# ActiveCollab PROJECT

# USING DEVOPS TOOLS CHAIN

**By**

**Sani Ranpariya (Roll NO: MT2021116)**

**Vishwajeet Deulkar (Roll NO: MT2021154)**

**A project submitted**

**in**

**partial fulfillment of the requirements**

**for the course of**

**SOFTWARE PRODUCTION ENGINEERING**

| Mentor/Guide | Teaching Assistant |
|---|---|
| *Prof. B. Thangaraju* | *Neha Kothari* |


**International Institute of Information**
**And Technology, Bangalore**
**April 2022**

# Abstract

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to serve their customers better and compete more effectively.

Here we create an ActiveCollab project with some basic functionality to demonstrate how the DevOps toolchain can automate the CI/CD pipeline.

Further, we used the ELK stack to monitor the project logs and visualize data with charts and graphs.

# Table of Contents

# Chapter 1 Introduction

To embrace automation in projects, many companies start with a system to manage their project processes to scale up their productivity. Whatever the state of your workflow management, using the right Workflow Management System with the right features is critical. Workflow management is creating and optimizing the paths for data to complete items in a given process. Workflow Management System involves creating a form to hold data and automating a sequential path of tasks for the data to follow until it is fully processed. Thus, such a system allows individuals to automate repetitive processes, follows up automatically on uncompleted tasks in the process, and gives an overall picture of the workflow along with performance metrics. **ActiveCollab** provides this type of feature in a very efficient way and it will make work easier and increase productivity, The scope of the system is very large as it applies to any industry trying to manage workflow and embrace automation, be it a small scale or large scale industry.

## 1.1 Features

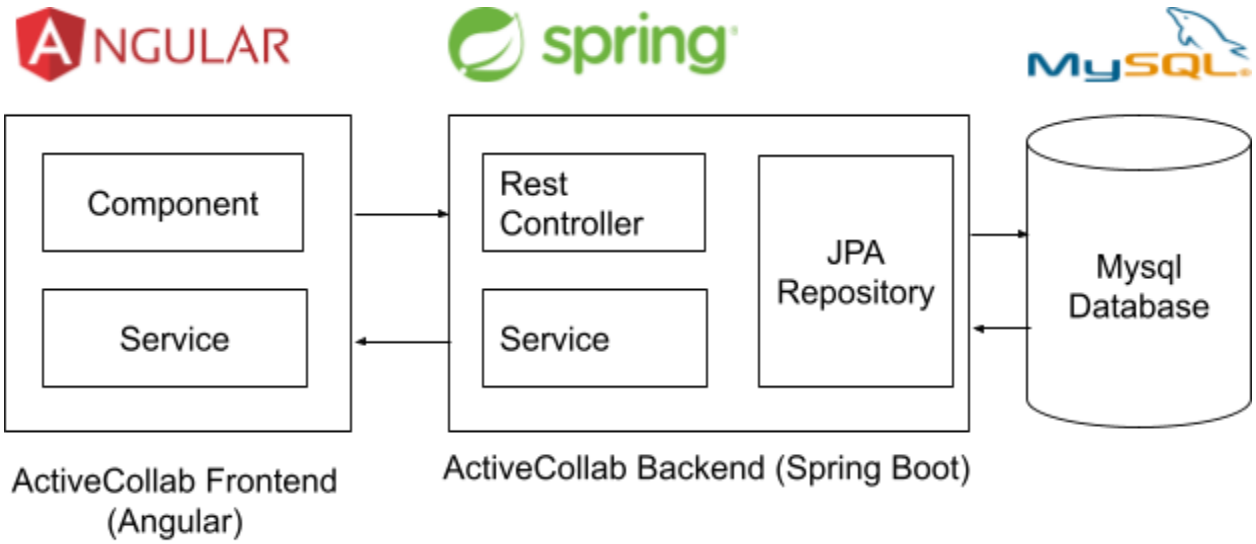
- **Admin**
    - Can see all manager user lists
    - Can see all employee user lists
    - Can add manager user data
    - Can add employee user data

- **Manager**
    - Can create, update and delete project
    - Can see all project lists under them
    - Can add, or remove employees from the project
    - Can add, or remove tasks into the project
    - Can assign an employee to a task
    - Can update the distribution of time in the project effort table
    - Can see effort distribution in visualization

- **Employee**
    - Can see all projects which have tasks assigned to them.
    - Can see all tasks assigned to them in the project.
    - Can update the status of tasks assigned to them.
    - Can see the performance matrix in visualization.

# Chapter 2 Architecture

## 2.1 System Architecture



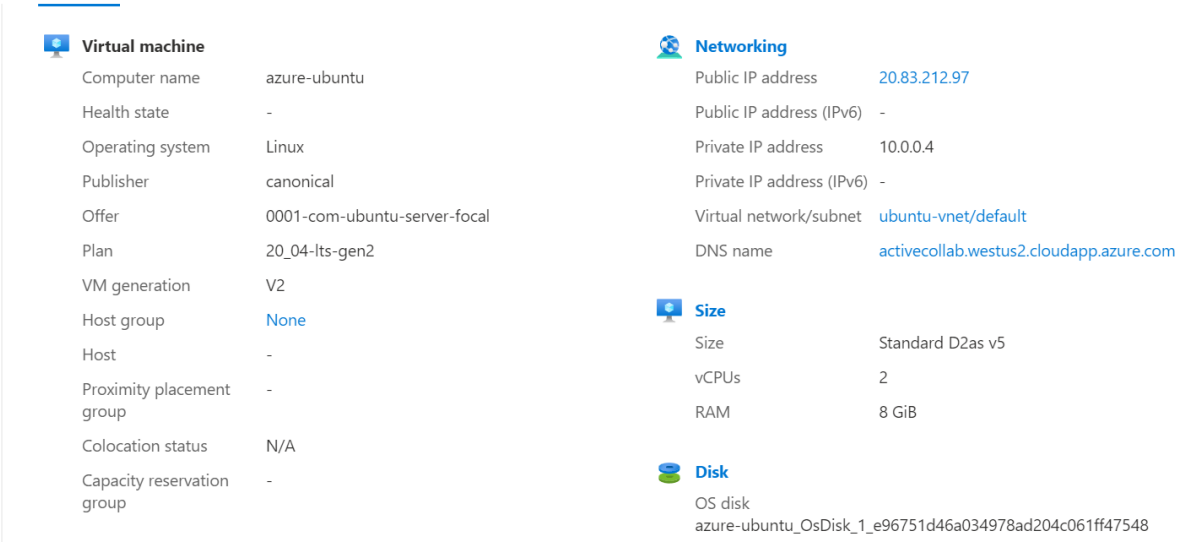

- **ActiveCollab - Frontend:** It is an Angular-based single-page application.
- **ActiveCollab - Backend:** It is a Spring boot Web Rest Controllers for API calls via HTTP calls.
- **ActiveColab - Database:** The data is stored in the MySql Server.

