

SPE Final Project Report: Elective Management Application

Siddharth Kothari (IMT2021019), Sankalp Kothari (IMT2021028)

December 10, 2024

Contents

1	Introduction	2
1.1	Application Introduction	2
1.2	Tools Utilized	2
2	Codebase	3
2.1	Directory Structure	3
2.2	Database	3
2.3	Backend	4
2.4	Frontend	5
3	DevOps Tools and Setup	6
3.1	Git and Github for Version Control	6
3.2	Maven - Build Automation Tool	7
3.3	Node Package Manager (NPM)	8
3.4	Ngrok for Secure Tunneling	8
3.5	Jenkins for CI/CD	9
3.6	Docker	10
3.7	Ansible for Application Deployment	11
3.8	Kubernetes	12
3.9	Grafana, Prometheus and Loki	15
4	Additional Configuration Steps	16
4.1	Credentials in Jenkins	16
4.2	Jenkinsfile Environment Variables	17
4.3	Creating a Pipeline Project Jenkins	17
4.4	Running the Application	17
5	Results	18
5.1	Outputs of the Run Stages from Jenkins	18
5.2	Outputs on Dockerhub	18
5.3	Images Created	18
5.4	Outputs on Kubernetes	20
5.5	Outputs inside the Database	20
5.6	Outputs of the Frontend	21
5.7	Outputs from Logging	21
6	Relevant Links	21
6.1	Github Link	21
6.2	Dockerhub Link	21

1 Introduction

1.1 Application Introduction

The Elective Management Application is designed to make managing elective courses simple and efficient for everyone involved—students, instructors, and administrators. With secure role-based access, each user has tools tailored to their needs.

1. Administrators can easily manage user accounts, assign roles, and oversee all activities, ensuring the system runs smoothly. Instructors can add and manage elective courses, handle student enrollment requests, and decide which requests to approve or decline.
2. For students, the application offers options to browse available electives, view detailed information about each course, and request enrollment. They can also check the status of their requests, keeping everything transparent and stress-free.

This report gives an overview of the application along with the DevOps tools used in the development of this project. The primary tools I have utilised in this project are Jenkins for CI/CD pipelines, Github for Version Control, Docker for Virtualization, Ansible for Configuration Management, Maven for Project Management in the Backend, Npm for Project Management in the frontend, Kubernetes for Container Orchestration and Deployment, and Grafana, Prometheus and Loki for Monitoring and Log Management. An intro to each of them is provided in the next section.

1.2 Tools Utilized

The tools utilized in the project are listed below -

- **Jenkins:** An open-source automation server widely used for implementing CI/CD pipelines. It helps automate tasks like building, testing, and deploying code, ensuring smoother development and delivery cycles.
- **GitHub:** A cloud-based platform for version control and collaboration. It allows the team to store and manage code, track changes, and contribute seamlessly to the project's source repository.
- **Ngrok:** Tool that creates secure tunnels to local servers, allowing us to expose our local development environment to the internet. It simplifies testing webhooks, sharing local projects, and remote access to services.
- **Docker:** A containerization platform that packages applications and their dependencies into portable containers. This ensures that the application runs consistently across different environments, from development to production.
- **Ansible:** An automation tool for configuration management and application deployment. It simplifies managing server setups and automating repetitive tasks, making the infrastructure more reliable and efficient.
- **Maven:** A build and project management tool used in Java applications. It handles dependencies, compiles code, and manages the backend project lifecycle, streamlining development.
- **Npm (Node Package Manager):** A package manager for JavaScript that facilitates dependency management and script execution in the frontend. It is crucial for handling libraries and building the frontend efficiently.
- **Kubernetes:** A container orchestration platform that automates deploying, scaling, and managing containerized applications. It ensures high availability and simplifies the deployment process.
- **Grafana:** A monitoring and analytics tool that creates visual dashboards. It provides insights into application performance and system metrics by integrating with various data sources.
- **Prometheus:** A monitoring system that collects and stores metrics in a time-series database. It is particularly useful for tracking application performance and resource usage.
- **Loki:** A log aggregation tool that works seamlessly with Prometheus and Grafana. It helps collect and analyze logs for troubleshooting and performance monitoring.

2 Codebase

The Backend is written in Java, and uses Maven for project management. The frontend is written in React, and uses Bootstrap for Styling. Project Management is done using Npm. The database used is mysql.

2.1 Directory Structure

Figure 1 shows the project structure. Each folder is described in later sections.

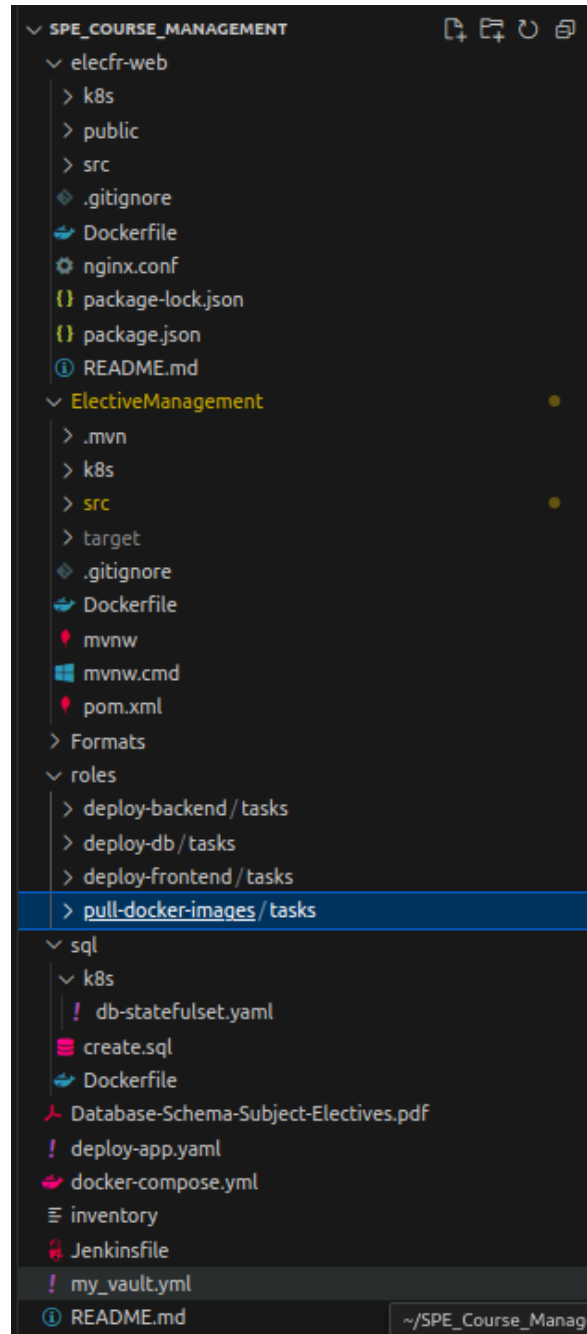


Figure 1: Project Directory Structure

2.2 Database

The database is written using MySQL, and has the following features -

1. The schema for the application is structured to handle users, their roles, and course-related information. It has tables to manage users, their roles (Student, Instructor, or Admin), and how these roles connect.
2. Students and instructors have dedicated tables to store their details, which are linked back to the main user table. The subjects table keeps track of all courses, including the instructor responsible for each.
3. Students can request enrollment in courses through the request table, where details like the course, student, and request period are stored. Once a request is approved, the assignment is recorded in the student_subject table, which tracks which students are enrolled in which courses.
4. The dockerfile for the sql database uses a create.sql script to initialise all the tables along with their relationships, and also initialises the database with some dummy values.

The schema for the database is provided in Figure 2.

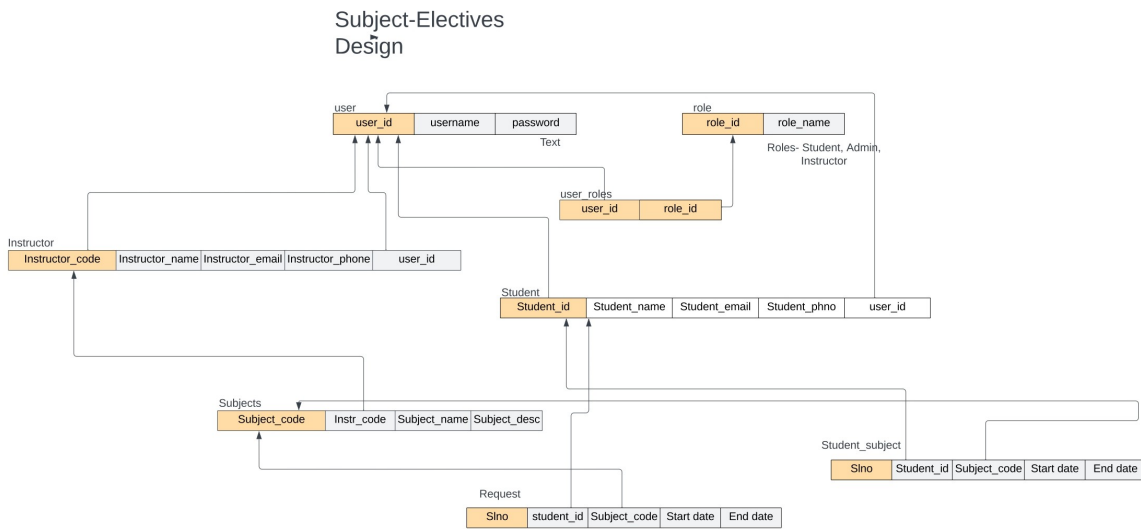


Figure 2: Database Schema

2.3 Backend

Directory Structure

The `ElectiveManagement` folder houses all backend-related files essential for running and managing the application. It includes a `Dockerfile` for containerizing the backend and automating its deployment process. Additionally, the `Formats` directory provides sample JSON object formats for various API requests to facilitate interaction with the backend (used for testing via Postman). The directory structure is shown in Figure 3.

The `Dockerfile` creates a multi-stage build for a Spring Boot application using Maven to compile the project and OpenJDK to run the application. It first builds the JAR file in a Maven container and then copies it into a minimal OpenJDK container to run the application.

The `k8s` folder contains 2 files - one for the deployment and creation of the backend service for Kubernetes, and one for using Horizontal Pod Autoscaler (HPA) for the backend deployment.

Overview of Functionalities

The application backend, written in Spring Boot, is designed to manage user roles, subject assignments, and administrative tasks effectively. It supports three primary roles—students, instructors, and admins—with varying access levels. Using RESTful endpoints, the backend facilitates essential operations, such as user registration, authentication, subject and instructor management, request handling, and student-subject associations. The architecture comprises:

- **REST Controllers:** Expose endpoints for interaction with the web interface, invoking appropriate service-layer functions.

