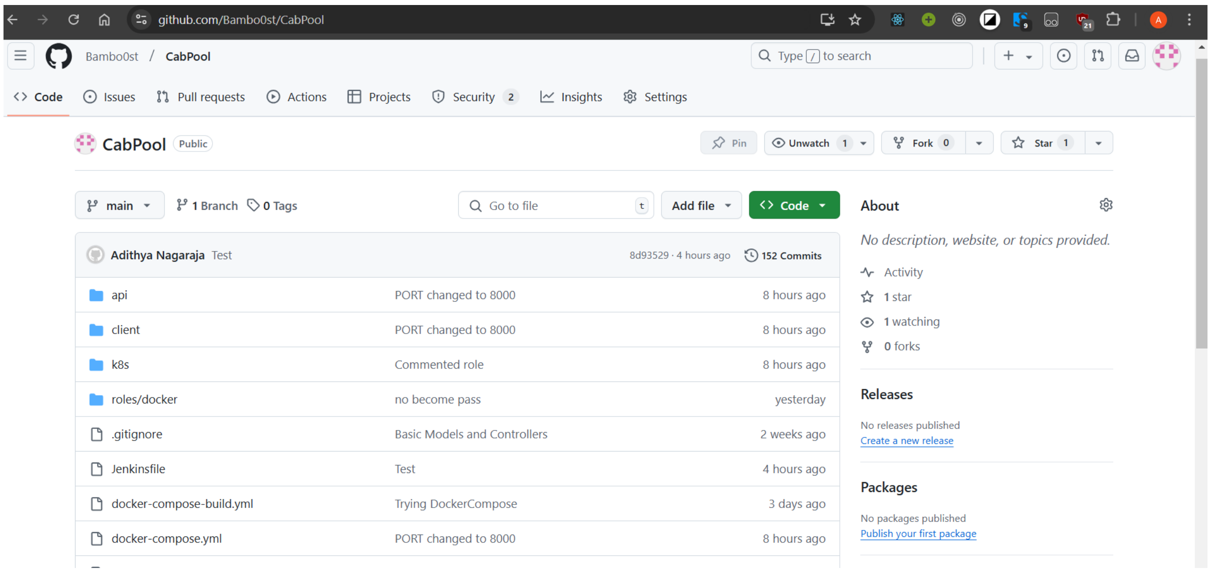
- **Git and Github:** For version control.
- **VSCode:** IDE suitable for developing MERN stack application
- **Jekins:** Used for continuous integration and deployment. Automates the pipeline for testing, building, and deploying the product.
- **k3s:** used to deploy orchestrated kubernetes containers on a single node.
- **webhooks:** Github webhook is used to trigger a build in Jenkins whenever there is a push to the remote GitHub repository.
- **ngrok:** Used to forward requests received at a public IP address to the localhost.
- **Docker and DockerCompose:** Used to containerize the application.
- **Ansible:** Used to easily configure hosts.
- **ELK Stack:** Tool used to analyze logs generated by the application for monitoring purposes.
- **Chai, ChaiHttp, Mocha** : Framework used for testing the backend api routes and functionalities.
- **Jest**: Framework used for testing the frontend (React componenents)
- **Kubernetes** : Container Orchestration Platform.

2.3 Setting Up

2.3.1 GitHub

- Open an empty directory and run the following command

```
git pull https://github.com/Bambo0st/CabPool
cd /Cabpool
```



No description, website, or topics provided.

No releases published
Create a new release

No packages published
Publish your first package

Adithya Nagaraja Test 8d93529 · 4 hours ago 152 Commits

api	PORT changed to 8000	8 hours ago
client	PORT changed to 8000	8 hours ago
k8s	Commented role	8 hours ago
roles/docker	no become pass	yesterday
.gitignore	Basic Models and Controllers	2 weeks ago
Jenkinsfile	Test	4 hours ago
docker-compose-build.yml	Trying DockerCompose	3 days ago
docker-compose.yml	PORT changed to 8000	8 hours ago

Figure 2.1: GitHub Repo

2.3.2 Installing Jenkins

Use the following commands to install Jenkins on the system:

```
sudo apt install ca-certificates
sudo apt update
sudo apt install jenkins
```

To start Jenkins, execute the following command:

```
sudo service start jenkins
```

2.3.3 Installing Docker

First, install the required dependencies:

```
sudo apt install curl
```

Then, run the following commands to install Docker:

```
curl -fsSL "https://get.docker.com" -o get-docker.sh
sh get-docker.sh
```

To verify if Docker was installed, you can check its version using:

```
docker --version
```

2.3.4 Installing Ansible

Run the following commands to install Ansible:

```
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt install ansible
```

2.3.5 Installing ngrok

To install ngrok, follow these steps:

First, download the ngrok zip file from ngrok.com. Once downloaded, extract the zip file. The instructions are available on the download page, so follow them to complete the extraction.

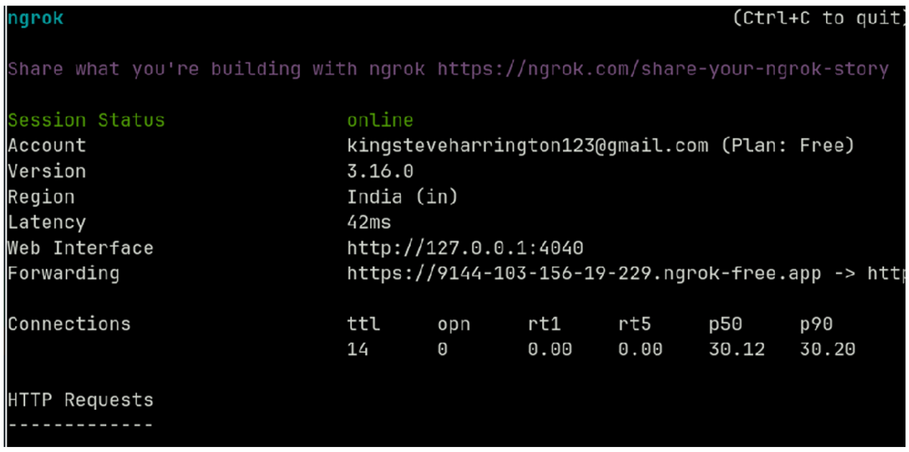
Once ngrok is available in the command line, sign up for a free account at ngrok.com and visit your account dashboard. There, you can find your auth token.

Then, run the following command to extract ngrok:

```
sudo tar xvzf ~/Downloads/ngrok-v3-stable-linux-amd64.tgz -C /usr/local/bin
```

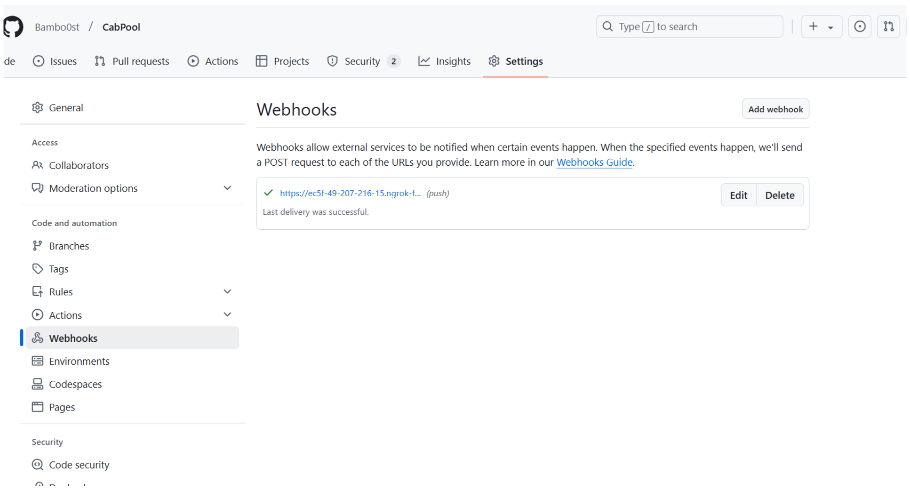
To connect your ngrok account with the auth token, use the following command:

```
ngrok config add-authtoken <your-authtoken>
```



```
ngrok                                                        (Ctrl+C to quit

Share what you're building with ngrok https://ngrok.com/share-your-ngrok-story

Session Status                online
Account                       kingsteveharrington123@gmail.com (Plan: Free)
Version                       3.16.0
Region                        India (in)
Latency                       42ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://9144-103-156-19-229.ngrok-free.app -> htt

Connections                   ttl     opn     rt1     rt5     p50     p90
                              14      0       0.00    0.00    30.12   30.20

HTTP Requests
-------------
```

Use this ngrok URL to create a GitHub webhook. For this go to settings of your GitHub repo and choose WebHook.



Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

✓ https://ec5f-49-207-216-15.ngrok-f... (push)

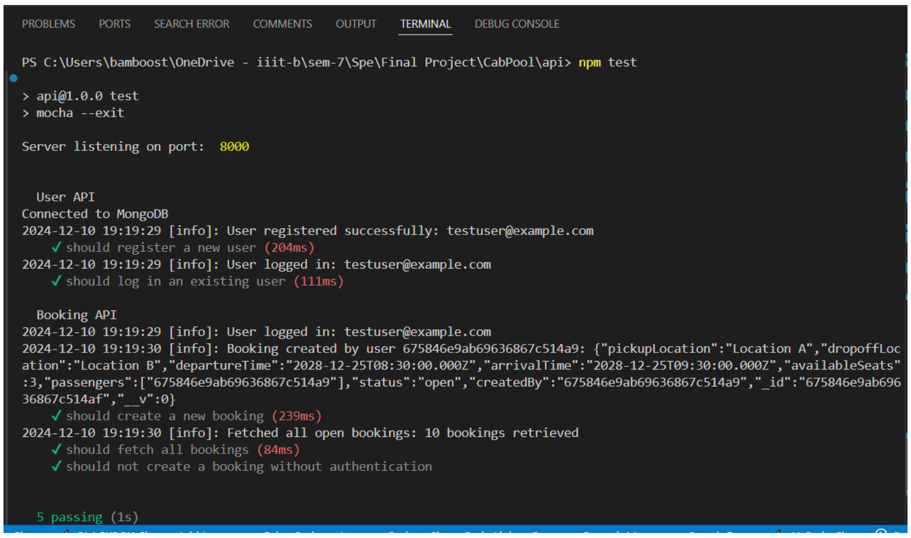
Last delivery was successful.

2.4 Testing

- The test suite thoroughly validates core functionalities like user authentication, booking creation, and data fetching while covering both positive and negative scenarios to ensure robust error handling.
- Backend tests use Mocha, Chai, and chai-http for API validation, while React Testing Library ensures that the front-end UI and interactions are user-friendly and error-resilient.
- Clear separation of test cases and realistic scenarios make the codebase maintainable and aligns well with CI / CD pipelines for early bug detection.

2.4.1 Backend Testing

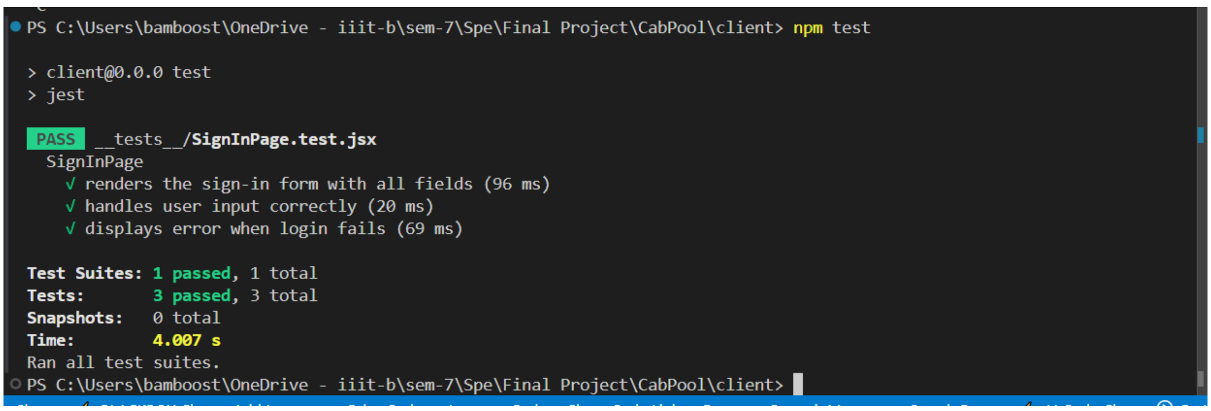
Run *npm test* in api and client.



```
PS C:\Users\bamboost\OneDrive - iiit-b\sem-7\Spe\Final Project\CabPool\api> npm test

> api@1.0.0 test
> mocha --exit

Server listening on port:  8000


  User API
Connected to MongoDB
2024-12-10 19:19:29 [info]: User registered successfully: testuser@example.com
    ✓ should register a new user (204ms)
2024-12-10 19:19:29 [info]: User logged in: testuser@example.com
    ✓ should log in an existing user (111ms)

  Booking API
2024-12-10 19:19:29 [info]: User logged in: testuser@example.com
2024-12-10 19:19:30 [info]: Booking created by user 675846e9ab69636867c514a9: {"pickupLocation":"Location A","dropoffLoc
ation":"Location B","departureTime":"2028-12-25T08:30:00.000Z","arrivalTime":"2028-12-25T09:30:00.000Z","availableSeats"
:3,"passengers":["675846e9ab69636867c514a9"],"status":"open","createdBy":"675846e9ab69636867c514a9","_id":"675846e9ab696
36867c514af","__v":0}
    ✓ should create a new booking (239ms)
2024-12-10 19:19:30 [info]: Fetched all open bookings: 10 bookings retrieved
    ✓ should fetch all bookings (84ms)
    ✓ should not create a booking without authentication


  5 passing (1s)
```

Figure 2.2: BackEnd Testing

2.4.2 Frontend Testing



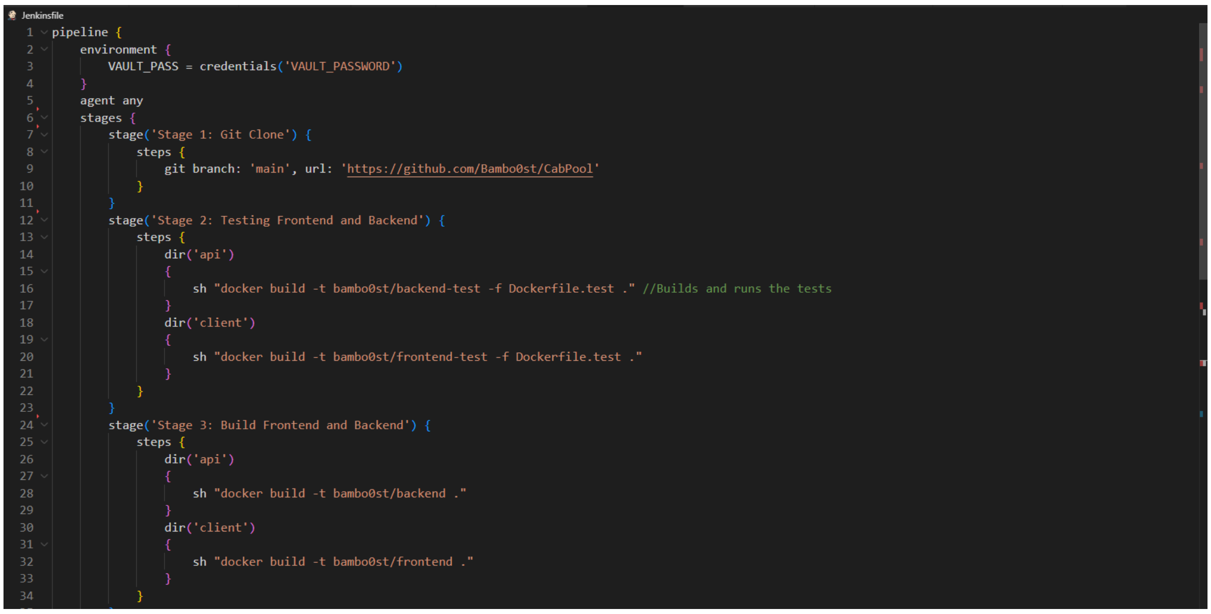
```
PS C:\Users\bamboost\OneDrive - iiit-b\sem-7\Spe\Final Project\CabPool\client> npm test

> client@0.0.0 test
> jest

 PASS  __tests__/SignInPage.test.jsx
  SignInPage
    √ renders the sign-in form with all fields (96 ms)
    √ handles user input correctly (20 ms)
    √ displays error when login fails (69 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        4.007 s
Ran all test suites.
PS C:\Users\bamboost\OneDrive - iiit-b\sem-7\Spe\Final Project\CabPool\client>
```

