SOFTWARE PRODUCTION ENGINEERING

PLACEMENT MANAGER WEBAPP

A platform connecting recruiters and students

By

Karthik Hegde IMT2018509

Jishnu V Kumar IMT2018033

Under the guidance of

Prof B.Thangaraju

Teaching Assistant

Vaibhav

Table of Contents

ABSTRACT

In this project we have developed a platform that connects recruiters of different companies and aspiring students applying for the companies. This website manages student profiles including their resumes and eases out the process of applying to an interview. Administrators (placement committee members) can open up company recruitment drives on behalf of recruiters. Recruiters can append meeting links, online test links and other guidelines regarding interview, job and compensation. Moreover the students are automatically updated regarding new companies starting out recruitment drives.

The project has been developed using DevOps model and automation tools reducing the release time of new features. The project has been built using Springboot backend, React frontend and MySQL database.

INTRODUCTION

About the project

Placement Manager webapp is a website built using 3-tier architecture involving React frontend, Spring Boot backend and MySQL database.

The platform currently accepts three types of users - Admin, Recruiter and Student. Admin manages creation of new recruitment drives - a drive is an end-to-end process involving many rounds of interviews. The recruitment drive will be managed by recruiters who will update the general guideline regarding the interview. The recruiters can also update other details regarding job description, job compensation, meeting link (if the interview is going to be online), test links. While the interview continues, the recruiters can also choose to eliminate some of the candidates. The Students are updated about the latest company drives and a deadline date before which they can apply for a company and participate in the interview process. The student can observe the recruitment drive details until he/she is eliminated. Students can update/edit their profiles and upload/update their resumes. Currently the site accepts three different types of resumes - software , data-science and ece. Each recruitment drive can be one of the three streams - software, data-science and ece. When the student applies for a recruitment drive, recruiters will be able to review the student's resume. The recruiters can only review those types of resumes that match with the recruitment drive's stream.

Admin's functionalities

- Create users - admin, recruiter, student
- Create a new recruitment drive
- Delete a recruitment drive

Recruiter's functionalities

- Update the general guidelines regarding interview process
- Update links for meeting (if it is going to be virtual interview)
- Update test links (if there is going to be online test rounds)
- Update job description
- Update job pay
- Review applicant's resume
- Eliminate some or all applicants

Student's functionalities

- Update their profile details
- Update their resume or upload a new one

- Apply for a new recruitment drive
- Track recruitment drive's information

What is DevOps?

Devops is the set of software principles that extend agile principles beyond the boundaries of "the code" to the entire delivered service. It applies the agile methodology to deployment, environment configurations, monitoring as well as maintenance tasks enhancing the quality and lowering the time required to make the product available for the customers/users.

Why DevOps?

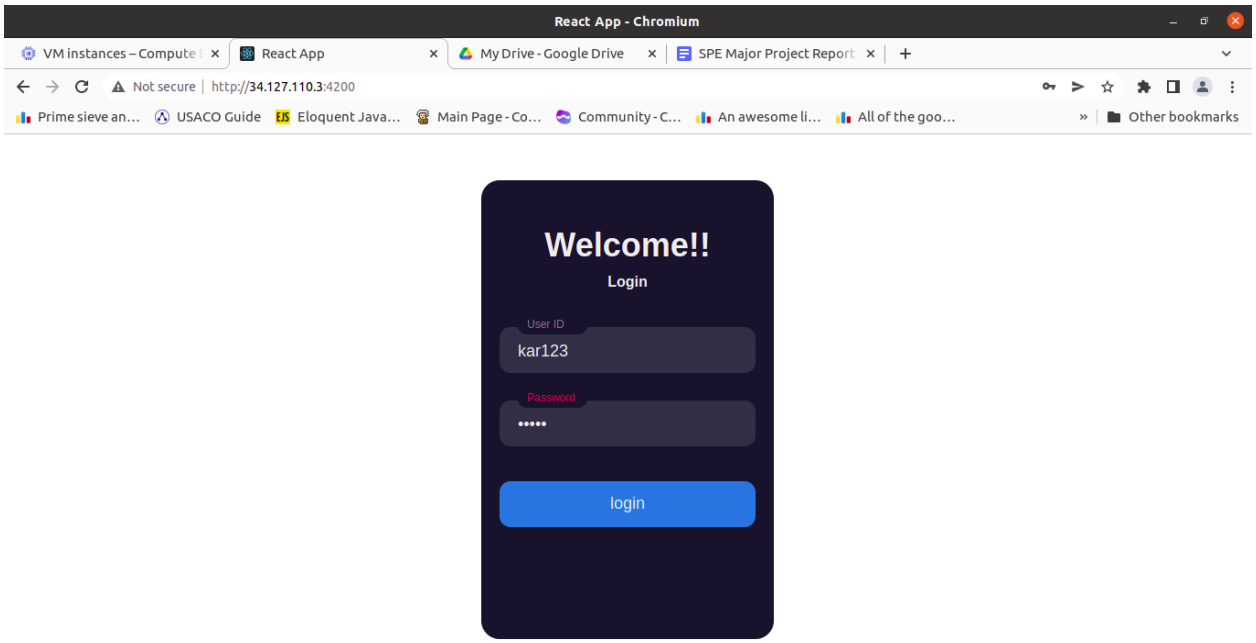
The primary goal of Devops is to reduce the time to market the service.

- It improves the deployment frequency
- Lower failure rate of new release
- Improves mean time to recovery in the event of a new release crashing

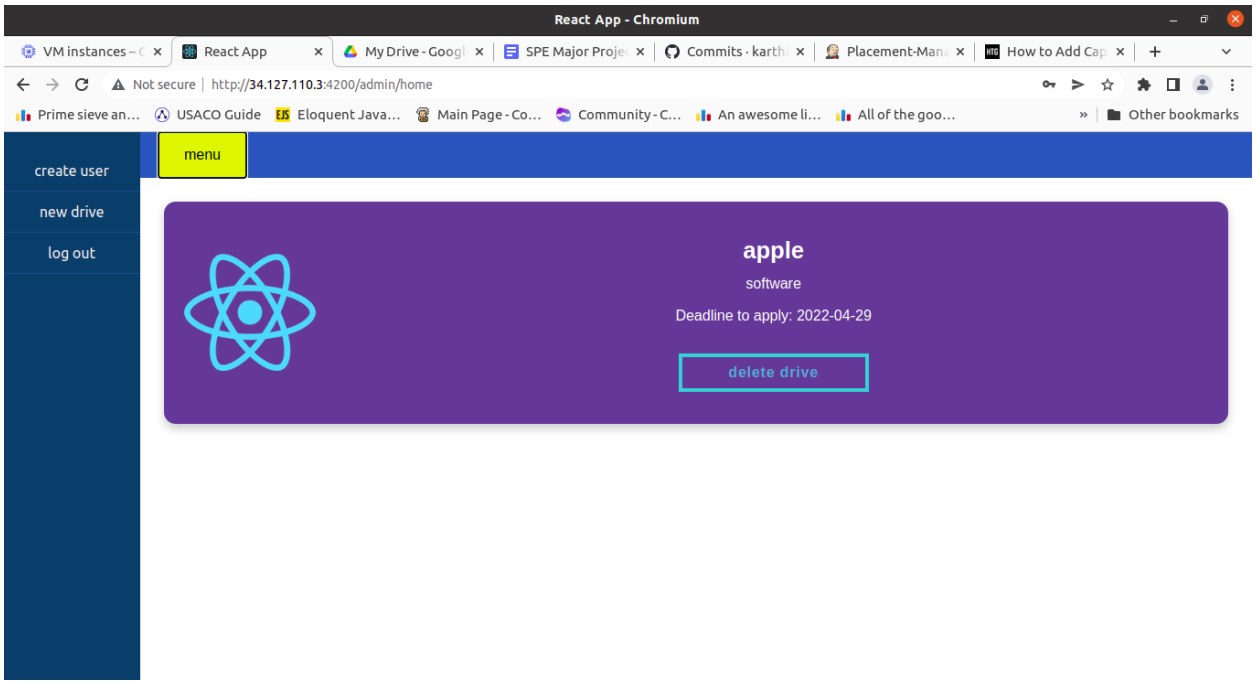
Technology Stack

- Frontend → React
- Backend → Java Spring Boot
- Database → MySQL
- Testing → Junit
- Source Code Management → GitHub
- Containerization → Docker, Container image repo → Docker Hub
- Automation of Continuous Integration, Continuous Deployment → Jenkins/Github Actions
- Continuous delivery and deployment → Ansible
- Deployment platform → Google Cloud Platform/ Google Kubernetes Engine
- Continuous Monitoring → ELK stack + filebeat

