SPE Final Project

Motor Vehicle Ticket Resolver

Madhav Sood (IMT2021009)
Shlok Agrawal (IMT2021103)

Abstract:

The Motor Vehicle Ticket Resolver (MVTR) is a website where users can create service tickets for two-wheelers or four-wheelers.

Available services include Cleaning, Deep Cleaning, Painting, Engine Repair, A/C Repair, Tyre Repair, and Oil Change. Users can track the status of their tickets, such as whether a request is pending or completed, along with acknowledgments from the owner. They can also raise multiple tickets for various vehicles, providing details like the vehicle number plate, vehicle type (two-wheeler/four-wheeler), remarks, and their address.

Owners can view user requests in a dashboard that displays details such as Request ID, Vehicle ID, remarks (if any), and other user information. They can resolve requests using a "Mark as Done" button and send acknowledgments, which users can view on their dashboards.

Tech Stack Used:

1. Version Control System: Git and GitHub
2. Continuous Integration/Continuous Delivery: Jenkins
3. Containerization: Docker and Docker Compose
4. Deployment: Ansible
5. Orchestration and Scaling: Kubernetes (K8s through Minikube)
6. Monitoring: ELK(Elasticsearch, Logstash, Kibana)
7. Git SCM Polling and Build Automation: Ngrok, GitHub webhooks
8. Front-end: React JS
9. Back-end: Spring Boot

10. Database: MySQL

Features:

1. User :
    a. The user can register to the portal to raise a ticket
    b. The user can raise a new ticket
    c. The user can view the status of their requested ticket
    d. The user can create multiple tickets for different vehicles

2. Owner:
    a. Owner can view the tickets raised by the user
    b. Owner can send acknowledgment to the user
    c. Owner can resolve a ticket raised by the user

Introduction:

The entire codebase is first pushed to a GitHub repository, serving as the source code management tool, in two phases: frontend and backend. Jenkins, a continuous integration tool, pulls the code from the repository and performs tasks such as compilation (build), code review, testing, and ultimately generates a JAR file.

After a successful build, the JAR file is used to create a Docker image. This image is The resulting Docker image is pushed to Docker Hub and later deployed to production servers using automation tools like Ansible or Rundeck.

The final step involves monitoring the system using tools like the ELK Stack.

We have deployed the application using Ansible in 2 ways:

1. Docker Compose with Jenkins
2. Kubernetes with Jenkins

Implementation:

Github:

a. Backend: https://github.com/madhav0407/TicketResolver_Backend.git
b. Frontend (Docker Compose): https://github.com/madhav0407/TicketResolver_Frontend.git
c. Frontend (Kubernetes): https://github.com/madhav0407/TicketResolver_Kube.git

Docker:

a. Backend: https://hub.docker.com/repository/docker/madhavsood04/ticketresolver-backend/general
b. Frontend: https://hub.docker.com/repository/docker/madhavsood04/ticketresolver_frontend/general

MySQL:

MySQL is an open-source relational database management system (RDBMS) that organizes data into related tables, providing structure and ease of access. Using SQL, it enables programmers to manage, query, and control access to the database. MySQL also integrates with the operating system to handle storage, manage users, enable network access, ensure data integrity, and support backups.

Screenshots of the database:

mysql> show tables;

Tables_in_ticketresolver
complaints
custumer
hibernate_sequence
roles
user_details
users_roles

mysql> select * from complaints;

com_id	address	aknow	remark	services	status	vehicle_no	vehicle_type	customer_id
1	M2UlGwjLJag3D5fHL/Jh+g==	Done!!	Ax1MJ6XRMrUdgN1XfbeD1A==	0xACED0005737200136A6176612E7574696C2E41727261794C6973747881D21D99C7619D03000149000473697A65787000000002770400000002740008436C65616E696E677400085061696E74696E6778	8z364LFC0lGMRrJzPLMcuw==	IK5mG9507TgDjVL0nJ563w==	8iU1bg7pS0Eh5/hKyvqW3UNqMFa60B9zbAHWLgb2w5o=	2
2	kqpcolUJEa2jBy9RA09IMA==	Done	Ax1MJ6XRMrUdgN1XfbeD1A==	0xACED0005737200136A6176612E7574696C2E41727261794C6973747881D21D99C7619D03000149000473697A65787000000003770400000003740008436C65616E696E6774000A412F43207265706169727400D4465657020436C65616E696E6778	zwxtDF59KYw+YPYEltP7HdC/saPEwnkBKSfOfISGh5vDWVtxb86bhr5elJpjz/Eg	/OsEJdh8TTl2+KFNBDc4CQ==	8iU1bg7pS0Eh5/hKyvqW3UNqMFa60B9zbAHWLgb2w5o=	3

mysql> select * from custumer;

id	email	first_name	last_name	password	username
1	NULL	NULL	NULL	owner	owner
2	madhav.sood0407@gmail.com	Madhav	Sood	1234	maddy
3	addy@gmail.com	Aditya	Sood	1234	addy

Source Code Management:

A Source Code Management is an application used by developers to collaborate and work on the project together. We have used GIT for Source code Management. For our project we have created a repository and cloned them on our local host and worked on them individually.

Some basics commands used

1. git init : It starts a new repository locally.
2. git add . : This command adds a change in the working directory to the staging area.
3. git commit -m “message” :This command is used to save your changes to the local repository with -m used to provide a message which tells you what change you made in this commit.
4. git remote add origin : Add the repository given as the working remote repository.
5. git pull :This command is used to update the local version of a repository from a remote.
6. git push origin master: This command will push all the recent code to the repository.

Testing:

For testing, we used JUnit, an open-source unit testing framework for Java. Java developers rely on JUnit to write and execute automated tests. In Java, test cases must be re-executed whenever new code is added to ensure nothing is broken.

A "unit" refers to the smallest testable piece of code, such as a line, method, or class. Testing smaller units is preferred because they execute faster and provide clearer insights into code functionality and performance. JUnit streamlines this process, enabling efficient and reliable testing.

The screenshots of testing are attached:

```
@Test
void testSave() {
    Customer customer = new Customer();
    ArrayList<Camplaints> camplaintsList = new ArrayList<>();
    customer.setCamplaints(camplaintsList);
    customer.setEmail("jane.doe@example.org");
    customer.setFirstName("Jane");
    customer.setId(1);
    customer.setLastName("Doe");
    customer.setPassword("iloveyou");
    customer.setUsername("janedoe");
    when(customerRepository.save((Customer) any())).thenReturn(customer);

    Customer customer1 = new Customer();
    customer1.setCamplaints(new ArrayList<>());
    customer1.setEmail("jane.doe@example.org");
    customer1.setFirstName("Jane");
    customer1.setId(1);
    customer1.setLastName("Doe");
    customer1.setPassword("iloveyou");
    customer1.setUsername("janedoe");
    customerServiceImpl.save(customer1);
    verify(customerRepository).save((Customer) any());
    assertEquals(camplaintsList, customer1.getCamplaints());
    assertEquals("janedoe", customer1.getUsername());
    assertEquals("iloveyou", customer1.getPassword());
    assertEquals("Doe", customer1.getLastName());
    assertEquals(1, customer1.getId());
    assertEquals("Jane", customer1.getFirstName());
    assertEquals("jane.doe@example.org", customer1.getEmail());
```

```java
@Test
void testFindById() {
    Customer customer = new Customer();
    customer.setCamplaints(new ArrayList<>());
    customer.setEmail("jane.doe@example.org");
    customer.setFirstName("Jane");
    customer.setId(1);
    customer.setLastName("Doe");
    customer.setPassword("iloveyou");
    customer.setUsername("janedoe");
    when(customerRepository.getReferenceById((Integer) any())).thenReturn(customer);
    assertSame(customer, customerServiceImpl.findById(1));
    verify(customerRepository).getReferenceById((Integer) any());
}
```

```java
@Test
void testCostumerComplaint() {
    Customer customer = new Customer();
    ArrayList<Camplaints> camplaintsList = new ArrayList<>();
    customer.setCamplaints(camplaintsList);
    customer.setEmail("jane.doe@example.org");
    customer.setFirstName("Jane");
    customer.setId(1);
    customer.setLastName("Doe");
    customer.setPassword("iloveyou");
    customer.setUsername("janedoe");
    when(customerRepository.getReferenceById((Integer) any())).thenReturn(customer);

    Customer customer1 = new Customer();
    customer1.setCamplaints(new ArrayList<>());
    customer1.setEmail("jane.doe@example.org");
    customer1.setFirstName("Jane");
    customer1.setId(1);
    customer1.setLastName("Doe");
    customer1.setPassword("iloveyou");
    customer1.setUsername("janedoe");
    List<Camplaints> actualCostumerComplaintResult = customerServiceImpl.costumerComplaint(customer1);
    assertSame(camplaintsList, actualCostumerComplaintResult);
    assertTrue(actualCostumerComplaintResult.isEmpty());
    verify(customerRepository).getReferenceById((Integer) any());
}
```