Figure 2.3: FrontEnd Testing

Chapter 3

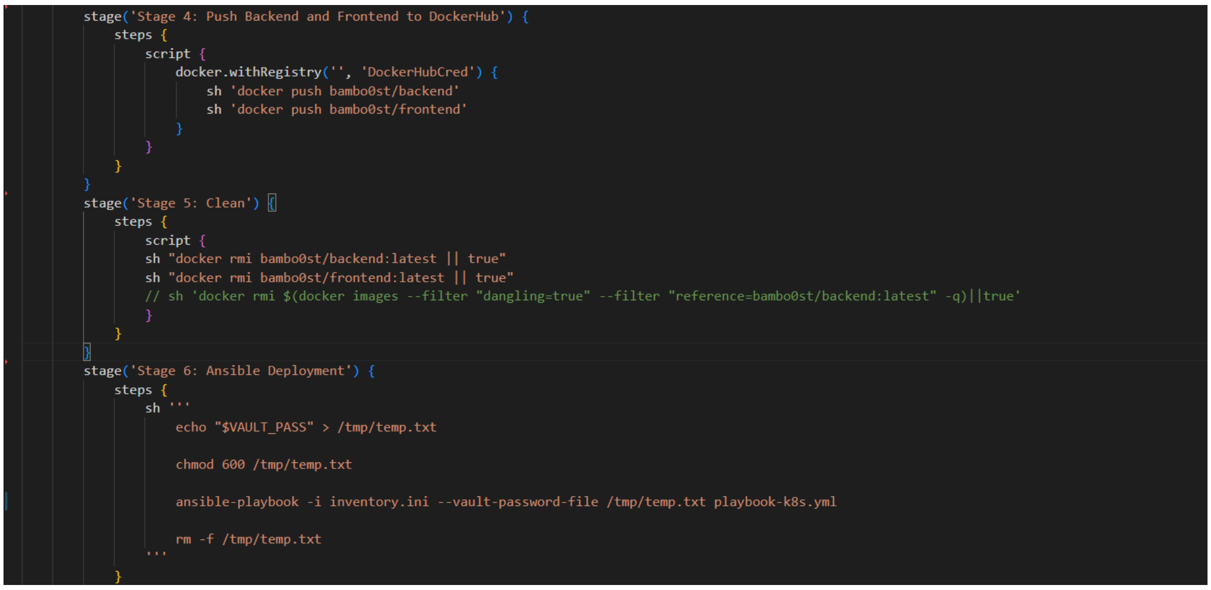
CI/CD and Container Orchestration

3.1 Jenkins Pipeline



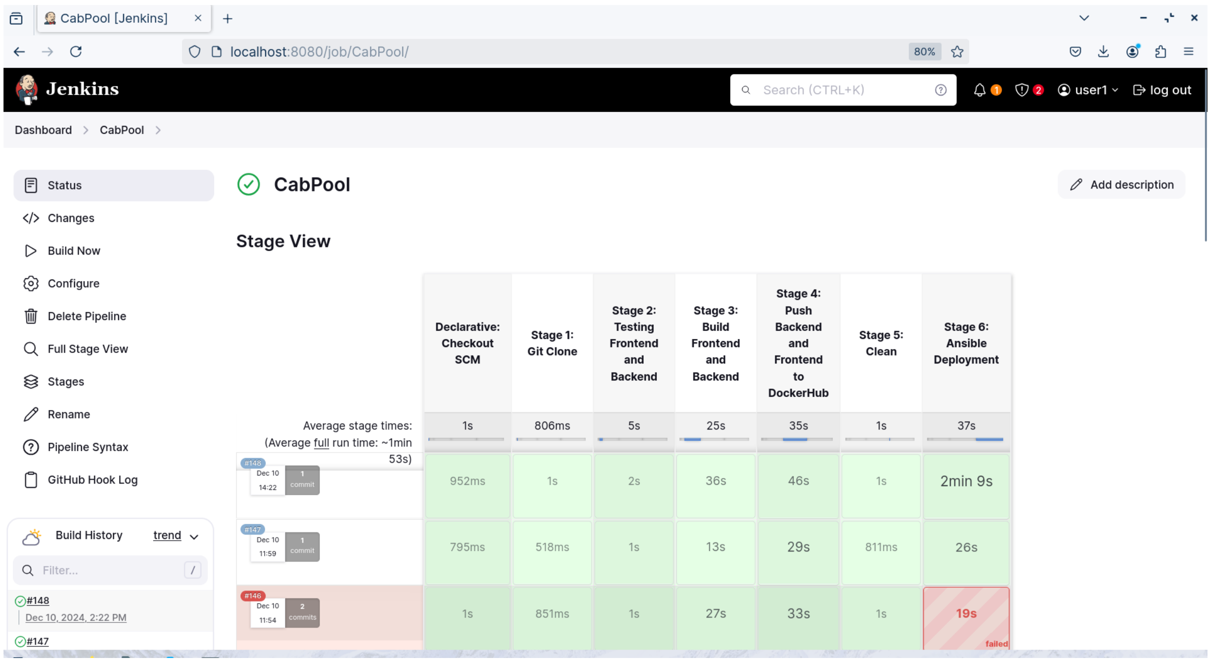
Jenkinsfile

```
pipeline {
    environment {
        VAULT_PASS = credentials('VAULT_PASSWORD')
    }
    agent any
    stages {
        stage('Stage 1: Git Clone') {
            steps {
                git branch: 'main', url: 'https://github.com/Bambo0st/CabPool'
            }
        }
        stage('Stage 2: Testing Frontend and Backend') {
            steps {
                dir('api')
                {
                    sh "docker build -t bambo0st/backend-test -f Dockerfile.test ." //Builds and runs the tests
                }
                dir('client')
                {
                    sh "docker build -t bambo0st/frontend-test -f Dockerfile.test ."
                }
            }
        }
        stage('Stage 3: Build Frontend and Backend') {
            steps {
                dir('api')
                {
                    sh "docker build -t bambo0st/backend ."
                }
                dir('client')
                {
                    sh "docker build -t bambo0st/frontend ."
                }
            }
```



```
        stage('Stage 4: Push Backend and Frontend to DockerHub') {
            steps {
                script {
                    docker.withRegistry('', 'DockerHubCred') {
                        sh 'docker push bamboOst/backend'
                        sh 'docker push bamboOst/frontend'
                    }
                }
            }
        }
        stage('Stage 5: Clean') {
            steps {
                script {
                sh "docker rmi bamboOst/backend:latest || true"
                sh "docker rmi bamboOst/frontend:latest || true"
                // sh 'docker rmi $(docker images --filter "dangling=true" --filter "reference=bamboOst/backend:latest" -q)||true'
                }
            }
        }
        stage('Stage 6: Ansible Deployment') {
            steps {
                sh '''
                    echo "$VAULT_PASS" > /tmp/temp.txt

                    chmod 600 /tmp/temp.txt

                    ansible-playbook -i inventory.ini --vault-password-file /tmp/temp.txt playbook-k8s.yml

                    rm -f /tmp/temp.txt
                '''
            }
```

This is the output that follows when the Jenkins pipeline is run.