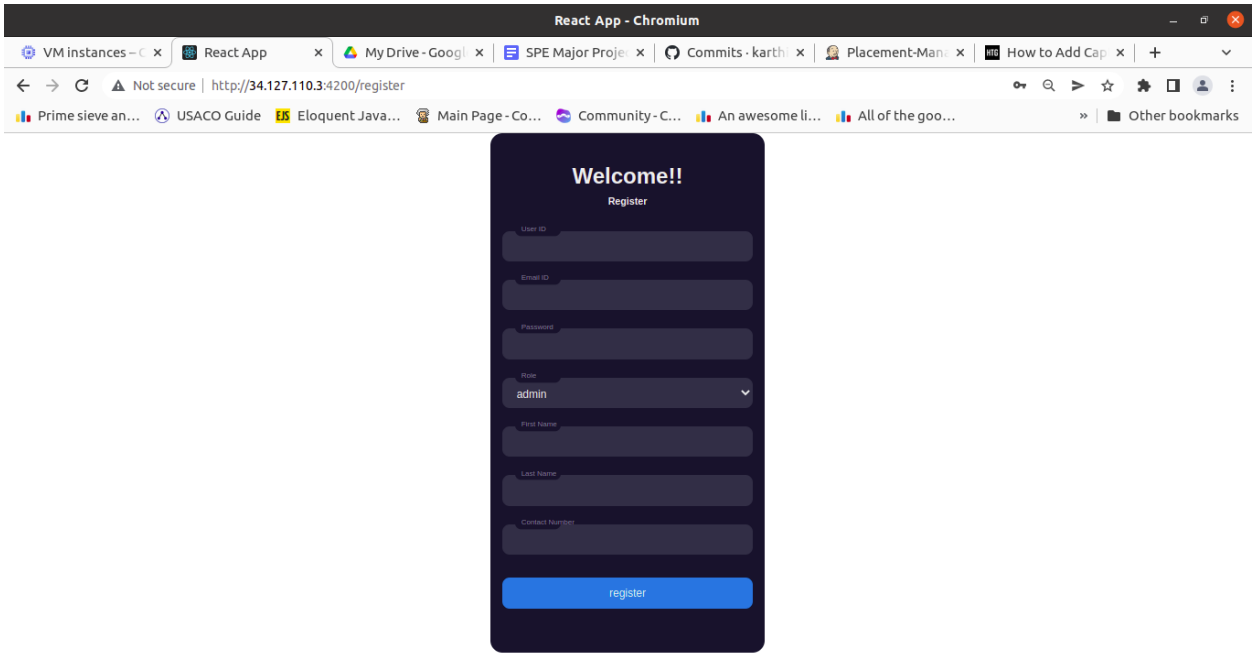
Snapshots of working website



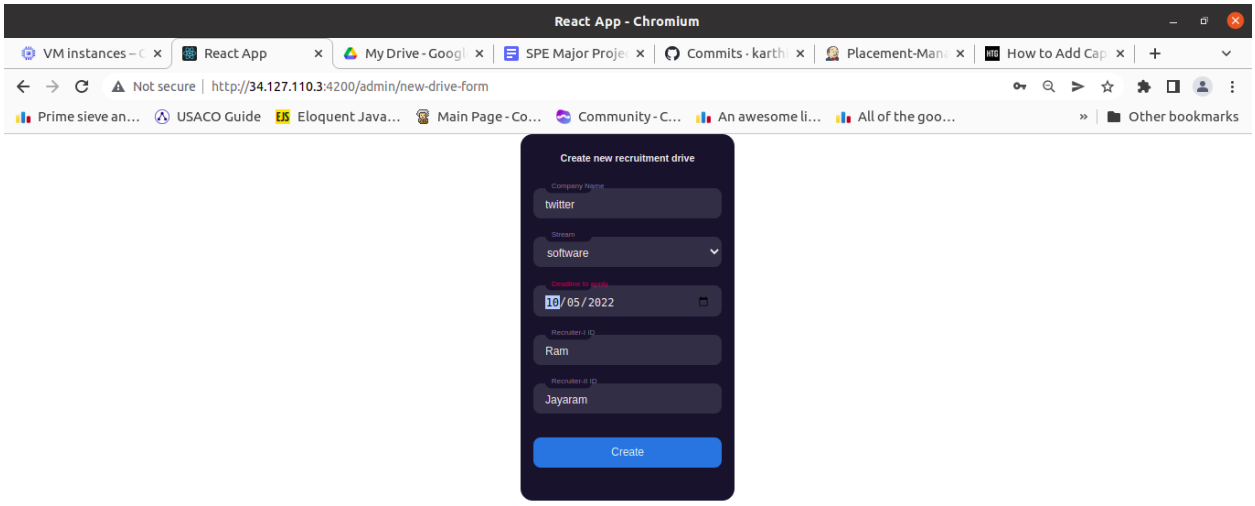
Login page



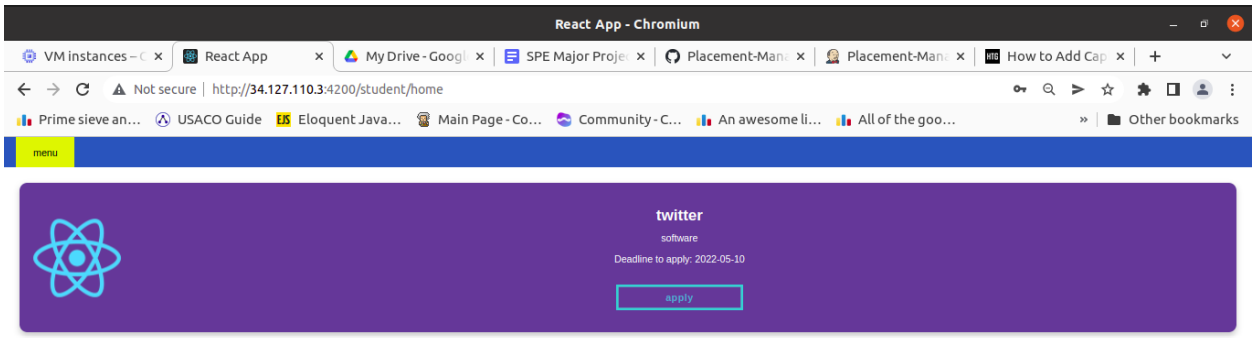
Admin home page



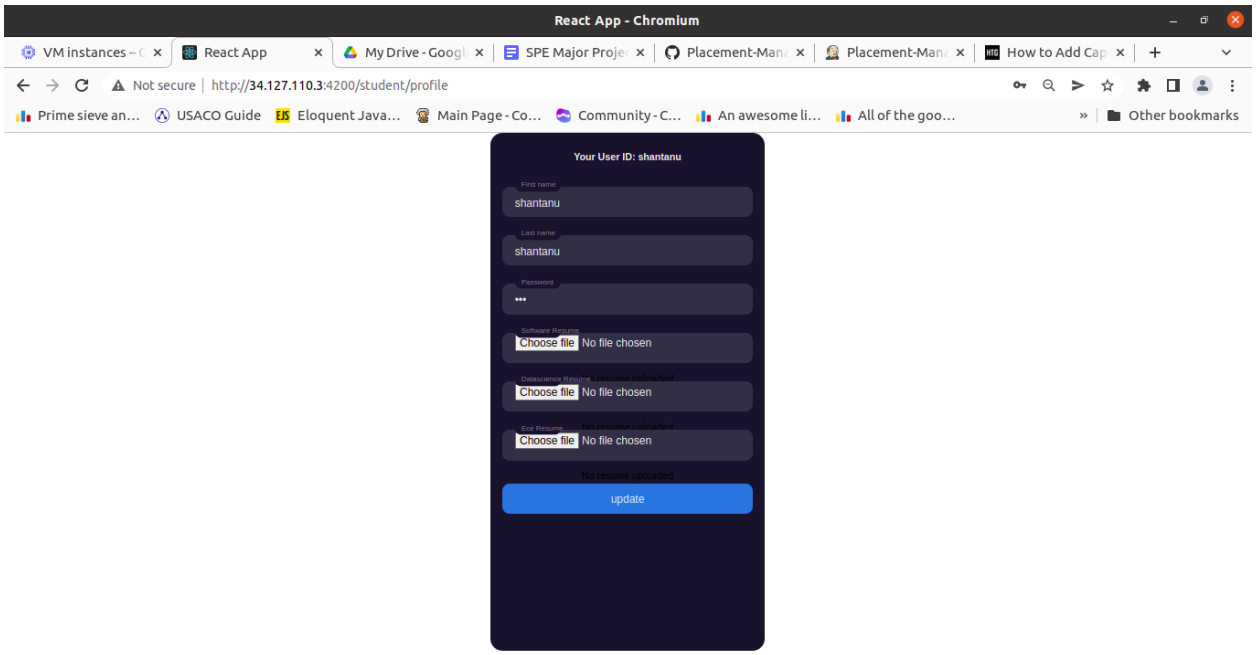
Register new users



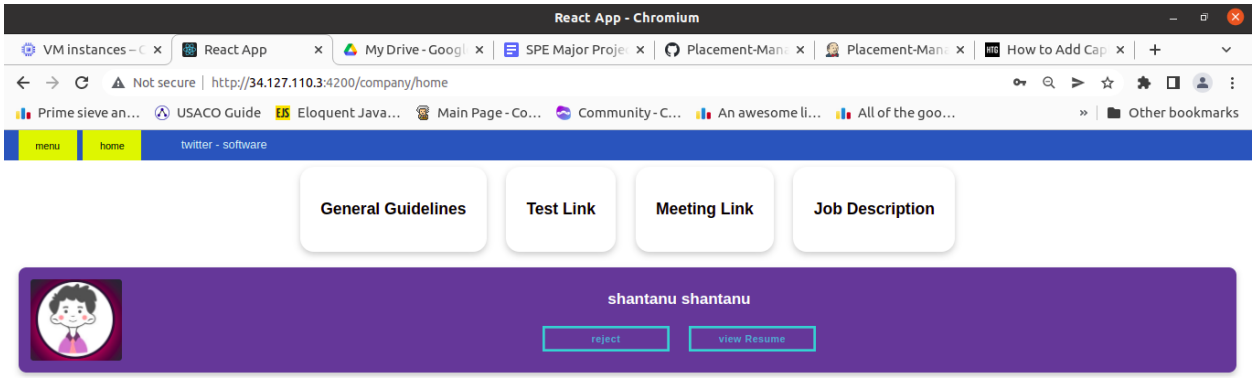
Create new recruitment drive



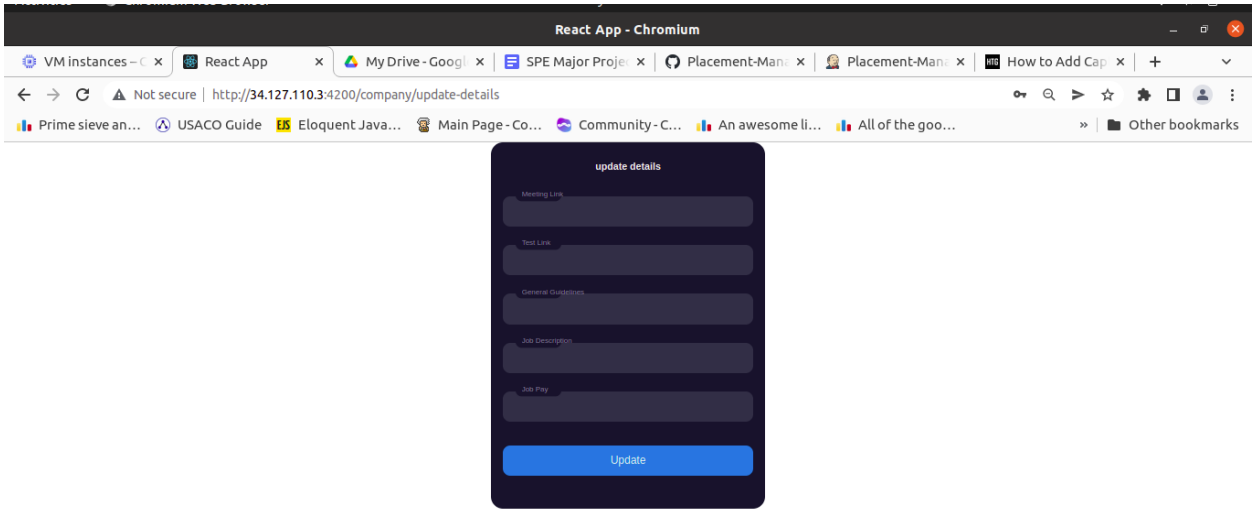
Student home page



Student profile page



Company home page - visible to recruiters and admin



Update details and guidelines about the recruitment drive

SOFTWARE DEVELOPMENT LIFE CYCLE

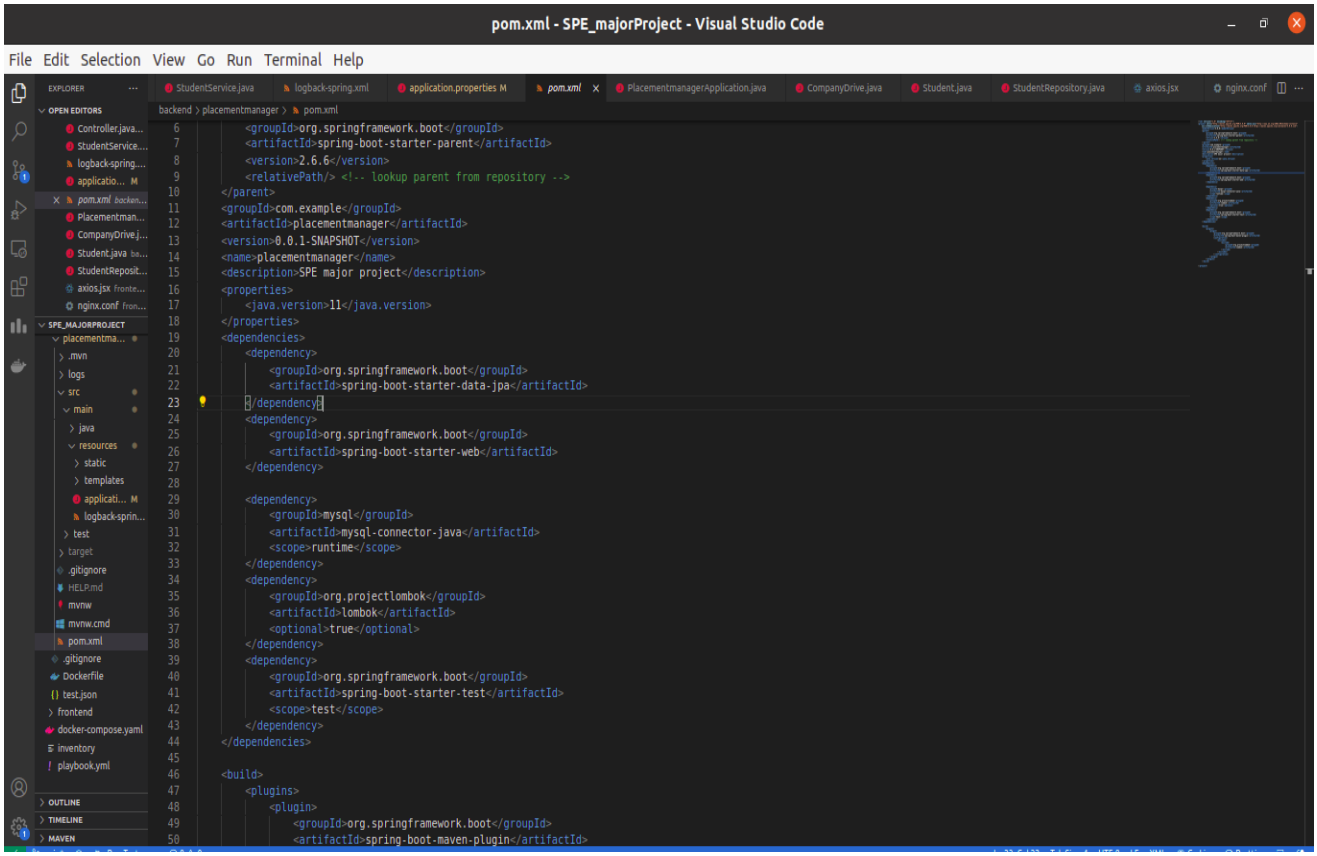
In this section, we will describe the entire workflow of software development of this project.

Spring Boot Backend

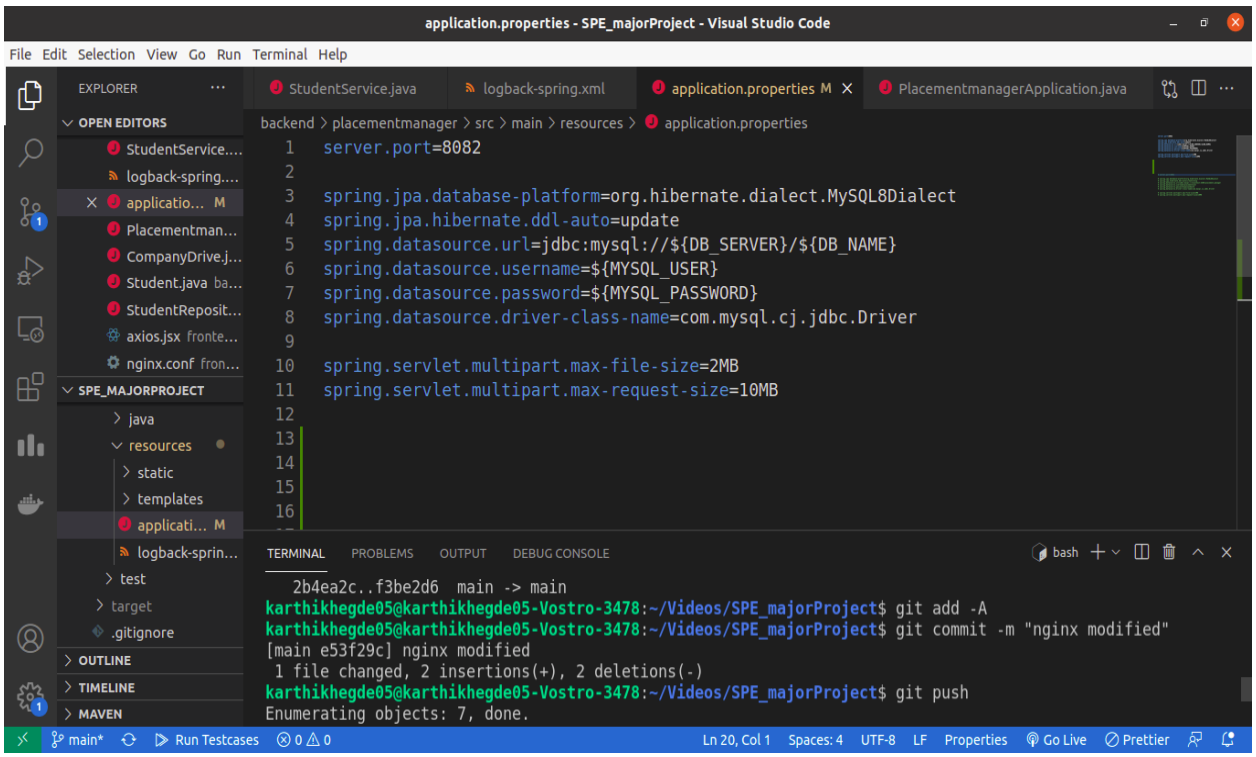
Spring boot is a framework based on Java that allows building MVC applications quickly and easily. Benefits of using spring boot was we get support of embedded tomcat server which makes the developed code production ready, opinionated starter dependencies that simplify the build process. It can be easily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

Steps to setup and configure

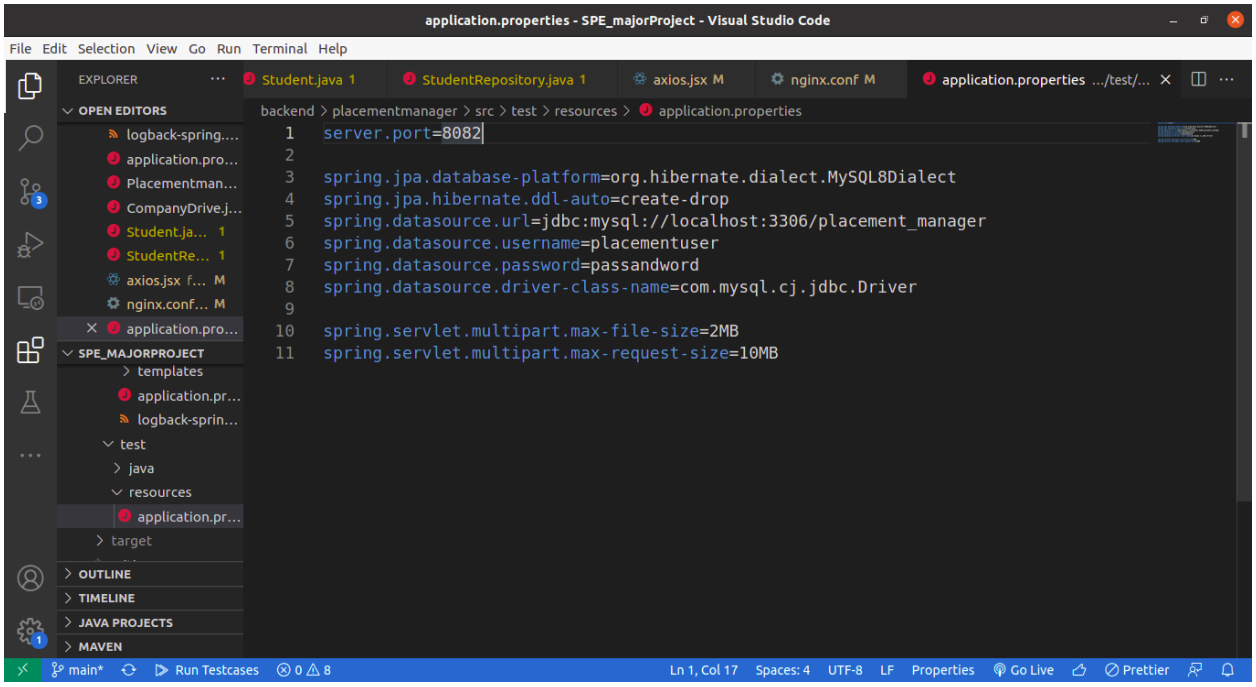
- First we need to set up a spring boot project. For this we can either install appropriated spring extensions in our IDE or directly download the project from the Spring initializer - https://start.spring.io/
- In the initializer we can add dependencies and java version → We chose java 11 and used Spring Web, Lombok, MySQL driver, Spring data JPA dependencies for the development
- We can set application properties (in src/main/resources/application.properties file) such as port (default - 8080) that backend is accessible, mysql database configurations (database url, type of database etc), maximum file size that can uploaded as resume (Multipart file upload)
- As it is visible in the image, the src/main/resources/application.properties are the properties used in deployment. Here we can see we have set the mysql url, username and password as environment variables which are configured at the time of docker container initialization. But this is not suitable for testing the database since these variables are required to be defined at the time of testing. For this we have added src/test/resources/application.properties that override the earlier configurations at the time of testing.



pom.xml content



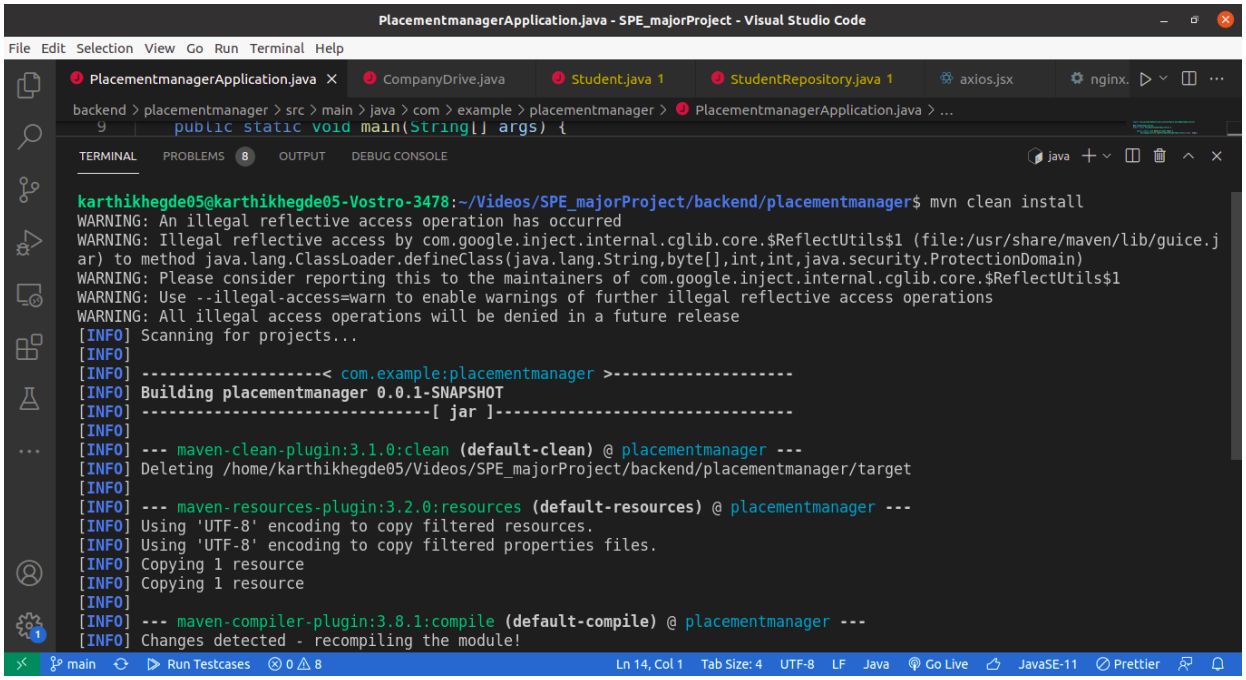
src/main/resources/application.properties file



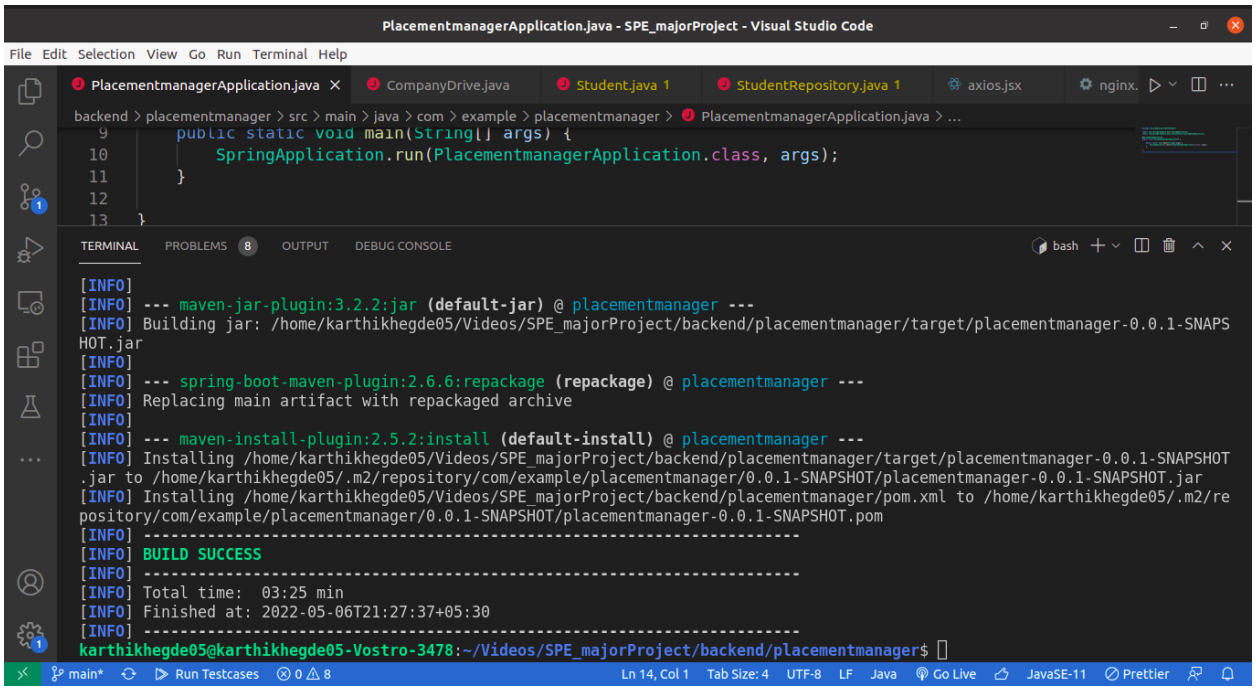
src/test/resources/application.properties file