The ActiveCollab application is dockerized in a different docker image and published on to **DockerHub** and deployed on **Microsoft Azure Ubuntu-server virtual machine**. We are using **Ansible** to deploy images for the docker hub to Azure ubuntu-server and **Docker-compose** for defining and running the multiple Docker images under the same network.

## 2.2 System Configuration

### 2.2.1 Microsoft Azure Ubuntu-Server Configuration

- Standard D2as v5
- Platform: Ubuntu Server 20.04 LTS
- Kernel Details: Linux/UNIX
- RAM: 8 GiB
- No of vCPU: 2
- Memory volume: 300GiB
- IPv6 Support: Yes



**Virtual machine**

| | |
|---|---|
| Computer name | azure-ubuntu |
| Health state | - |
| Operating system | Linux |
| Publisher | canonical |
| Offer | 0001-com-ubuntu-server-focal |
| Plan | 20_04-lts-gen2 |
| VM generation | V2 |
| Host group | None |
| Host | - |
| Proximity placement group | - |
| Colocation status | N/A |
| Capacity reservation group | - |

**Networking**

| | |
|---|---|
| Public IP address | 20.83.212.97 |
| Public IP address (IPv6) | - |
| Private IP address | 10.0.0.4 |
| Private IP address (IPv6) | - |
| Virtual network/subnet | ubuntu-vnet/default |
| DNS name | activecollab.westus2.cloudapp.azure.com |

**Size**

| | |
|---|---|
| Size | Standard D2as v5 |
| vCPUs | 2 |
| RAM | 8 GiB |

**Disk**

OS disk
azure-ubuntu_OsDisk_1_e96751d46a034978ad204c061ff47548

### 2.2.2 Project Technology Stack

- **Frontend:** Angular (HTML, CSS, TypeScript)
- **Backend:** SpringBoot Web Service (Java, Maven)
- **Database:** MySql
- **Testing:** Junit, Karma, Jasmin
- **Logging:** log4j2
- **API documentation:** Postman API Platform

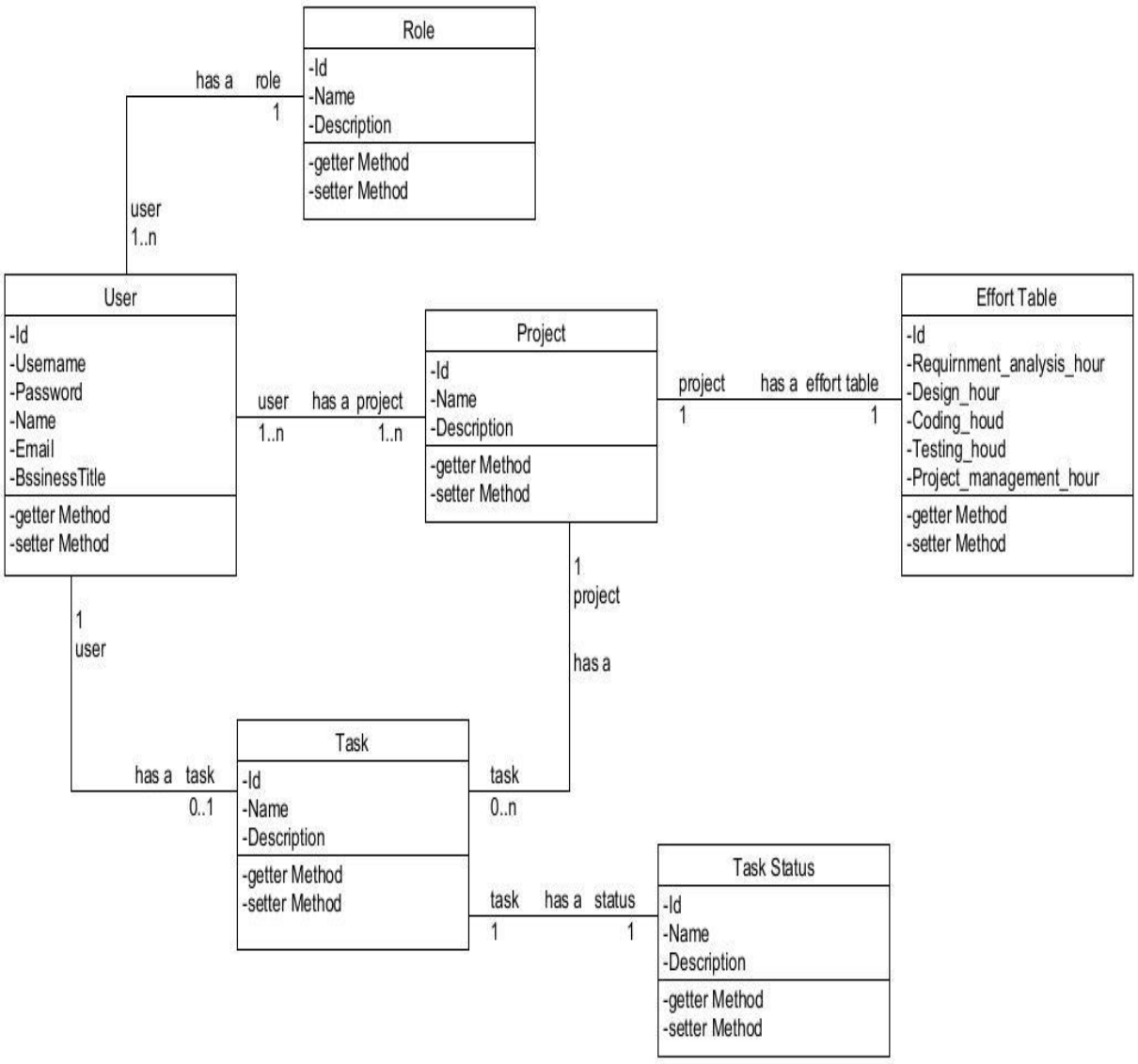
### 2.2.3 DevOps Tools

- **Source Code Management:** Git, Github
- **Containerization:** Docker, DockerHub
- **Continuous Integration (CI):** Github Action
- **Continuous Deployment (CD):** DockerCompose, Ansible, Azure VM
- **Monitoring:** ELK stack, Crontab