CabPool

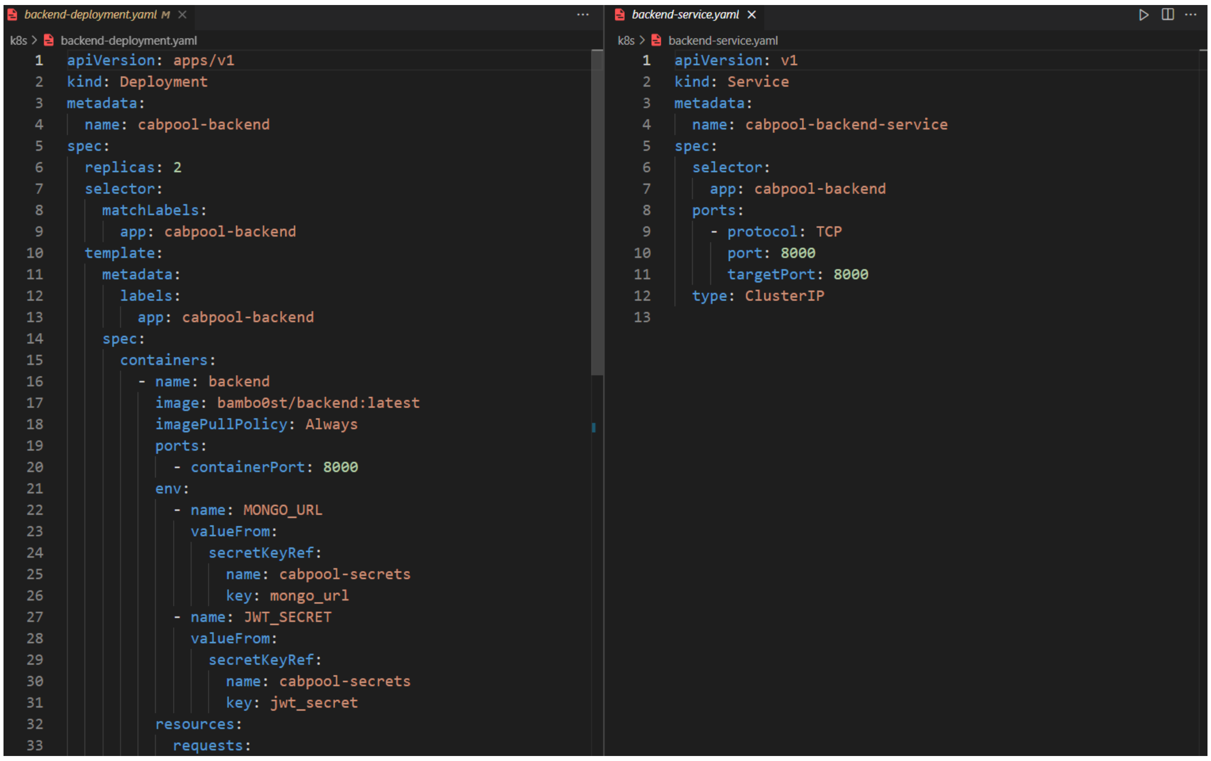
Stage View

	Declarative: Checkout SCM	Stage 1: Git Clone	Stage 2: Testing Frontend and Backend	Stage 3: Build Frontend and Backend	Stage 4: Push Backend and Frontend to DockerHub	Stage 5: Clean	Stage 6: Ansible Deployment
Average stage times: (Average full run time: ~1min 53s)	1s	806ms	5s	25s	35s	1s	37s
#148 Dec 10 14:22 1 commit	952ms	1s	2s	36s	46s	1s	2min 9s
#147 Dec 10 11:59 1 commit	795ms	518ms	1s	13s	29s	811ms	26s
#146 Dec 10 11:54 2 commits	1s	851ms	1s	27s	33s	1s	19s failed

#148
Dec 10, 2024, 2:22 PM
#147

3.2 Containerization

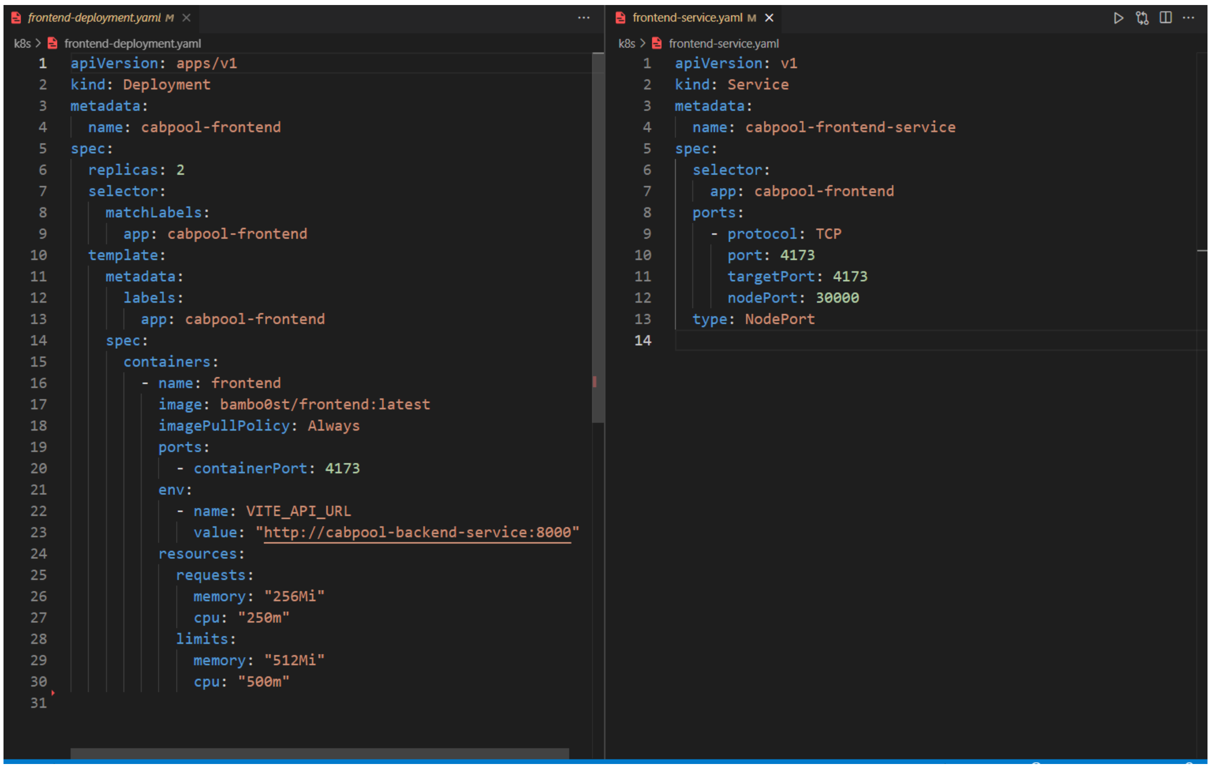
The following are backend-deployment.yaml and backend-service.yaml files