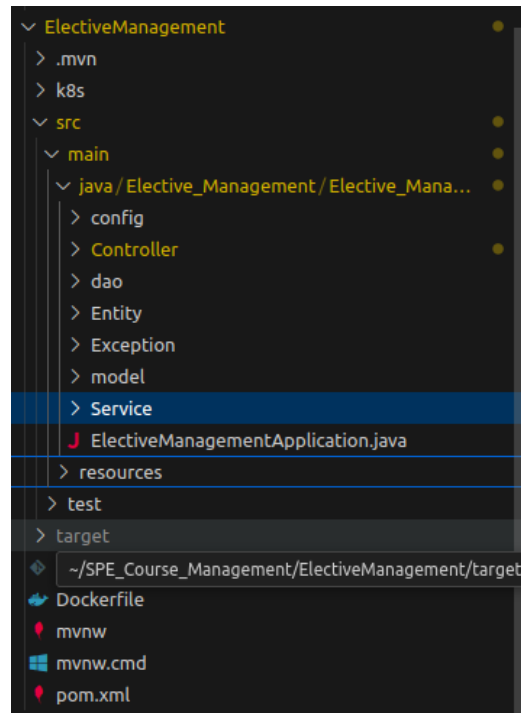


Figure 3: Spring Boot App Directory Structure

- **Services:** Act as intermediaries between controllers and database repositories, ensuring data flow and logic execution.
- **Repositories/DAOs:** Handle database communication, managing all CRUD operations for various entities.

The endpoints allow admins to oversee user registrations, manage subjects and instructors, and handle student requests. Instructors and students interact with the system to view, update, or manage their respective records, leveraging a token-based authentication system powered by JWT for secure access. The application supports granular access control, ensuring endpoint functionality aligns with user roles and permissions.

2.4 Frontend

Directory Structure

The `elecfr_web` folder contains the frontend-related files for the application, written in React.js. It houses all components, services, and validation functions required for managing the user interface. A `Dockerfile` is also included to containerize the frontend for consistent deployment. The main subdirectories are:

- **validators:** Contains validation functions for various forms, ensuring that user inputs meet the required criteria before submission. This includes validation for login, registration (admin, instructor, student), and subject requests.
- **services:** Handles authentication, API requests, and local storage management. Key files include `auth.header.js`, `auth-services.js`, `localStorage-services.js`, and `user-services.js`.
- **common:** Contains reusable list item components that display summaries of entities like students, subjects, and requests. These list items are clickable and lead to detailed views.
- **components:** Contains the main screens of the application, such as forms for registration, subject management, request handling, and detailed entity views. These screens are conditionally rendered based on the user's role (student, instructor, admin).

The directory structure for the frontend is shown in Figure 4.

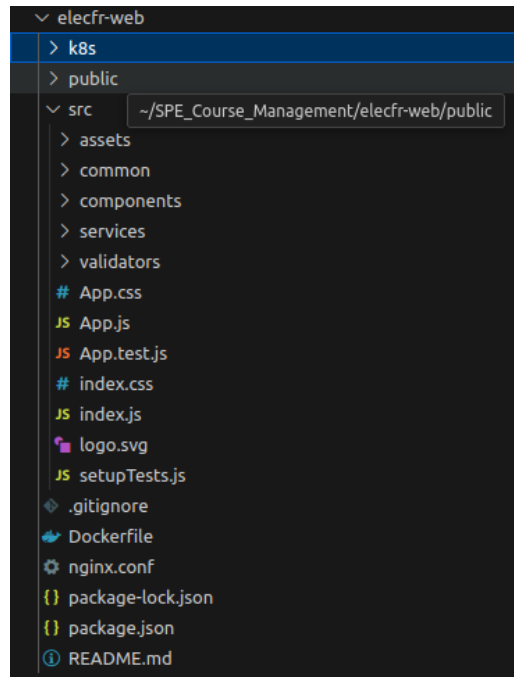


Figure 4: React App Directory Structure

Overview of Functionalities

The frontend is designed to provide a seamless user experience by interacting with the backend and presenting the data appropriately to the user. It handles user authentication (login/signup), form validation, and manages state using local storage. The UI adjusts based on the user's role, offering specific access to various functionalities like viewing subjects, making requests, and managing profiles. The key functionalities include:

- **User Authentication:** Allows users to log in and access the application with a JWT token, which is stored in local storage and sent with each API request.
- **Role-based Navigation:** The navbar adapts based on the user role (student, instructor, admin, or guest), providing relevant links for each role.
- **Forms and Validation:** Handles user input for registration, login, subject management, and requests with validation functions to ensure correct and complete data submission.
- **List Items and Entity Management:** Displays summaries of entities such as students, subjects, and requests, with links to view detailed information.
- **Interaction with Backend:** Sends HTTP requests to the backend for data management, such as saving/updating subjects, registering users, and creating requests.

3 DevOps Tools and Setup

This section provides an overview of the various tools used for this project.

3.1 Git and Github for Version Control

Description

Git is a distributed version control system that allows multiple developers to work on a project simultaneously. It tracks changes in the source code, enabling developers to collaborate on a project, manage different versions of files, and revert to earlier versions if needed. **Github** is a web-based platform built around Git that provides hosting for Git repositories. It offers additional tools for collaboration, code review, and project management, making it a key resource for both open-source and private projects. Figure 5 shows a snapshot of the repo.

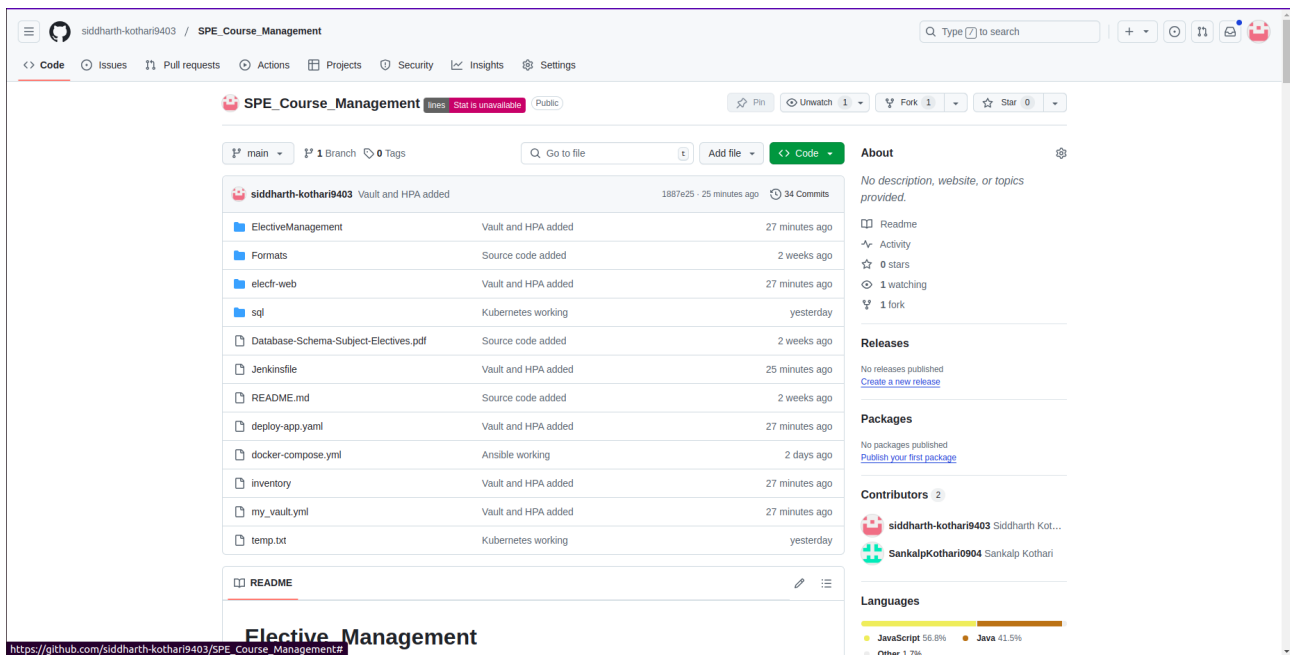


Figure 5: Screenshot of Repository

Setup

Ubuntu

Git is most likely already installed on Ubuntu, and it can be checked using the following command:

Listing 1: Command for Git Version

```
1 $ git --version
```

If not found, it can be installed using the following commands:

Listing 2: Command for Git Installation

```
1 $ sudo apt update
2 $ sudo apt install git
```

Mac

The command to check the git version is the same. Git can be installed in Mac using the following command -

Listing 3: Command for Git Installation

```
1 $ brew install git
```

Windows

The installer for Windows can be found at the following link - <https://git-scm.com/download/win>.

3.2 Maven - Build Automation Tool

Description

Maven is a build automation and project management tool primarily used for Java projects, though it can be used with other programming languages as well. It simplifies the process of building, managing dependencies, and handling project lifecycle tasks, such as compiling code, running tests, packaging applications, and deploying them. Maven uses a `pom.xml` file that contains project information, a list of external dependencies, build plugins and the phases of the build lifecycle. When Maven runs, it reads the `pom.xml` file and executes tasks according to the build lifecycle.

Maven and IntelliJ IDEA Installation

The zip archives for downloading Maven can be found at the following website - <https://maven.apache.org/download.cgi>, while the guide for installation can be found at the following link - <https://maven.apache.org/install.html>.

The complete installation guide for IntelliJ IDEA can be found here - <https://www.jetbrains.com/help/idea/installation-guide.html#toolbox>.

3.3 Node Package Manager (NPM)

Description

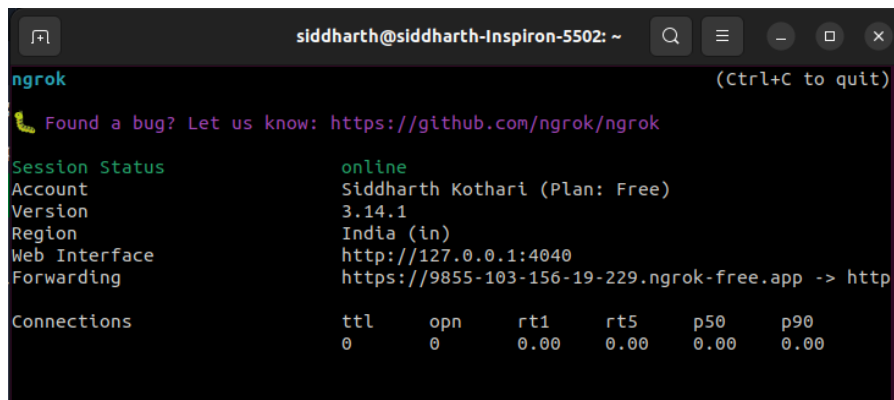
npm (Node Package Manager) is the default package manager for Node.js. It is used to manage and distribute JavaScript libraries and tools. With npm, developers can:

- **Install Packages:** Download and install reusable JavaScript libraries or tools for their projects, including dependencies.
- **Share Code:** Publish and share their own packages with the developer community.
- **Manage Dependencies:** Handle project dependencies efficiently, ensuring all required libraries are installed and up-to-date.
- **Run Scripts:** Automate project tasks like building, testing, and deployment through custom scripts.