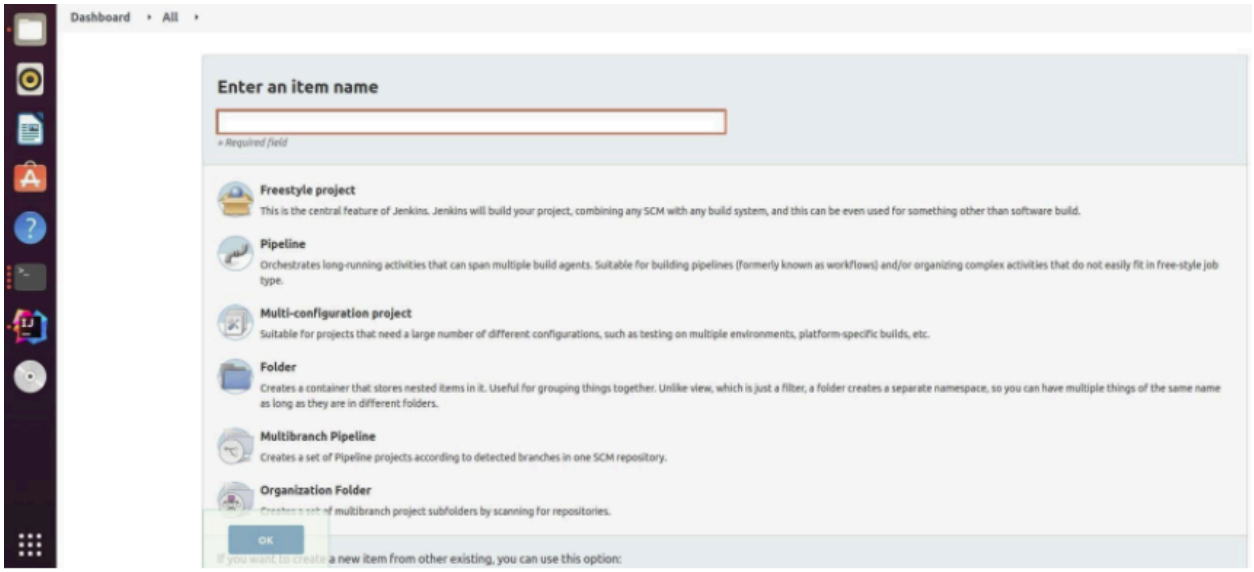
CI/CD Pipeline:

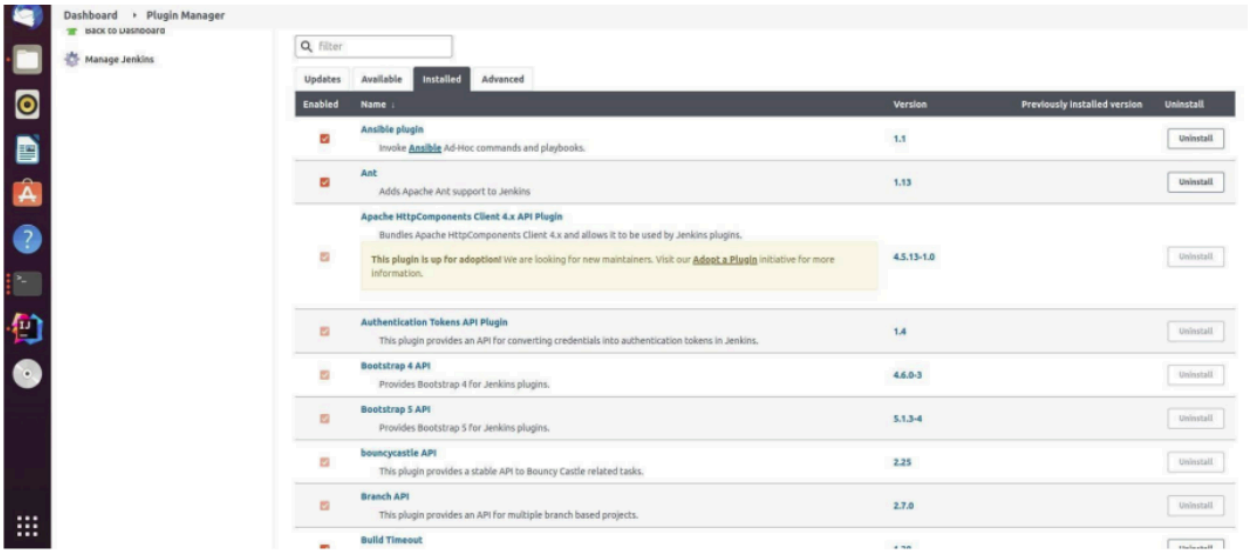
Jenkins Setup:

For continuous integration, we use Jenkins, which facilitates seamless integration and deployment through pipeline scripts. First, Jenkins needs to be installed on the localhost and configured to run on port 8080.

To access Jenkins, open a browser and navigate to localhost:8080, then sign in to use the tool. For creating a pipeline project, select "Pipeline" when creating a new item in Jenkins.

To run the pipeline, the Docker plugin is required. For deployment, the Ansible plugin must be installed, and to integrate Jenkins with Kubernetes, the Kubernetes plugin is also necessary.





Above pic shows the installation of plugins. Then we have to write a pipeline script which is present in the configure part of our created project.
It has several stages based on our requirement. Firstly, we write the script for cloning the project from our git repo by specifying our git repo link.

```
stages {
    stage('Git clone') {
        steps {
        git branch: 'master',url: 'https://github.com/madhav0407/TicketResolver_Backend.git'
        }
    }
```

```
stages {
    stage('Git clone') {
        steps {
        git branch: 'master',url: 'https://github.com/madhav0407/TicketResolver_Frontend.git'
        }
    }
```

```
stages {
    stage('Git clone') {
        steps {
        git branch: 'master',url: 'https://github.com/madhav0407/TicketResolver_Kube.git'
        }
    }
```

Build:

To compile the backend and run tests on it:

```
stage("Running Test cases"){
    steps{
        sh "mvn clean test"
    }
}

stage("Maven Build"){
    steps{
        sh "mvn clean install"
    }
}
```

Docker Containerisation:

Docker is a tool for creating, deploying, and running applications using containers. A container is a lightweight package that includes the application along with all its dependencies and libraries, ensuring it can run on any system. Unlike virtual machines, containers do not replicate an entire operating system, focusing only on the essentials needed to run the application.

Dockerfiles are used to build images, which package the application into containers. These images are then pushed to Docker Hub, where they can be pulled onto any host system for execution.

In our project, which involves a server, client, and database, we create two Dockerfiles. Before starting, Docker must be installed on the system. The Docker daemon binds to a Unix socket by default rather than a TCP port. This socket is owned by the root user, so other users need sudo to access it.

To grant access without sudo, the user must be added to the Docker group using:

sudo usermod –aG docker ${USER}

Similarly, Jenkins is added to the Docker group to enable container builds within Jenkins:

sudo usermod -aG docker jenkins

Docker File:

For the Backend:

```
# Fetching latest version of Java
FROM openjdk:11

# Setting up work directory
WORKDIR .

# Copy the jar file into our app
COPY ./target/ticketresolver-0.0.1-SNAPSHOT.jar .

# Exposing port 8081
EXPOSE 8081

# Starting the application
CMD ["java", "-jar", "ticketresolver-0.0.1-SNAPSHOT.jar"]
```

This Dockerfile creates a container using the openjdk:11 image to run a Java application. It sets the current directory as the working directory, copies the JAR file (ticketresolver-0.0.1-SNAPSHOT.jar) into the container, and exposes port 8081. The application is started using the java -jar command. The resulting container is ready to run the Java application independently.

For the frontend:

```
FROM node:alpine
WORKDIR /app
COPY package.json ./
COPY package-lock.json ./
COPY ./ ./
RUN npm i
EXPOSE 3000
CMD ["npm", "run", "start"]
```

The Dockerfile creates a lightweight container using the node:alpine image for running a Node.js application.
It sets /app as the working directory, copies application files, installs dependencies with npm install, and exposes port 3000.
Finally, it runs the application using npm run start. The container is self-sufficient and ready to execute the app independently.