Commands to generate a jar file

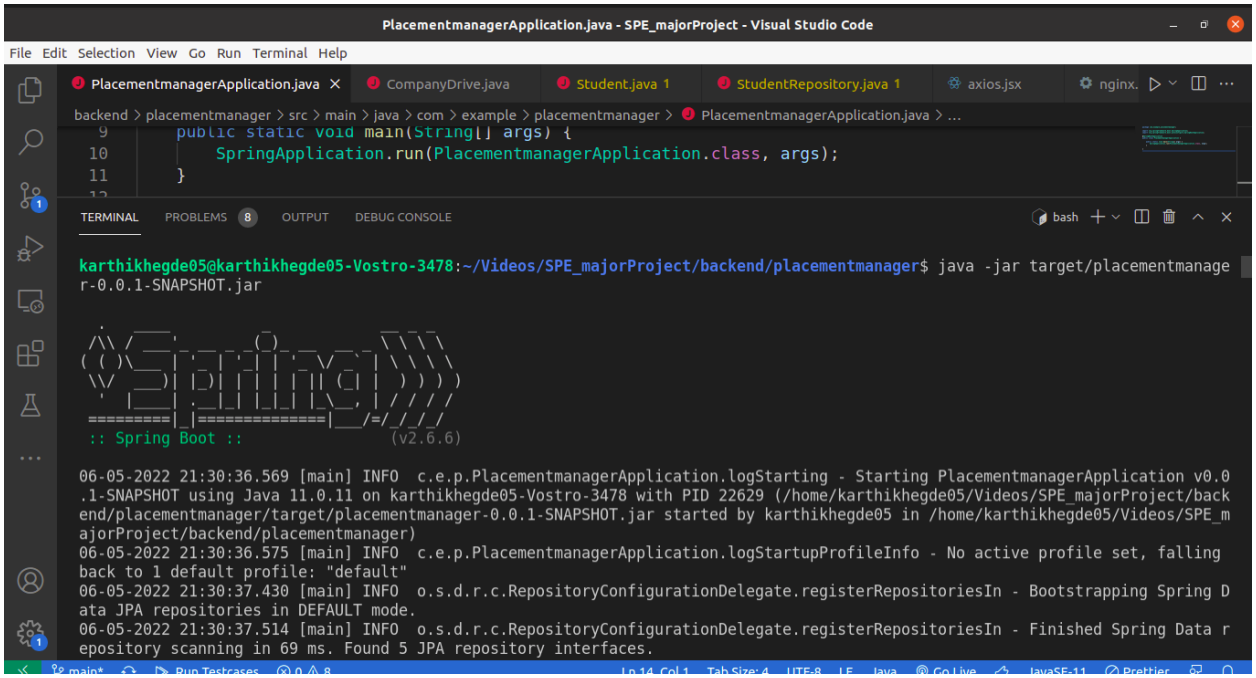
- Go to the project directory where src directory resides → $ cd backend/placementmanager/
- Type $ mvn clean install → this will build and also test
- After this in the target folder, we can see the generated jar file → in our case it is placementmanager-0.0.1-SNAPSHOT.jar . command to execute this is → $ java -jar target/placementmanager-0.0.1-SNAPSHOT.jar



Maven clean install command execution



Maven build Test result



Executing the jar file

React Frontend

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

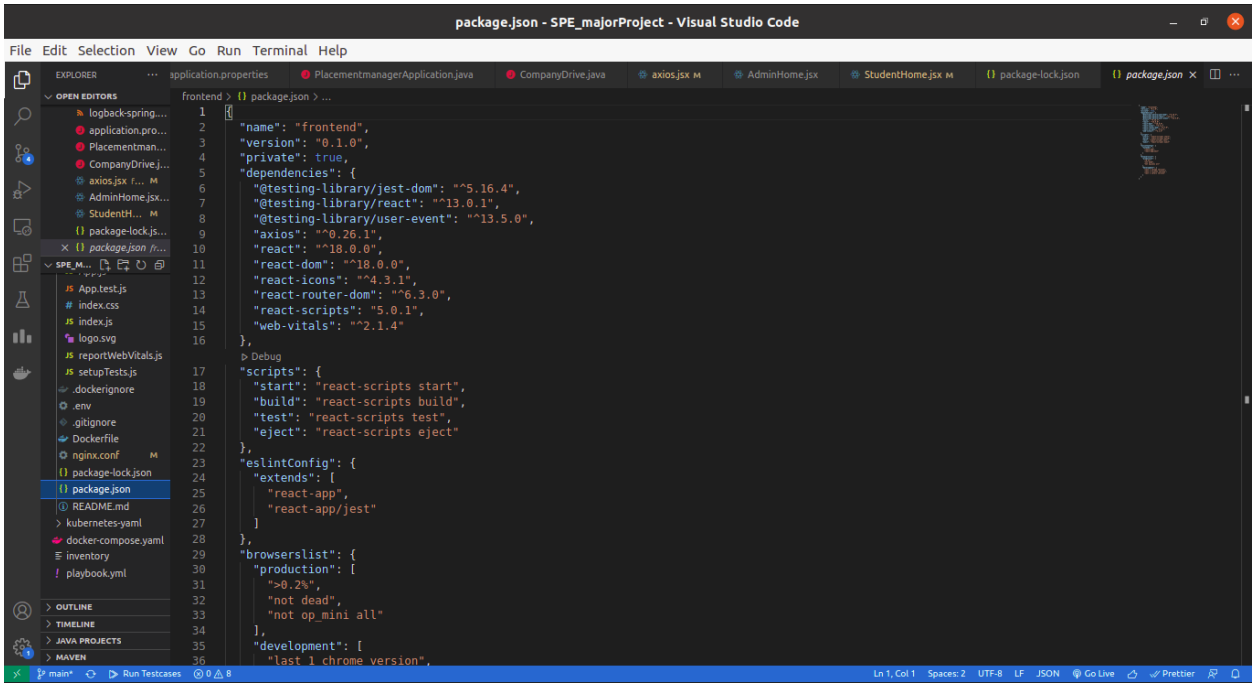
React needs a web server tool to be deployed. For this, we have used Nginx open-source web server tool for routing http requests.

Axios is another Javascript library used to make http requests from node or XMLHttpRequests from the browser. We are using this to make http requests from the react frontend to backend rest api.

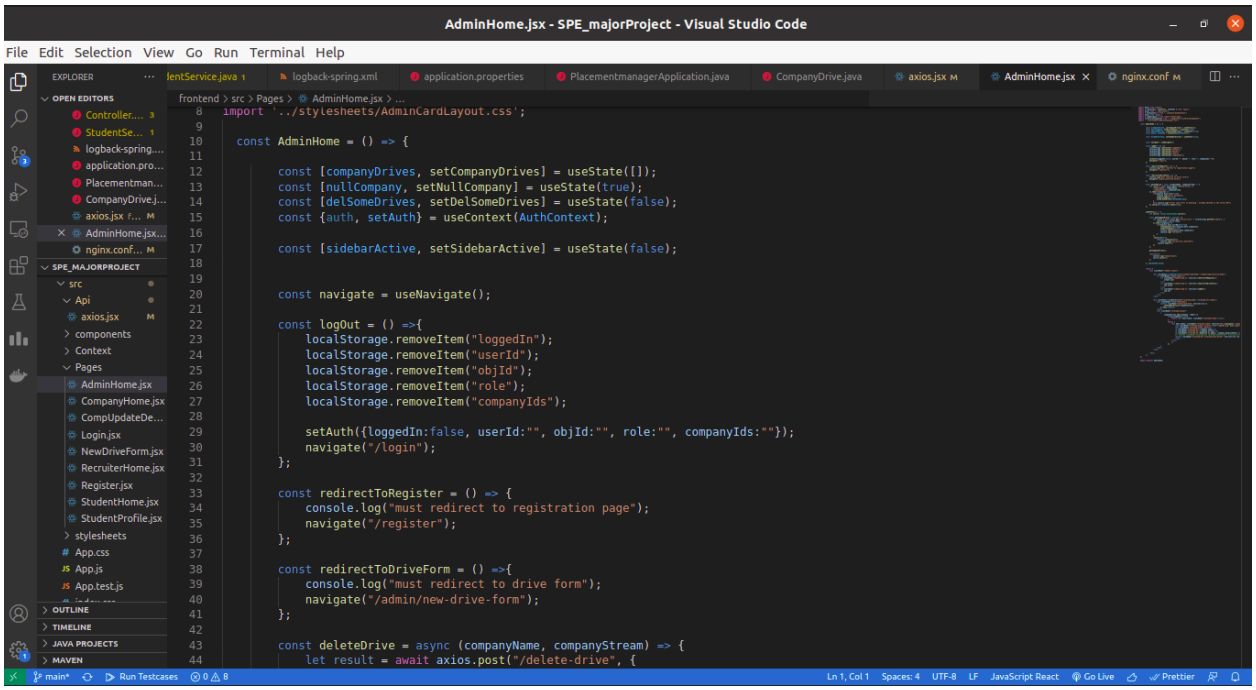
Setup and configuration

- First we need to install Node.js and Node Package Manager (npm) https://nodejs.org/en/
- Install react using npm → $ npm install -g create-react-app
- Create react app workspace → $ create-react-app <project Name> → (In our case project Name is frontend)
- Install axios (required for routing GET and POST calls to backend REST API)
  → $ npm install axios

→ upload images of react frontend code



Package json file defining npm dependencies



admin-home frontend code

Source Code Management

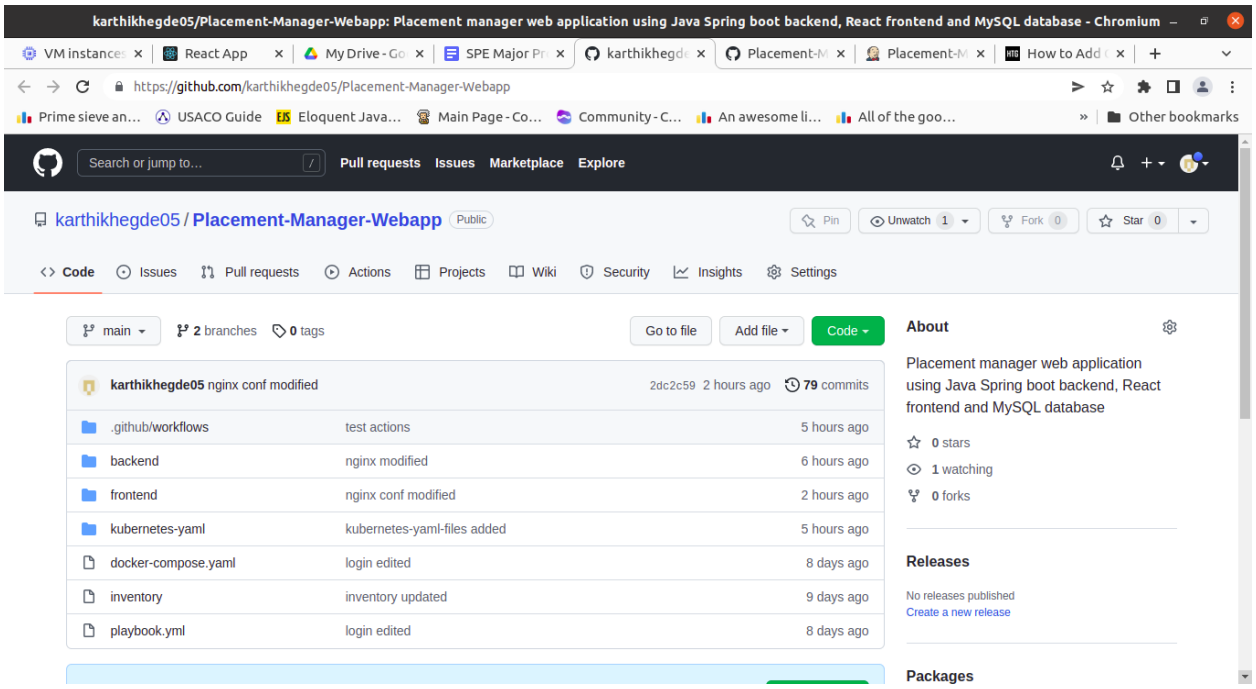
For source code management we are using Git which is a distributed version control system. For making it public to others, we are using GitHub which is a web based git file hosting service.

Our git repository link → https://github.com/karthikhegde05/Placement-Manager-Webapp

Setup and configuration

- Install Git in the local machine
- In the project folder initialize git repository → $ git init .
- Create a main branch → $ git branch -m main
- Add the changes to the staging area → $ git add .
- Commit the changes → $ git commit -m “<appropriate message>”
- Login to the gitHub and create a new repository
- To set the remote repo type the command in the local machine
  → $ git remote add origin <remote-url>
- Push the commits to remote repo → $ git push origin -u main

In this way, whenever a new change is committed it can easily be pushed to the remote repo.



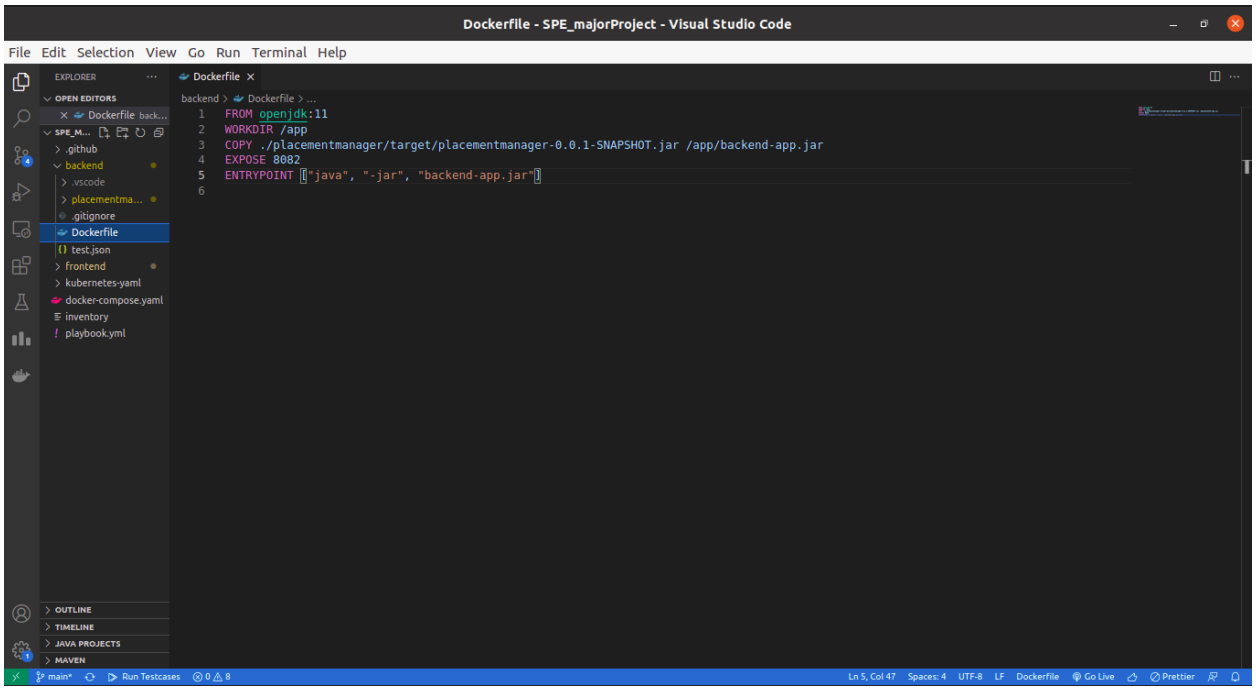
Github repo

Containerization

To describe in brief containerization is a modern virtualization method defined as a form of operating system virtualization. Containers access a single OS kernel to power multiple distributed applications that are each developed and run in their own container. An application with all its dependencies and libraries needed are packed as an image that can be run as a container anywhere - desktop, traditional IT, or the cloud.

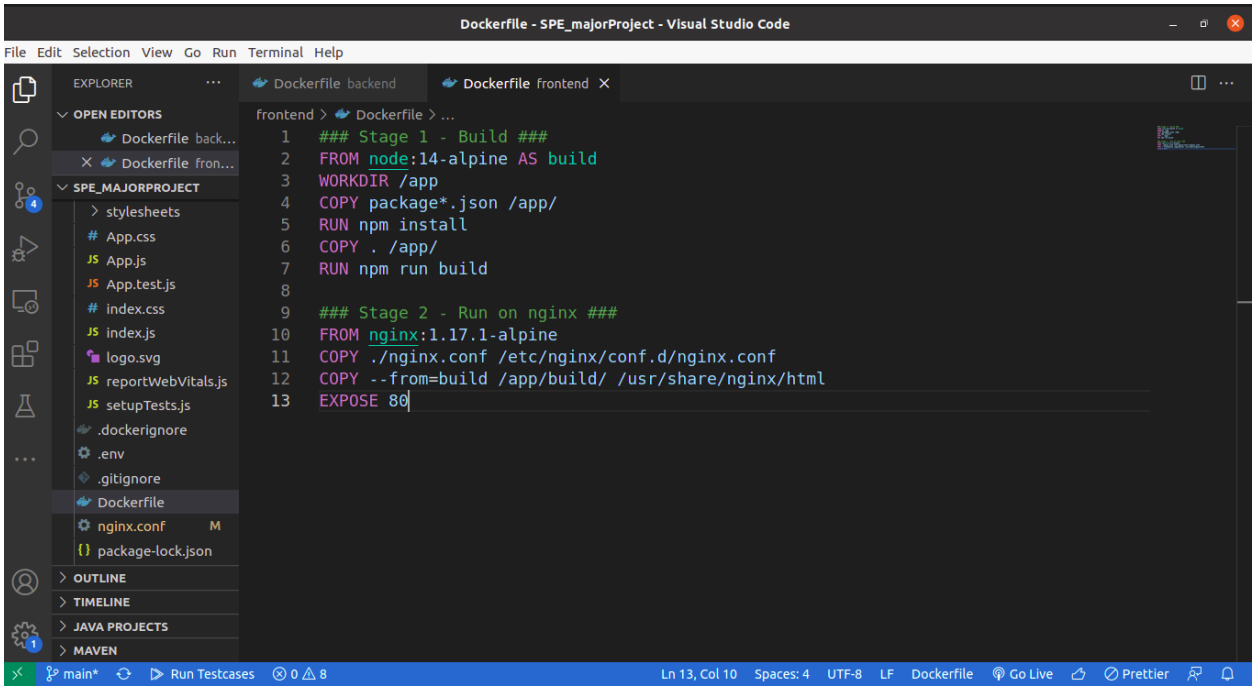
For containerization we are using Docker and for remote management of docker images we are using Docker Hub.