3.4 Ngrok for Secure Tunneling

Description

Ngrok is a tool that provides secure tunneling from the public internet to a local development environment. It allows developers to expose their local web servers, APIs, or applications running on their local machine to the internet via a temporary, secure URL. Ngrok is particularly useful when you want to share your development work with others or when testing webhooks and third-party integrations that require a publicly accessible URL.



```
siddharth@siddharth-Inspiron-5502: ~  
ngrok (Ctrl+C to quit)  
📧 Found a bug? Let us know: https://github.com/ngrok/ngrok  
Session Status      online  
Account             Siddharth Kothari (Plan: Free)  
Version             3.14.1  
Region              India (in)  
Web Interface        http://127.0.0.1:4040  
Forwarding           https://9855-103-156-19-229.ngrok-free.app -> http  
  
Connections          ttl    opn    rt1    rt5    p50    p90  
                     0      0      0.00   0.00   0.00   0.00
```

Figure 6: Ngrok Running

Ngrok Installation

The guides for installation for Ngrok for Windows, Mac and Ubuntu can be found here - <https://ngrok.com/download>.

Setup of Webhooks

Setting Up of Webhooks involves the following steps -

1. Open your Github Profile and go to Settings → Developer Settings → Personal Access Tokens → Tokens (Classic). You can create a new Personal Access Token from here. We should ensure to provide it the required permissions. This will be used later.

2. Open the terminal, and run the command below. This will provide us with a temporary url TEMPORARY_URL which we will use in this step.

Listing 4: Command for Ngrok Tunneling

```
1 $ ngrok http 8080
```

3. Open the repository which we want to attach webhooks to. Go to Settings → Webhooks → Add Webhook. Here we can specify the URL to be TEMPORARY_URL/github-webhooks/ This allows us to notify the port 8080 (where jenkins will be running) about any changes in the Repository. The page for the same is shown in Figure 7.

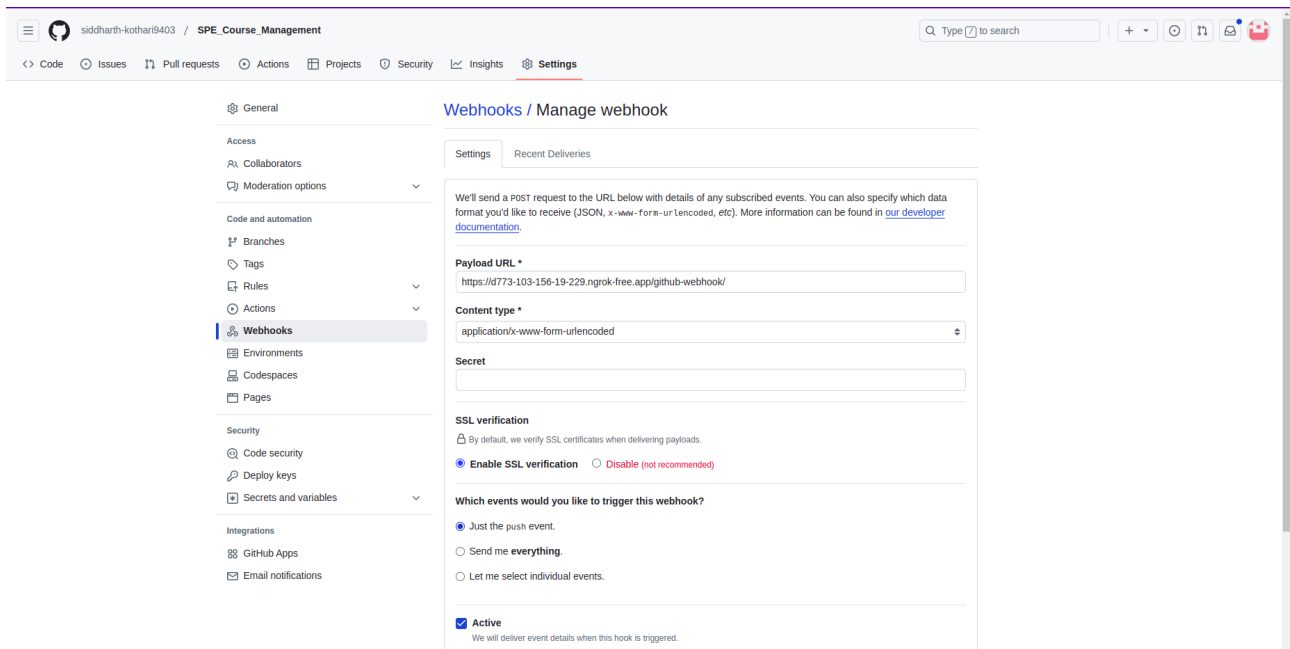


Figure 7: Setting up a Webhook

3.5 Jenkins for CI/CD

Description

Jenkins is an open-source automation server that facilitates Continuous Integration (CI) and Continuous Delivery (CD) for software development. It automates tasks like building, testing, and deploying applications. Jenkins is highly customizable through plugins and can be integrated with a variety of tools and technologies.

Jenkinsfile Description

The Jenkinsfile I have used for this project is shown in Figure 8. The stages in this pipeline are described below

1. **Build and Tag Images:** This stage builds Docker images for the `sql`, `ElectiveManagement`, and `elecfr-web` directories and tags them with the `latest` tag. Each directory contains a dedicated Dockerfile for creating the respective images.
2. **Push to Docker Hub:** The Docker images are pushed to a Docker Hub repository. Credentials for Docker Hub are securely managed using Jenkins' credential store. The pipeline logs into Docker Hub and pushes the images for `mysql`, `elective-management`, and `elecfr-web`.
3. **Clean Local Docker Images:** To save local disk space, this stage removes the Docker images created during the build stage. The `docker rmi` command is used, and failures are ignored using the `|| true` construct.

```

1 pipeline {
2
3     environment {
4         DOCKERHUB_CRED = credentials("Dockerhub-Credentials-ID") // Jenkins credentials ID for Docker Hub
5         DOCKER_HUB_REPO = 'siddharthkothari9403' // Docker Hub username or repo name
6         MINIKUBE_HOME = '/home/jenkins/minikube'
7         VAULT_PASS = credentials("ansible_vault_pass")
8     }
9
10    agent any
11    stages {
12
13        stage('Build and Tag Images') {
14            steps {
15                dir('sql') {
16                    sh "docker build -t ${DOCKER_HUB_REPO}/mysql:latest ."
17                }
18                dir('ElectiveManagement') {
19                    sh "docker build -t ${DOCKER_HUB_REPO}/elective-management:latest ."
20                }
21                dir('elecfr-web') {
22                    sh "docker build -t ${DOCKER_HUB_REPO}/elecfr-web:latest ."
23                }
24            }
25        }
26
27        stage('Push to Docker Hub') {
28            steps {
29                sh "echo $DOCKERHUB_CRED_PSW | docker login -u $DOCKERHUB_CRED_USR --password-stdin"
30                sh "docker push ${DOCKER_HUB_REPO}/elecfr-web:latest"
31                sh "docker push ${DOCKER_HUB_REPO}/elective-management:latest"
32                sh "docker push ${DOCKER_HUB_REPO}/mysql:latest"
33            }
34        }
35
36        stage('Clean Local Docker Images') {
37            steps {
38                sh "docker rmi ${DOCKER_HUB_REPO}/elecfr-web:latest || true"
39                sh "docker rmi ${DOCKER_HUB_REPO}/elective-management:latest || true"
40                sh "docker rmi ${DOCKER_HUB_REPO}/mysql:latest || true"
41            }
42        }
43
44        stage('Deploy Ansible Vault with Kubernetes'){
45            steps {
46                sh '''
47                echo "$VAULT_PASS" > /tmp/vault_pass.txt
48                chmod 600 /tmp/vault_pass.txt
49                ansible-playbook -i inventory --vault-password-file /tmp/vault_pass.txt deploy-app.yaml
50                rm -f /tmp/vault_pass.txt
51                '''
52            }
53        }
54    }
55 }

```

Figure 8: Jenkinsfile

4. **Deploy Ansible Vault with Kubernetes:** This stage uses Ansible Vault for secure management of sensitive data, like passwords, during the deployment process. The deployment is orchestrated using a Kubernetes configuration defined in `deploy-app.yaml`. A temporary file stores the vault password securely for the playbook execution and is deleted afterward.

Jenkins Installation

The guide to install Jenkins can be found at the following link - <https://www.jenkins.io/doc/book/installing/>. The command to start Jenkins is given below.

Listing 5: Jenkins Service Start

```
1 $ sudo systemctl start jenkins.service
```

Further steps for configuring Jenkins can be found by opening the browser, and entering the url `localhost : 8080`. It will guide you through the entire setup.

3.6 Docker

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. A container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system dependencies. Docker containers are isolated from each other and the host system, providing a consistent environment for the application to run.

Docker Installation

The guide to install Docker can be found at the following link - <https://docs.docker.com/engine/install/>. Following the installation, we can install docker desktop by following the guide at the link - <https://docs.docker.com/desktop/>.

Following this, we can create a docker account, and use the credentials to login to docker in the local system using the following command -

Listing 6: Docker Login Command

```
1 $ docker login -u USERNAME
```

Finally, we can add the jenkins user to the docker user group using the following command. This is necessary to allow Jenkins to push images to Dockerhub.

Listing 7: Add Jenkins to Docker

```
1 $ sudo usermod -a -G docker jenkins
```

Following this, we should restart the Jenkins service for the changes to take effect, using the following command -

Listing 8: Restart Jenkins

```
1 $ sudo systemctl restart jenkins
```

3.7 Ansible for Application Deployment

Ansible is an open-source automation tool used for configuration management, application deployment, and task automation. It allows users to define tasks in a simple YAML-based language called **Ansible Playbooks**, which describe the desired state of a system. Ansible is agentless, meaning it does not require any software to be installed on the nodes it manages, relying on SSH (for Linux/Unix systems) or WinRM (for Windows systems) for communication.

Ansible Installation

Ansible can be installed using the following commands -

Listing 9: Docker Login Command

```
1 $ sudo apt update
2 $ sudo apt install ansible
```

Playbook using Ansible Roles

The .yaml file for this project is provided below. It makes use of roles as shown in Figure 9.