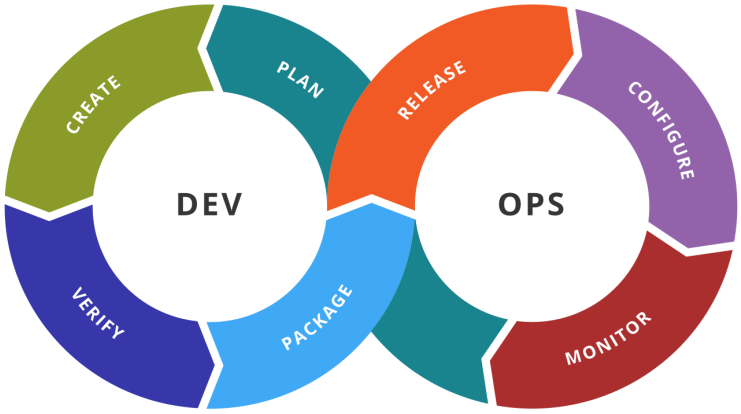
# Chapter 3 Design

## 3.1 Use Case Diagram

## 3.2 Class Diagram



# Chapter 4 Software Development Life Cycle




In the Software Development Life Cycle there are many stages. Using DevOps we can automate stages in a pipeline. For example in our project for code SCM we are using Git and GitHub, Maven & Junit have been used for build and test respectively. DockerHub is used at the delivery stage, Deployment is done using Ansible, in our project we are deploying Scientific Calculator inside a Ubuntu server within a docker container. The ELK stack is used for monitoring and visualizing logs.

## 4.1 Frontend

### 4.1.1 Installations

Angular Framework

Angular is a TypeScript-based free and open-source web application framework. Google maintains it, and its primary purpose is to develop single-page applications. As a framework, Angular has clear advantages while also providing a standard structure for developers to work with. It enables users to create large applications in a maintainable manner. Following are steps for installing Angular(base: UBUNTU 20.04 LTS)

1. Update the system

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

2. Pre-requisite installations

Before starting with Angular, need to make sure that already have **Node.js** installed in the system.

```
$ sudo apt-get install software-properties-common
$ curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
$ sudo apt-get install nodejs
```

To verify installations

```
vishwa@vishwa-VirtualBox:~$ node --version
v12.22.12
vishwa@vishwa-VirtualBox:~$
```

3. Install Angular CLI

```
$ npm install -g @angular/cli
```

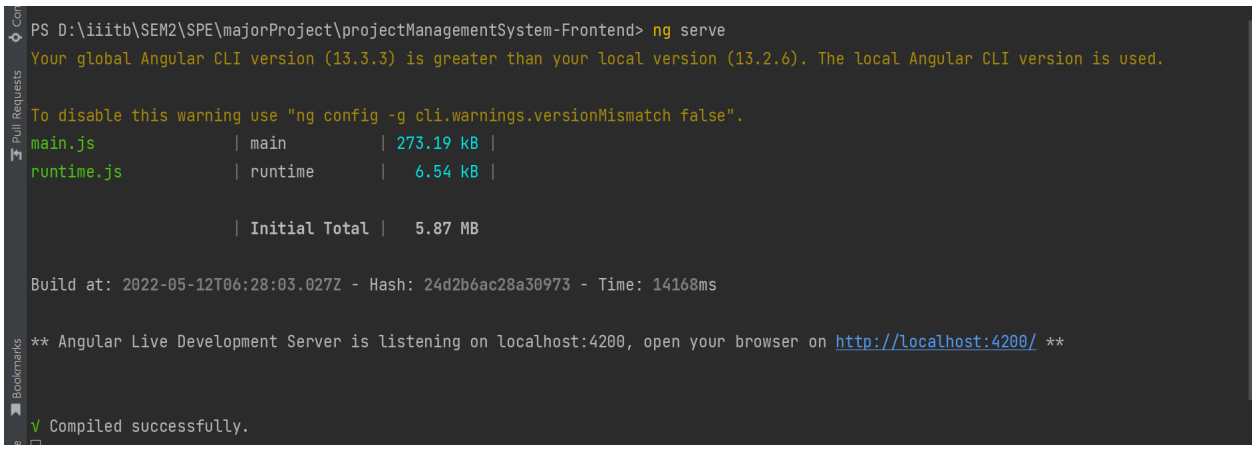
4. Verify angular installations

```
vishwa@vishwa-VirtualBox:~$ ng --version

     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 13.3.3
Node: 12.22.12
Package Manager: npm 8.7.0
OS: linux x64

Angular:
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1303.3 (cli-only)
@angular-devkit/core         13.3.3 (cli-only)
@angular-devkit/schematics   13.3.3 (cli-only)
@schematics/angular          13.3.3 (cli-only)
```

5. Running the application on localhost

```
$ ng serve
```

Output




## 4.1.2 Source Control Management

Source code management is the practice of tracking modifications to source code. Keeping a running history of the changes made to a codebase helps programmers, developers, and testers ensure that they're always working with accurate and up-to-date code and helps resolve conflicts when merging code from multiple sources. Here, we used Git as an SCM and GitHub for online repository hosting.

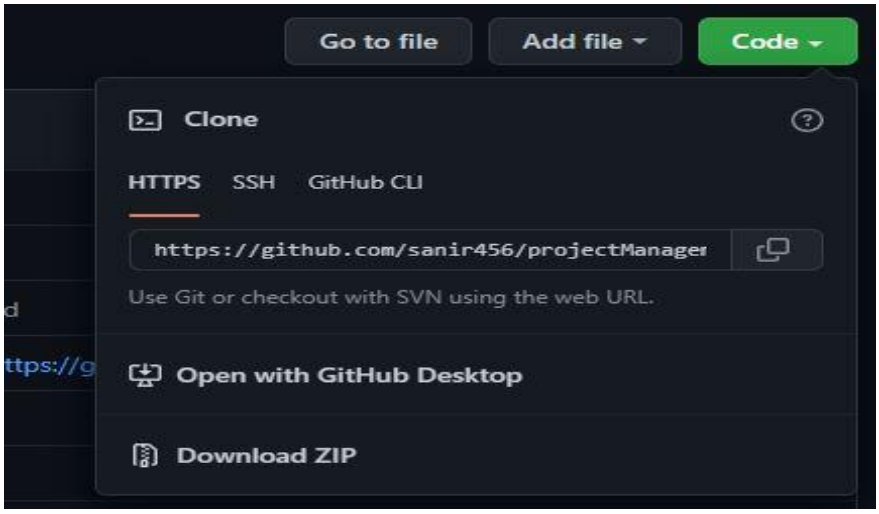
**Repository Link:** ActiveCollab - Frontend