In our project we are using dockerfiles to bundle the backend and frontend as separate docker images. The dockerfile - /backend/Dockerfile will build the backend docker image pushed as karthikhegde2022/placement-manager-backend image (https://hub.docker.com/repository/docker/karthikhegde2022/placement-manager-backend) The dockerfile - /frontend/Dockerfile will build the frontend docker image pushed as karthikhegde2022/placement-manager-frontend image (https://hub.docker.com/repository/docker/karthikhegde2022/placement-manager-frontend)

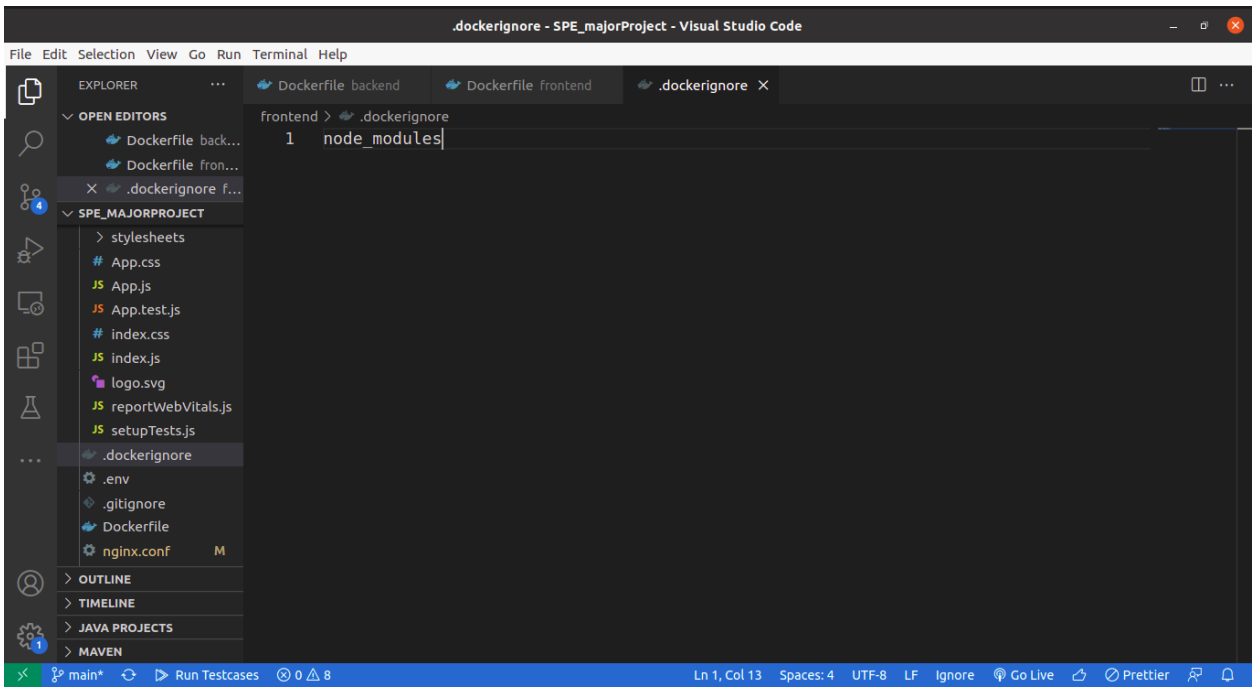


Backend dockerfile

We are building from openjdk:11 image and our working directory is set to /app
The jar file is moved to the appropriate place in the working directory and port 8082 is exposed for communication. As an entry point command the jar file is executed.

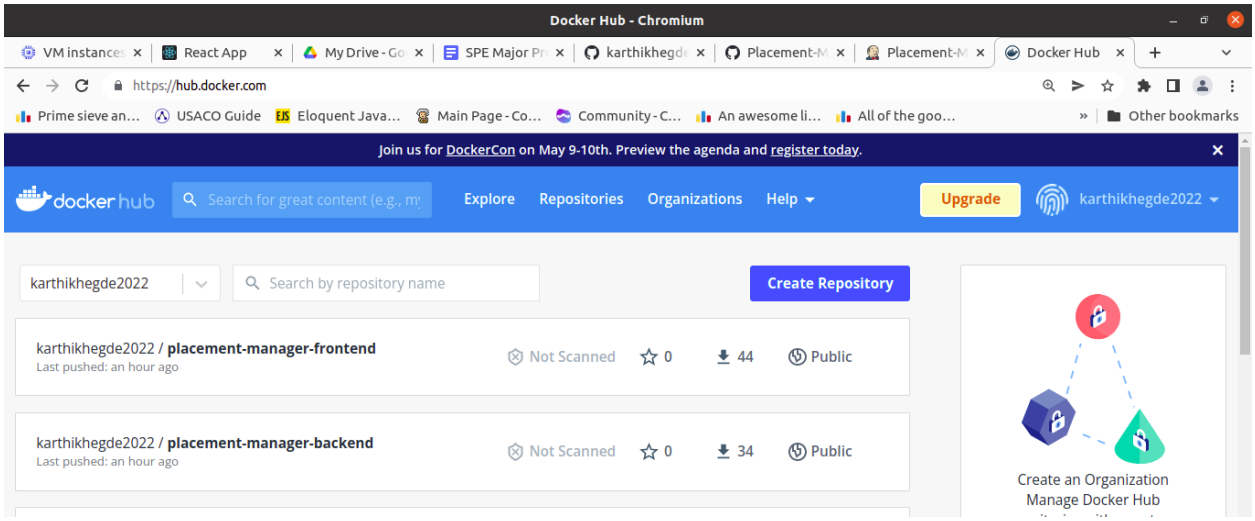


Frontend dockerfile



Dockerignore file to ignore copying of node_modules to work directory

This is a two stage docker file and the first stage is regarding building the react deployment and second is to setup nginx and move the nginx .conf file. '.dockerignore' here ignores the node_modules in moving the files to the working directory in the image.



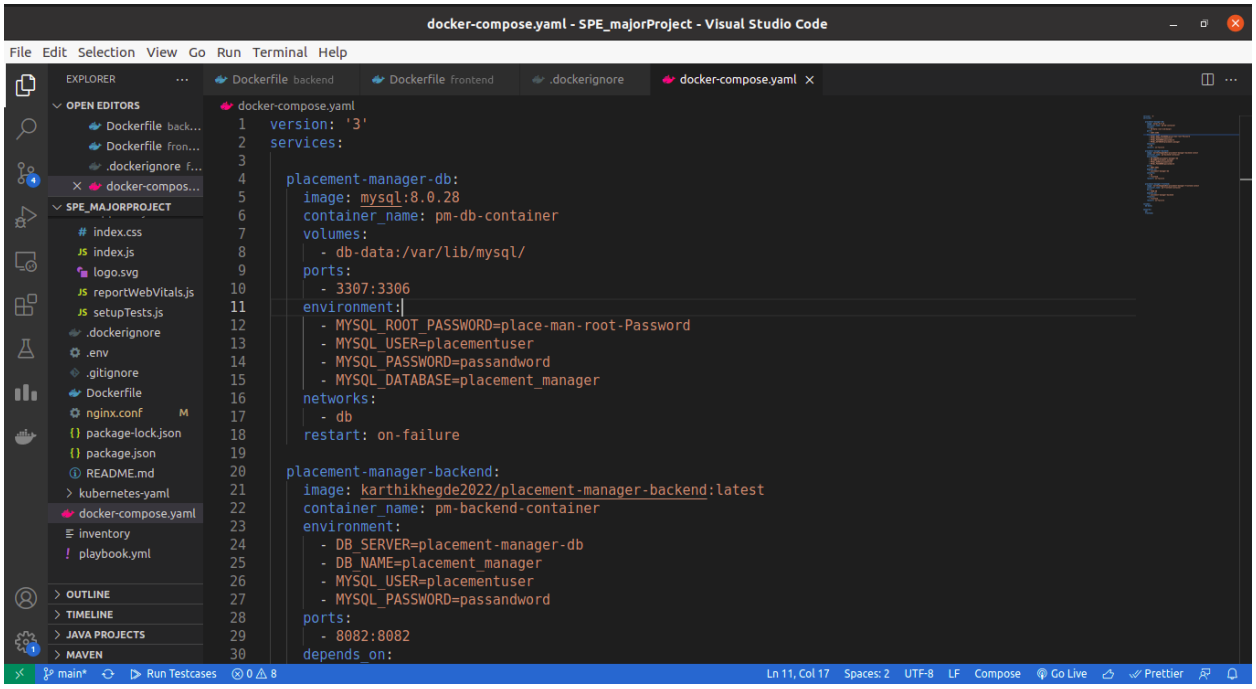
Docker hub repo

Creation of Containers - Docker Compose

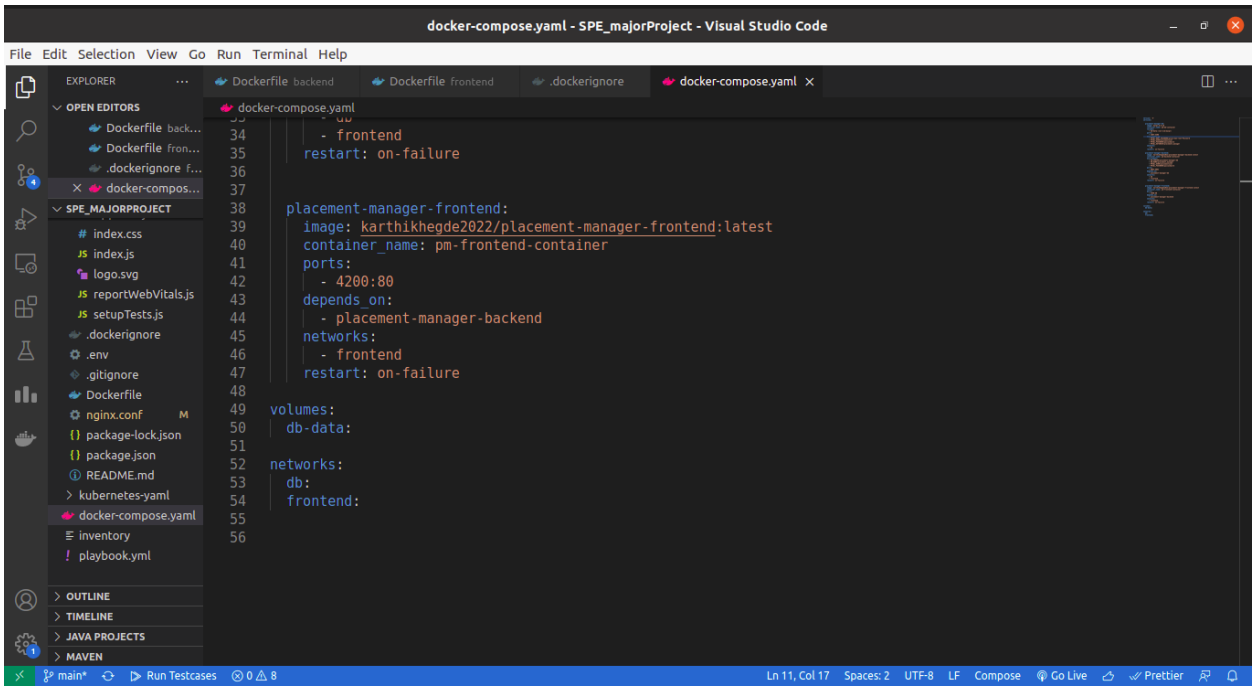
Docker compose is a tool for defining and running multi-container Docker applications. Docker-compose lets one use a single YAML file to configure the application's services. Then, with a single command, we can create and start all the services from the configuration. [3]

Our main aim in this project is to use ansible to copy this docker-compose YAML file to the remote machine and run the "docker-compose up" command to spin the three containers corresponding to three images - frontend, backend, mysql image and to allow networking between them.

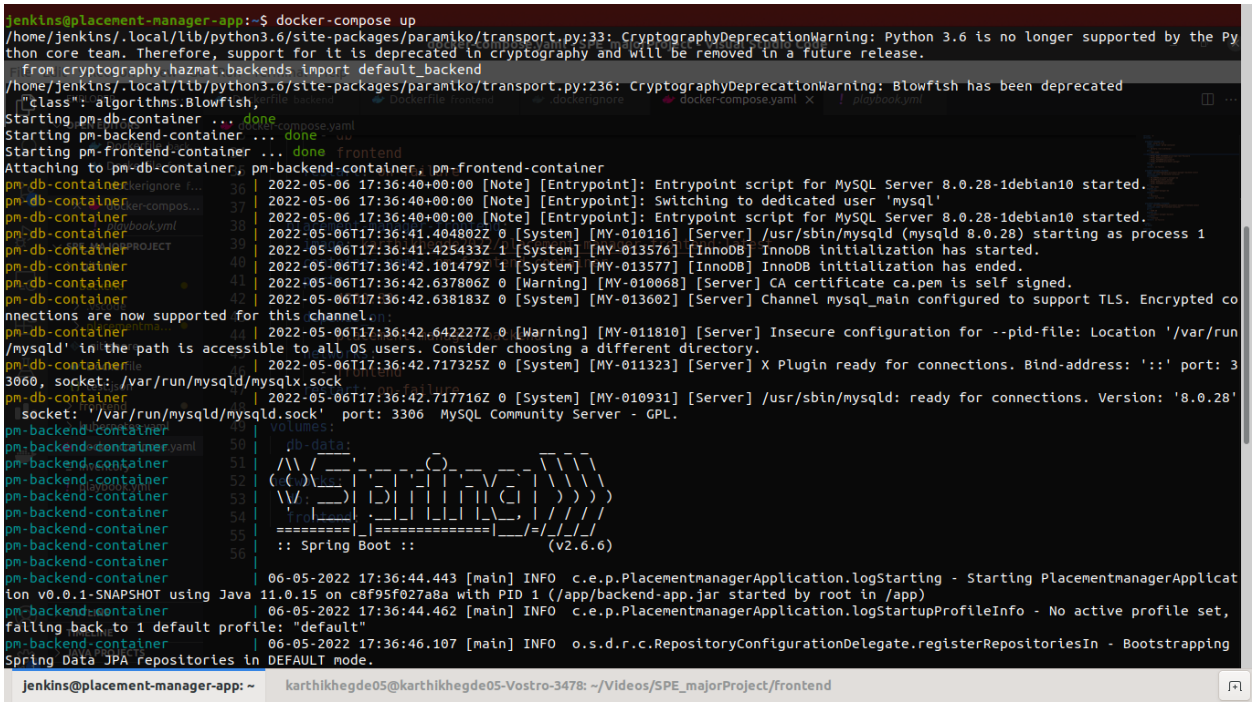
→ upload image of docker-compose YAML file and if possible show the commands to run



Docker compose file



Docker compose file continued



Docker-compose up command execution on gcp vm instance through ssh

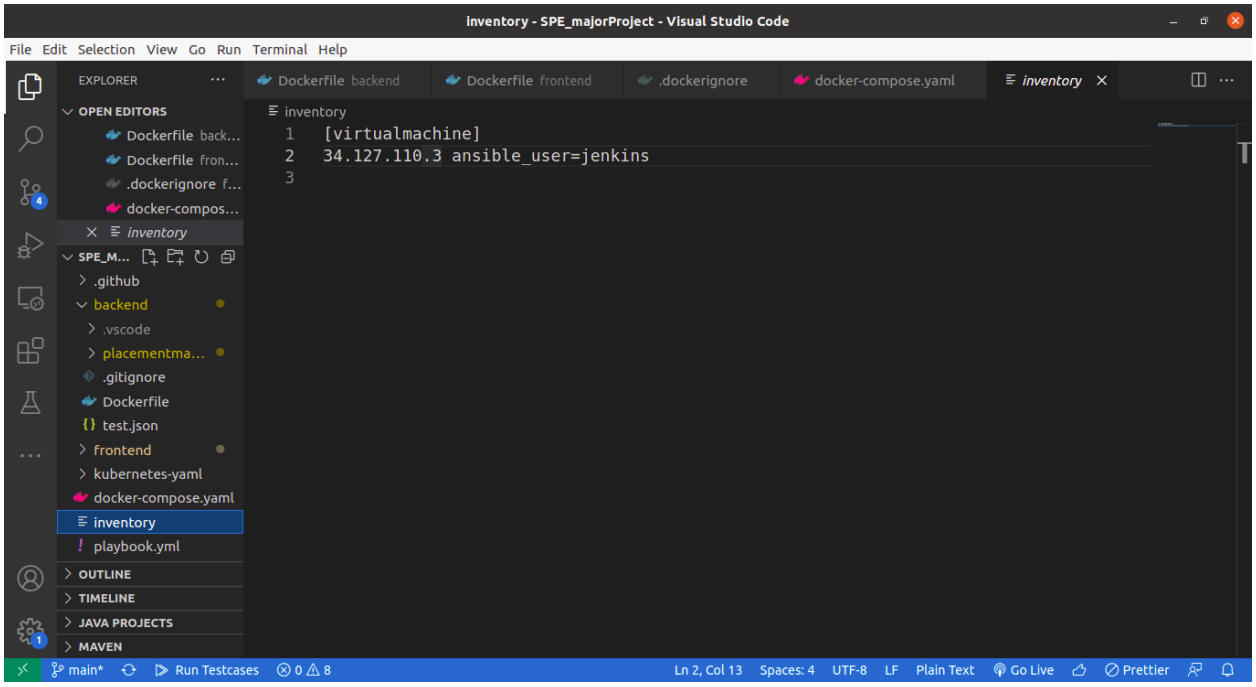
Continuous Delivery and Deployment - Ansible

While continuous integration is the process of integrating code to the repository and testing, continuous delivery ensures that the incremental features that are integrated are ready to be delivered to the remote machines that eventually gets served to the end users.

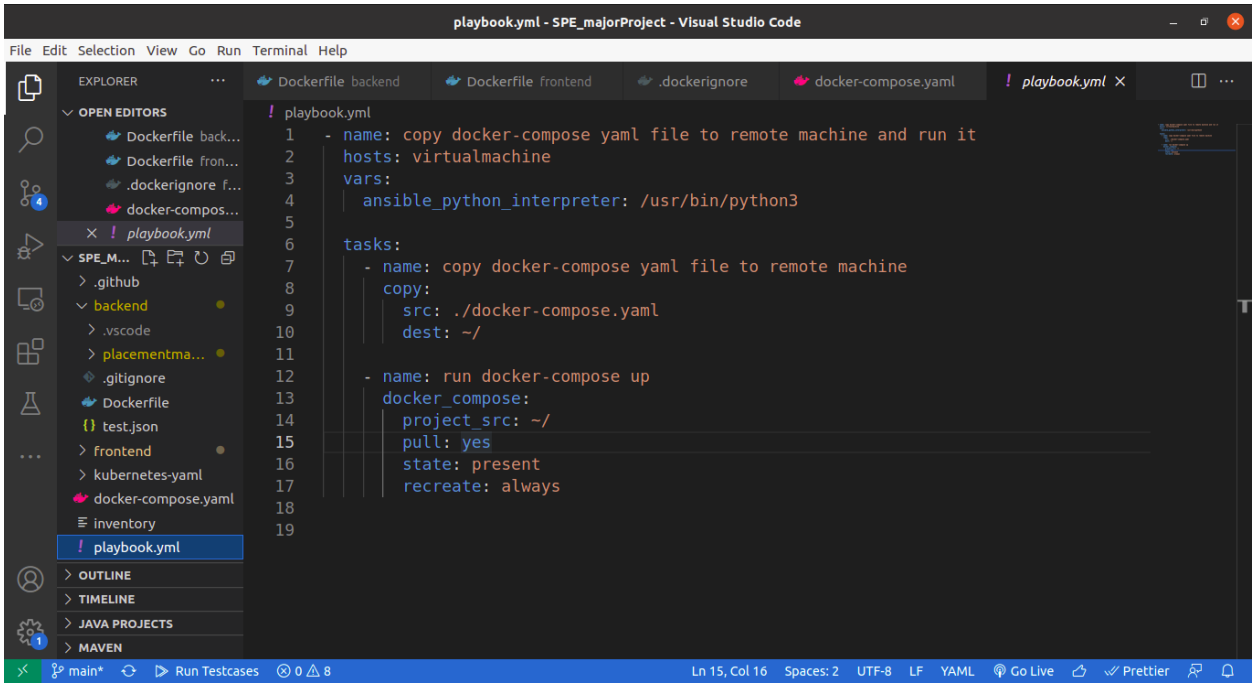
Continuous deployment on the other hand is the process through which this deployment of incremental features of the software to the end users is completely automated.