The ansible playbook does the following tasks -

1. **Pull Docker Images:** The playbook pulls the required Docker images (elecfr-web, elective-management, and mysql) from the Docker registry using the specified Docker username.
2. **Deploy MySQL StatefulSet:** It deploys the MySQL StatefulSet to Kubernetes by applying the db-statefulset.yaml file located within the deploy-db role.
3. **Deploy Backend Application:** The backend application is deployed to Kubernetes using the backend-deployment.yaml file from the deploy-backend role.
4. **Deploy Backend Horizontal Pod Autoscaler:** It applies the backend-hpa.yaml file from the deploy-backend role to deploy the Horizontal Pod Autoscaler for the backend service.
5. **Deploy Frontend Application:** The frontend application is deployed to Kubernetes using the frontend-deployment.yaml file from the deploy-frontend role.
6. **Deploy Frontend Horizontal Pod Autoscaler:** It applies the frontend-hpa.yaml file from the deploy-frontend role to deploy the Horizontal Pod Autoscaler for the frontend service.

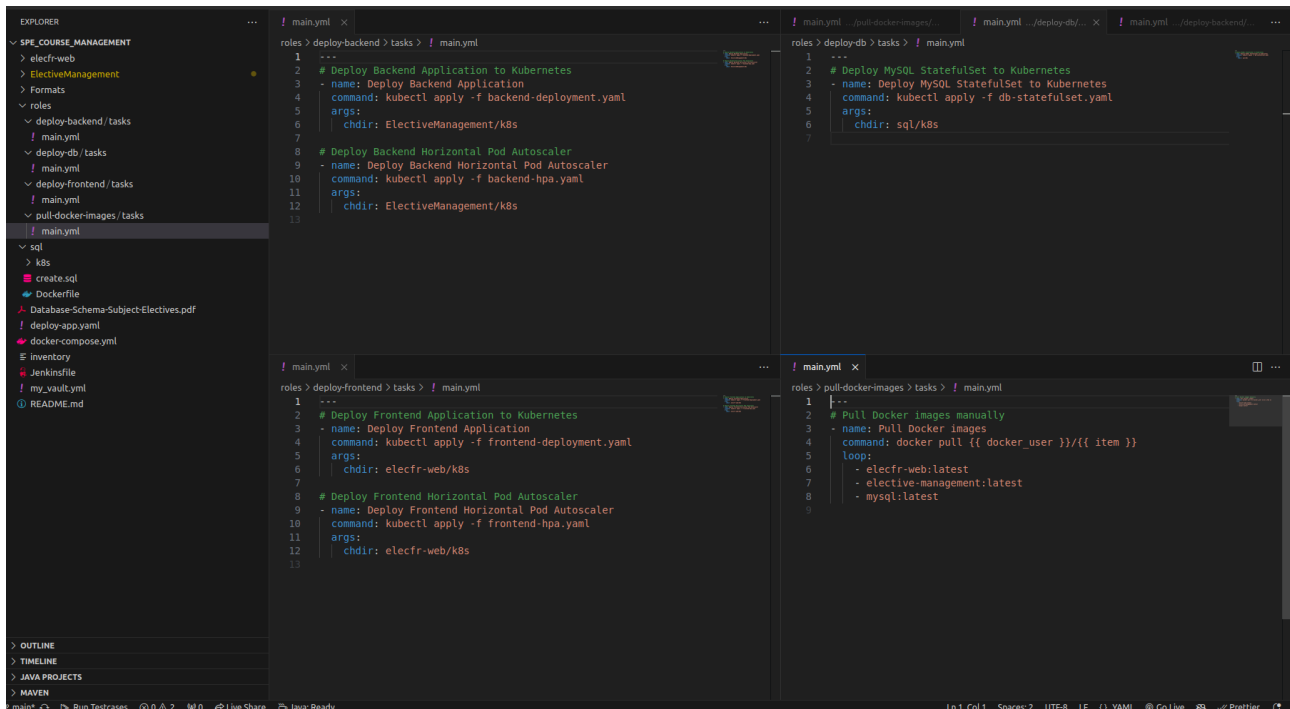


Figure 9: Ansible Roles

Inventory file using Ansible Vaults

The inventory code for the project is given below. It makes use of Ansible vaults.

Listing 10: inventory file

```
1 [localhost]
2 127.0.0.1 ansible_connection=local ansible_user=siddharth ansible_password=@my_vault.yml
```

3.8 Kubernetes

Description

Kubernetes is a powerful open-source platform designed to manage containerized applications across distributed environments. It provides tools to automate the deployment, scaling, and maintenance of these applications, ensuring reliability and efficiency. Kubernetes enables seamless scaling to handle increased demand, self-healing to recover from failures, and simplified application updates through rolling deployments.

Minikube is a tool that allows users to create a local Kubernetes cluster on their machine. It is particularly useful for developers to learn Kubernetes concepts, test configurations, and develop containerized applications without requiring access to a full multi-node cluster.

Kubectl is the command-line interface for Kubernetes, used to interact with clusters. With `kubectl`, users can deploy applications, view cluster resources, update configurations, monitor logs, and troubleshoot issues. It serves as the primary tool for managing all aspects of a Kubernetes environment.

Installation and Setup

For our application, we use Minikube as the cluster, along with `kubectl` as the command line tool. The commands to install them are listed below. We will install minikube inside the Jenkins user for easy configuration. To change to the Jenkins user, we can use the command -

Listing 11: Command to Change User to Jenkins

```
1 $ sudo su - jenkins
```

We should first install the virtualization hypervisor (in this case, it is Docker). The commands are - To change to the Jenkins user, we can use the command -

Listing 12: Virtualization Hypervisor installation commands

```
1 $ sudo apt update
2 $ sudo apt install docker.io
```

The commands to install Kubectl are -

Listing 13: Kubectl Installation Commands

```
1 $ sudo snap install kubectl --classic
```

The commands to install Minikube are -

Listing 14: Kubectl Installation Commands

```
1 $ curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-
  -amd64
2 $ chmod +x minikube
3 $ sudo mv minikube /usr/local/bin/
```

The command to get Minikube container running is -

Listing 15: Kubectl Installation Commands

```
1 $ minikube start --driver=docker
```

We can then see the ip address that Minikube is running on using the command -

Listing 16: Kubectl Installation Commands

```
1 $ minikube ip
```

Configuration yaml files

Figure 10 provides the code used to create a deployment for the frontend and the backend, along with the statefulset for the database. It also contains the code to create the corresponding services.

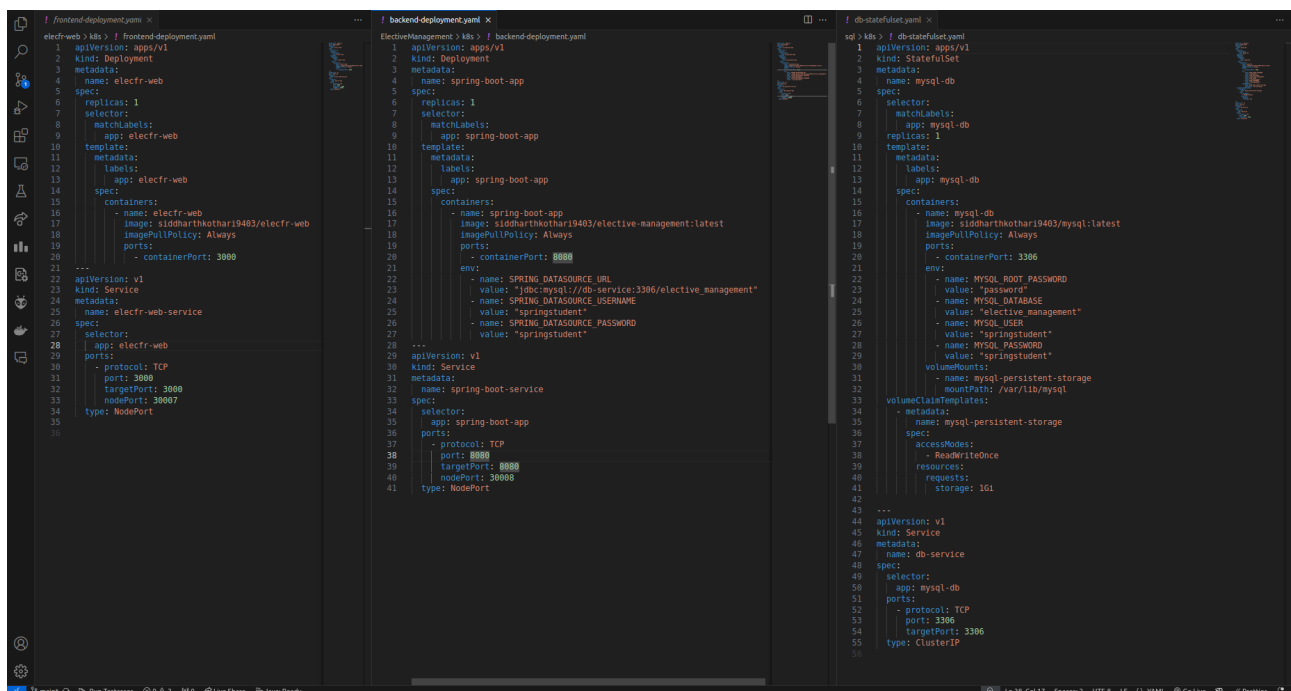


Figure 10: Yaml files for deployments, services and stateful sets

The following yaml files have been used to create the pods and services for the frontend, the backend and the database -

1. The **frontend-deployment.yaml** file defines the deployment and service configuration for the `elecfr-web` frontend application in Kubernetes:
 - The **Deployment** creates one replica of the containerized application, using the Docker image `siddharthkothari9403/elecfr-web`, and exposes it on port 3000 within the container.
 - The **Service** of type `NodePort` exposes the application on port 30007, enabling external access to the frontend.
2. The **backend-deployment.yaml** file configures the deployment and service for the `spring-boot-app` backend application in Kubernetes:
 - The **Deployment** creates one replica of the application using the image `siddharthkothari9403/elective-management:latest`. It listens on port 8080 and includes environment variables for database connectivity.
 - The **Service** of type `NodePort` exposes the backend on port 30008, allowing external access to the application.
3. The **db-statefulset.yaml** file sets up a `StatefulSet` and `Service` for the `mysql-db` database in Kubernetes:
 - The **StatefulSet** deploys one replica of the MySQL database using the image `siddharthkothari9403/mysql:latest`. It uses environment variables for configuration and attaches persistent storage of 1Gi to the MySQL data directory.
 - The **Service** of type `ClusterIP` exposes the database internally within the cluster on port 3306, enabling connectivity for other applications.

Horizontal Pod Autoscaler for Frontend and Backend

As shown in Figure 11, we have created autoscalers for the frontend and backend pods, which will introduce new pods when the cpu usage increases beyond 50%.