backend-deployment.yaml M
k8s > backend-deployment.yaml
backend-service.yaml
k8s > backend-service.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cabpool-backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cabpool-backend
  template:
    metadata:
      labels:
        app: cabpool-backend
    spec:
      containers:
        - name: backend
          image: bambo0st/backend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8000
          env:
            - name: MONGO_URL
              valueFrom:
                secretKeyRef:
                  name: cabpool-secrets
                  key: mongo_url
            - name: JWT_SECRET
              valueFrom:
                secretKeyRef:
                  name: cabpool-secrets
                  key: jwt_secret
          resources:
            requests:
```

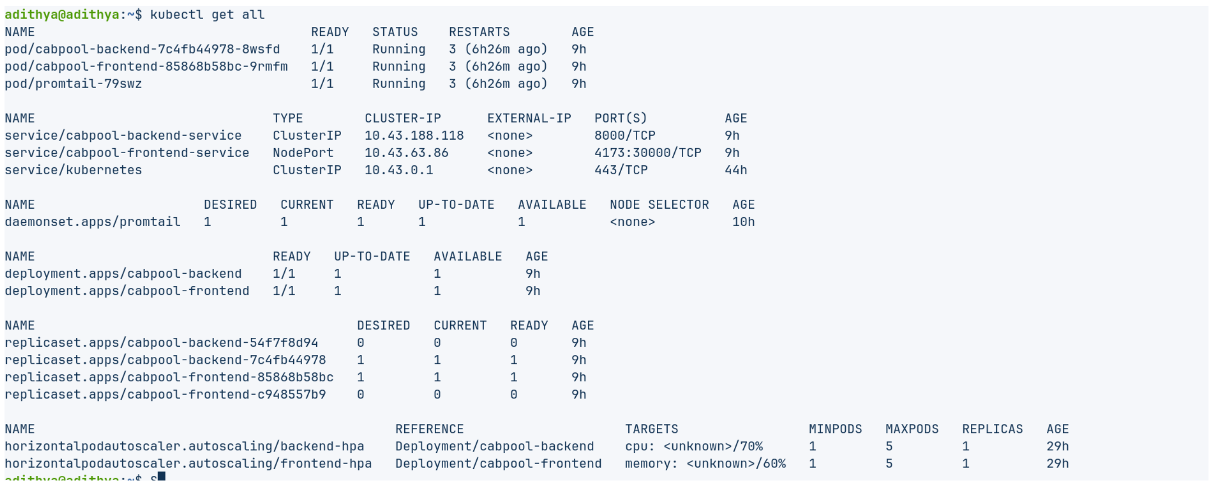
```yaml
apiVersion: v1
kind: Service
metadata:
  name: cabpool-backend-service
spec:
  selector:
    app: cabpool-backend
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
  type: ClusterIP
```

The following are frontend-deployment.yaml and frontend-service.yaml files

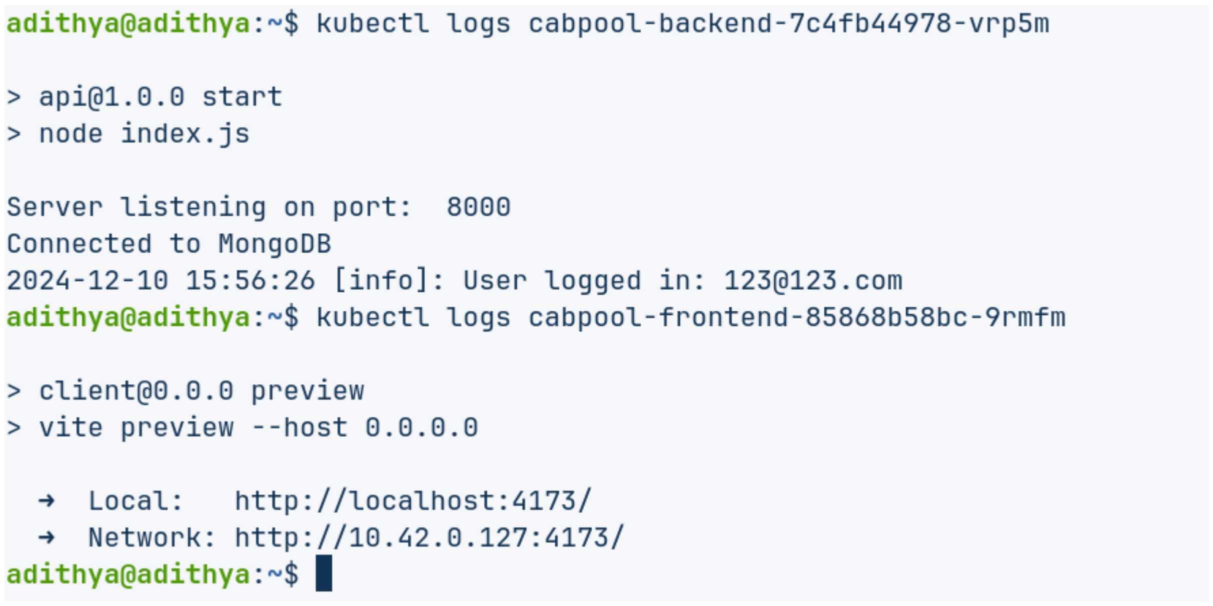

frontend-deployment.yaml M
k8s > frontend-deployment.yaml
frontend-service.yaml M
k8s > frontend-service.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cabpool-frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cabpool-frontend
  template:
    metadata:
      labels:
        app: cabpool-frontend
    spec:
      containers:
        - name: frontend
          image: bambo0st/frontend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 4173
          env:
            - name: VITE_API_URL
              value: "http://cabpool-backend-service:8000"
          resources:
            requests:
              memory: "256Mi"
              cpu: "250m"
            limits:
              memory: "512Mi"
              cpu: "500m"
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: cabpool-frontend-service
spec:
  selector:
    app: cabpool-frontend
  ports:
    - protocol: TCP
      port: 4173
      targetPort: 4173
      nodePort: 30000
  type: NodePort
```



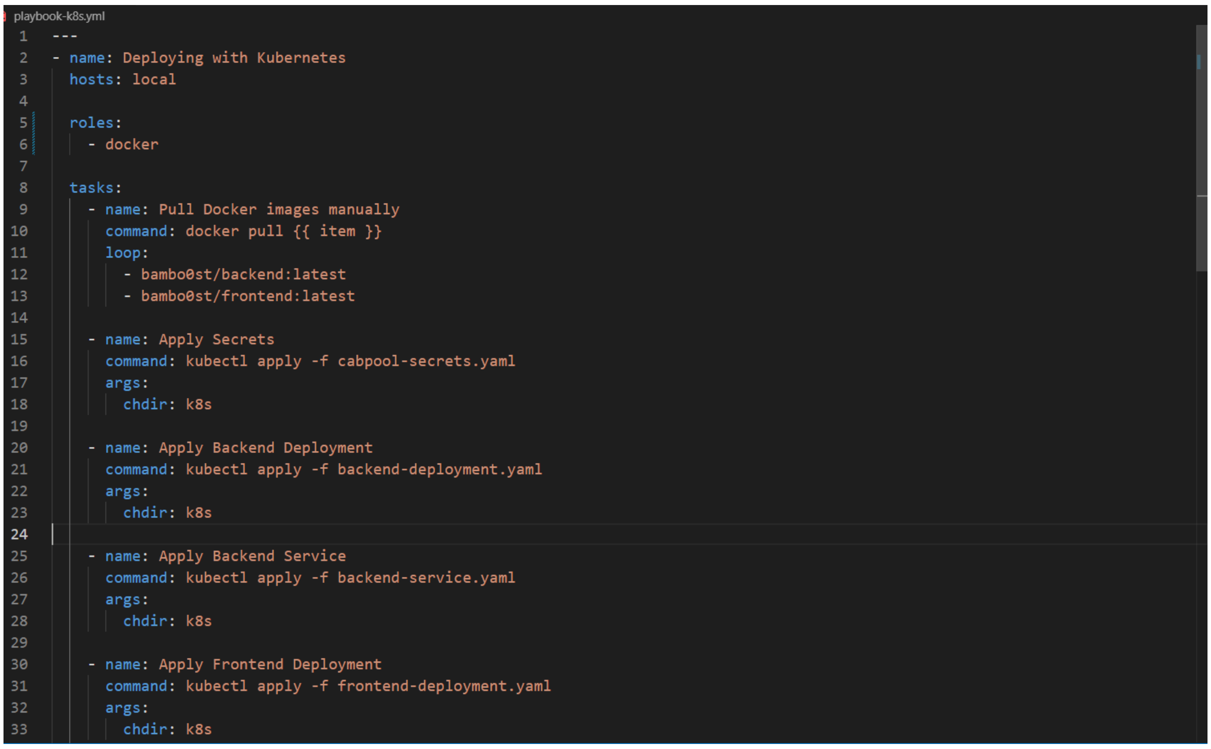
```
adithya@adithya:~$ kubectl get all
NAME                                  READY   STATUS    RESTARTS        AGE
pod/cabpool-backend-7c4fb44978-8wsfd    1/1     Running   3 (6h26m ago)   9h
pod/cabpool-frontend-85868b58bc-9rmfm   1/1     Running   3 (6h26m ago)   9h
pod/promtail-79swz                      1/1     Running   3 (6h26m ago)   9h

NAME                               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/cabpool-backend-service    ClusterIP   10.43.188.118   <none>        8000/TCP         9h
service/cabpool-frontend-service   NodePort    10.43.63.86     <none>        4173:30000/TCP   9h
service/kubernetes                 ClusterIP   10.43.0.1       <none>        443/TCP          44h

NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/promtail   1         1         1       1            1           <none>          10h

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cabpool-backend    1/1     1            1           9h
deployment.apps/cabpool-frontend   1/1     1            1           9h

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/cabpool-backend-54f7f8d94     0         0         0       9h
replicaset.apps/cabpool-backend-7c4fb44978    1         1         1       9h
replicaset.apps/cabpool-frontend-85868b58bc   1         1         1       9h
replicaset.apps/cabpool-frontend-c948557b9    0         0         0       9h

NAME                                                   REFERENCE                      TARGETS                  MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/backend-hpa    Deployment/cabpool-backend    cpu: <unknown>/70%      1         5         1          29h
horizontalpodautoscaler.autoscaling/frontend-hpa   Deployment/cabpool-frontend   memory: <unknown>/60%   1         5         1          29h
adithya@adithya:~$
```



```
adithya@adithya:~$ kubectl logs cabpool-backend-7c4fb44978-vrp5m

> api@1.0.0 start
> node index.js

Server listening on port:  8000
Connected to MongoDB
2024-12-10 15:56:26 [info]: User logged in: 123@123.com
adithya@adithya:~$ kubectl logs cabpool-frontend-85868b58bc-9rmfm

> client@0.0.0 preview
> vite preview --host 0.0.0.0

  ➜  Local:   http://localhost:4173/
  ➜  Network: http://10.42.0.127:4173/
adithya@adithya:~$
```