Ansible here is an IT automation tool which primarily eases the process of continuous delivery of the latest Docker images to the remote machines (specified by IP addresses) and continuous deployment of the docker images to running containers.

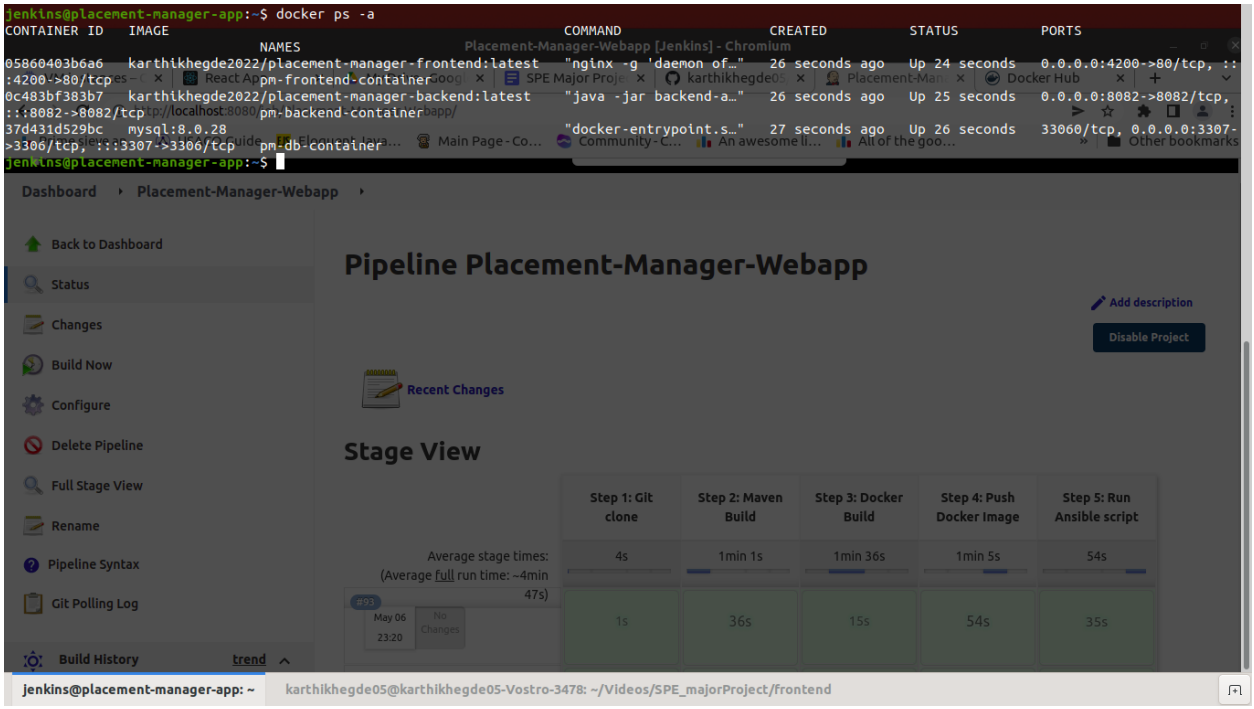
In our case, Ansible uses python3 and ssh to deliver the docker-compose.yaml file to the remote machine (specified by IP address) and run the “docker-compose up” command to run the three Docker containers corresponding to frontend, backend and mysql database.



Ansible inventory file



Ansible playbook file



Result of ansible step in jenkins pipeline. Docker containers running in the gcp vm instance

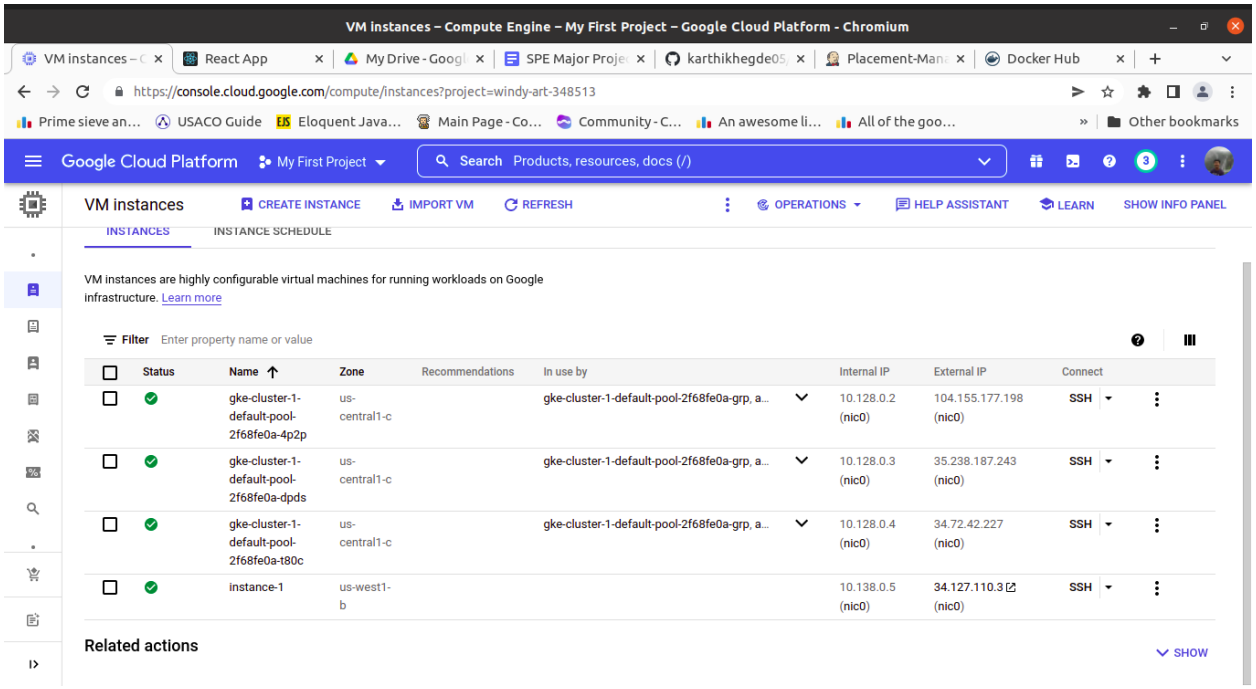
Deployment Platform - Google Cloud Platform

"The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers. Cloud servers are located in data centers all over the world. By using cloud computing, users and companies do not have to manage physical servers themselves or run software applications on their own machines.[1]

Google cloud platform, offered by Google, is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google search, Gmail, Google drive, and youtube.[2]

In our case, we have spun a VM instance on the Google cloud platform where the ansible delivers the docker-compose.yaml file and runs the containers.

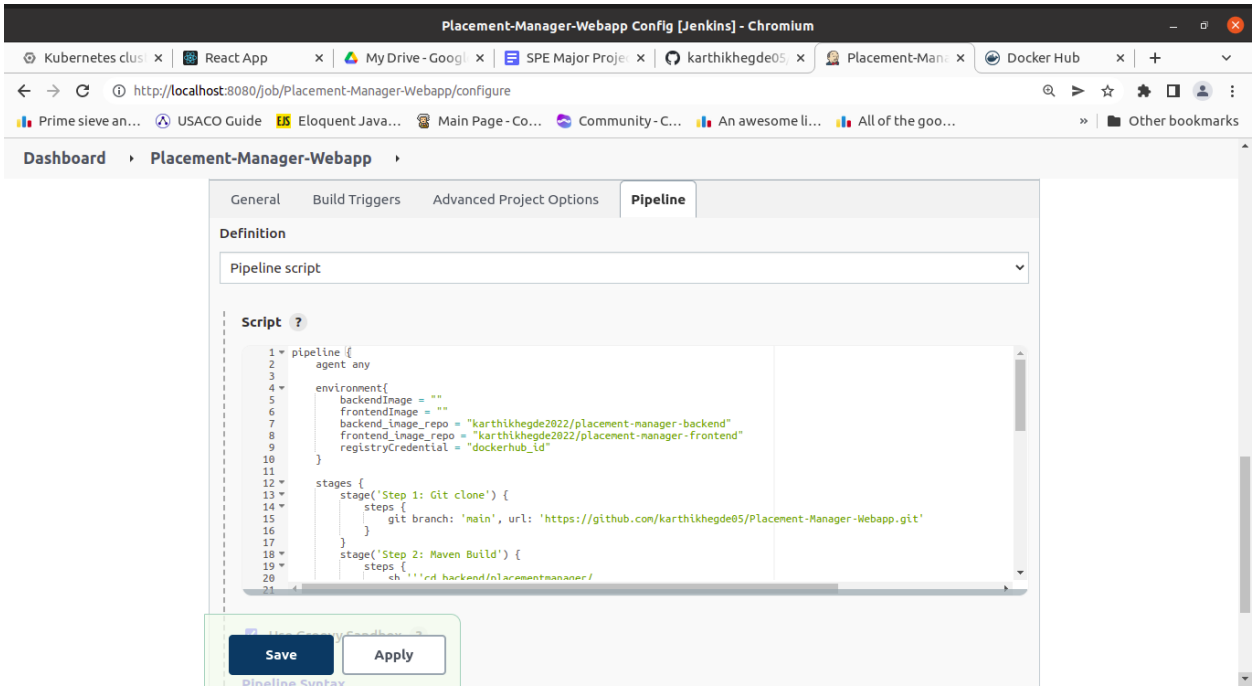
Through the external ip provided by the GCP, and port 4200, one can use the placement-manager website.



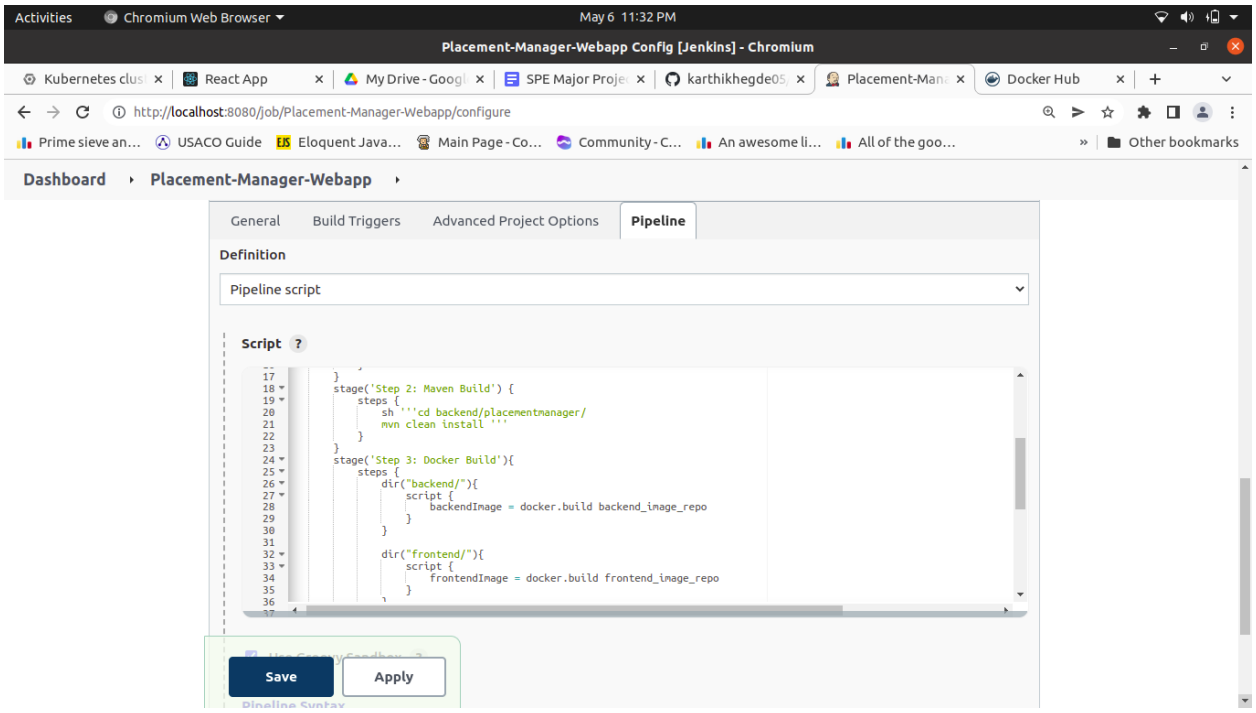
GCP vm instance (instance -1)

Automating Continuous Integration and Continuous Deployment - Jenkins

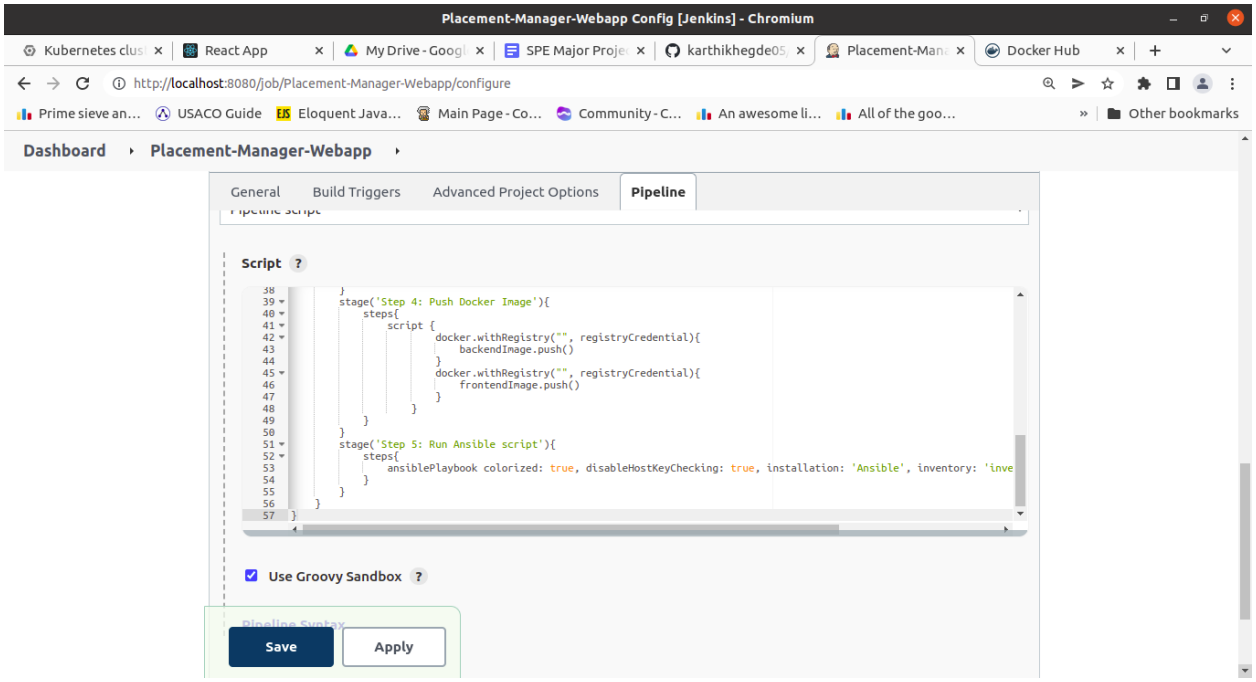
Jenkins is a popular tool for automating the process of building, testing, delivering and deploying software. We have used a pipeline script in Jenkins to automate the continuous integration of maven build and test, docker image creation and image pushing to the docker hub, and to automate the continuous delivery and deployment process carried on by the ansible playbook.



Jenkins pipeline



Jenkins pipeline continued



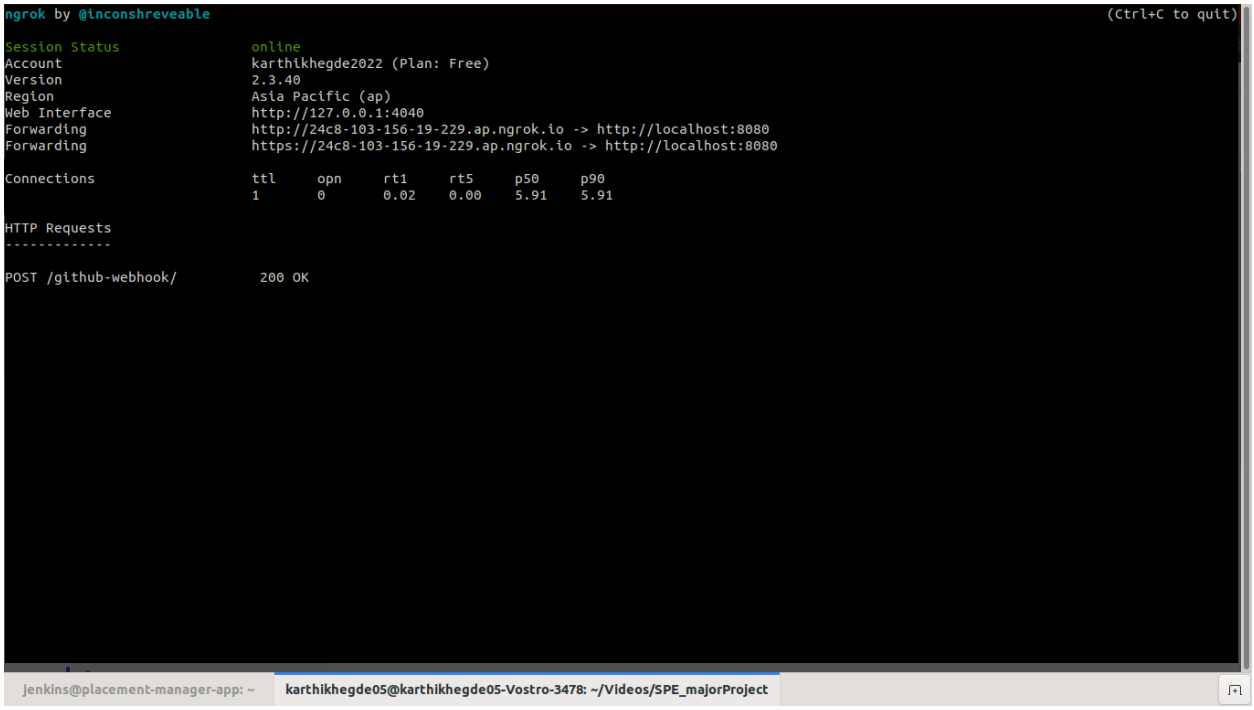
Jenkins pipeline continued

We are using GitHub hook trigger for GITScm polling as a triggering option for the pipeline. Since the jenkins server originally runs on localhost, we are using ngrok to expose the localhost through a tunnel. With this the github repo can send the request to the ngrok exposed url whenever a commit is pushed to the github repo.