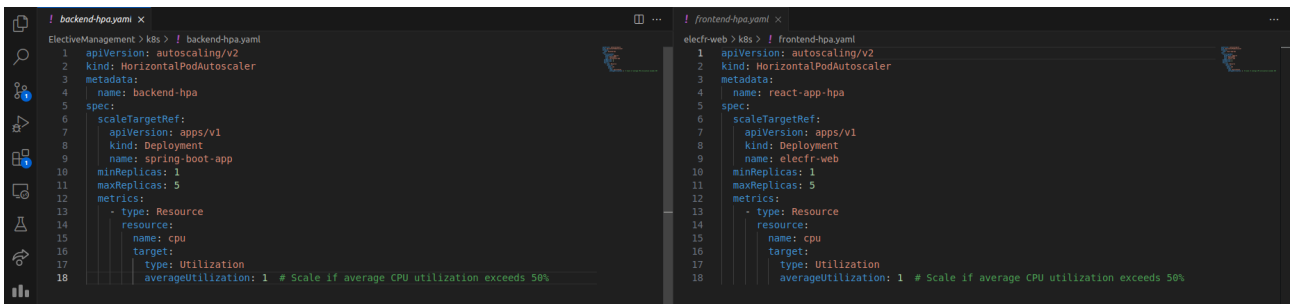


Figure 11: Yaml files for Frontend and Backend HPA

The explanations for the HPA files are provided below -

1. The **backend-hpa.yaml** file configures a Horizontal Pod Autoscaler (HPA) for the `spring-boot-app` deployment:
 - The **HPA** dynamically adjusts the number of replicas between 1 and 5 based on CPU utilization.
 - Scaling occurs when the average CPU utilization exceeds the target threshold, ensuring optimal resource usage and application performance.
2. The **frontend-hpa.yaml** file configures a Horizontal Pod Autoscaler (HPA) for the `elecfr-web` deployment:
 - The **HPA** adjusts the number of replicas dynamically between 1 and 5, responding to changes in resource usage.
 - It scales up when the average CPU utilization exceeds the target threshold, ensuring efficient resource allocation for the application.

3.9 Grafana, Prometheus and Loki

Descriptions

1. **Grafana:** Grafana is an open-source platform for monitoring, visualization, and alerting, often used to create dynamic dashboards. It integrates seamlessly with various data sources, including Prometheus, Loki, and Elasticsearch, allowing users to visualize metrics, logs, and traces in a unified interface. With features like alerts, role-based access control, and panel customization, Grafana is widely adopted for system monitoring and troubleshooting in DevOps and cloud-native environments.
2. **Promtail:** Promtail is a lightweight log collector designed to work with Grafana Loki. It reads logs from sources like files or the system journal, attaches labels to them, and pushes them to Loki for storage and querying. Promtail supports dynamic label assignment through pipelines and can integrate with Kubernetes to collect pod logs, enhancing log organization and searchability.
3. **Prometheus:** Prometheus is an open-source monitoring and alerting toolkit tailored for reliability and scalability. It collects time-series data by scraping metrics from instrumented applications, storing the data in a highly efficient database. With its robust query language (PromQL) and alerting capabilities, Prometheus is a cornerstone of cloud-native observability, often paired with Grafana for visualization.
4. **Loki:** Loki is a log aggregation system that is part of the Grafana stack. Unlike traditional log systems, it focuses on storing logs alongside associated labels without indexing the content, making it cost-effective and efficient. Loki integrates seamlessly with Prometheus and Grafana, providing a unified approach to correlating logs with metrics and enabling effective debugging in distributed environments.

Installation

We first pull the docker images for grafana, loki and prometheus using the commands -

Listing 17: Commands to pull relevant docker images

```
1 $ docker pull grafana/loki:2.9.3
2 $ docker pull grafana/promtail:2.9.3
3 $ docker pull grafana/grafana:latest
```

Following this, we can then install helm using the command -

Listing 18: Helm Installation Command

```
1 $ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

We can then add helm charts for Grafana using the command -

Listing 19: Command to add Helm Charts for Grafana

```
1 $ helm repo add grafana https://grafana.github.io/helm-charts
```

We can now get everything running using the command -

Listing 20: Command to run all Pods and Containers

```
1 $ helm install loki grafana/loki-stack --namespace monitoring --create-namespace --
  set loki.image.tag=2.9.3 --set promtail.enabled=true --set promtail.config.server.
  http_listen_port=3101 --set promtail.config.clients[0].url=http://loki:3100/loki/api
  /v1/push --set promtail.config.positions.filename=/run/promtail/positions.yaml --
  set promtail.config.scrape_configs[0].job_name="kubernetes-pods" --set promtail.
  config.scrape_configs[0].kubernetes_sd_configs[0].role="pod" --set promtail.config.
  scrape_configs[0].relabel_configs[0].action="keep" --set promtail.config.
  scrape_configs[0].relabel_configs[0].source_labels=["_meta_kubernetes_namespace"] --
  set promtail.config.scrape_configs[0].relabel_configs[0].regex=".*" --set promtail.
  config.scrape_configs[0].relabel_configs[1].source_labels=["_meta_kubernetes_pod_name"]
  --set promtail.config.scrape_configs[0].relabel_configs[1].target_label="job" --
  set promtail.config.scrape_configs[0].relabel_configs[2].source_labels=["_
  meta_kubernetes_namespace"] --set promtail.config.scrape_configs[0].relabel_configs
  [2].target_label="namespace" --set promtail.config.scrape_configs[0].relabel_configs
  [3].source_labels=["_meta_kubernetes_pod_name"] --set promtail.config.scrape_configs
  [0].relabel_configs[3].target_label="pod" --set grafana.enabled=true --set
  prometheus.enabled=true
```

This command will create all the necessary pods and services for Grafana and all tools, in the "monitoring" namespace. It also configures Loki and Prometheus to be able to obtain logs and metrics for the pods running outside the monitoring namespace, allowing for us to get logs and metrics from all our pods. Finally, we can enable port forwarding for Prometheus, Grafana and Loki using the commands below. These allow us to run and visualise the logging and metrics.

Listing 21: Port forwarding commands

```
1 $ kubectl port-forward --namespace monitoring svc/loki-prometheus-server 9090:80
2 $ kubectl port-forward --namespace monitoring service/loki-grafana 3000:80
3 $ kubectl port-forward svc/loki -n monitoring 3100:3100
```

We can now login to Grafana at localhost:3000. The username is admin, and the password is obtained using the command below -

Listing 22: Command to get password for Grafana

```
1 $ kubectl get secret loki-grafana -n monitoring -o jsonpath="{.data.admin-password}" |
   base64 --decode
```

4 Additional Configuration Steps

Aside from the installations and setup, we need to run additional steps to configure the entire project. These steps are listed below -

4.1 Credentials in Jenkins

We now need to add Github and Docker Credentials to Jenkins, to ensure that it has the necessary permissions to perform relevant tasks. We also need to add our sudo user credentials to ensure that the ansible playbook can run smoothly. For this, go to Manage Jenkins → Credentials → System → Global credentials (unrestricted). Here we can add new credentials. Figures 12 and 13 show where and how to add credentials to Jenkins.

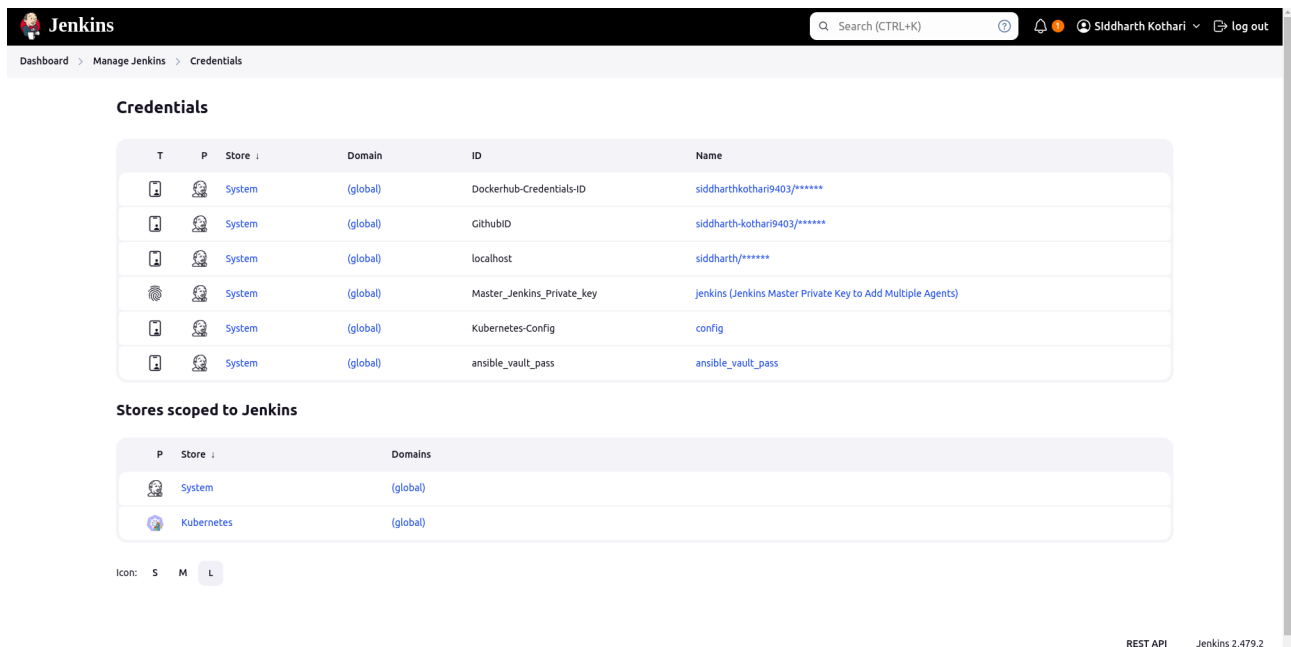


Figure 12: Where to add Credentials in Jenkins

For Github, we can select the Secret Kind to be Username and Password. Here, the username is the Github Username, and Password is the Access Token created previously. Any ID can be provided. For Docker, the steps are similar. The username is the Docker Username, and password is the Docker password. The ID should be kept as "Dockerhub-Credentials-ID", as it has been configured accordingly in the Jenkinsfile provided.

Figure 13: Adding Credentials in Jenkins

For Ansible, the steps are again similar. The username and password are the sudo username and password of the system.

For the Vault password, we need to create a credential of type "Secret Text", and provide the password to the vault in it. The ID should be kept as "ansible_vault_pass", as that has been configured in the Jenkinsfile.

4.2 Jenkinsfile Environment Variables

- `DOCKERHUB_CRED`: Jenkins credentials ID for Docker Hub access.
- `DOCKER_HUB_REPO`: Specifies the Docker Hub repository for storing images.
- `MINIKUBE_HOME`: Defines the Minikube home directory for Kubernetes. This should be replaced by the location of Minikube in the local system.
- `VAULT_PASS`: Jenkins credentials ID for accessing the Ansible Vault password.

4.3 Creating a Pipeline Project Jenkins

To create a pipeline project in Jenkins, open the Jenkins Dashboard. Go to New Item → Pipeline Project, and enter a Project Name. We should check the following options - Github Project, GitHub hook trigger for GITScm polling, and Definition to be "Pipeline Script from SCM". In this one, SCM should be chosen as Git. When prompted, we should provide the Github repository URL, and the Github Credentials created in the previous step, and choose the correct branch of the Github Repository. This will complete the necessary setup of the pipeline project.

4.4 Running the Application

The application can be run by either running the pipeline script in Jenkins, or manually by running the commands listed in the jenkinsfile.

To create the images, we use the following commands -

Listing 23: Commands to Build Images