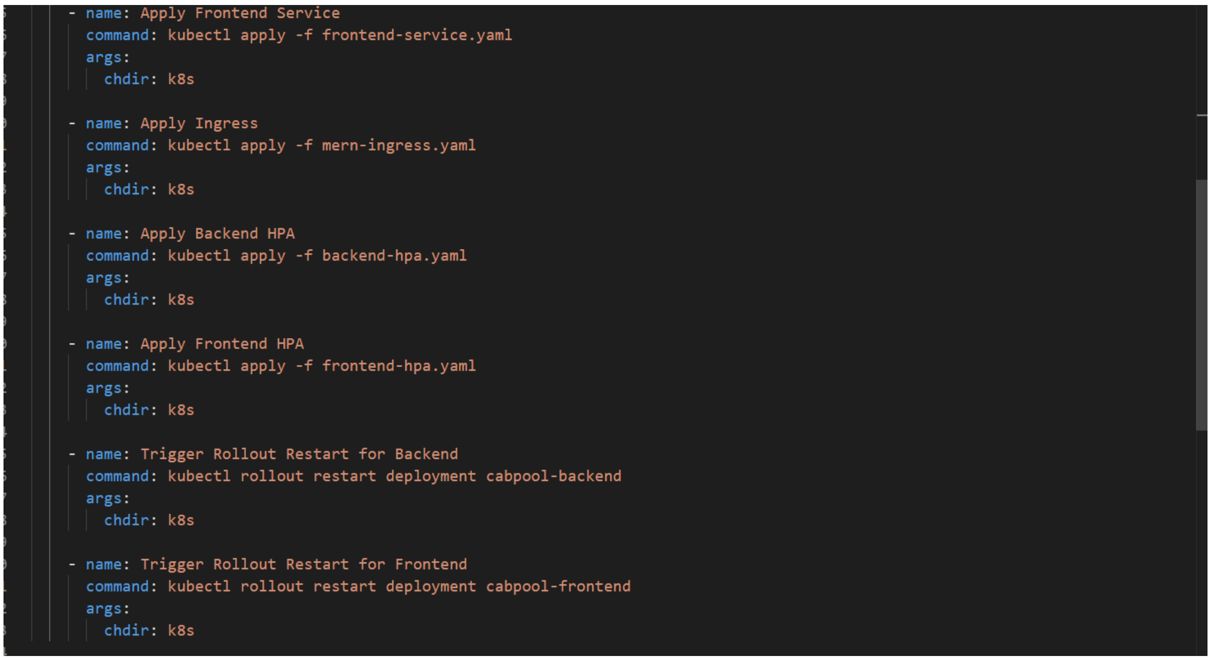
The above figure shows that both frontend and backend pods work properly.

3.3 Ansible playbook for Kubernetes



playbook-k8s.yml

```yaml
---
- name: Deploying with Kubernetes
  hosts: local

  roles:
    - docker

  tasks:
    - name: Pull Docker images manually
      command: docker pull {{ item }}
      loop:
        - bambo0st/backend:latest
        - bambo0st/frontend:latest

    - name: Apply Secrets
      command: kubectl apply -f cabpool-secrets.yaml
      args:
        chdir: k8s

    - name: Apply Backend Deployment
      command: kubectl apply -f backend-deployment.yaml
      args:
        chdir: k8s

    - name: Apply Backend Service
      command: kubectl apply -f backend-service.yaml
      args:
        chdir: k8s

    - name: Apply Frontend Deployment
      command: kubectl apply -f frontend-deployment.yaml
      args:
        chdir: k8s
```



```yaml
    - name: Apply Frontend Service
      command: kubectl apply -f frontend-service.yaml
      args:
        chdir: k8s

    - name: Apply Ingress
      command: kubectl apply -f mern-ingress.yaml
      args:
        chdir: k8s

    - name: Apply Backend HPA
      command: kubectl apply -f backend-hpa.yaml
      args:
        chdir: k8s

    - name: Apply Frontend HPA
      command: kubectl apply -f frontend-hpa.yaml
      args:
        chdir: k8s

    - name: Trigger Rollout Restart for Backend
      command: kubectl rollout restart deployment cabpool-backend
      args:
        chdir: k8s

    - name: Trigger Rollout Restart for Frontend
      command: kubectl rollout restart deployment cabpool-frontend
      args:
        chdir: k8s
```

The above playbook initializes the kubernetes deployment and services of frontend and backend. The playbook adds Trigger Rollout which makes sure to implement the changes without in downtime.

Chapter 4

Monitoring with ELK stack

Tools

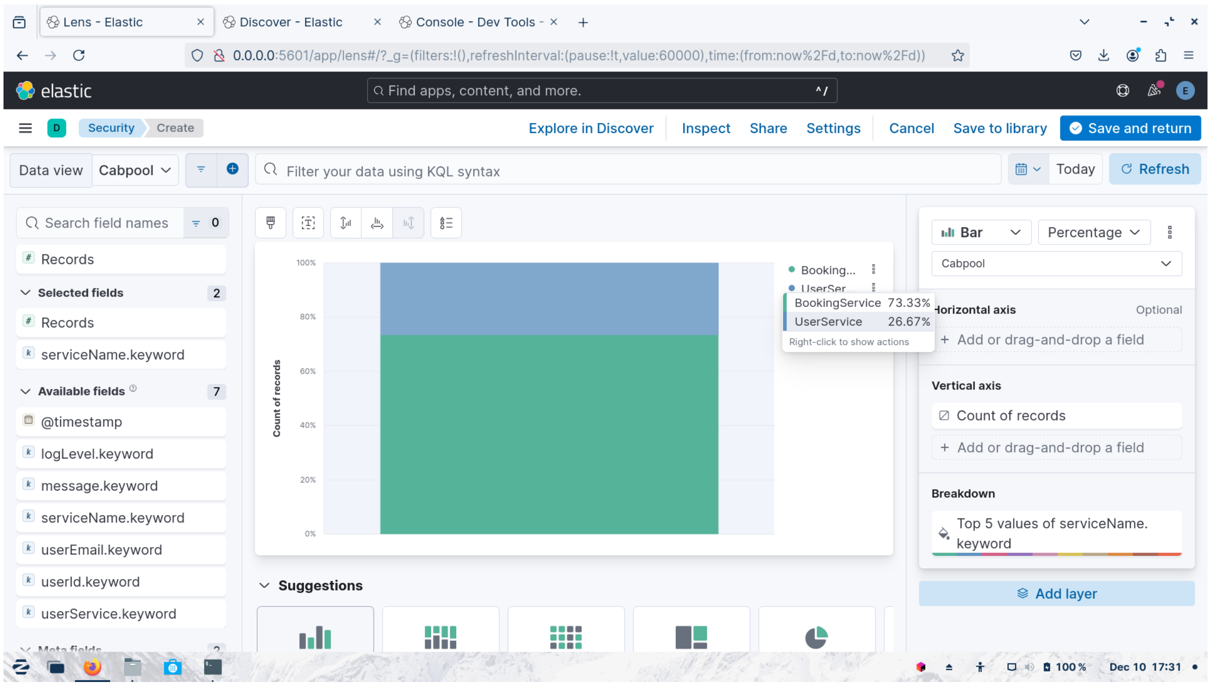
Logs are managed using the ELK stack:

- **Logstash:** Filters and processes logs.
- **Elasticsearch:** Indexes logs.
- **Kibana:** Provides a dashboard for real-time log visualization.