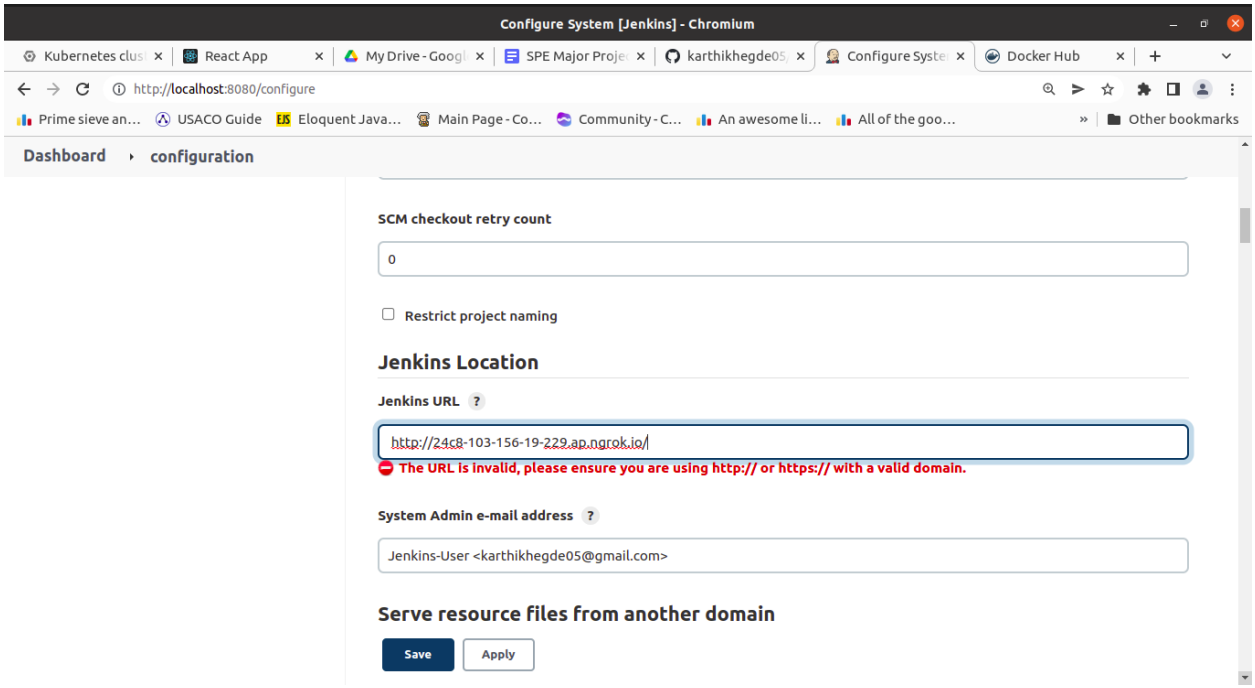
Github hook trigger necessary configuration and setup

- Go to github → go to developer settings → personal access token → create one token and save it as this is required for further steps
- Go to github repo → go to settings → go to webhooks → add a webhook by filling the “secret” with the generated personal access token, filling the “payload url” with the ngrok url
- Go to Jenkins → manage jenkins → configure system → enter the jenkins url with the ngrok url
- In the configure system (Jenkins) → go to Github add credentials → credentials will be a secret containing the generated persona access token (1st step)

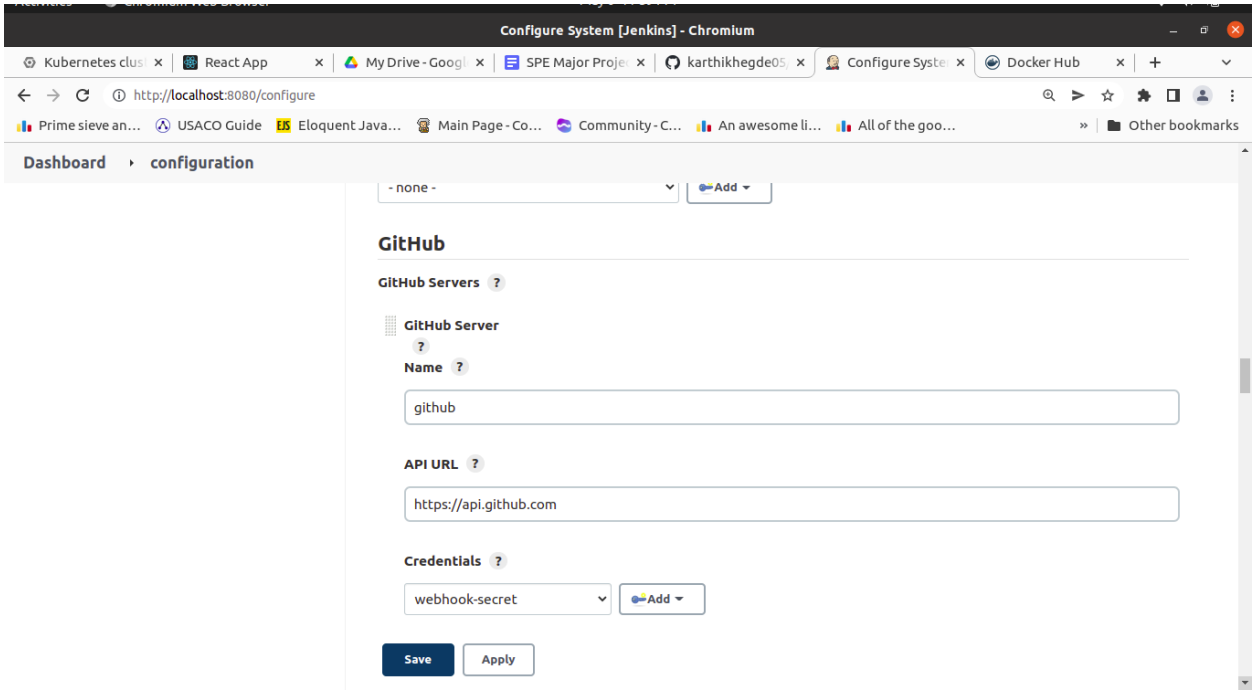
→ upload images involving setting up of ngrok, configuring personal access token, webhooks and jenkins configure systemsGitHub hook trigger for GITScm polling



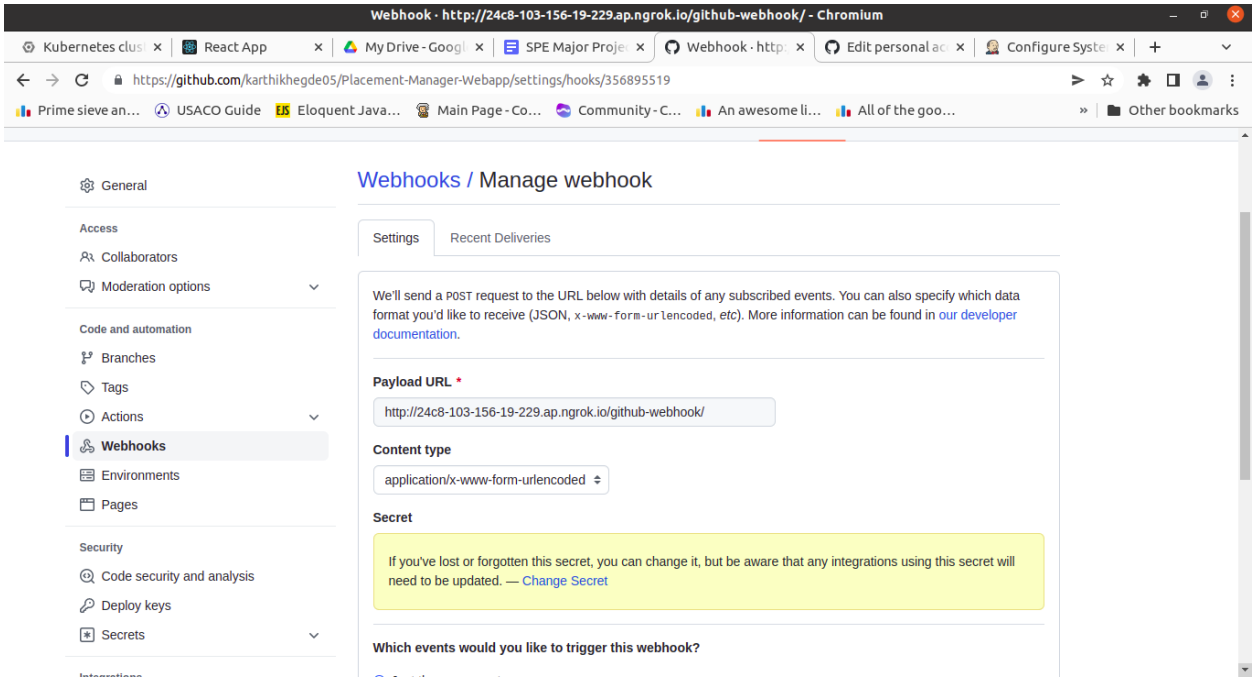
Ngrok exposing 8080 port



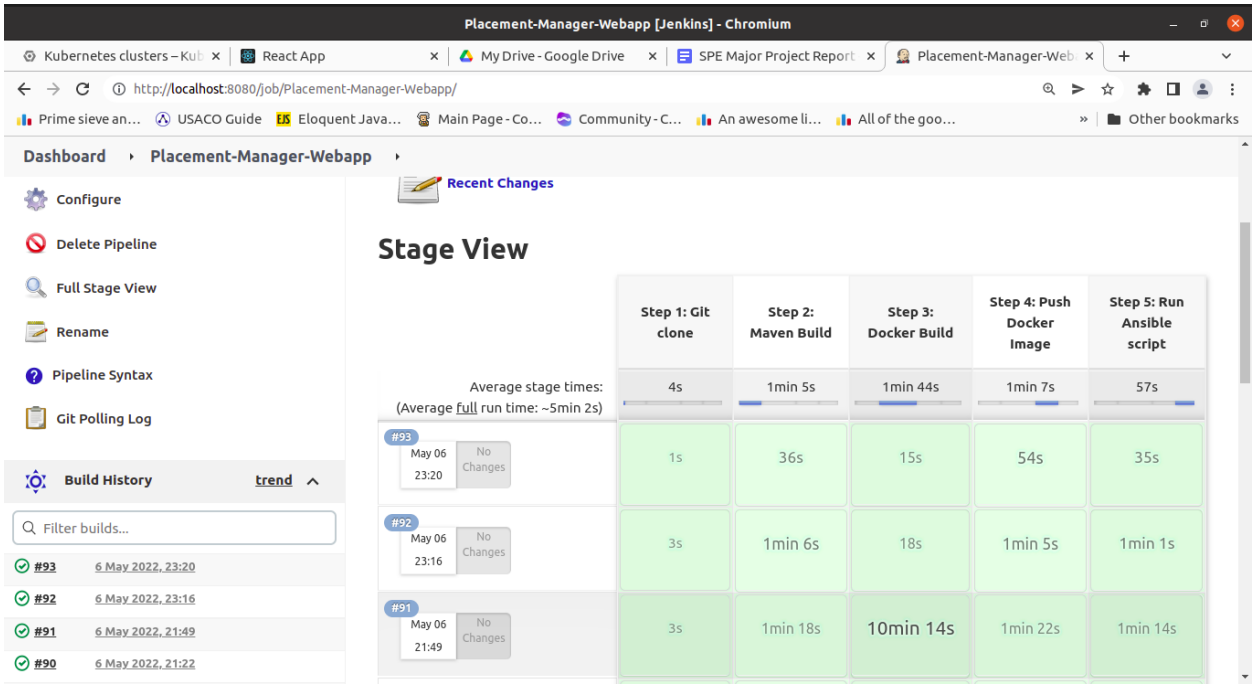
System configuration jenkins



Adding personal access token as secret



Webhook configuration in github repo settings



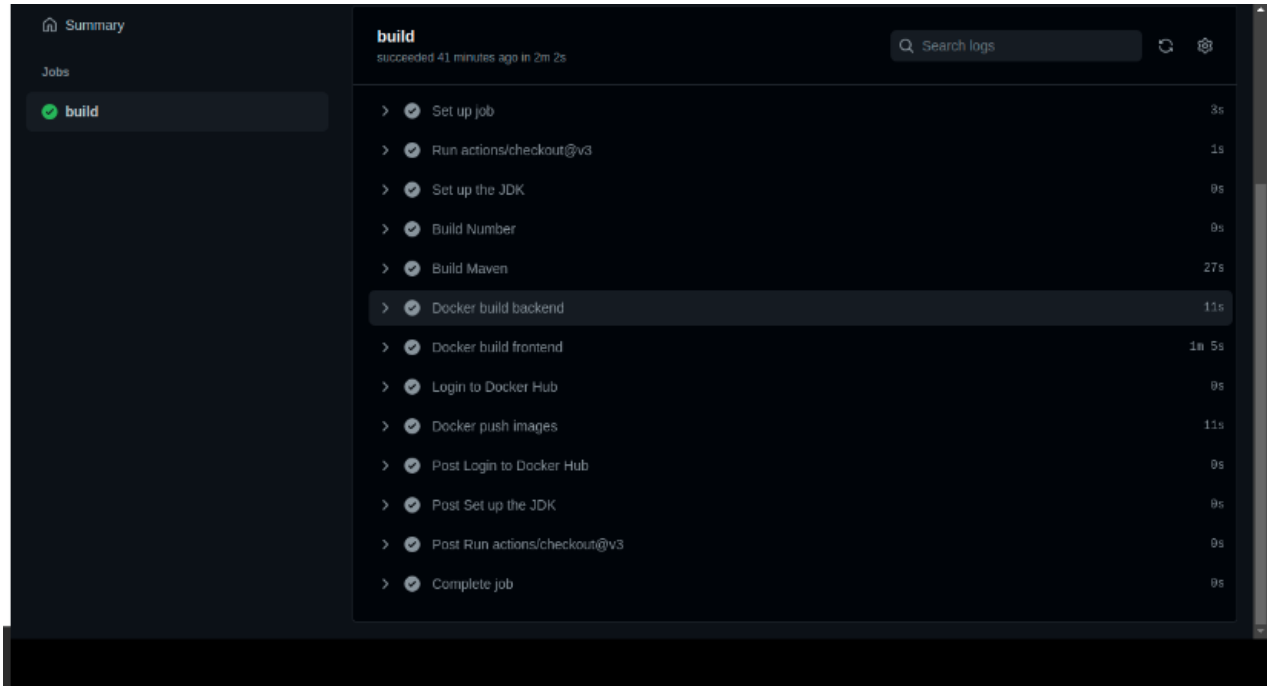
Jenkins build execution

Github Actions

We also explored github actions for creating a CI pipeline. The workflow will be triggered automatically when there is a push to the branch. It can also be triggered manually by the user. It includes the following steps:

1) Set up a JDK
2) Generate build number
3) Build the backend server using maven (mvn clean install)
4) Docker build backend image
5) Docker build frontend image
6) Authorize access to dockerhub
7) Push backend image to dockerhub (https://hub.docker.com/repository/docker/jishnuvk/spe-main-project)
8) Push frontend image to dockerhub

The workflow file is available in the /.github/workflows/main.yml of the actions branch.



Successful workflow

Since the workflow file is publicly available on the github repo, it is not safe to write usernames and passwords directly as plain text. As a more secure alternative, an access token was created from docker hub. The Dockerhub username and password were stored in github secrets which are visible only to the repo owner.

Deployment on Kubernetes Cluster

With the popularity of Docker and the concept of microservices, the use of containers have increased exponentially. For a large-scale application, multiple containers are used to achieve high availability, failover mechanisms, scalability, high performance, etc. In such cases, the number of containers deployed is huge, and depends on the requirements, and in some cases the containers are deployed on multiple hosts.

Kubernetes emerged as the open source tool to orchestrate containers. Kubernetes is a popular container orchestration tool with many features to help us manage docker containers.

In this project, we have also tried using Google Cloud's Kubernetes service where this service creates a cluster and we can configure different service and deployment yaml files to spin up k8 pods.

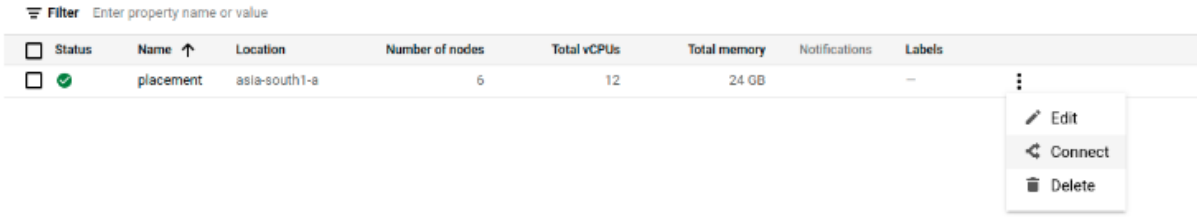
Setup and configuration:

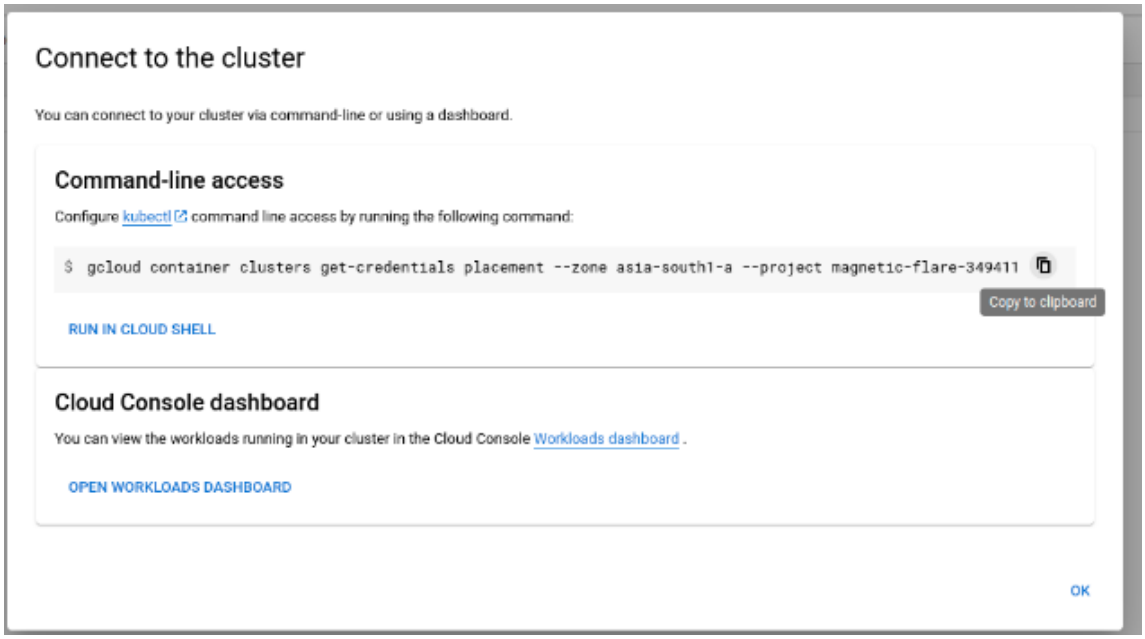
Creating a cluster on GKE:

1) You need a Google cloud platform account for this. There is a 90 day free trial, but credit card information is required.
2) From the google cloud platform page, go to console.
3) Create a project
4) Go to kubernetes Engine from the sidebar.
5) Click on create cluster
6) Choose GKE-standard.
7) Give a name, and chose a region of your preference.
8) Wait for a few minutes for the cluster to wind up.

Connecting to the cluster from command line:

1) Download kubectl. (Available with apt-get)
2) From the GKE GUI get the connection command as shown below.