The **Jenkins Pipeline script** has been configured to include the Dockerfile and Docker commands, automating the process of building and pushing the Docker Image to DockerHub. Below are the stages in the Jenkins to do so:

```
stage('Docker Build Image') {
    steps {
        script{
            dockerimage=docker.build "madhavsood04/ticketresolver-backend"
        }
    }
}
stage('Push Docker Image') {
    steps {
        script{
            docker.withRegistry('','DockerHubCred'){
            dockerimage.push()
            }
        }
    }
}
```

```
stage('Docker Build Image') {
    steps {
        script{
            dockerimage=docker.build "madhavsood04/ticketresolver_frontend"
        }
    }
}
stage('Push Docker Image') {
    steps {
        script{
            docker.withRegistry('','DockerHubCred'){
            dockerimage.push()
            }
        }
    }
}
```

After successful build the images are pushed to docker hub with the credentials specified,

Name	Last Pushed ↑	Contains	Visibility	Scout
madhavsood04/ticketresolver_frontend	about 1 hour ago	IMAGE	Public	Inactive
madhavsood04/ticketresolver-backend	about 4 hours ago	IMAGE	Public	Inactive

Docker Compose:

Docker Compose simplifies the process of managing and running multiple containers, such as frontend and backend services, on different ports. While containers can be created and run using separate Dockerfiles in multiple terminals, Docker Compose allows you to define and manage all containers in a single configuration file.

With Docker Compose, you can spin up and run multiple containers using a single command. Configuration is done through a YAML file that specifies container details. Below are some commonly used Docker Compose commands:

1. docker-compose up Start a specific Service.
2. docker-compose up <service-name> To see all the images.
3. docker-compose images To stop running containers.
4. docker-compose stop To remove stopped containers.
5. docker-compose rm To remove images and volumes.
6. docker-compose down For configuring the Docker-compose we use a YAML file

```yaml
version: '3'
services:
  mysql_db:
    image: mysql
    container_name: mysql_db
    restart: always
    networks:
      - spe-network
    environment:
      MYSQL_DATABASE: ticketresolver
      MYSQL_ROOT_PASSWORD: Abcd@1234
      MYSQL_PASSWORD: Abcd@1234
      MYSQL_USER: spe-project
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
```

```yaml
ticketresolver_backend:
  image: madhavsood04/ticketresolver-backend:latest
  container_name: ticketresolver_backend
  restart: always
  networks:
    - spe-network
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql_db:3306/ticketresolver?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true
    SPRING_DATASOURCE_USERNAME: spe-project
    SPRING_DATASOURCE_PASSWORD: Abcd@1234
  ports:
    - "8081:8081"
  volumes:
    - logs-volume:/logs  # Mount the logs directory for sharing with Logstash
  depends_on:
    - mysql_db

ticketresolver_frontend:
  image: madhavsood04/ticketresolver_frontend:latest
  container_name: ticketresolver_frontend
  restart: always
  networks:
    - spe-network
  ports:
    - "3000:3000"
  depends_on:
    - ticketresolver_backend
```

```yaml
elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch:8.10.2
  container_name: elasticsearch
  environment:
    discovery.type: single-node
    xpack.security.enabled: "false"  # Disable security for simplicity
  ports:
    - "9200:9200"
    - "9300:9300"
  networks:
    - spe-network

logstash:
  image: docker.elastic.co/logstash/logstash:8.10.2
  container_name: logstash
  networks:
    - spe-network
  ports:
    - "5044:5044"
  volumes:
    - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
    - logs-volume:/logs  # Mount logs directory from backend
  depends_on:
    - elasticsearch
```

```yaml
  kibana:
    image: docker.elastic.co/kibana/kibana:8.10.2
    container_name: kibana
    networks:
      - spe-network
    ports:
      - "5601:5601"
    environment:
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200
    depends_on:
      - elasticsearch

networks:
  spe-network:
    driver: bridge

volumes:
  mysql-data:
    driver: local
  logs-volume:
    driver: local
```

The Docker Compose configuration defines a multi-container application for the Ticket Resolver Project, including the backend, frontend, database, and logging/monitoring stack. It uses a custom network (spe-network) for communication between containers and persistent volumes for data storage.

Services:

1. **MySQL Database (mysql_db)**:
    - Runs a MySQL container to store application data.
    - Configured with database name, user credentials, and persistent storage (mysql-data).
    - Exposes port 3306.
2. **Backend (ticketresolver_backend)**:
    - Hosts the backend service using the provided Docker image.
    - Connects to the MySQL database using Spring DataSource.
    - Logs are stored in a shared volume (logs-volume) for integration with Logstash.
    - Exposes port 8081 and depends on mysql_db.
3. **Frontend (ticketresolver_frontend)**:
    - Runs the frontend service using the specified image.
    - Exposes port 3000.

    - Depends on the backend service (ticketresolver_backend).
4. **Elasticsearch (elasticsearch)**:
    - Handles centralized search and analytics.
    - Configured as a single-node cluster with security disabled for simplicity.
    - Exposes ports 9200 and 9300.
5. **Logstash (logstash)**:
    - Processes and sends logs from the backend to Elasticsearch.
    - Reads configuration from logstash.conf and shares logs via logs-volume.
    - Exposes port 5044 and depends on elasticsearch.
6. **Kibana (kibana)**:
    - Provides a web interface for visualizing Elasticsearch data.
    - Configured to connect to Elasticsearch.
    - Exposes port 5601 and depends on elasticsearch.

Networking: A custom bridge network (spe-network) ensures secure communication between containers.

Volumes:

- **mysql-data**: Stores MySQL database data persistently.
- **logs-volume**: Shares logs between the backend and Logstash.

Kubernetes:

Kubernetes (K8s) is an open-source platform for container orchestration that simplifies the deployment, scaling, and management of containerized applications. In this project, Kubernetes was utilized to manage the application's backend and frontend services, ensuring high availability, scalability, and optimal resource utilization.

Containerization of the Application: Before deploying to Kubernetes, the application was containerized using Docker:

1. **Backend**: A Spring Boot application was containerized as madhavsood04/ticketresolver_backend:latest.
2. **Frontend**: A React application was containerized as madhavsood04/ticketresolver_frontend:latest. These Docker images were built and pushed to Docker Hub for accessibility by Kubernetes.

Setup: Kubernetes was configured on the system to orchestrate the containerized application:

1. **kubectl**: The Kubernetes command-line tool was installed and configured to manage the cluster.
2. **Cluster**: A local Kubernetes cluster, set up using Minikube, was used for deploying the application.

Kubernetes Resources:

Following are the kubernetes resources that are made:

1. HorizontalPodAutoscaler (HPA)

- **backend-hpa.yaml**: Scales the **Spring Boot backend application** pods based on CPU utilization. It has a minimum of 1 replica and a maximum of 5 replicas.
- **frontend-hpa.yaml**: Scales the **React frontend application** pods based on CPU utilization. It has a minimum of 1 replica and a maximum of 5 replicas.

2. StatefulSet

- **mysql-db-deployment.yaml**: Deploys the **MySQL database** as a **StatefulSet** for persistent storage, with a **volumeClaimTemplate** that requests 1Gi of storage. It defines environment variables for the MySQL root password, database, and user credentials.

3. Deployment

- **ticketresolver-backend-deployment.yaml**: Deploys the **Spring Boot backend** application, exposing it on port 8081. The backend connects to the MySQL service and requires environment variables for database connection details.
- **ticketresolver-frontend-deployment.yaml**: Deploys the **React frontend** application, exposing it on port 3000. It defines an environment variable (REACT_APP_BACKEND_URL) to point to the backend service.

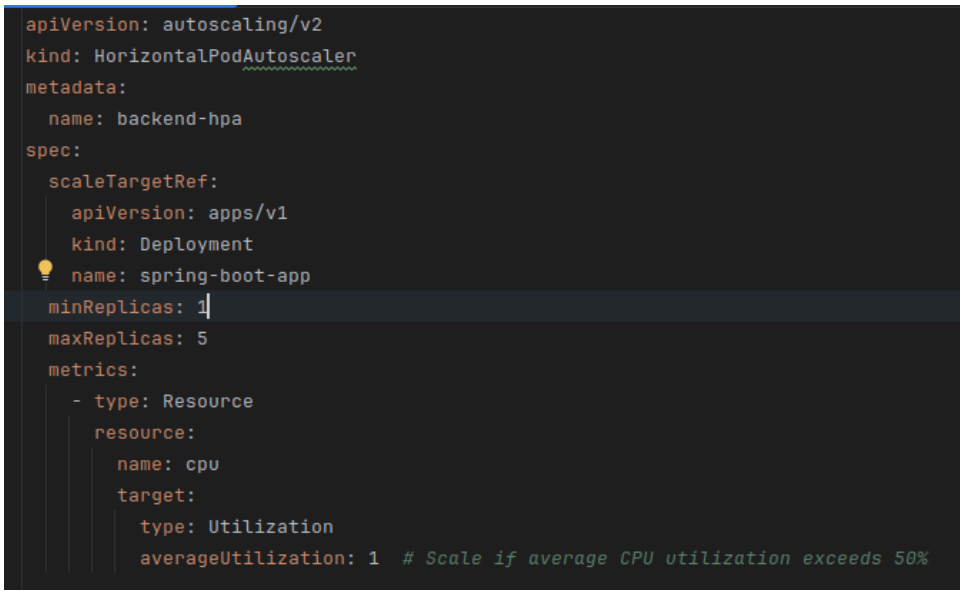
4. Service

- **mysql-db-deployment.yaml**: Defines a **ClusterIP Service** (db-service) to expose the MySQL database internally on port 3306.
- **ticketresolver-backend-deployment.yaml**: Defines a **NodePort Service** (spring-boot-service) to expose the backend application on port 8081, with external access on port 30008.
- **ticketresolver-frontend-deployment.yaml**: Defines a **NodePort Service** (ticketresolver-service) to expose the frontend application on port 3000, with external access on port 30007.