Visualizations

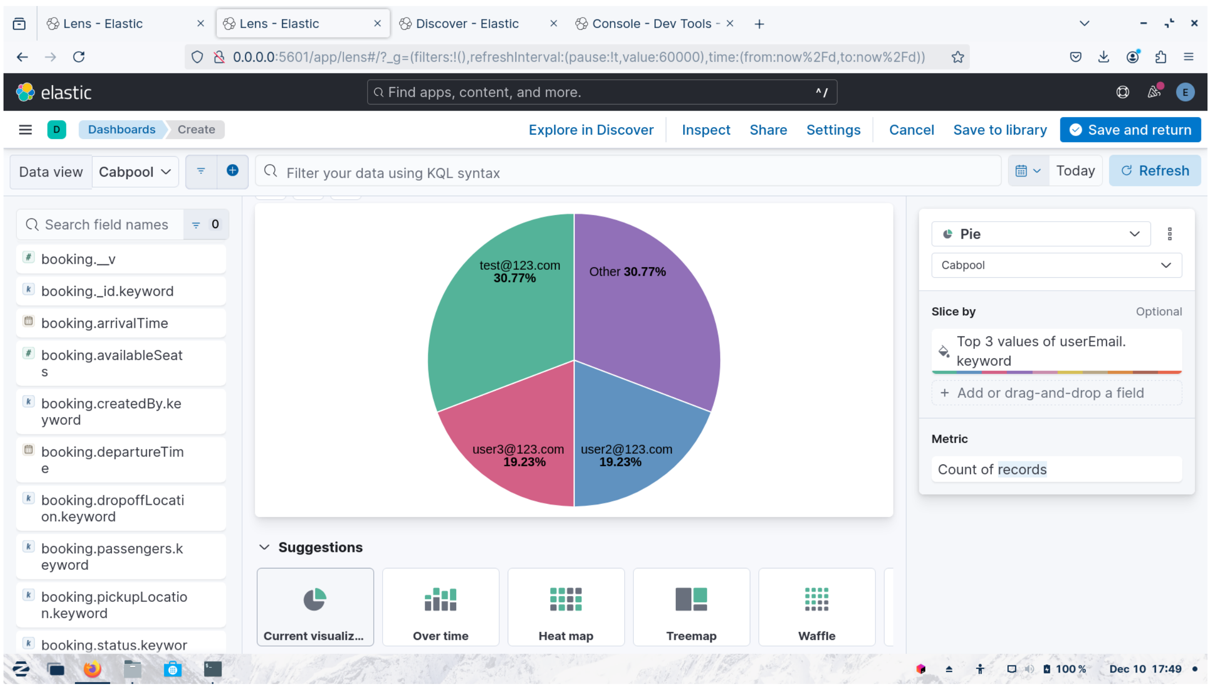
Following are the visualizations on Kibana dashboard.

- **UserServices vs BookingServices**



This visualization compares the requests made to user services and request made to Booking services.

- **Comparison of requests made by users**


0.0.0.0:5601/app/lens#/?_g=(filters:!(),refreshInterval:(pause:!t,value:60000),time:(from:now%2Fd,to:now%2Fd))

Data view Cabpool

Filter your data using KQL syntax

Pie

Cabpool

Slice by Optional

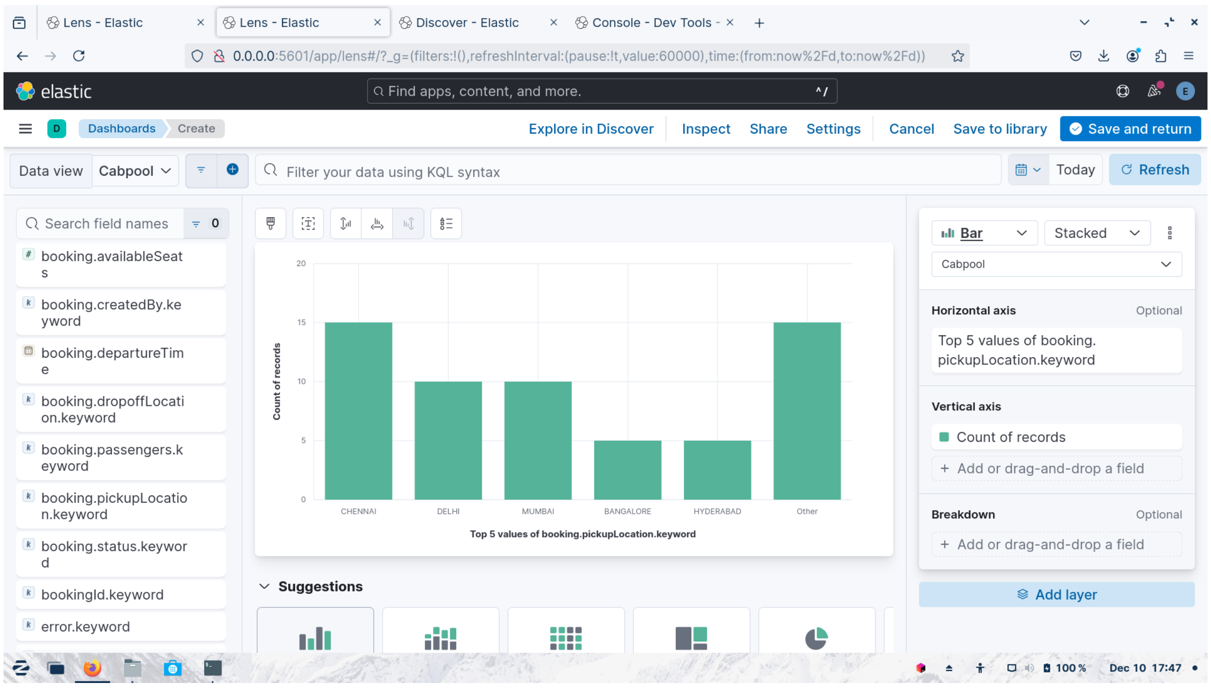
Top 3 values of userEmail.keyword

Metric

Count of records

From the above visualization it can be infered that the user with email test@123.com likes our web application as he is actively using it.

- **Comparison of pickUpLocation picked**


0.0.0.0:5601/app/lens#/?_g=(filters:!(),refreshInterval:(pause:!t,value:60000),time:(from:now%2Fd,to:now%2Fd))

Data view Cabpool

Filter your data using KQL syntax

Bar Stacked

Cabpool

Horizontal axis Optional

Top 5 values of booking.pickupLocation.keyword

Vertical axis

Count of records

Breakdown Optional

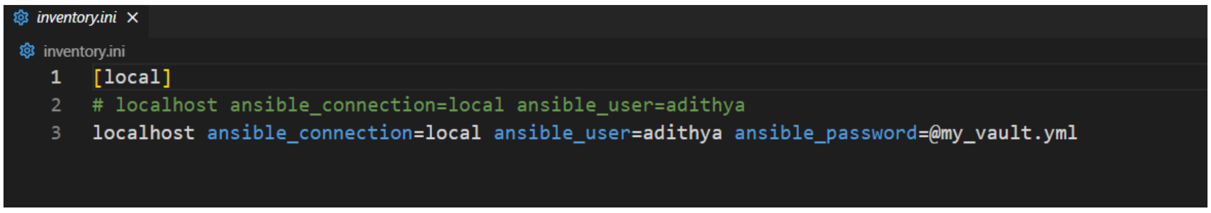
From the above visualization it is clear that Chennai is highly chosen as a pick up location.

Chapter 5

Evaluation Expectations

5.1 Vault

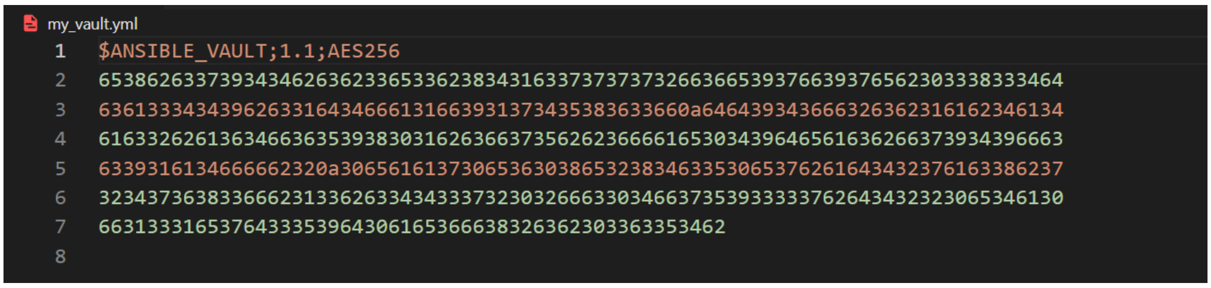
Vault has been implemented to store the localhost user credentials.



inventory.ini

```
[local]
# localhost ansible_connection=local ansible_user=adithya
localhost ansible_connection=local ansible_user=adithya ansible_password=@my_vault.yml
```

The vault stores the localhost credentials in encrypted file.