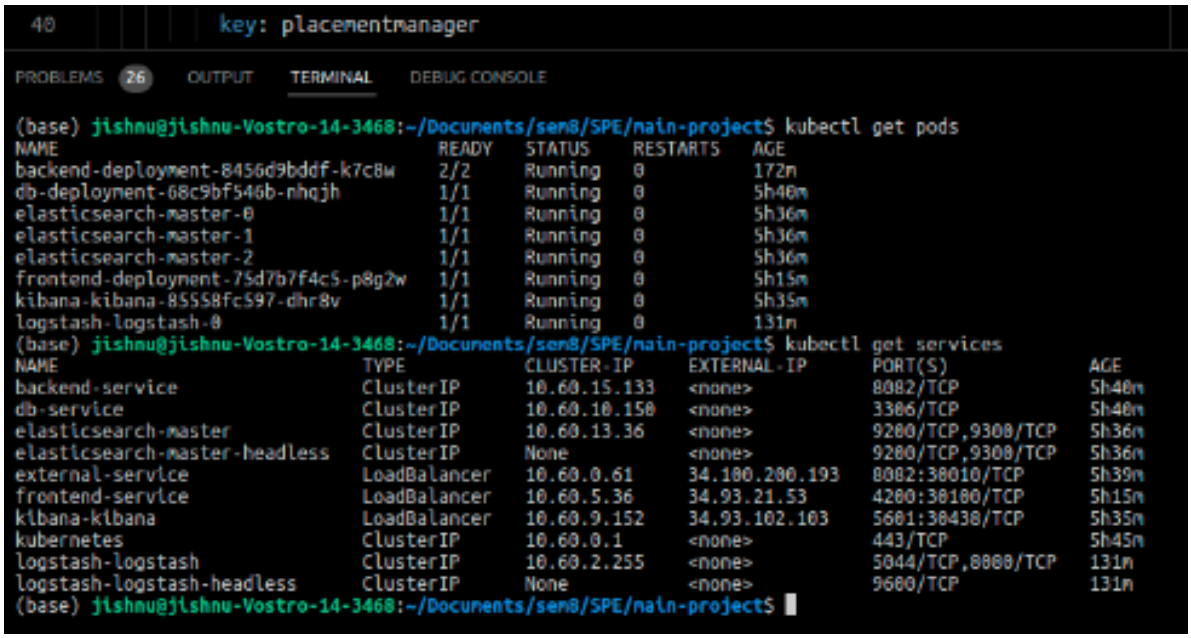


3) Paste the command in the local terminal and you will be connected.
4) To verify, run “kubectl get all” to display the details of the cluster.

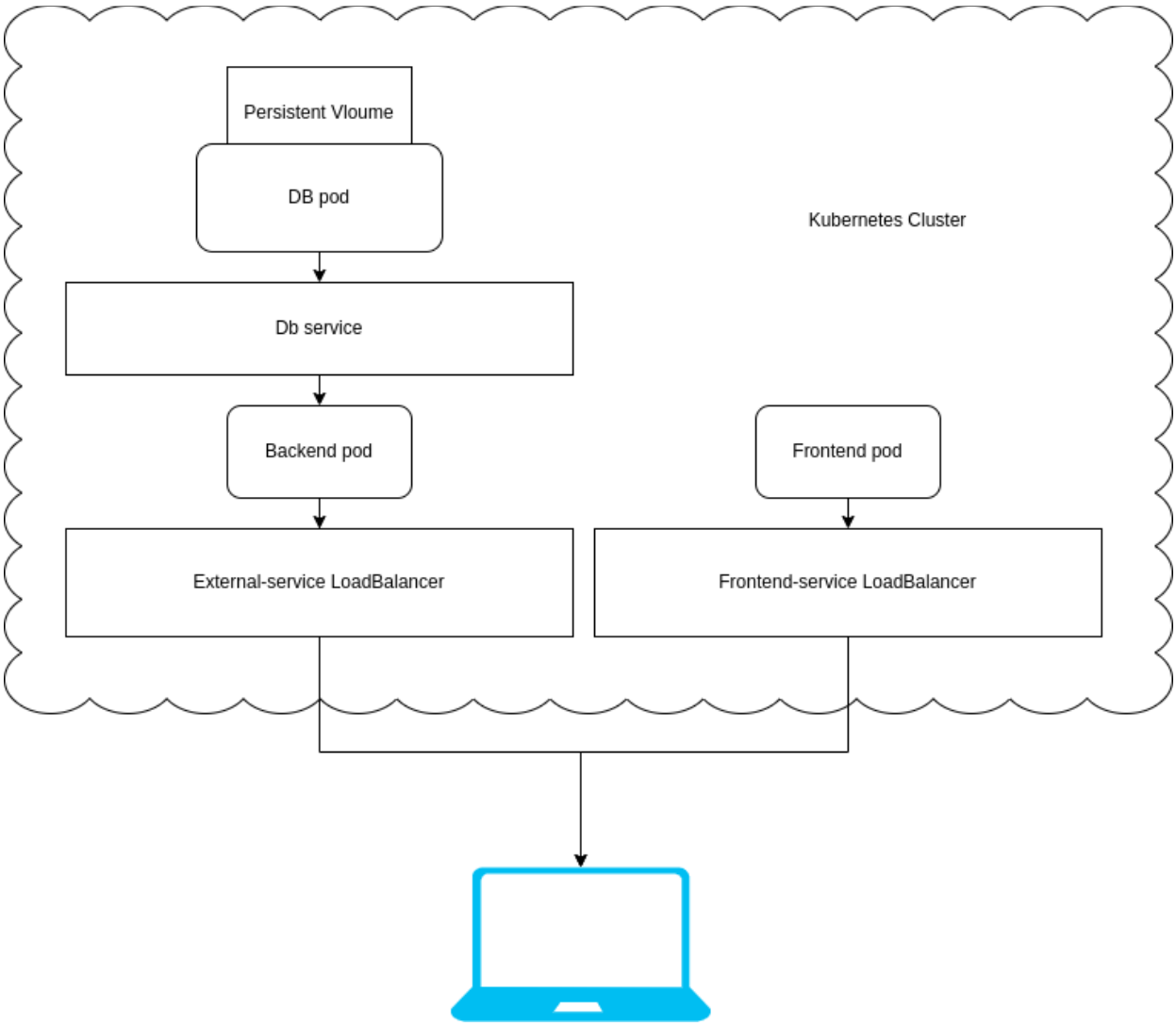
Deploying to the cluster

1) To deploy docker containers on the cluster, we need to write configuration yaml files.
2) All files we used are available in the github repo at /kubernetes-yaml/

3) To run these config files, the command “kubectl apply -f <yaml-file>” is used.
4) After running the files, many pods and services will be created.
5) To view pods and services, we can use the commands “kubectl get pods” and “kubectl get services”.



In the image, the important core pods are backend-deployment, db-deployment and frontend-deployment. And the important core services are frontend-service, external-service and db-service. The rough architecture diagram is given below.



Continuous Monitoring - ELK Stack

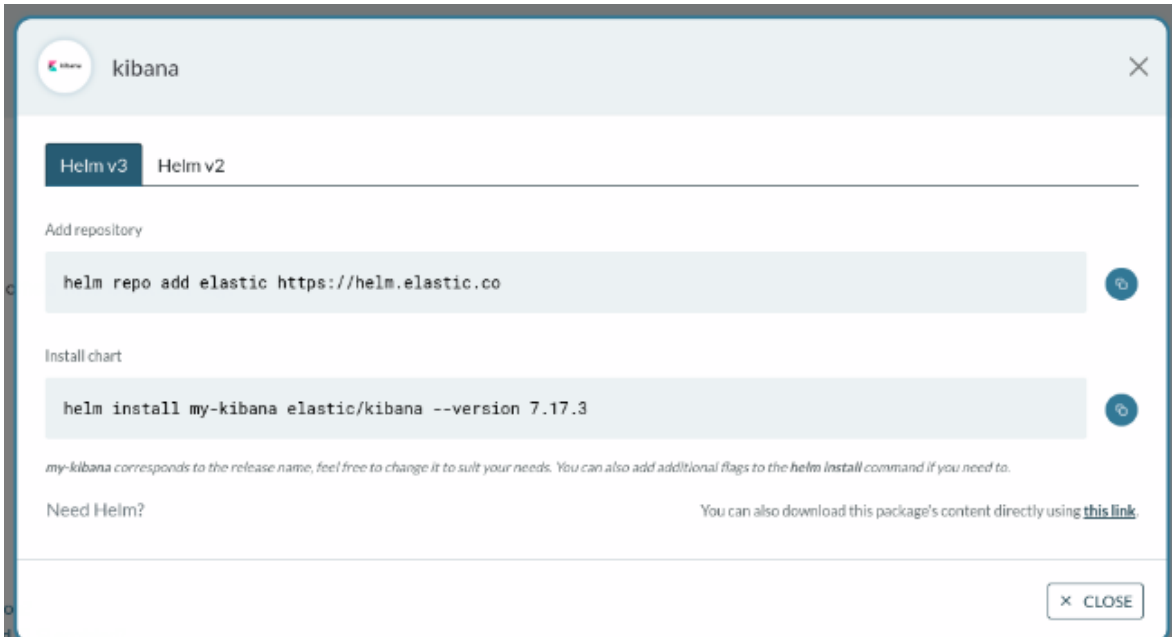
ELK consists of three open source softwares:

- Elasticsearch: It is a search and analytics engine

- Logstash: It is a server side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a “Stash” like Elasticsearch
- Kibana: It lets users visualize data with charts and graphs made from elasticsearch

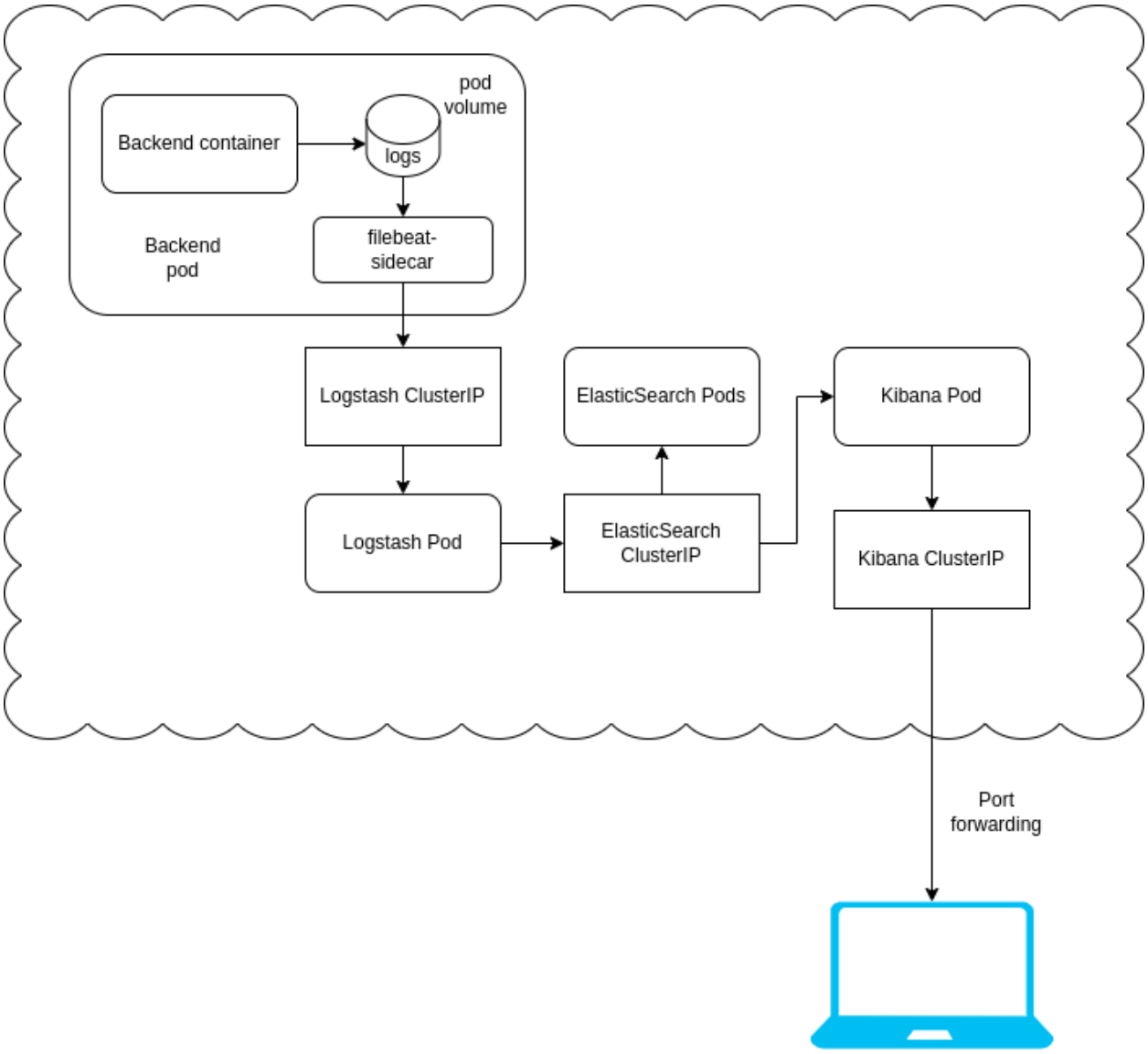
This is used for continuous monitoring of the deployment and creating visualizations for better understanding of the working and business profits from the deployment.

We deployed an ELK stack directly onto our kubernetes cluster and used it for monitoring. For this, we used Helm, a package manager for kubernetes. It can be installed on the terminal using snap package manager. It automatically uses the kubectl configurations. We used Helm to deploy the elasticsearch, logstash and kibana pods into the cluster. We get the helm packages from artifacthub.io. Using the search, we can find official packages from elastic.co. We can download packages directly to our computer by clicking the install button on the right and then using the link on the bottom right(show in image).

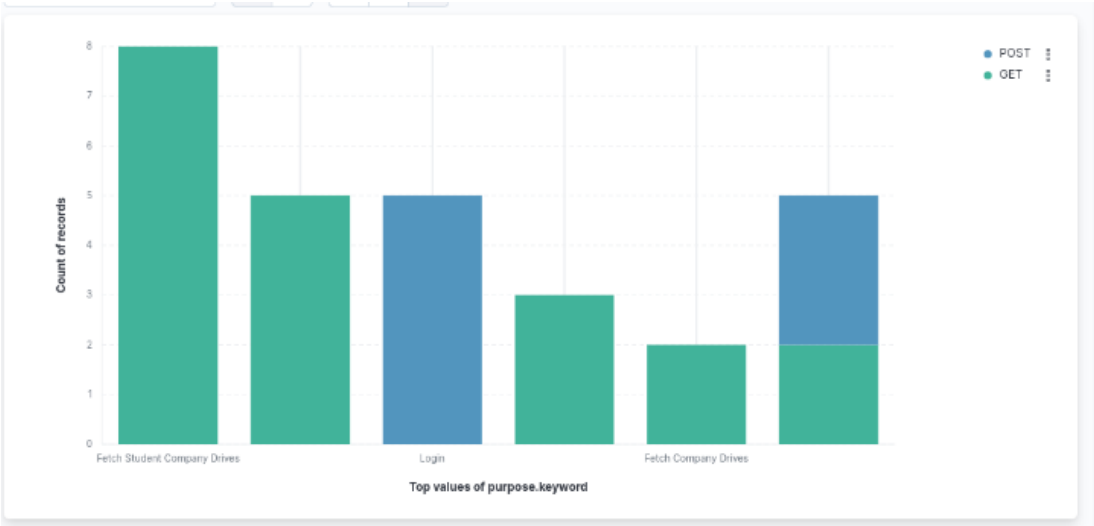


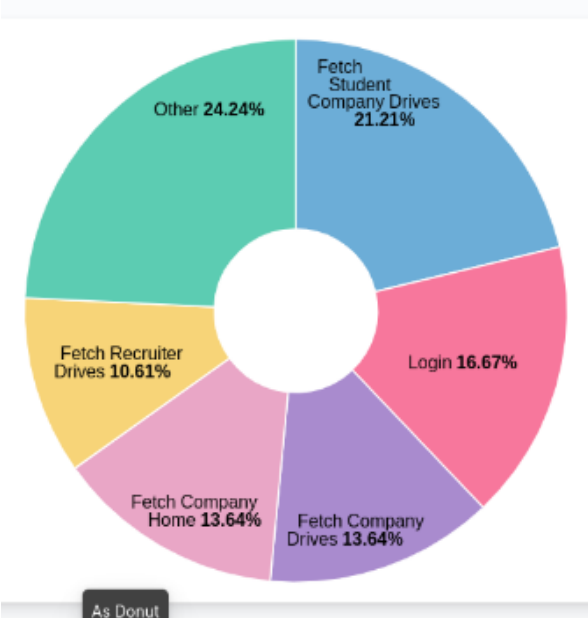
To configure helm packages, we modify the values.yaml file. We can set the resources it should take and the config files for the software itself. For example. The grok pattern for logstash can be found in the github repo at /ELK/logstash/values.yaml. This grok pattern is used to create structured data out of the log files.

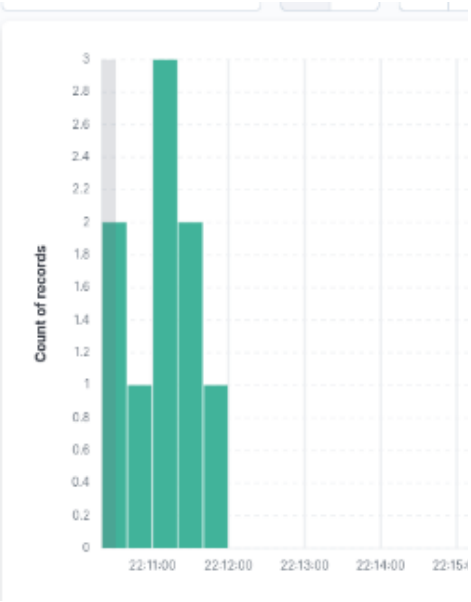
We also need filebeat to read log files from within containers and send it to logstash. To achieve this, the sidecar design pattern was used. In the sidecar pattern, the filebeat container is deployed in the same pod as the backend as a “sidecar”. Using pod volumes, the filebeat container is able to access the log files in the backend container. The rough architecture for this is given below.