**Commits:** ActiveCollab - Frontend Commits

1. **Initializing the project with git**

```
$ git init
$ git remote add origin
https://github.com/sanir456/projectManagementSystem-Frontend.git
```

2. **Cloning the project**




```
$ git clone
https://github.com/sanir456/projectManagementSystem-Frontend.git
```

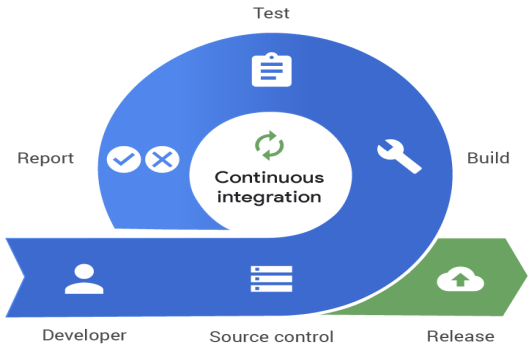
3. **Workflow**

```
$ git add <files>
$ git commit -m "commit message"
$ git pull origin master
$ git push origin master
```

The code is first pulled before making a push to make sure that our project is in the latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “master” branch.

## 4.1.3 Continuous Integration with Github Actions

Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied. Few Popular tools for CI are Jenkins, Travis CI, GitLab CI, Github actions, and many more.




**GitHub Actions**

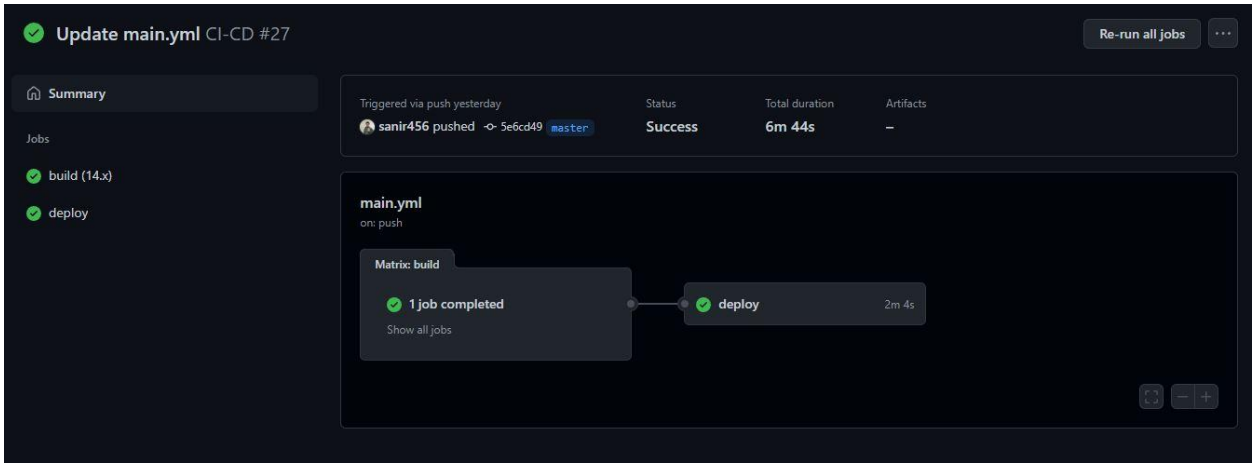
Continuous Integration using GithubActions gives us workflows that can build & test code in the SCM repository. Workflows can run on GitHub-hosted virtual machines, or on machines that you host yourself. Here, we are using GitHub-hosted virtual machines.

**Benefits of GitHub Actions**

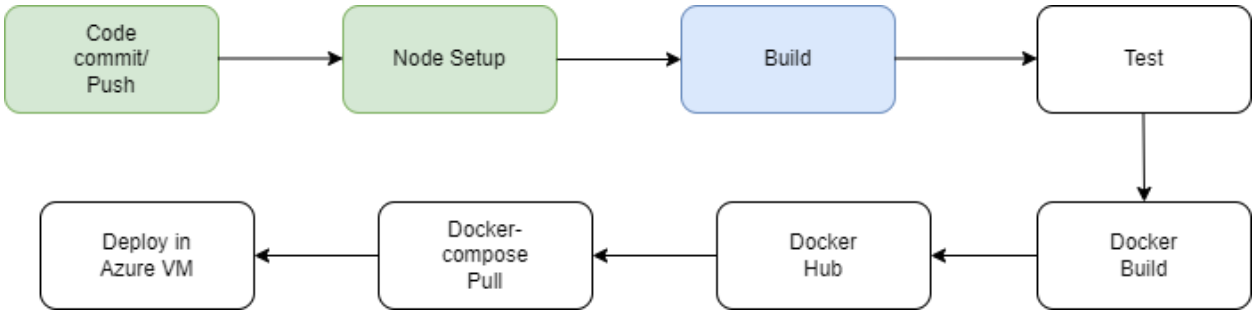
- CI/CD pipeline set-up is simple
- to Respond to any webhook on GitHub
- Community-powered, reusable workflows
- Support for any platform, any language, and any cloud

**Setting up GitHub Actions**

For pipeline scrip we need to create a YAML file that is placed in .github/ directory which will be triggered. Workflow is a collection of jobs and these jobs will run on the trigger of an event, that we are using on the Push event trigger.