```
1 $ docker build -t ${DOCKER_HUB_REPO}/mysql:latest ./sql
2 $ docker build -t ${DOCKER_HUB_REPO}/mysql:latest ./ElectiveManagement
3 $ docker build -t ${DOCKER_HUB_REPO}/mysql:latest ./elecfr-web
```

We can then push the images to dockerhub using the commands -

Listing 24: Commands to Push to Dockerhub

```
1 $ docker push ${DOCKER_HUB_REPO}/elecfr-web:latest
2 $ docker push ${DOCKER_HUB_REPO}/elective-management:latest
3 $ docker push ${DOCKER_HUB_REPO}/mysql:latest
```

We can then run the ansible playbook to deploy on kubernetes using the command -

Listing 25: Commands to Push to Dockerhub

```
1 $ ansible-playbook -i inventory --ask-vault-pass deploy-app.yaml
```

5 Results

5.1 Outputs of the Run Stages from Jenkins

Figure 14 shows the outputs of the various pipeline stages. As we observe, the outputs of all the stages are green, indicating that all stages ran without any issues. We will now confirm the various changes this script was supposed to carry out, and verify whether everything works.

✓ SPE_Final_Project

Stage View

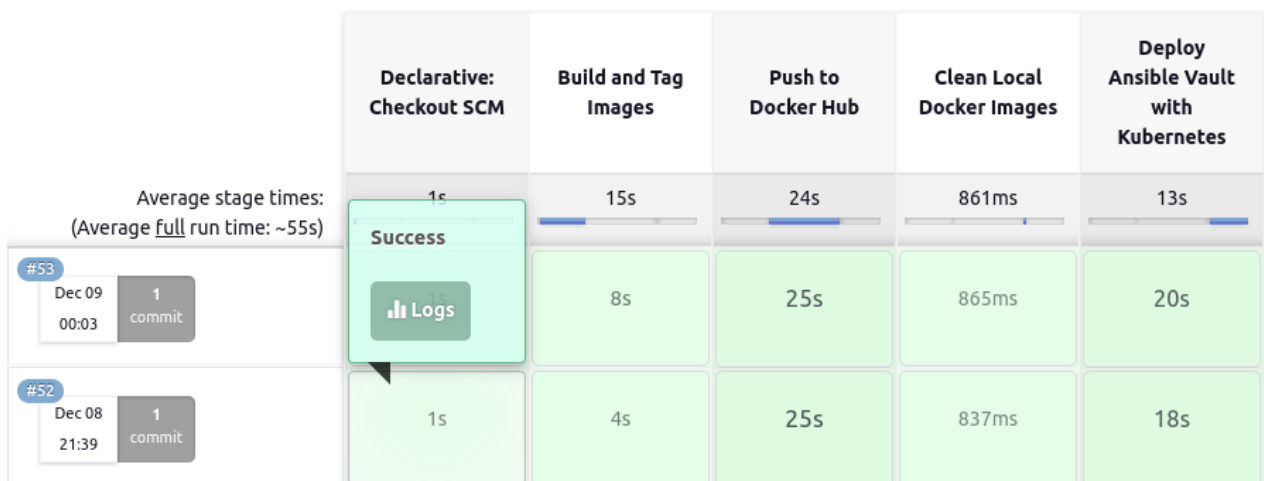


Figure 14: Outputs of the various pipeline stages

5.2 Outputs on Dockerhub

The 3rd stage in the Jenkins pipeline requires us to push the created image to dockerhub, while the 5th stage requires us to pull the images back on the system, after the previous images have been pruned.

As we observe from Figure 15 both the pull and the push occur successfully, and we observe the terminal output for the same in the next section.

5.3 Images Created

Figure 16 contains the outputs of the following command to view the images created.

Listing 26: Docker Command to View All Images

```
1 $ docker images
```

As we observe from the figure, the three images have been pulled from Dockerhub.

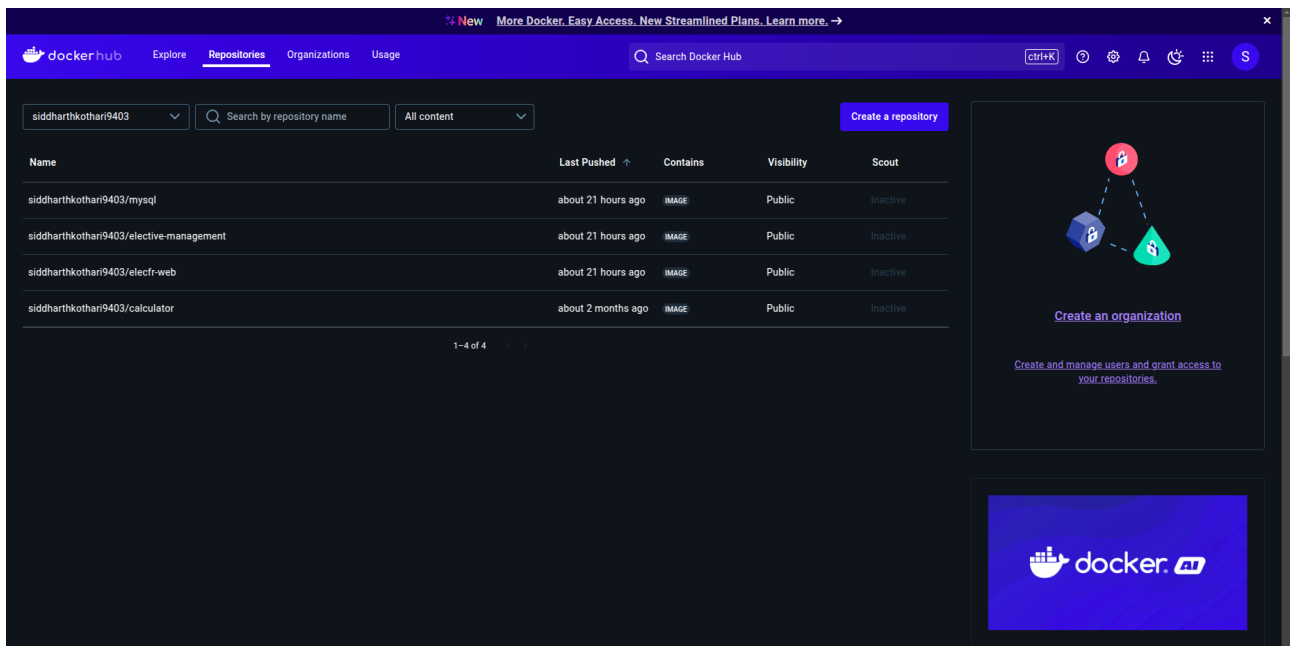


Figure 15: Outputs on Dockerhub

```
siddharth@siddharth-Inspiron-5502:~/SPE_Course_Management$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
siddharthkothari9403/elecfr-web    latest     999b28460b28  30 hours ago  195MB
siddharthkothari9403/elective-management  latest     2a2e59d87aa5  30 hours ago  458MB
siddharthkothari9403/mysql         latest     a6fbee93fca1  33 hours ago  501MB
grafana/grafana             latest     c0b69935a246  4 days ago    486MB
gcr.io/k8s-minikube/kicbase    v0.0.45    aeed0e1d4642  3 months ago  1.28GB
grafana/loki                 2.9.3      6a8de175ce0d  12 months ago  74.6MB
grafana/promtail             2.9.3      6860eccd9725  12 months ago  198MB
opensearch/origin-node        v3.11      c415722faf12  22 months ago  1.2GB
opensearch/origin-control-plane v3.11      567a6d92bc6e  22 months ago  839MB
opensearch/origin-cli         v3.11      3cf32924ae12  22 months ago  390MB
siddharth@siddharth-Inspiron-5502:~/SPE_Course_Management$
```

Figure 16: Docker Images

5.4 Outputs on Kubernetes

We can observe the outputs on Kubernetes by running the following commands -

Listing 27: Docker Command to View All Images

```
1 $ kubectl get all
2 $ kubectl get all -n monitoring
```

This will provide us with the list of running pods, services etc. in the default namespace, along with the logging and monitoring pods and services (Grafana, Loki etc.) in the monitoring namespace. The outputs are shown in Figures 17 and 18.

```
jenkins@siddharth-Inspiron-5502:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/electr-web-cb9f8f9b9-bktr	1/1	Running	0	38h
pod/mysql-db-0	1/1	Running	0	38h
pod/spring-boot-app-5897985fc-kxr6d	1/1	Running	0	38h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/db-service	ClusterIP	10.107.242.65	<none>	3306/TCP	38h
service/electr-web-service	NodePort	10.105.240.63	<none>	3000:30007/TCP	38h
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	38h
service/spring-boot-service	NodePort	10.100.100.28	<none>	8080:30008/TCP	38h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/electr-web	1/1	1	1	38h
deployment.apps/spring-boot-app	1/1	1	1	38h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/electr-web-cb9f8f9b9	1	1	1	38h
replicaset.apps/spring-boot-app-5897985fc	1	1	1	38h

NAME	READY	AGE
statefulset.apps/mysql-db	1/1	38h

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/backend-hpa	Deployment/spring-boot-app	cpu: <unknown>/1%	1	5	1	38h
horizontalpodautoscaler.autoscaling/react-app-hpa	Deployment/electr-web	cpu: <unknown>/1%	1	5	1	38h

Figure 17: Application Pods, Services and Deployments

```
jenkins@siddharth-Inspiron-5502:~$ kubectl get all -n monitoring
```

NAME	READY	STATUS	RESTARTS	AGE
pod/loki-0	1/1	Running	0	2d
pod/loki-alertmanager-0	1/1	Running	0	2d
pod/loki-grafana-557c4ff765-4z8qz	2/2	Running	0	2d
pod/loki-kube-state-metrics-85d4bdc5-7n2g9	1/1	Running	0	2d
pod/loki-prometheus-node-exporter-dnd6p	1/1	Running	0	2d
pod/loki-prometheus-pushgateway-7999888c99-7q5qh	1/1	Running	0	2d
pod/loki-prometheus-server-598f8bc7d5-jlbjt	2/2	Running	0	2d
pod/loki-prontail-99wf6	1/1	Running	0	2d