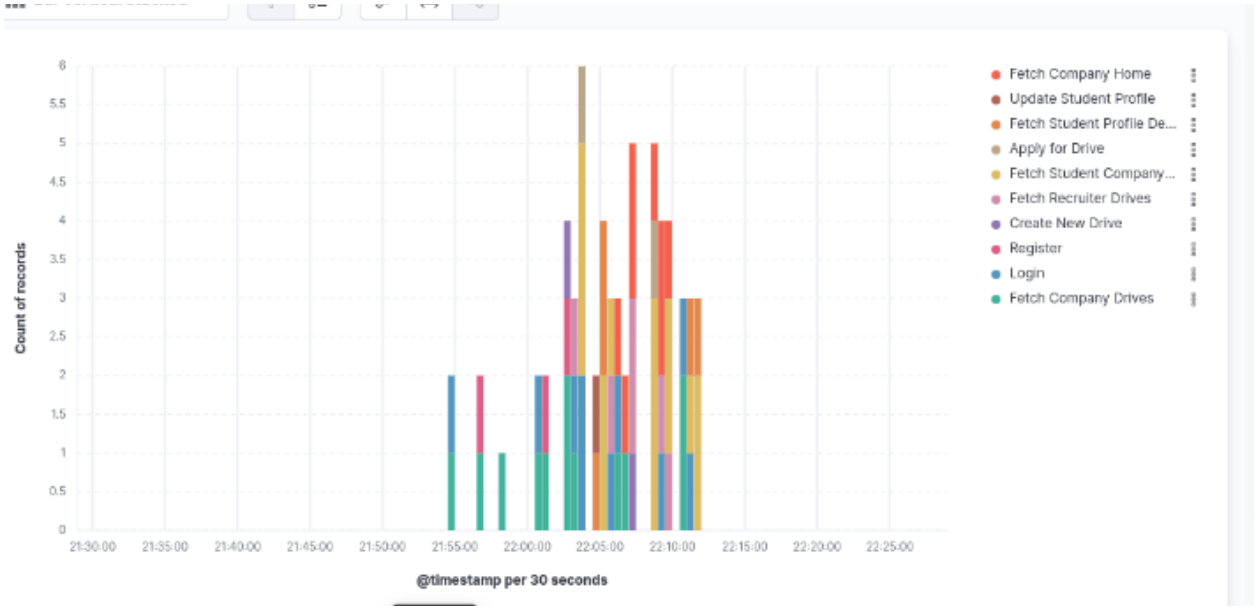


To connect to the kibana GUI, we forwarded the kibana service port to localhost. Then we use localhost to connect to kibana, view the log files and create visualizations for them.

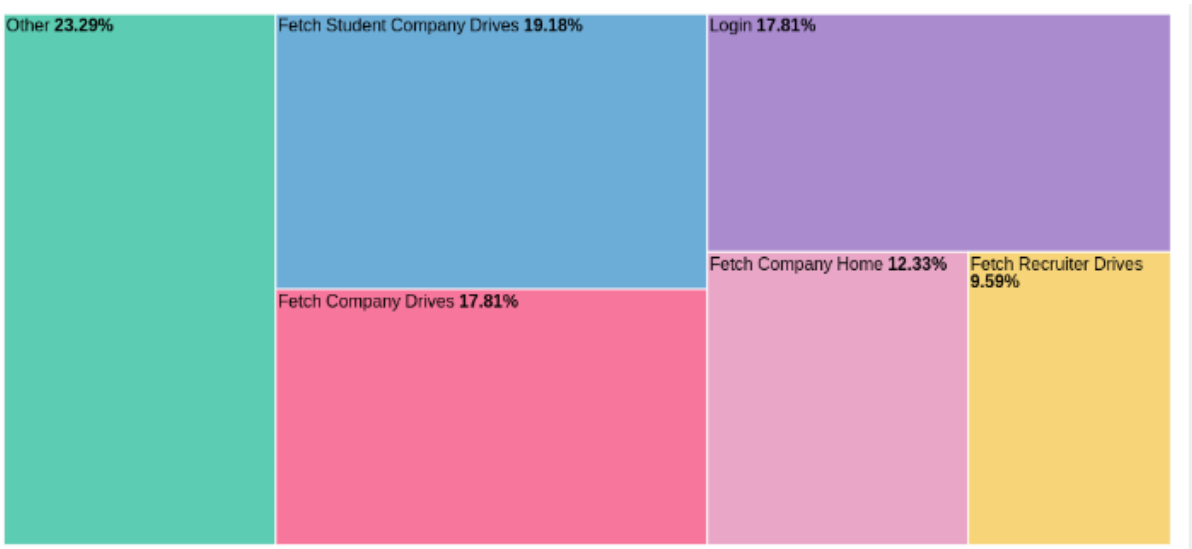

POST
GET
Count of records
0
1
2
3
4
5
6
7
8
Fetch Student Company Drives
Login
Fetch Company Drives
Top values of purpose.keyword

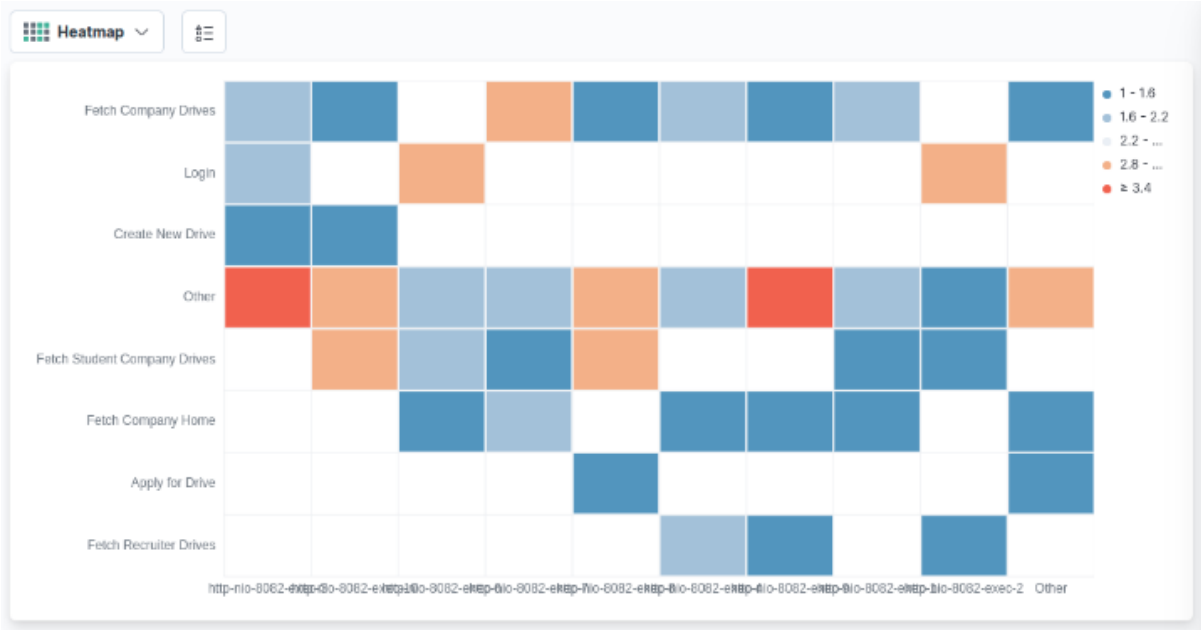

Fetch Student Company Drives 21.21%
Login 16.67%
Fetch Company Drives 13.64%
Fetch Company Home 13.64%
Fetch Recruiter Drives 10.61%
Other 24.24%
As Donut


Count of records
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8
3
22:11:00
22:12:00
22:13:00
22:14:00


Fetch Company Home
Update Student Profile
Fetch Student Profile De...
Apply for Drive
Fetch Student Company...
Fetch Recruiter Drives
Create New Drive
Register
Login
Fetch Company Drives
Count of records
@timestamp per 30 seconds


POST
GET
Count of records
@timestamp per 30 seconds


Other 23.29%
Fetch Student Company Drives 19.18%
Login 17.81%
Fetch Company Drives 17.81%
Fetch Company Home 12.33%
Fetch Recruiter Drives 9.59%


Heatmap
Fetch Company Drives
Login
Create New Drive
Other
Fetch Student Company Drives
Fetch Company Home
Apply for Drive
Fetch Recruiter Drives
1 - 1.6
1.6 - 2.2
2.2 - ...
2.8 - ...
≥ 3.4
Other

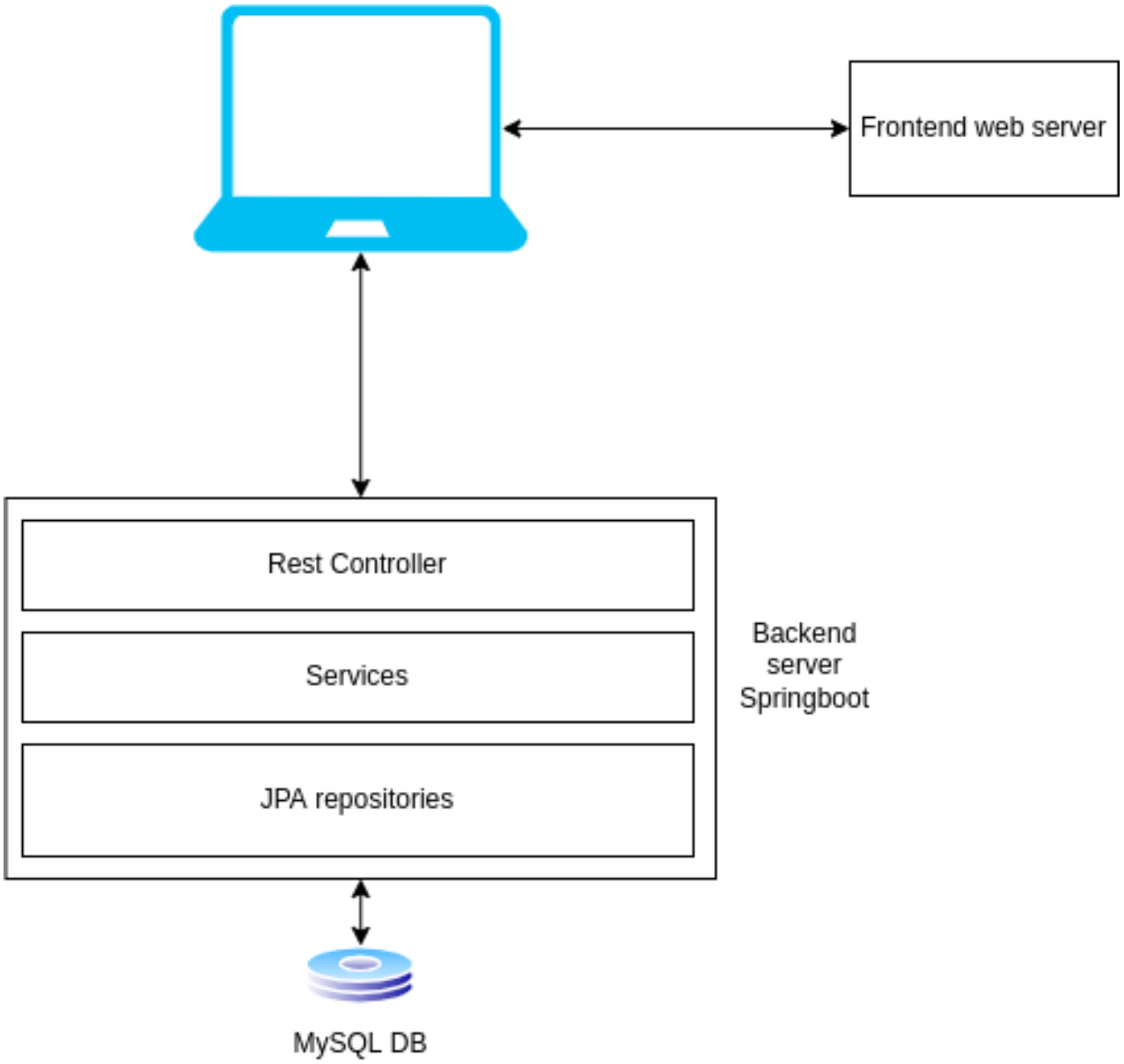
PROJECT DESIGN SPECIFICATIONS

This section describes the requirements gathered before starting the project, architecture design diagram of the application, relational database design diagram.

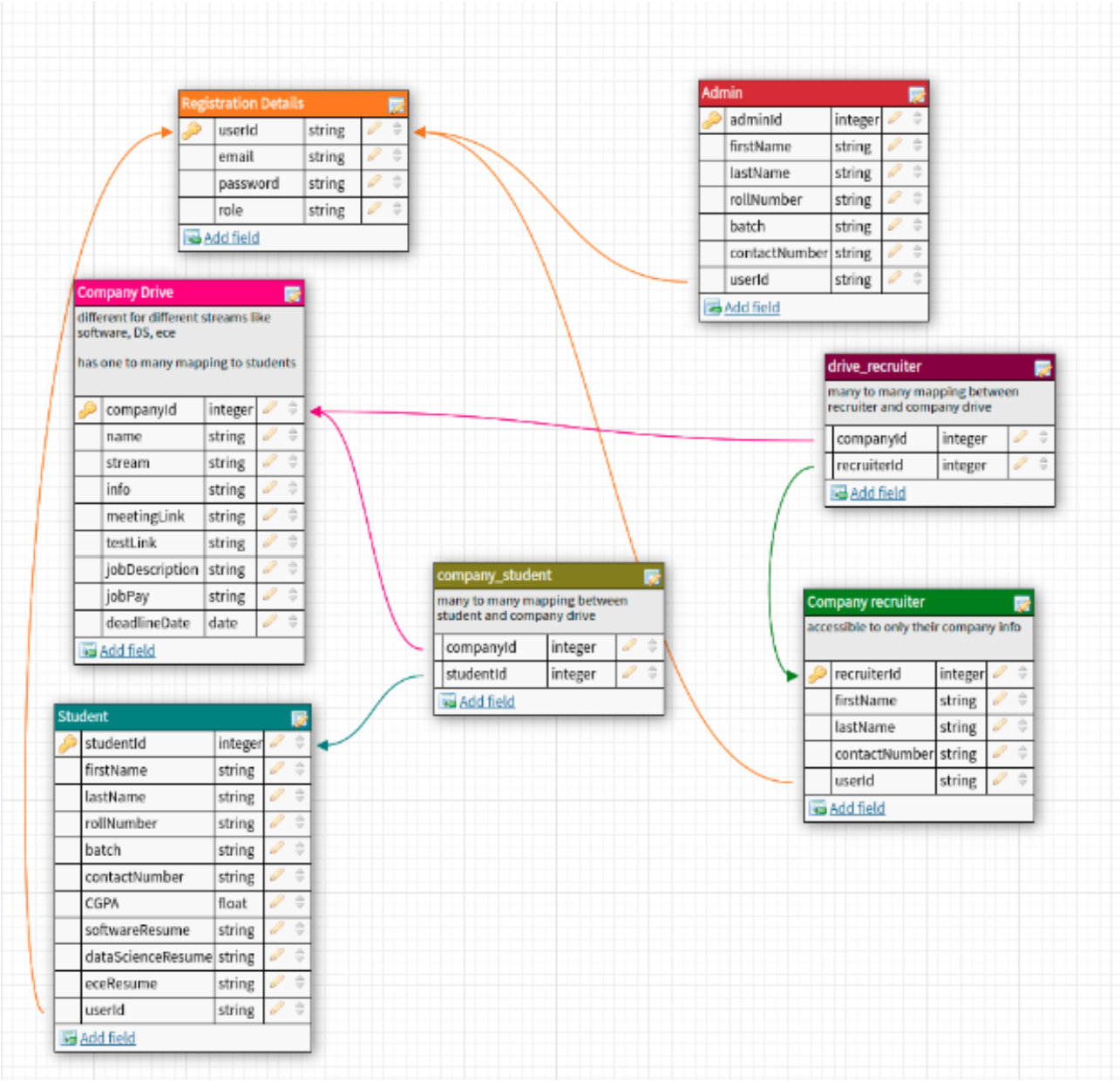
Functional requirements

FR no	Description	Status
ADMIN		
A0	Admin should be able to create accounts	Admins can create admins, students and recruiters
A1	Admin should be able to create placement drives	Complete
A2	Admin should be able to customize drives	Can chose profile type, update any info related to drive
STUDENT		
S0	Student need a profile with details related to placements	Student has an editable profile
S1	Student can upload multiple resumes	Student can upload seperate resumes for software, ece and DS
S2	Student is notified of new drives	New drives appear on the home page
S3	Student can apply for a drive with a few clicks	Students can apply directly from the home screen
RECRUITER		
R0	Recruiters are able to communicate with students in their drive	Recruiters are given a page for thier drive. They have full control over this page. They can use this to share meeting links, info, test links etc.
R1	Recuiters can see students applying for their drive	Recruiters can see a list of students on thier page. They can view resumes from there with a click of a button
R2	Recruiters are able to reject students from their drive (in case of failng interview round, test etc)	Recuiter can reject students with a click of a button

Architecture diagram



Database diagram



API documentation

Base path: /api/

- POST: /register : register a new user into the system, add details into database
    - Request: rcvRegDto
    - Response: sndRegAckDto
- POST: /login: check login for username and password
    - Request: cvLgDto
    - Response: sndProfileDto
- GET: /drive-list/{objID}: Gets the list of all placement drives (for admin)
    - Response: sndDriveLstDto
- POST: /create-new-drive: create a new placement drive
    - Request: rcvCrtDriveDto
    - Response: sndCrtDriveAckDto
- POST: /delete-drive: delete a placement drive
    - Request: rcvDelDriveDto
    - Response: sndAckDto
- GET: /company-home/{compID}: get the homepage for company
    - Response: sndCompDetailsDto
- POST: /company-details-update: update company details in the db
    - Request: rcvCompDetailsDto
    - Response:sndAckDto
- POST: /company-reject-student: remove a student from a company's drive
    - Request: rcvStudentCompDto
    - Response: sndAckDto
- GET: /student-company-drives/{objId}: get the home page for a student
    - Response: sndStDriveLstDto
- POST: /student-apply-drive: for student to apply for a placement drive
    - Request: rcvStudentCompDto
    - Response: sndAckDto
- GET: /student-profile-details/{studId}: get student profile
    - Response:sndStudProfDto
- POST: /student-profile-update/{objId}: update students profile including their resumes
    - Request: rcvStudProfDto
    - Response: sndAckDto
- GET: /recruiter-drives/{objId}: recruiters home page
    - Response: sndDriveLstDto
- GET: /ping: returns pong (for testing deployments)
    - Response: string

Data transmission objects:

Incoming DTOS:

- rcvLgDto : username and password
- rcvRegDto: userID, email, password, role, firstName, lastName, contactNumber.
- rcvCrtDriveDto: companyName, companyStream, deadlineDate, list of affiliated recruiters
- rcvStudentCompDto: companyId, studentId
- rcvStudProfDto: userId, password, firstName, lastName, rollNumber, batch, contactNumber, cgpa, resumes as multipart files.
- rcvCompDetailsDto: string companyId, meetingLink, testLink, info, jobDescription, pay

Response DTOs:

- sndProfileDto: nullObj (boolean flag), role, userId, objId(adminId/recruiterId/studentId), list of companyIds (only for recruiter)
- sndRegAckDto: regComplete, alreadyTaken both are boolean flags
- sndCrtDriveAckDto: createDriveComplete, alreadyTaken
- sndAckDto: ack
- sndStDriveLstDto: nullObj, list of companies(boll applied, CompanyDrive)
- sndStudProfDto: similar to student object

REFERENCES

[1] https://www.cloudflare.com/en-in/learning/cloud/what-is-the-cloud/
[2] https://en.wikipedia.org/wiki/Google_Cloud_Platform
[3] https://docs.docker.com/compose/
[4] https://cloud.google.com/sdk/docs/install
[5] https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/