5. Persistent Volume Claim

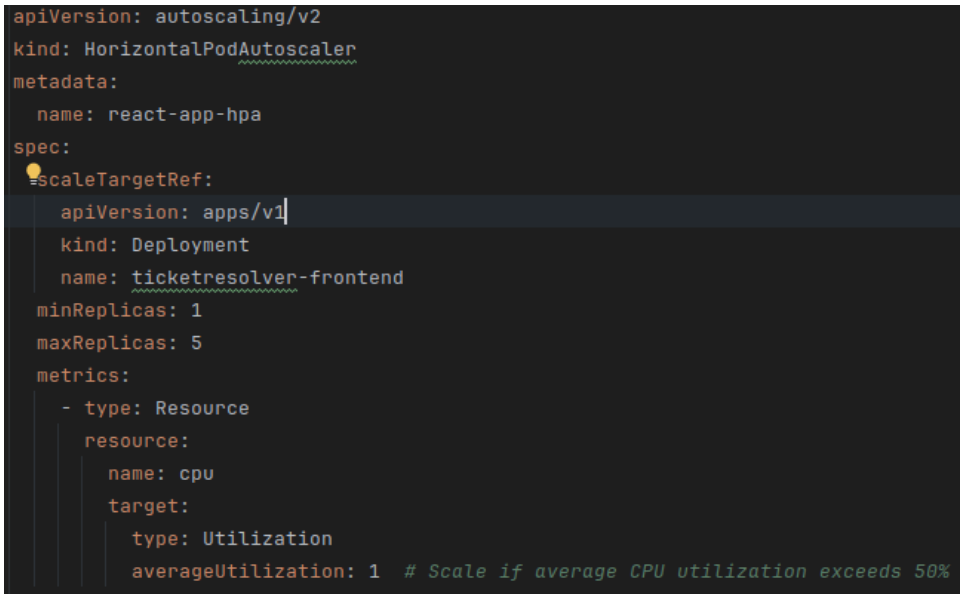
- **mysql-db-deployment.yaml**: Uses a **Persistent Volume Claim (PVC)** (mysql-persistent-storage) to persist MySQL data across pod restarts, with a storage request of 1Gi.

backend-hpa.yaml



```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: backend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: spring-boot-app
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 1  # Scale if average CPU utilization exceeds 50%
```

frontend-hpa.yaml



```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: react-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: ticketresolver-frontend
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 1  # Scale if average CPU utilization exceeds 50%
```

Mysql-db-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-db
spec:
  selector:
    matchLabels:
      app: mysql-db
  replicas: 1
  template:
    metadata:
      labels:
        app: mysql-db
    spec:
      containers:
        - name: mysql-db
          image: mysql
          imagePullPolicy: Always
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "Abcd@1234"
            - name: MYSQL_DATABASE
              value: "ticketresolver"
            - name: MYSQL_USER
              value: "spe-project"
            - name: MYSQL_PASSWORD
              value: "Abcd@1234"
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
  volumeClaimTemplates:
    - metadata:
        name: mysql-persistent-storage
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
```

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

Ticketresolver-backend-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-boot-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: spring-boot-app
  template:
    metadata:
      labels:
        app: spring-boot-app
    spec:
      containers:
        - name: spring-boot-app
          image: madhavsood04/ticketresolver-backend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8081
          env:
            - name: SPRING_DATASOURCE_URL
              value: "jdbc:mysql://db-service:3306/ticketresolver?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true"
            - name: SPRING_DATASOURCE_USERNAME
              value: "spe-project"
            - name: SPRING_DATASOURCE_PASSWORD
              value: "Abcd@1234"
---
apiVersion: v1
kind: Service
metadata:
  name: spring-boot-service
spec:
  selector:
    app: spring-boot-app
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30008
  type: NodePort
```

Ticketresolver-frontend-deployment.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ticketresolver-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ticketresolver-frontend
  template:
    metadata:
      labels:
        app: ticketresolver-frontend
    spec:
      containers:
        - name: ticketresolver-frontend
          image: madhavsood04/ticketresolver_frontend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 3000
          env:
          - name: REACT_APP_BACKEND_URL
            value: http://spring-boot-service:8081
---
apiVersion: v1
kind: Service
metadata:
  name: ticketresolver-service
spec:
  selector:
    app: ticketresolver-frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 30007
  type: NodePort
```

Ansible for Deployment:

To install Ansible and check the version, we use the following commands:

1. Add the Ansible Personal Package Archive (PPA) to your system: sudo add-apt-repository --yes --update ppa:ansible/ansible
2. Update Package List: sudo apt update
3. Install Ansible: sudo apt install ansible
4. To verify installation run: ansible --version

Ansible requires inventory and YAML files are used to specify the actions that should be done on the managed nodes.

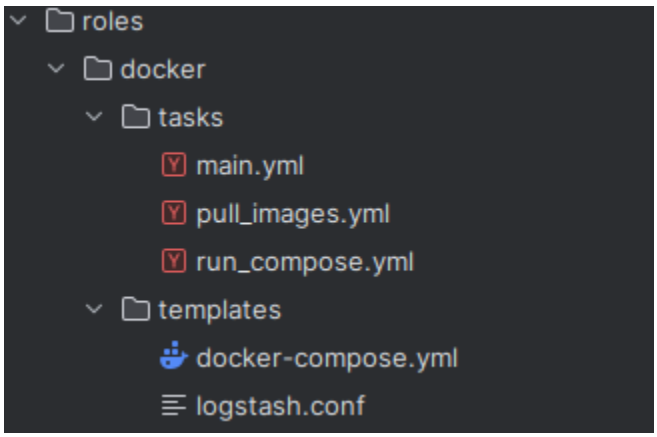
Inventory file: It keeps track of all the hosts for which we define our architecture. We can specify hosts under different groups and can write playbooks such that the containers can run in specified groups only.

```
[localhost]
127.0.0.1 ansible_connection=local ansible_user=madhav ansible_password=@vault.yml
```

Password refers to the **vault** where the user password is securely stored.

1. **Deployment using role ‘docker’**

   The following is the role structure:



   Pull_images.yml: Contains task to pull the images that will be required to run the docker containers

```yaml
---
- name: Pull backend image
  docker_image:
    name: madhavsood04/ticketresolver-backend:latest
    source: pull

- name: Pull frontend image
  docker_image:
    name: madhavsood04/ticketresolver_frontend:latest
    source: pull

- name: Pull mysql image
  docker_image:
    name: mysql
    source: pull

- name: Pull elasticsearch image
  docker_image:
    name: docker.elastic.co/elasticsearch/elasticsearch:8.10.2
    source: pull

- name: Pull logstash image
  docker_image:
    name: docker.elastic.co/logstash/logstash:8.10.2
    source: pull

- name: Pull kibana image
  docker_image:
    name: docker.elastic.co/kibana/kibana:8.10.2
    source: pull
```

run_compose.yml: Copies the docker compose template and runs the docker compose file

```yaml
---
- name: Copy compose file to remote host
  template:
    src: docker-compose.yml
    dest: ./docker-compose.yml

- name: run docker-compose file
  command: docker compose up -d --build
```

Main.yml: Runs the tasks in both the files

```yaml
---
- include_tasks: pull_images.yml
- include_tasks: run_compose.yml
```

Docker-compose.yml: docker-compose file explained earlier

Logtstash.conf: Logstash configuration file used to process and send log data to ElasticSearch

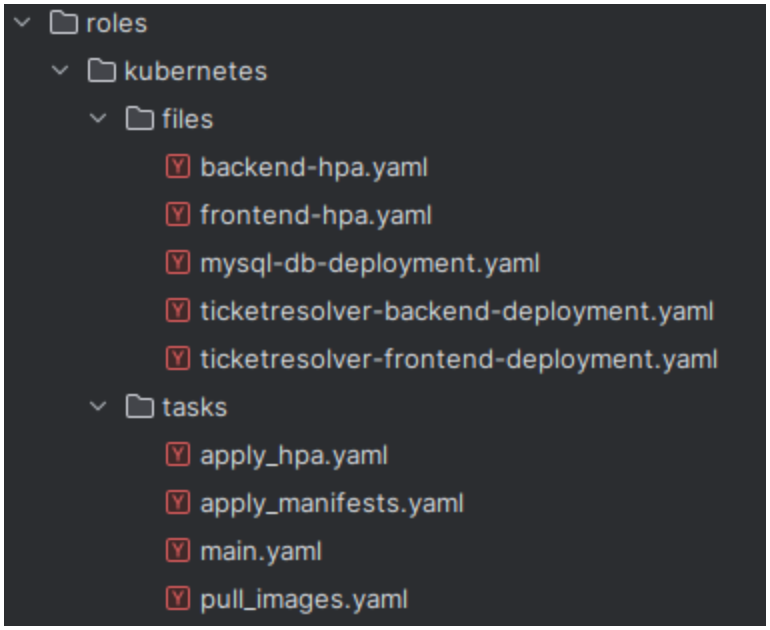
```
input {
  file {
    path => "/logs/app.log"  # Path to the log file inside the Logstash container
    start_position => "beginning"  # Read logs from the beginning
    sincedb_path => "/dev/null"   # Prevent Logstash from persisting state between restarts
  }
}

filter {
  grok {
    match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} \[%{DATA:thread}\] %{LOGLEVEL:log_level}  %{JAVACLASS:class} - %{GREEDYDATA:message}" }
  }
  date {
    match => ["timestamp", "yyyy-MM-dd HH:mm:ss"]
    target => "@timestamp"
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "ticketresolver-logs-%{+YYYY.MM.dd}"
  }
  stdout {
    codec => rubydebug
  }
}
```

Deploy.yml**:** Ansible playbook for deploying the application through docker