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/loki	ClusterIP	10.106.50.115	<none>	3100/TCP	2d
service/loki-alertmanager	ClusterIP	10.98.44.214	<none>	9093/TCP	2d
service/loki-alertmanager-headless	ClusterIP	None	<none>	9093/TCP	2d
service/loki-grafana	ClusterIP	10.97.90.229	<none>	80/TCP	2d
service/loki-headless	ClusterIP	None	<none>	3100/TCP	2d
service/loki-kube-state-metrics	ClusterIP	10.110.133.239	<none>	8080/TCP	2d
service/loki-memberlist	ClusterIP	None	<none>	7946/TCP	2d
service/loki-prometheus-node-exporter	ClusterIP	10.107.120.60	<none>	9100/TCP	2d
service/loki-prometheus-pushgateway	ClusterIP	10.99.188.41	<none>	9091/TCP	2d
service/loki-prometheus-server	ClusterIP	10.105.3.254	<none>	80/TCP	2d

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset.apps/loki-prometheus-node-exporter	1	1	1	1	1	<none>	2d
daemonset.apps/loki-prontail	1	1	1	1	1	<none>	2d

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/loki-grafana	1/1	1	1	2d
deployment.apps/loki-kube-state-metrics	1/1	1	1	2d
deployment.apps/loki-prometheus-pushgateway	1/1	1	1	2d
deployment.apps/loki-prometheus-server	1/1	1	1	2d

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/loki-grafana-557c4ff765	1	1	1	2d
replicaset.apps/loki-kube-state-metrics-85d4bdc5	1	1	1	2d
replicaset.apps/loki-prometheus-pushgateway-7999888c99	1	1	1	2d
replicaset.apps/loki-prometheus-server-598f8bc7d5	1	1	1	2d

NAME	READY	AGE
statefulset.apps/loki	1/1	2d
statefulset.apps/loki-alertmanager	1/1	2d

Figure 18: Monitoring Pods, Services and Deployments

5.5 Outputs inside the Database

We can connect to the database pod and observe changes via the following commands -

Listing 28: Docker Command to View All Images

```
1 $ kubectl exec -it mysql-db-0 -- bash
2 $ mysql -h 127.0.0.1 -P 3306 -u <MYSQLUSER> -p
```

We can then type in the password, and access the databases and tables. Figure 19 shows the output of the same.

```
jenkins@siddharth-Inspiron-5502:~$ kubectl exec -it mysql-db-0 -- bash
bash-4.2# mysql -h 127.0.0.1 -P 3306 -u springstudent -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 402
Server version: 5.7.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use elective_management1
ERROR 1044 (42000): Access denied for user 'springstudent'@'%' to database 'elective_management1'
mysql> use elective_management;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_elective_management |
+-----+
| instructor                     |
| request                       |
| role                          |
| student                       |
| student_subject               |
| subjects                      |
| user                          |
| user_roles                    |
+-----+
8 rows in set (0.00 sec)

mysql> select * from subjects;
+-----+
| subject_code | instructor_code | subject_name | subject_desc |
+-----+
| 1            | NULL           | DSA          | Data Structures and Algorithms |
| 2            | 7              | Web Dev      | Web Development in React and Spring Boot |
| 3            | 6              | DBMS         | Database Management Systems and their applications |
| 4            | NULL           | Let us C     | An introductory course in C programming |
| 5            | 2              | Probability  | Bayes Theorem, Conditional Probability |
| 6            | 3              | App Dev      | App development using React Native |
| 7            | 4              | Economics   | Micro and Macro Economics - an overview |
| 8            | 1              | Machine Learning | Intro to the idea of Supervised and Unsupervised ML algorithms |
| 9            | 1              | Math for ML  | Mathematics that goes into designing ML algorithms |
| 10           | 10             | Spoken English | Helps develop fluency in the English language, through various tasks and activities |
+-----+
10 rows in set (0.00 sec)
```

Figure 19: Outputs inside the Database

5.6 Outputs of the Frontend

Figures 20 and 21 show a few screens from the frontend. The frontend can be accessed on http://{Minikube_IP}:30007.

5.7 Outputs from Logging

Grafana, Prometheus and Loki can be accessed on <http://localhost:3000>, <http://localhost:9090> and <http://localhost:3100> respectively, due to port forwarding. Figures 22, 23 and 24 show the outputs from port forwarding.

We have also made a dashboard for visualising the metrics and logs from the pods. This is shown in Figure 25.

6 Relevant Links

6.1 Github Link

The Github Link for the Project is - https://github.com/siddharth-kothari9403/SPE_Course_Management.

6.2 Dockerhub Link

The Dockerhub Link for the hosted images is - <https://hub.docker.com/repositories/siddharthkothari9403>.