### 4.1.4 Build Process Angular application frontend



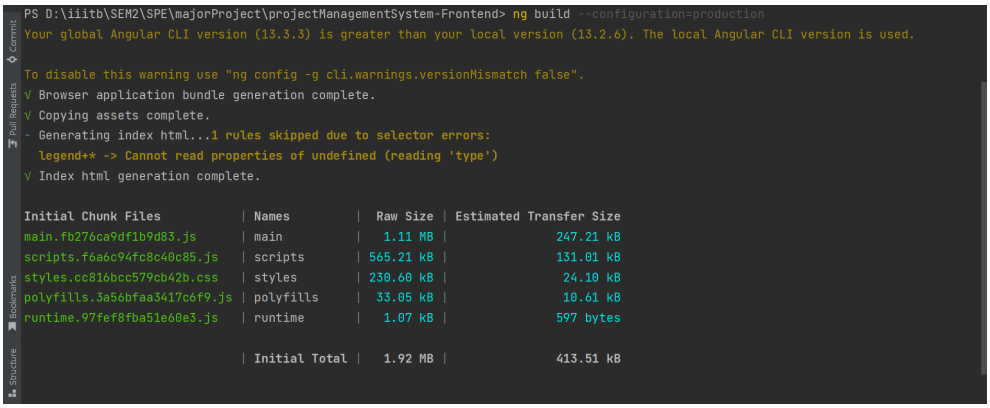

After the developer commits & pushes the code git action workflow will be triggered and the build will start.

```
$ ng serve
```

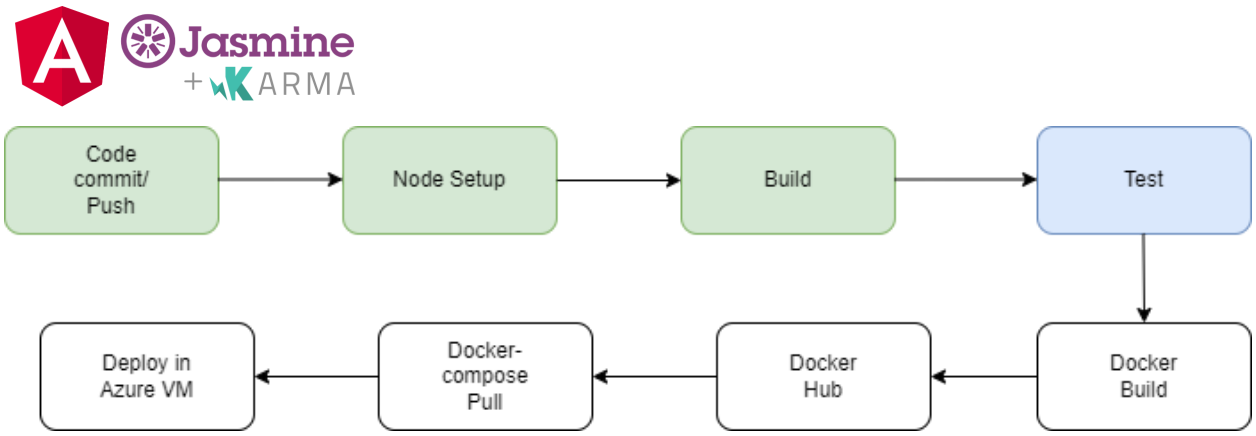
This command builds and serves the application. It rebuilds the application if changes occur. Here project is the name of the application as defined in angular.json.

```
$ ng build  - -configuration=production
```

This command compiles an angular application/library into an output directory named dist at the given path.
Options used & syntax:
– configuration=production: A named build target, as specified in the "configurations" section of angular.json. Each named target is accompanied by a configuration of option defaults for that target. Setting this explicitly overrides the "--prod" flag.d
Here, we are using productions we want to get production environment settings while building an angular application/binary.



### 4.1.5 Testing of Angular application frontend




**command: ng test --watch=false --browsers=ChromeHeadless**
Running the tests in the headerless chrome by keeping the watch flag to false. Through the pipeline, testing of the front end, the Angular application, can be automatically triggered after the build process. The tests are run using the Jasmine javascript test framework through the Karma test runner. Jasmine is a behavior-driven development framework for testing JavaScript code and Karma is a JavaScript test runner that runs the unit test snippet in Angular. It provides tools that make it easier to call Jasmine tests while writing code in Angular.

**Angular has the following files for each component:**

1. component-name.component.ts - a component file
2. component-name.component.spec.ts - a testing specification file
3. component-name.component.html - a template file
4. component-name.component.scss -a CSS file

The tests are written in the ‘component-name.component.spec.ts’ file. To run the test we use the ng test command. This command first builds the app in watch mode and then launches the Karma test runner.

```
F:\Term 2\SPE\Project management system\Frontend\projectManagementSystem>ng test --watch=false --browsers
=ChromeHeadless
✓ Browser application bundle generation complete.
12 05 2022 15:09:30.131:INFO [karma-server]: Karma v6.3.19 server started at http://localhost:9876/
12 05 2022 15:09:30.134:INFO [launcher]: Launching browsers ChromeHeadless with concurrency unlimited
12 05 2022 15:09:30.141:INFO [launcher]: Starting browser ChromeHeadless
12 05 2022 15:09:34.732:INFO [Chrome Headless 101.0.4951.64 (Windows 10)]: Connected on socket Y5cq0YciW5
Bnk1GrAAAB with id 90958139
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'input'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 1 of 19 SUCCESS (0 secs / 0.306 secs)
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'input'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 1 of 19 SUCCESS (0 secs / 0.306 secs)
ERROR: 'NG0304: 'mat-sidenav-container' is not a known element:
1. If 'mat-sidenav-container' is an Angular component, then verify that it is part of this module.
2. If 'mat-sidenav-container' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.sche
mas' of this component to suppress this message.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 2 of 19 SUCCESS (0 secs / 0.677 secs)
ERROR: 'NG0304: 'mat-sidenav-container' is not a known element:
1. If 'mat-sidenav-container' is an Angular component, then verify that it is part of this module.
2. If 'mat-sidenav-container' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.sche
```

At the end of the test, we get the following output, which suggests how many of the tests were successful and how many were not:

```
ERROR: 'NG0304: 'mat-form-field' is not a known element:
1. If 'mat-form-field' is an Angular component, then verify that it is part of this module.
2. If 'mat-form-field' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas' of
ERROR: 'NG0304: 'mat-label' is not a known element:
1. If 'mat-label' is an Angular component, then verify that it is part of this module.
2. If 'mat-label' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas' of this
Chrome Headless 101.0.4951.64 (Windows 10): Executed 17 of 19 SUCCESS (0 secs / 0.673 secs)
ERROR: 'NG0304: 'mat-label' is not a known element:
1. If 'mat-label' is an Angular component, then verify that it is part of this module.
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'input'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 17 of 19 SUCCESS (0 secs / 0.673 secs)
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'textarea'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 17 of 19 SUCCESS (0 secs / 0.673 secs)
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'input'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 18 of 19 SUCCESS (0 secs / 0.722 secs)
ERROR: 'NG0303: Can't bind to 'ngModel' since it isn't a known property of 'input'.'
Chrome Headless 101.0.4951.64 (Windows 10): Executed 18 of 19 SUCCESS (0 secs / 0.722 secs)
Chrome Headless 101.0.4951.64 (Windows 10): Executed 19 of 19 SUCCESS (1.135 secs / 0.733 secs)
TOTAL: 19 SUCCESS
```