```
---
- name: Deploy the application
  hosts: localhost
  vars:
    ansible_python_interpreter: /usr/bin/python3.8
  roles:
    - docker
```

2. **Deployment using role ‘kubernetes’**

The following is the role structure:



Files folder consists of all the kubernetes resources files.

Apply_hpa.yaml: Applies the horizontal pod scaler kubernetes manifests

```
---
- name: Apply Kubernetes HPA
  command: kubectl apply -f {{ playbook_dir }}/roles/kubernetes/files/{{ item }}
  with_items:
    - backend-hpa.yaml
    - frontend-hpa.yaml
```

Apply_manifests.yaml: Applies the rest of the kubernetes manifests

```
---
- name: Apply Kubernetes Manifests
  command: kubectl apply -f {{ playbook_dir }}/roles/kubernetes/files/{{ item }}
  with_items:
    - mysql-db-deployment.yaml
    - ticketresolver-backend-deployment.yaml
    - ticketresolver-frontend-deployment.yaml
```

Main.yml: Runs the tasks in both the files

```
---
#- include_tasks: pull_images.yaml
- include_tasks: apply_manifests.yaml
- include_tasks: apply_hpa.yaml
```

Deploy.yml**:** Ansible playbook for deploying the application through kubernetes

```
---
- name: Deploy the application
  hosts: localhost
  vars:
    ansible_python_interpreter: /usr/bin/python3.8
  roles:
    - kubernetes
```

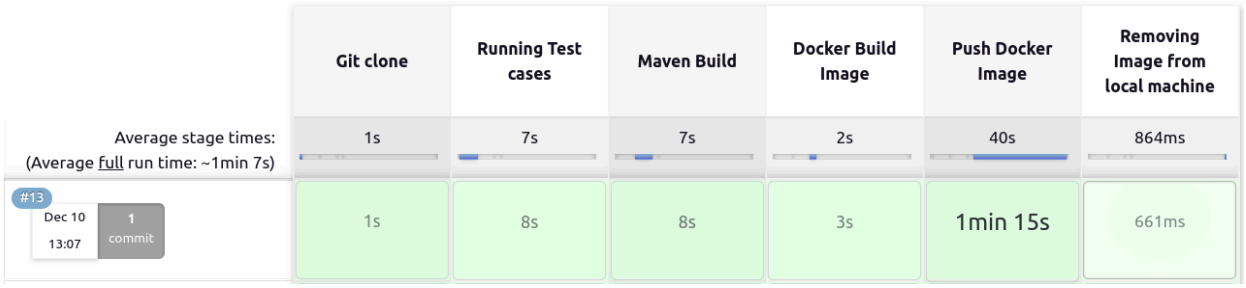
Pipeline Script for Ansible deployment:

```
stage('Ansible pull docker image') {
    steps {
        sh '''
        echo "$VAULT_PASS" > /tmp/vault_pass.txt
        chmod 600 /tmp/vault_pass.txt
        ansible-playbook -i inventory --vault-password-file /tmp/vault_pass.txt deploy.yml
        rm -f /tmp/vault_pass.txt
        '''
    }
}
```

Here $VAULT_PASS will the ansible-vault-pass configured in Jenkins.

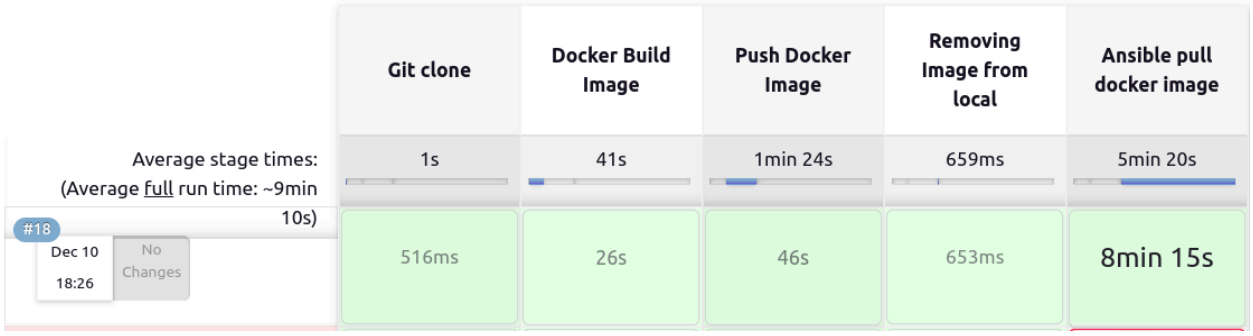
Jenkins Pipeline:

Backend:



	Git clone	Running Test cases	Maven Build	Docker Build Image	Push Docker Image	Removing Image from local machine
Average stage times: (Average full run time: ~1min 7s)	1s	7s	7s	2s	40s	864ms
#13 Dec 10 13:07 — 1 commit	1s	8s	8s	3s	1min 15s	661ms

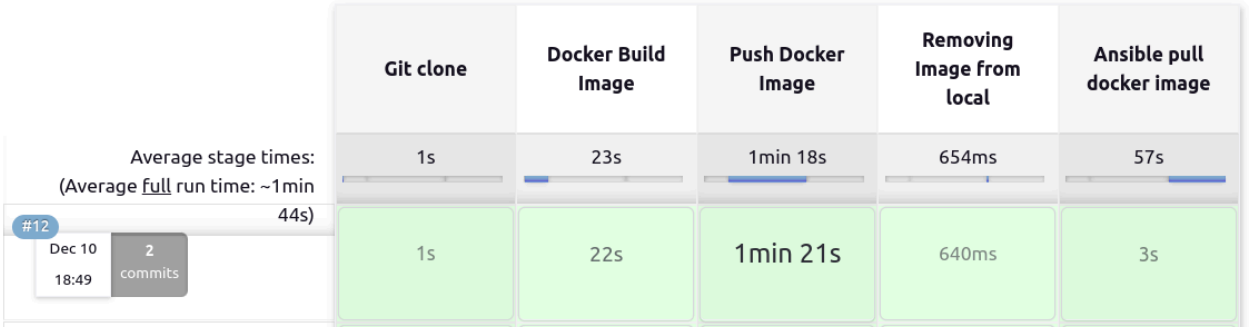
Frontend (Docker Compose):



	Git clone	Docker Build Image	Push Docker Image	Removing Image from local	Ansible pull docker image
Average stage times: (Average full run time: ~9min 10s)	1s	41s	1min 24s	659ms	5min 20s
#18 Dec 10 18:26 — No Changes	516ms	26s	46s	653ms	8min 15s

Frontend (Kubernetes):

Stage View



	Git clone	Docker Build Image	Push Docker Image	Removing Image from local	Ansible pull docker image
Average stage times: (Average full run time: ~1min 44s)	1s	23s	1min 18s	654ms	57s
#12 Dec 10 18:49 — 2 commits	1s	22s	1min 21s	640ms	3s

Docker Compose Output:

```
madhav@madhav-HP-Pavilion-Laptop-14-dv0xxx:~$ docker ps
CONTAINER ID   IMAGE                                                COMMAND                  CREATED         STATUS         PORTS
          NAMES
dba5548eefe5   madhavsood04/ticketresolver_frontend:latest           "docker-entrypoint.s…"   8 minutes ago   Up 8 minutes   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
          ticketresolver_frontend
bb4cd6ac7979   docker.elastic.co/logstash/logstash:8.10.2            "/usr/local/bin/dock…"   8 minutes ago   Up 8 minutes   0.0.0.0:5044->5044/tcp, :::5044->5044/tcp, 9600/tcp
          logstash
2d03362f0e4a   madhavsood04/ticketresolver-backend:latest           "java -jar ticketres…"   8 minutes ago   Up 8 minutes   0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
          ticketresolver_backend
0f393adf58f1   docker.elastic.co/kibana/kibana:8.10.2               "/bin/tini -- /usr/l…"   8 minutes ago   Up 8 minutes   0.0.0.0:5601->5601/tcp, :::5601->5601/tcp
          kibana
a38bd3bb86ad   mysql                                                "docker-entrypoint.s…"   8 minutes ago   Up 8 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
          mysql_db
80fac2bb239d   docker.elastic.co/elasticsearch/elasticsearch:8.10.2   "/bin/tini -- /usr/l…"   8 minutes ago   Up 8 minutes   0.0.0.0:9200->9200/tcp, :::9200->9200/tcp, 0.0.0.0:9300->9300/tcp, :::9300->9
300/tcp   elasticsearch
```

Kubernetes Output:

Workload Status

Deployments

Name	Images	Labels	Pods	Created
spring-boot-app	madhavsood04/ticketresolver-backend:latest	-	1 / 1	4 minutes ago
ticketresolver-frontend	madhavsood04/ticketresolver_frontend:latest	-	1 / 1	4 minutes ago

Pods

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
mysql-db-0	mysql	app: mysql-db apps.kubernetes.io/pod-index: 0 controller-revision-hash: mysql-db-658f66b94d Show all	minikube	Running	0	-	-	4 minutes ago
spring-boot-app-78bfb9dc65-55w72	madhavsood04/ticketresolver-backend:latest	app: spring-boot-app pod-template-hash: 78bfb9dc65	minikube	Running	1	-	-	4 minutes ago
ticketresolver-frontend-d6f7dcf99-jc26c	madhavsood04/ticketresolver_frontend:latest	app: ticketresolver-frontend pod-template-hash: d6f7dcf99	minikube	Running	0	-	-	4 minutes ago

Replica Sets

	Name	Images	Labels	Pods	Created ↑
●	spring-boot-app-78bfb9dc65	madhavsood04/ticketresolver-backend:latest	app: spring-boot-app pod-template-hash: 78bfb9dc65	1 / 1	4 minutes ago
●	ticketresolver-frontend-d6f7dcf99	madhavsood04/ticketresolver_frontend:latest	app: ticketresolver-frontend pod-template-hash: d6f7dcf99	1 / 1	4 minutes ago

Stateful Sets

	Name	Images	Labels	Pods	Created ↑
●	mysql-db	mysql	-	1 / 1	4 minutes ago

ELK:

ELK stands for Elasticsearch, Logstash and Kibana. It aggregates logs from systems and applications, analyzes these logs and creates a visualization for application and infrastructure monitoring, faster troubleshooting and security analytics.

sl4j is used to generate an app.log file in the backend.

App.log:

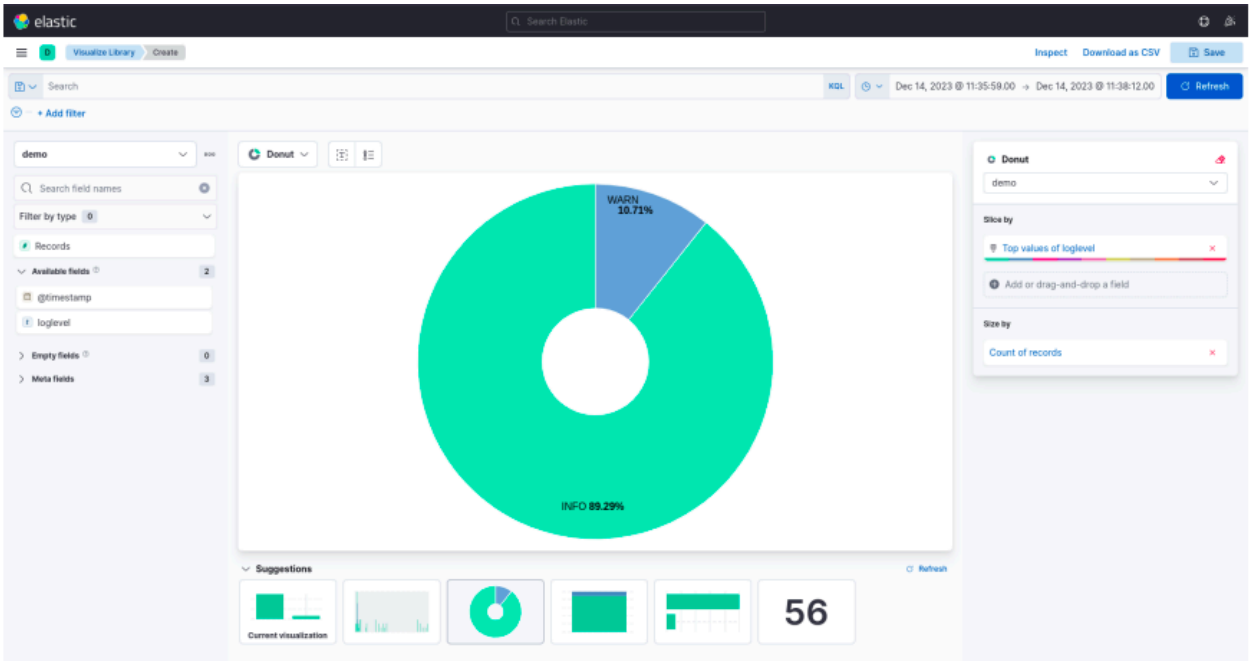
```
2024-12-08 16:52:21 [restartedMain] INFO  com.had.selfhelp.SelfhelpApplication - Starting SelfhelpApplication using Java 1.8.0_422 on madhav-HP-Pavilion-Laptop-14-dv0xxx with PID 21060 (/home/madhav/Desktop/SPE/
2024-12-08 16:52:21 [restartedMain] INFO  com.had.selfhelp.SelfhelpApplication - No active profile set, falling back to 1 default profile: "default"
2024-12-08 16:52:21 [restartedMain] INFO  o.s.b.d.e.DevToolsPropertyDefaultsPostProcessor - Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2024-12-08 16:52:21 [restartedMain] INFO  o.s.b.d.e.DevToolsPropertyDefaultsPostProcessor - For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2024-12-08 16:52:21 [restartedMain] INFO  o.s.d.r.c.RepositoryConfigurationDelegate - Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-12-08 16:52:21 [restartedMain] INFO  o.s.d.r.c.RepositoryConfigurationDelegate - Finished Spring Data repository scanning in 49 ms. Found 4 JPA repository interfaces.
2024-12-08 16:52:22 [restartedMain] INFO  o.s.b.w.e.tomcat.TomcatWebServer - Tomcat initialized with port(s): 8081 (http)