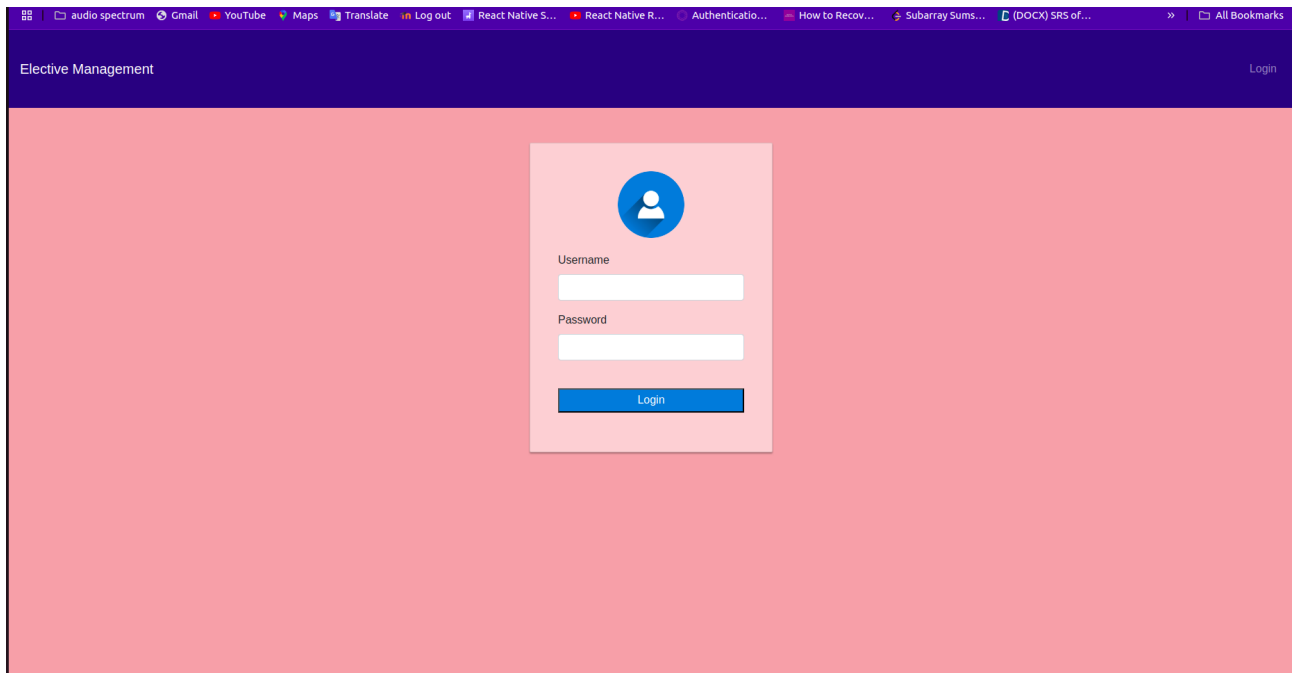


Figure 20: Frontend Login Page

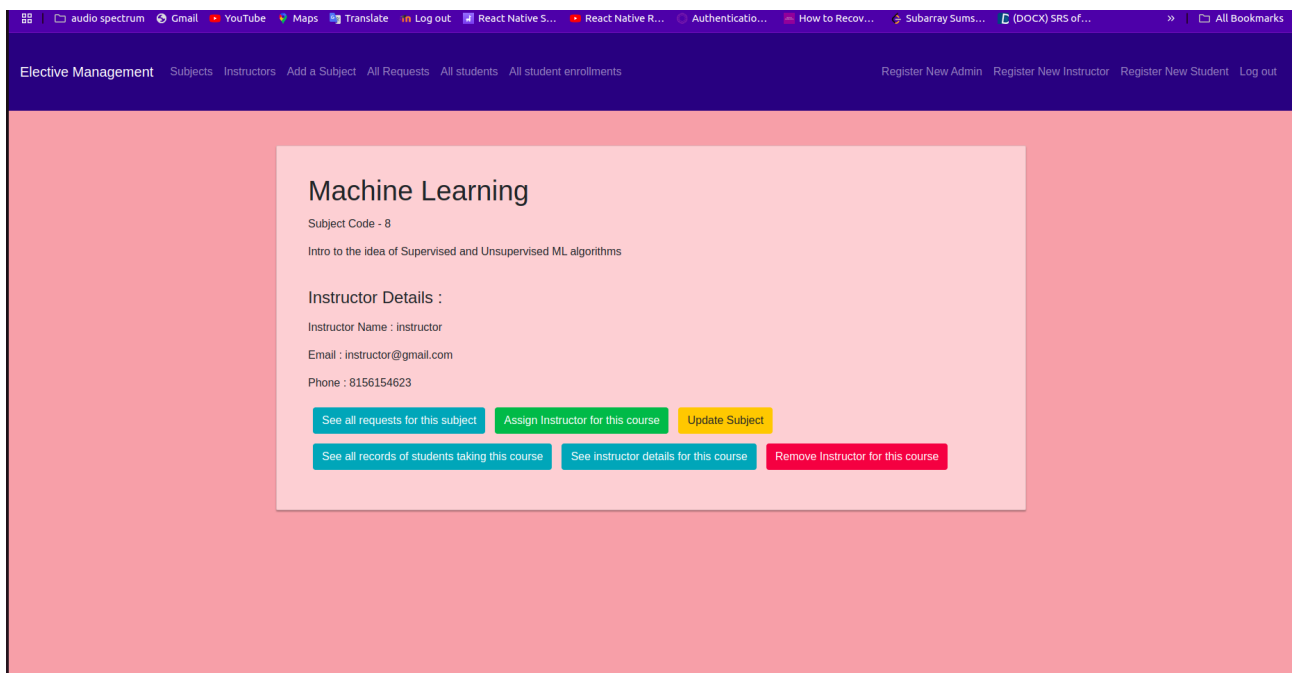


Figure 21: Frontend Screen

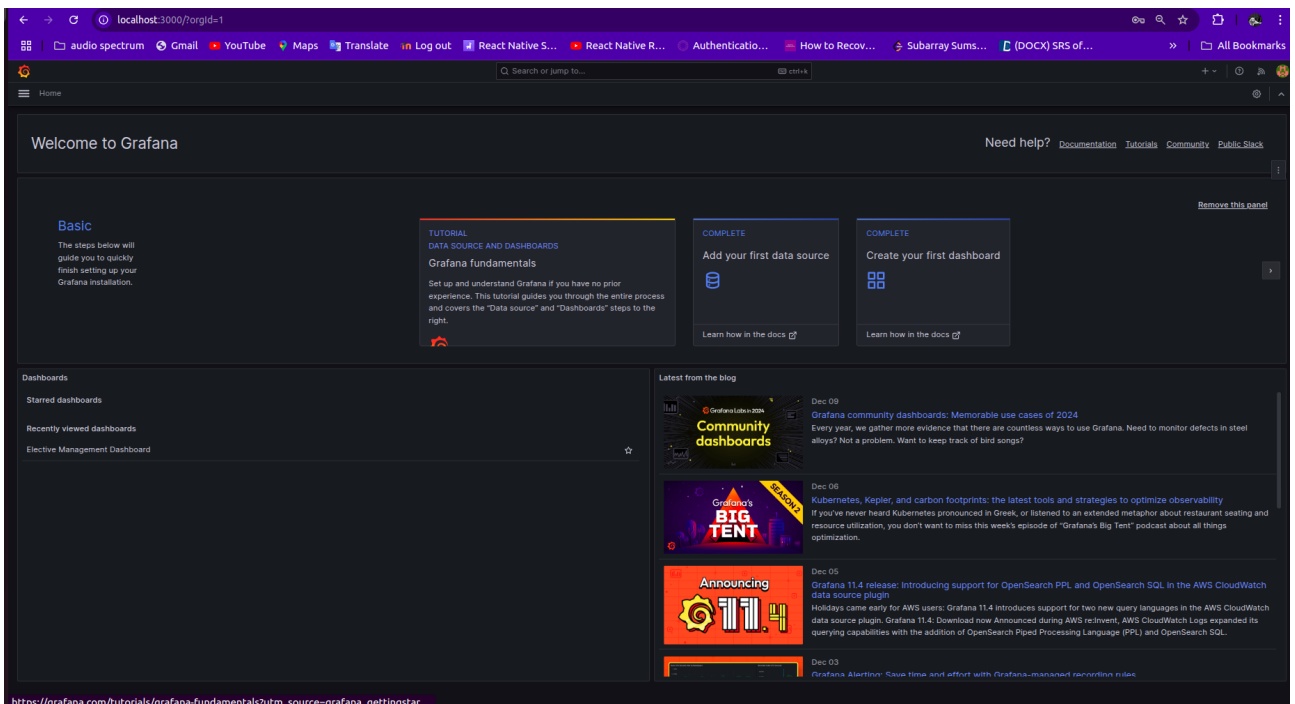


Figure 22: Grafana Running

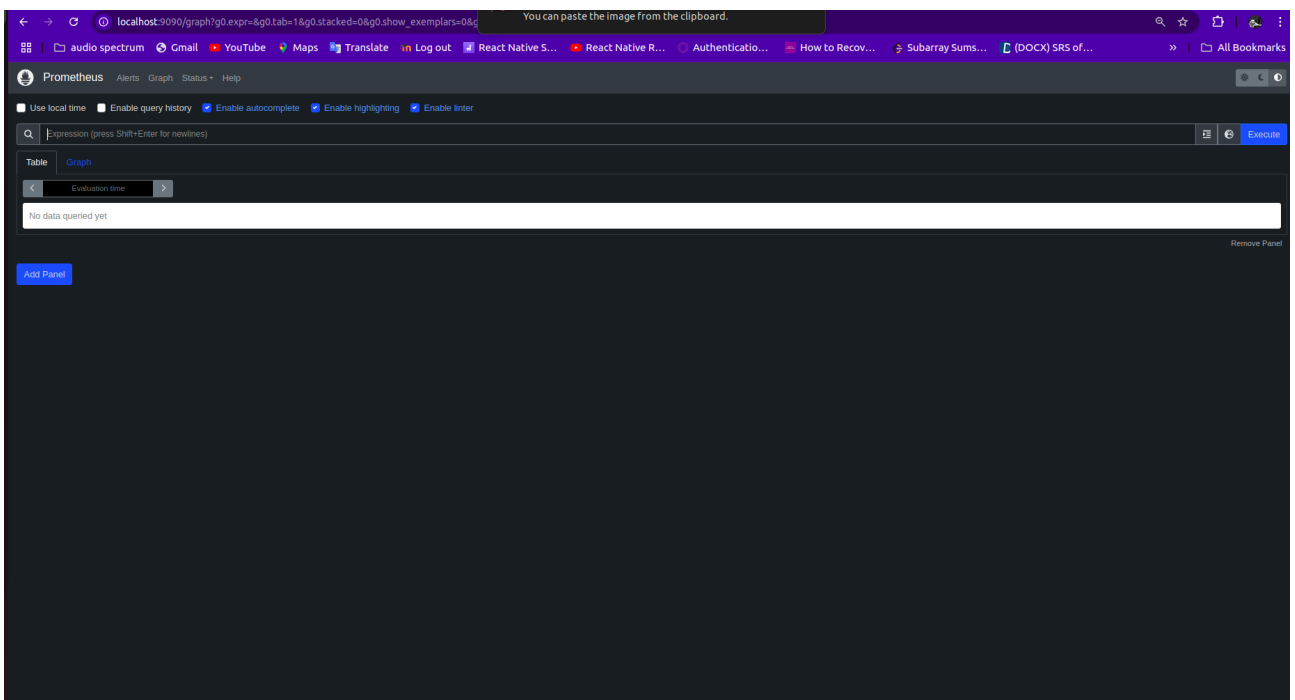


Figure 23: Prometheus Running

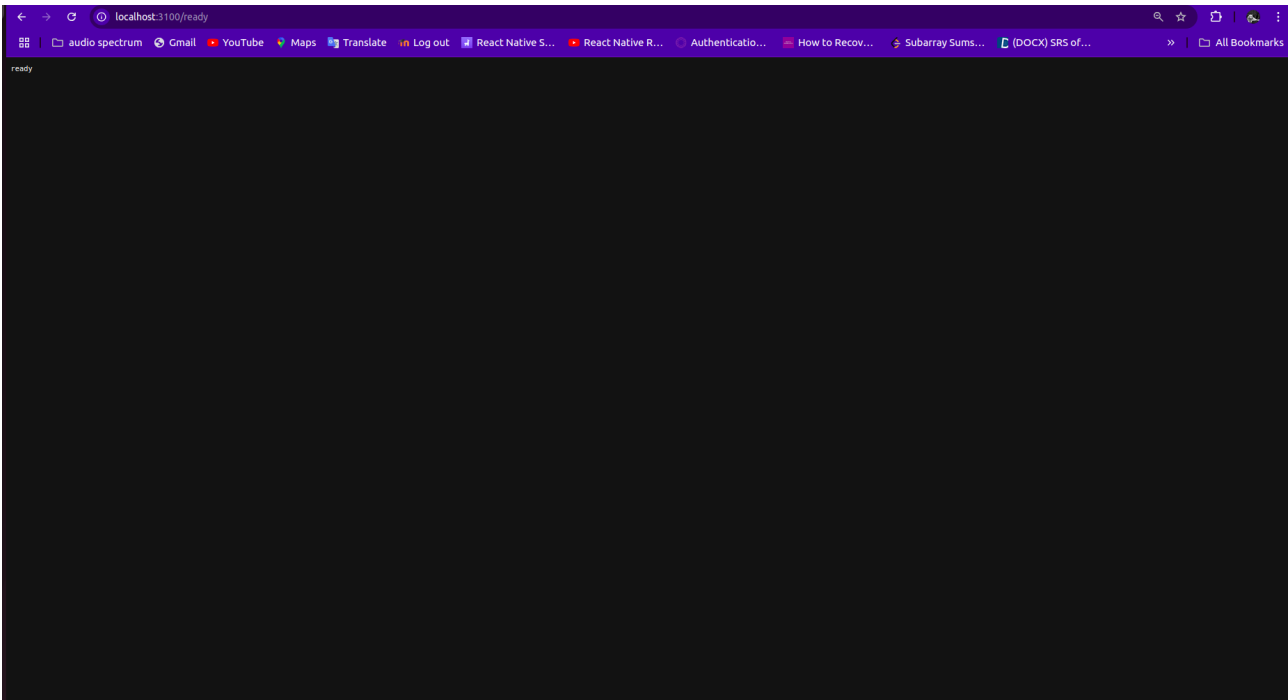


Figure 24: Loki Running

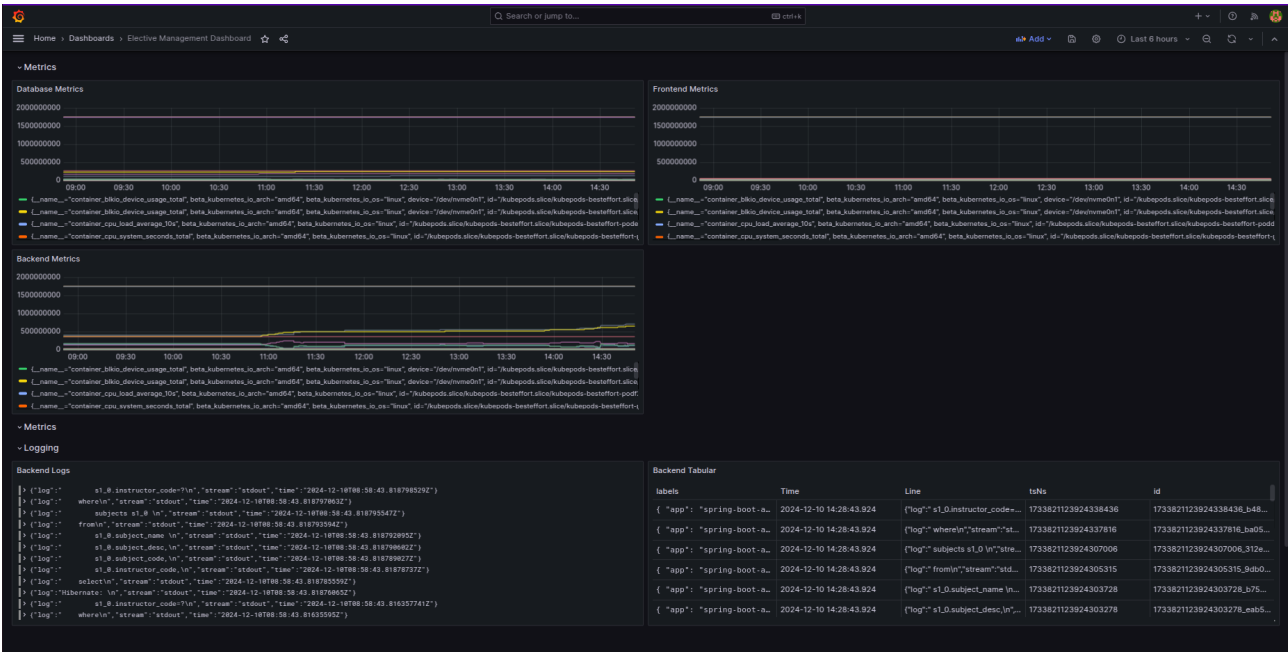


Figure 25: Grafana Dashboard