Individual components were tested. Two main functions are often used during testing, describe() and it(). describe() is used to define a group of related tests while it() is used to define a single test or spe

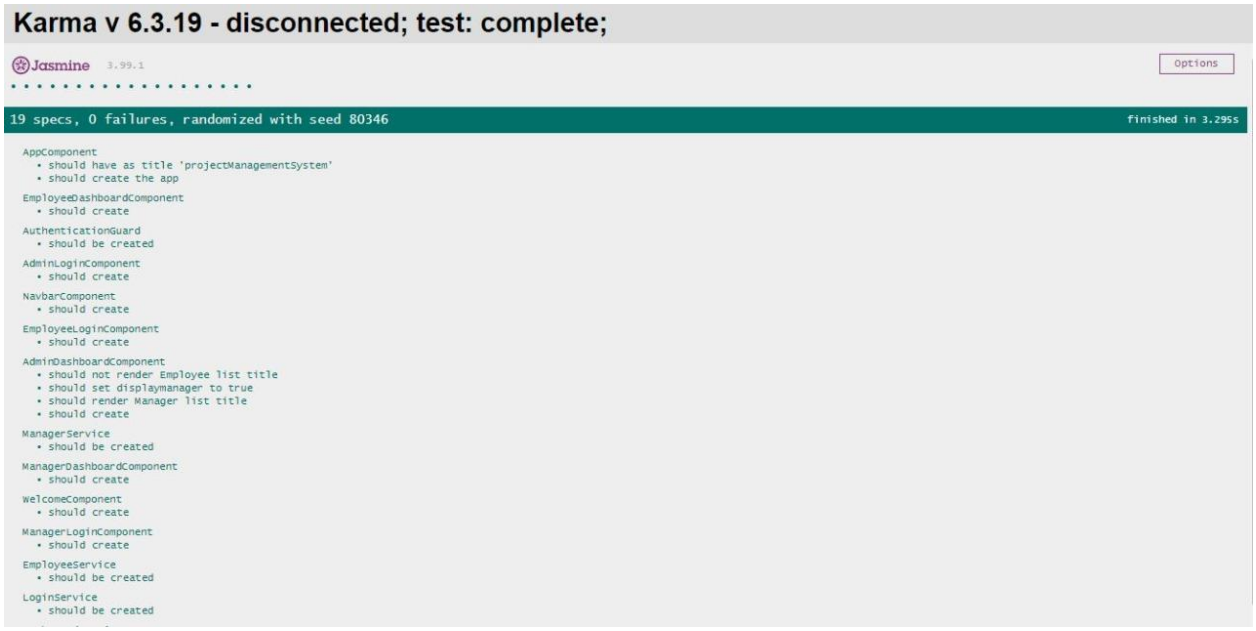
```
describe('AdminDashboardComponent', () => {
  let component: AdminDashboardComponent;
  let fixture: ComponentFixture<AdminDashboardComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports:[RouterTestingModule,HttpClientModule],
      declarations: [ AdminDashboardComponent ],
      providers:[AdminService ,MatSnackBar]
    })
    .compileComponents();
  });
```

Example of a spec.ts file:

```
import { HttpClientModule } from '@angular/common/http';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MatSnackBar } from '@angular/material/snack-bar';
import { Router } from '@angular/router';
import { RouterTestingModule } from '@angular/router/testing';
import { AdminService } from 'src/app/services/admin.service';

import { AdminDashboardComponent } from './admin-dashboard.component';

describe('AdminDashboardComponent', () => {
  let component: AdminDashboardComponent;
  let fixture: ComponentFixture<AdminDashboardComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports:[RouterTestingModule,HttpClientModule],
      declarations: [ AdminDashboardComponent ],
      providers:[AdminService ,MatSnackBar]
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(AdminDashboardComponent);
```

```
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should set displaymanager to true', ()=>{
    expect(component.displayManager).toBeTruthy();
  });

  it('should render Manager list title', ()=>{
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('p').textContent).toContain('Manager List');
  });

  it('should not render Employee list title', ()=>{
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('p').textContent=='Employee List').toBeFalsy();
  });
});
```

The tests outputs are also displayed on the browser in the “Jasmine HTML Reporter”:

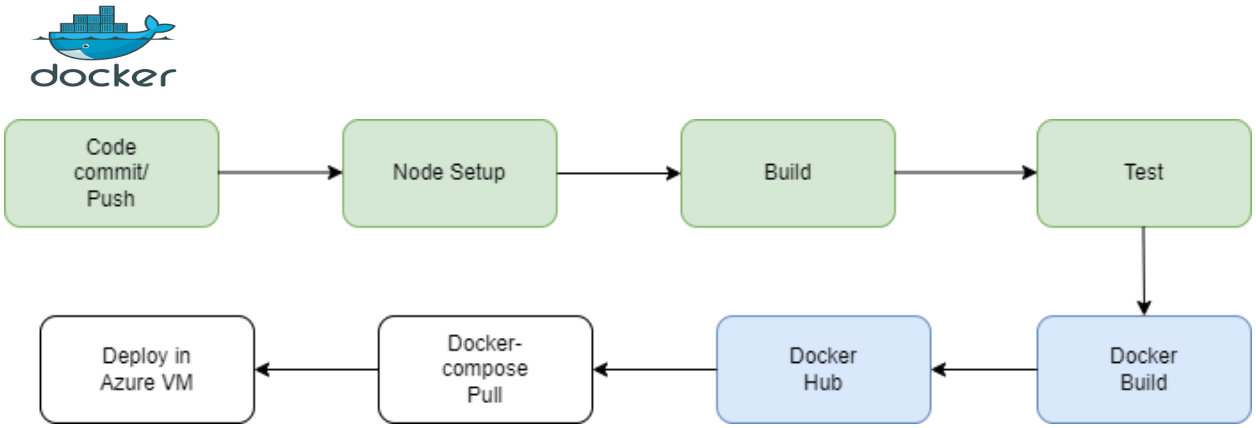


The configurations related to Karma and Jasmine are stored in a file named ‘karma.conf.js’.

```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      jasmine: {
      },
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    jasmineHtmlReporter: {
      suppressAll: true // removes the duplicated traces
    },
    coverageReporter: {
      dir: require('path').join(__dirname, './coverage/project-management-system'),
      subdir: '.',
      reporters: [
        { type: 'html' },
        { type: 'text-summary' }
      ]
    },
    browsers: ['Chrome', 'ChromeHeadless', 'ChromeHeadlessCI'],
    customLaunchers: {
    ChromeHeadlessCI: {
      base: 'ChromeHeadless',
      flags: ['--no-sandbox']
      }
    },
    reporters: ['progress', 'kjhtml'],
```

```
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['ChromeHeadless'],
    singleRun: false,
    restartOnFileChange: true
  });
};
```