2024-12-08 16:52:22 [restartedMain] INFO  o.a.catalina.core.StandardService - Starting service [Tomcat]
2024-12-08 16:52:22 [restartedMain] INFO  o.a.catalina.core.StandardEngine - Starting Servlet engine: [Apache Tomcat/9.0.71]
2024-12-08 16:52:22 [restartedMain] INFO  o.a.c.c.C.[Tomcat].[localhost].[/] - Initializing Spring embedded WebApplicationContext
2024-12-08 16:52:22 [restartedMain] INFO  o.s.b.w.s.c.ServletWebServerApplicationContext - Root WebApplicationContext: initialization completed in 1394 ms
2024-12-08 16:52:22 [restartedMain] INFO  c.had.selfhelp.util.SimpleCORSFilter - SimpleCORSFilter init
2024-12-08 16:52:22 [restartedMain] INFO  o.h.jpa.internal.util.LogHelper - HHH000204: Processing PersistenceUnitInfo [name: default]
2024-12-08 16:52:22 [restartedMain] INFO  org.hibernate.Version - HHH000412: Hibernate ORM core version 5.6.14.Final
2024-12-08 16:52:22 [restartedMain] INFO  o.h.annotations.common.Version - HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2024-12-08 16:52:22 [restartedMain] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
2024-12-08 16:52:23 [restartedMain] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
2024-12-08 16:52:23 [restartedMain] INFO  org.hibernate.dialect.Dialect - HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
2024-12-08 16:52:23 [restartedMain] WARN  o.h.t.d.j.JavaTypeDescriptorRegistry - HHH000481: Encountered Java type [class java.lang.Object] for which we could not locate a JavaTypeDescriptor and which does not ap
2024-12-08 16:52:23 [restartedMain] WARN  o.h.t.d.j.JavaTypeDescriptorRegistry - HHH000481: Encountered Java type [class java.lang.Object] for which we could not locate a JavaTypeDescriptor and which does not ap
2024-12-08 16:52:23 [restartedMain] WARN  o.h.t.d.j.JavaTypeDescriptorRegistry - HHH000481: Encountered Java type [class java.lang.Object] for which we could not locate a JavaTypeDescriptor and which does not ap
2024-12-08 16:52:23 [restartedMain] WARN  o.h.t.d.j.JavaTypeDescriptorRegistry - HHH000481: Encountered Java type [class java.lang.Object] for which we could not locate a JavaTypeDescriptor and which does not ap
2024-12-08 16:52:23 [restartedMain] WARN  o.h.t.d.j.JavaTypeDescriptorRegistry - HHH000481: Encountered Java type [class java.lang.Object] for which we could not locate a JavaTypeDescriptor and which does not ap
2024-12-08 16:52:23 [restartedMain] INFO  o.h.e.t.j.p.i.JtaPlatformInitiator - HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2024-12-08 16:52:23 [restartedMain] INFO  o.s.o.j.LocalContainerEntityManagerFactoryBean - Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-12-08 16:52:24 [restartedMain] WARN  o.s.b.a.o.j.JpaBaseConfiguration$JpaWebConfiguration - spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2024-12-08 16:52:24 [restartedMain] INFO  o.s.s.web.DefaultSecurityFilterChain - Will secure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@7c39c7a3, org.springframework.securi
2024-12-08 16:52:24 [restartedMain] INFO  o.s.b.d.a.OptionalLiveReloadServer - LiveReload server is running on port 35729
```

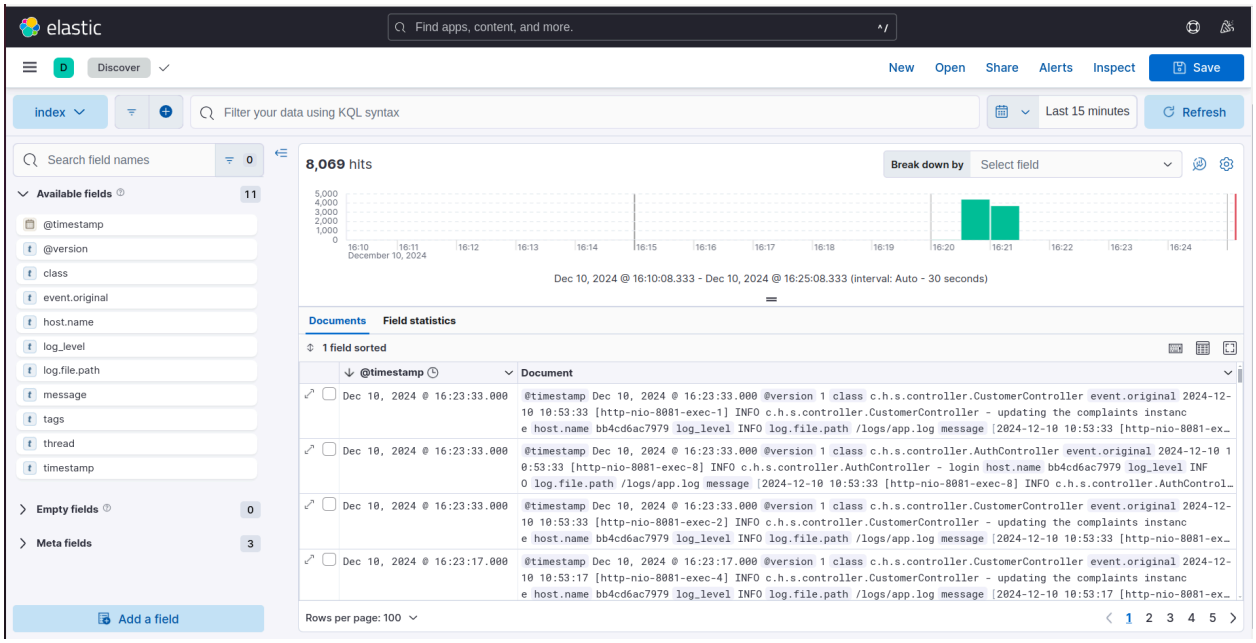
```
2024-12-08 16:52:24 [restartedMain] INFO  o.s.b.w.e.tomcat.TomcatWebServer - Tomcat started on port(s): 8081 (http) with context path ''
2024-12-08 16:52:24 [restartedMain] INFO  com.had.selfhelp.SelfhelpApplication - Started SelfhelpApplication in 4.224 seconds (JVM running for 4.665)
2024-12-08 16:53:29 [http-nio-8081-exec-1] INFO  o.a.c.c.C.[Tomcat].[localhost].[/] - Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-12-08 16:53:29 [http-nio-8081-exec-1] INFO  o.s.web.servlet.DispatcherServlet - Initializing Servlet 'dispatcherServlet'
2024-12-08 16:53:29 [http-nio-8081-exec-1] INFO  o.s.web.servlet.DispatcherServlet - Completed initialization in 2 ms
2024-12-08 16:53:29 [http-nio-8081-exec-2] INFO  c.h.s.controller.AuthController - login
2024-12-08 17:01:43 [http-nio-8081-exec-5] ERROR com.had.selfhelp.jwt.AuthTokenFilter - Cannot set user authentication: {}
```