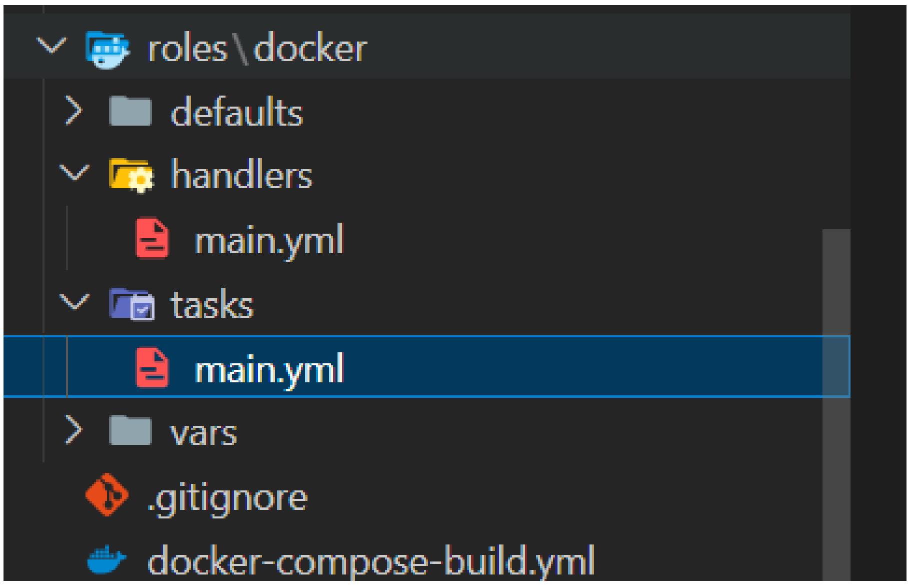


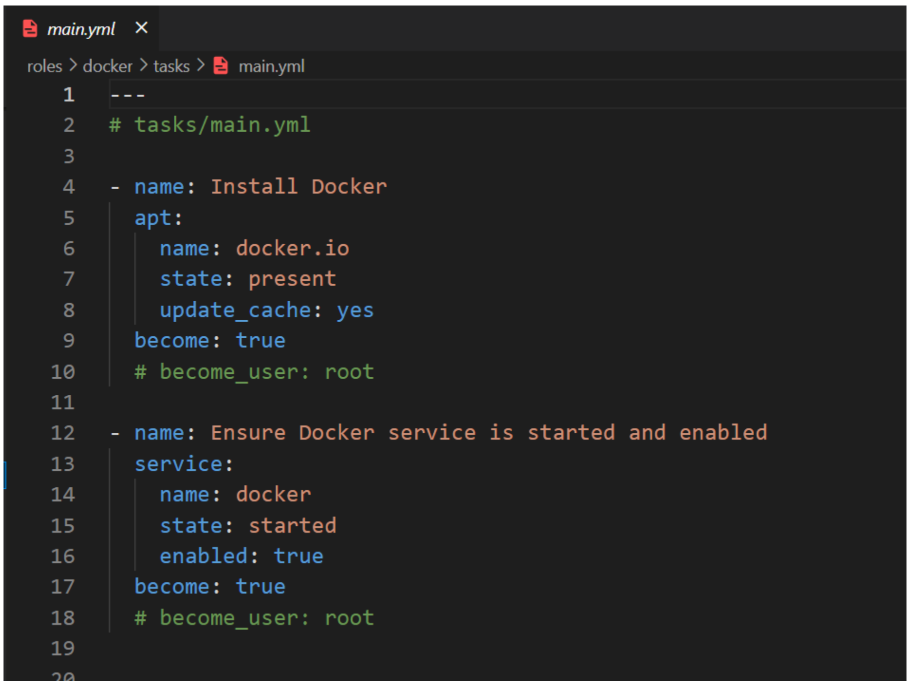
my_vault.yml

```
$ANSIBLE_VAULT;1.1;AES256
65386263373934346263623365336238343163373737373266366539376639376562303338333464
6361333434396263316434666131663931373435383633660a646439343666326362316162346134
61633262613634663635393830316263663735626236666165303439646561636266373934396663
6339316134666662320a306561613730653630386532383463353065376261643432376163386237
32343736383366623133626334343337323032666330346637353933333762643432323065346130
663133316537643335396430616536663832636230336363353462

```

5.2 Roles

Role called docker has been used which ensures that the docker has been installed in the system and docker service has started. This role is used in both playbook.yaml (docker-compose playbook) and playbook-k8s.yaml (Kuberenetes playbook). This way the code has been modularised rather than having to write the installation in both the files.

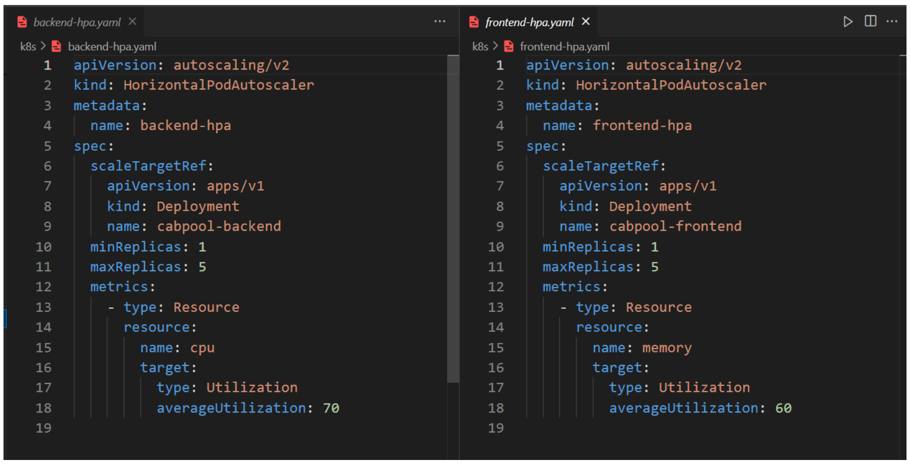




```yaml
---
# tasks/main.yml

- name: Install Docker
  apt:
    name: docker.io
    state: present
    update_cache: yes
  become: true
  # become_user: root

- name: Ensure Docker service is started and enabled
  service:
    name: docker
    state: started
    enabled: true
  become: true
  # become_user: root

```

5.3 High Availability and Scalability

For this Horizontal Pod Autoscaler has been deployed to make sure that pods are scaled based on the requirment.

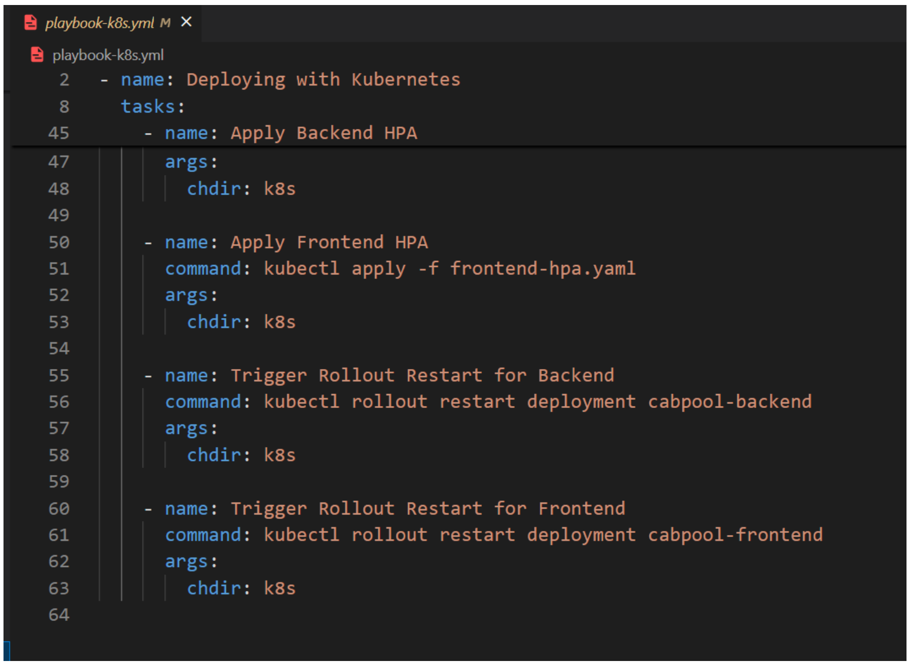


```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cabpool-backend
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70

```

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cabpool-frontend
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 60

```

5.4 Live Patching

The current deployment approach ensures that live patching works. The rollout feature of kubernetes has been used for this.



```yaml
- name: Deploying with Kubernetes
  tasks:
    - name: Apply Backend HPA
      args:
        chdir: k8s

    - name: Apply Frontend HPA
      command: kubectl apply -f frontend-hpa.yaml
      args:
        chdir: k8s

    - name: Trigger Rollout Restart for Backend
      command: kubectl rollout restart deployment cabpool-backend
      args:
        chdir: k8s

    - name: Trigger Rollout Restart for Frontend
      command: kubectl rollout restart deployment cabpool-frontend
      args:
        chdir: k8s

```

From the figure it is seen that rollout feature has been implemented for both frontend and backend, so even if you make a small change like changing the title name and push it to Github. Kubernetes makes sure that the changes take place within the services. This is done by using additional pod to avoid downtime.