### 4.1.6 Docker Build & Push



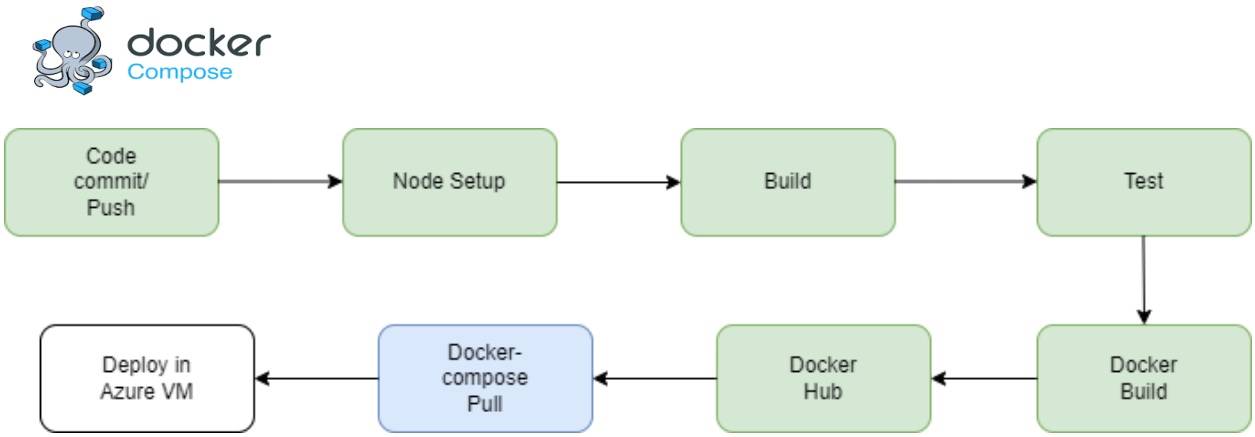

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run, providing flexibility of building & pushing images Dockerhub. Dockerfile is a simple text file that consists of a set of instructions to build Docker images.

Dockerfile :

```
# Stage 1: Compile and Build angular codebase

# Use official node image as the base image
FROM node:latest as build
# Set the working directory
WORKDIR /usr/local/app
# Add the source code to app
COPY ./ /usr/local/app/
```

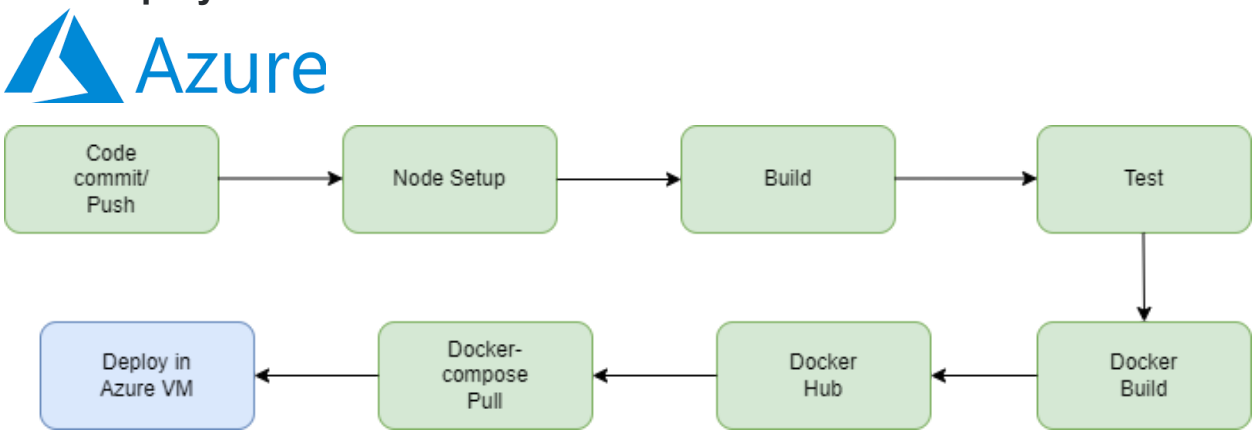
```
RUN npm config set legacy-peer-deps true
# Install all the dependencies
RUN npm install --configuration=production
# Generate the build of the application
RUN npm run build --configuration=production
# Stage 2: Serve app with nginx server

# Use official nginx image as the base image
FROM nginx:latest
RUN rm -rf /usr/share/nginx/html/* && rm -rf /etc/nginx/nginx.conf
COPY ./nginx.conf /etc/nginx/nginx.conf
# Copy the build output to replace the default nginx contents.
COPY --from=build /usr/local/app/dist/project-management-system /usr/share/nginx/html
# Expose port 80
EXPOSE 80
```


### 4.1.7 Docker Compose pull

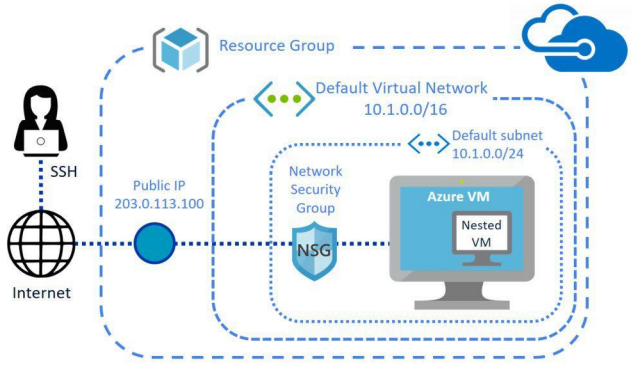



Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down. The big advantage of using Compose is you can define your application stack in a file, keep it at the root of your project repo (it's now version controlled), and easily enable someone else to contribute to your project. Someone would only need to clone your repo and start the compose app. We have created a docker-compose.yml file by which we pull the Dockerhub frontend container and connect it with the backend & database.