```
2024-12-08 17:01:43 [http-nio-8081-exec-5] INFO  c.h.s.controller.AuthController - registering the customer
2024-12-08 17:01:58 [http-nio-8081-exec-7] INFO  c.h.s.controller.AuthController - login
2024-12-08 17:01:58 [http-nio-8081-exec-10] INFO  c.h.s.controller.CustomerController - updating the complaints instance
2024-12-08 17:01:58 [http-nio-8081-exec-1] INFO  c.h.s.controller.CustomerController - updating the complaints instance
2024-12-08 17:15:36 [SpringApplicationShutdownHook] INFO  o.s.o.j.LocalContainerEntityManagerFactoryBean - Closing JPA EntityManagerFactory for persistence unit 'default'
2024-12-08 17:15:36 [SpringApplicationShutdownHook] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
2024-12-08 17:15:36 [SpringApplicationShutdownHook] INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.
```

Logstash.conf file is the configuration file used for ingesting, parsing and sending the data to elasticsearch. Screenshot attached earlier.

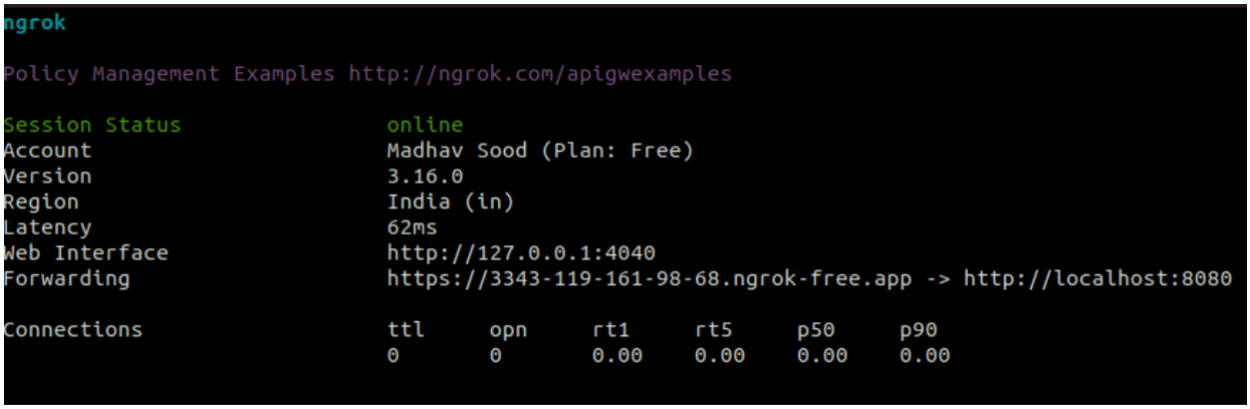
Kibana dashboard:



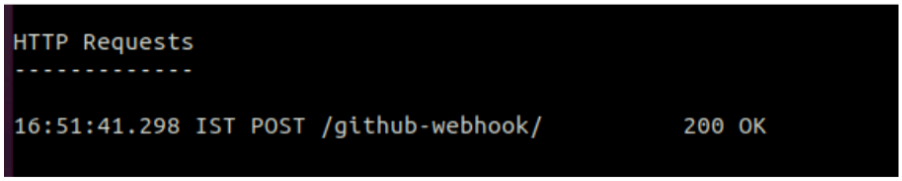


Executing the pipeline using ngrok and Git SCM Polling:

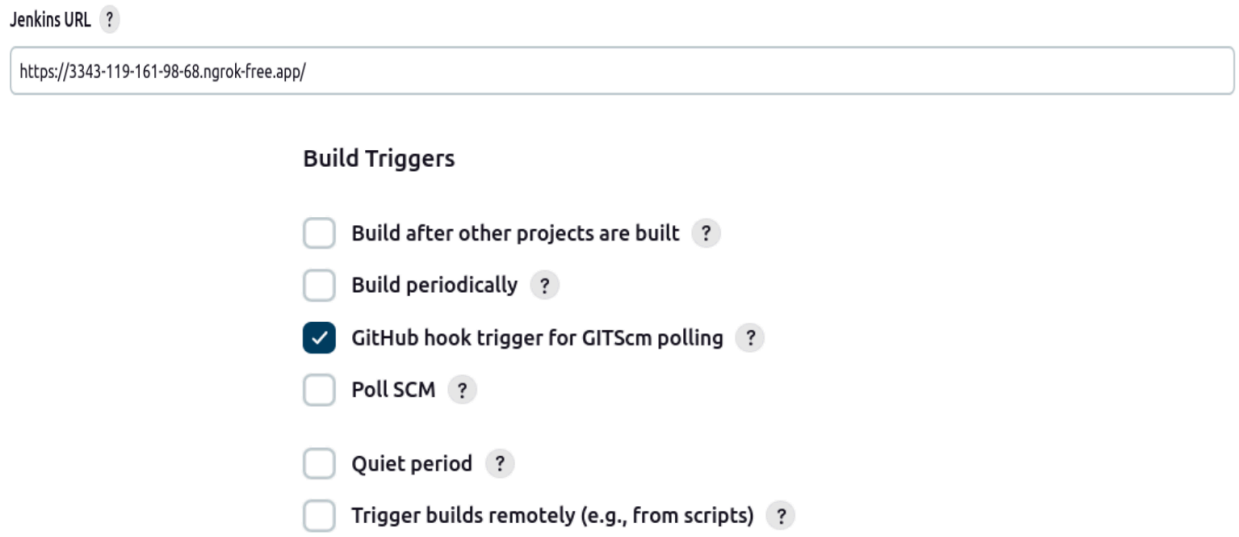
Open a terminal and run the command ngrok http 8080. This will create an HTTP tunnel using Ngrok, making the local server running on port 8080 accessible over the internet.

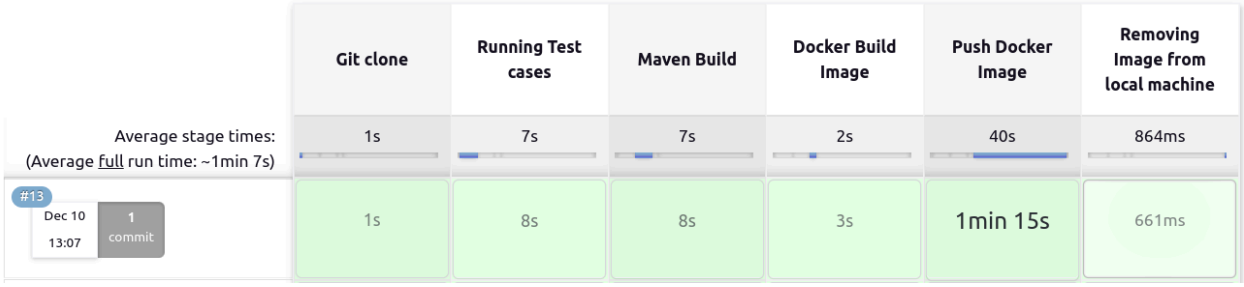


Copy the forwarding URL generated by Ngrok. Then, create a GitHub webhook and use this URL as the payload URL in the webhook configuration. GitHub will test the connection, and upon successful setup, a '200 OK' response will confirm that everything is configured correctly.

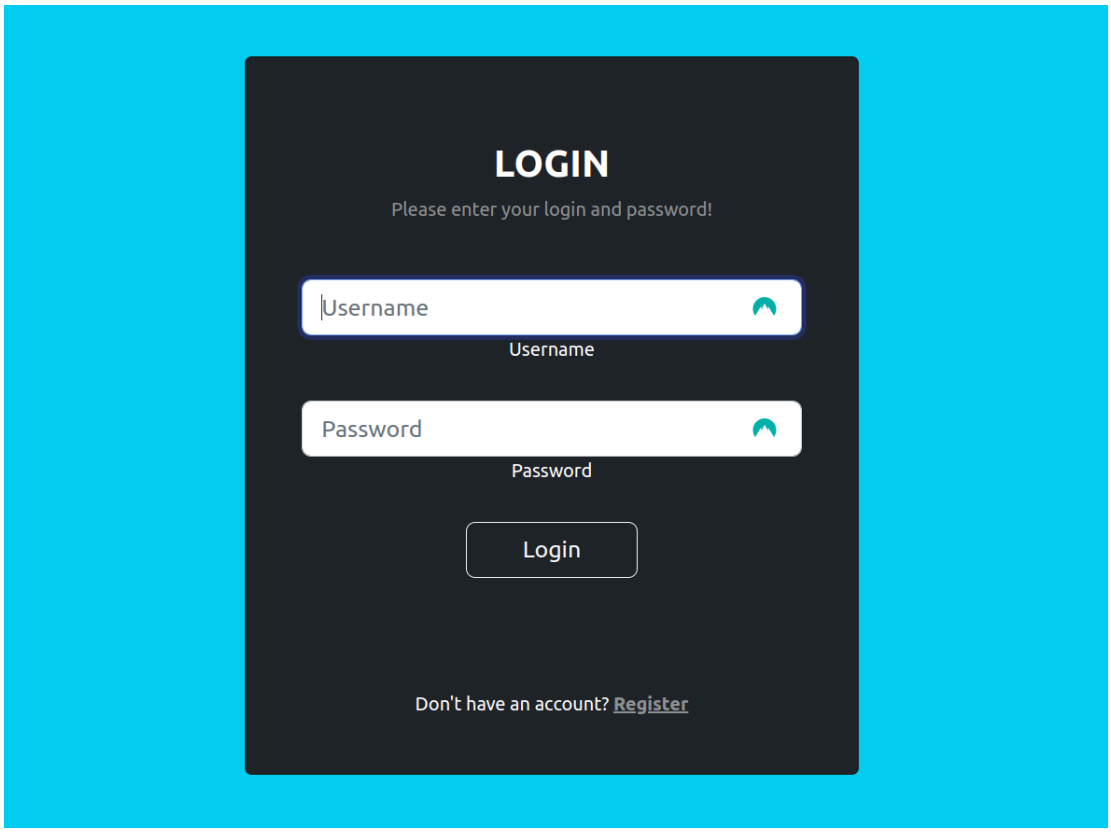


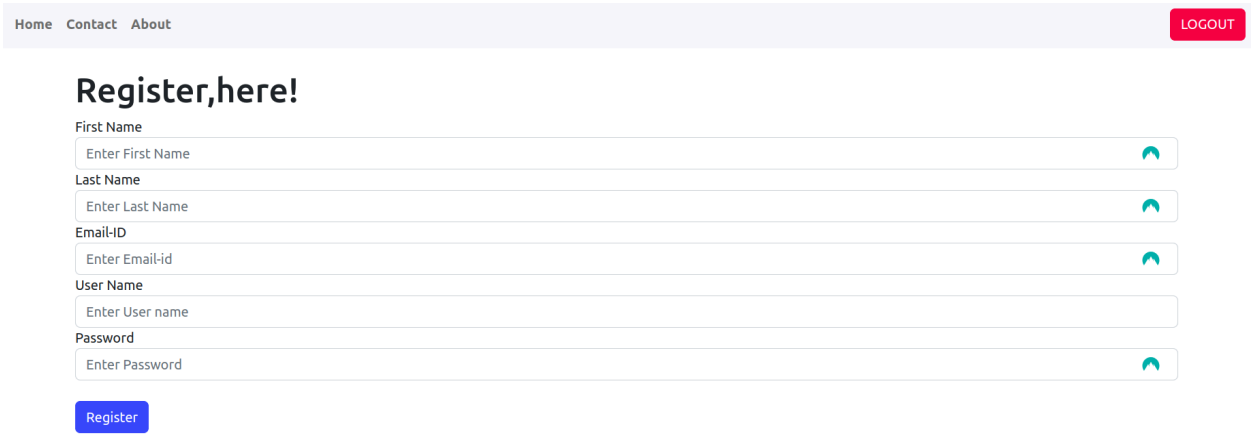
Next, we'll update the Jenkins URL with the forwarding URL obtained from Ngrok and configure a build trigger for Git SCM polling. This setup ensures that the pipeline automatically triggers the build process whenever Jenkins detects a new commit in the linked GitHub repository.





Working of Our Application:





Home Contact About LOGOUT

Customer Requests

Customer_ID	Vehicle_Id	Vehicle_Type	Services	Address	Remarks	Response	Acknowledgement
1	1234	Two-Wheeler	Cleaning,Painting	Mumbai	ASAP	Mark as Done	Type acknowledgement Send
2	12345	Two-Wheeler	Cleaning,A/C repair,Deep Cleaning	mumbai	ASAP	Mark as Done	Type acknowledgement Send

Home Contact About LOGOUT

Madhav's Service Requests

Vehicle_ID	Vehicle Type	Services	Remarks	Acknowledgement	Status
1234	Two-Wheeler	Cleaning,Painting	ASAP	Done!!	Done

Request New Service