### 4.1.8 Deploy in Azure VM

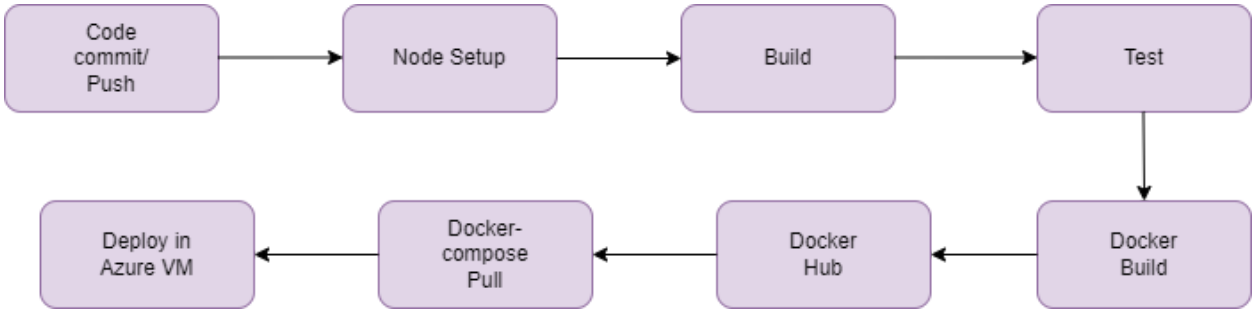



Azure Virtual Machines (VM) is one of several types of on-demand, scalable computing resources that Azure offers. Typically, you choose a VM when you need more control over the computing environment than the other choices offer. This article gives you information about what you should consider before you create a VM, how you create it, and how you manage it. An Azure VM gives you the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, you still need to maintain the VM by performing tasks, such as configuring, patching, and installing the software that runs on it.




We have used Azure VM for project deployment, with help of docker-compose we can deploy a 3-tier architecture project on a single instance.

### 4.1.8 Github Actions Code walkthrough (Frontend)




**Build & Tests :**

```
  build:
    # The type of runner that the job will run on
        runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [14.x]

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can
access it
      - uses: actions/checkout@v2

      - name: setup node
        uses: actions/setup-node@master
        with:
          node-version: ${{ matrix.node-version }}

      # install applicaion dependencies
      - name: Install dependencies
        run: |
          npm config set legacy-peer-deps true
          npm install --configuration=production
          npm ci
      # build and test the apps
      - name: build
```

```
      run: |
        npm run build --configuration=production
    - name: test
      run: npm test --watch=false --browsers=ChromeHeadCI
```

Node environment is set up before running the build and tests.

**Login to Docker**

```
- name: Login to DockerHub
      uses: docker/login-action@v1
      with:
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}
```

**Build and push to Docker**

Once the login is successful, The Image is built and pushed to dockerhub.

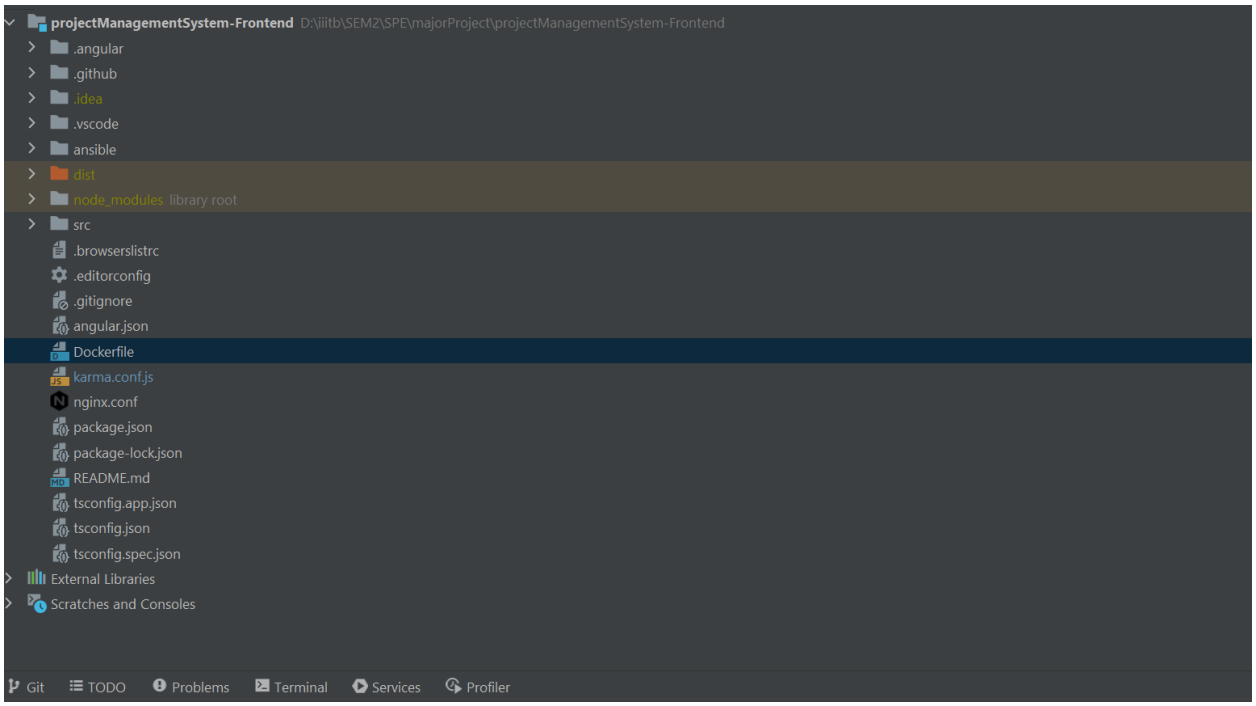
```
    - name: Build and push
      uses: docker/build-push-action@v2
      with:
        context: ./
        file: ./Dockerfile
        push: true
        tags: vishwajeet1321/project-management-system-frontend:latest
```

**Deploy in Azure VM**

```
deploy:
  needs: build
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2
    - uses: ./.github/actions/ansible
  env:
    SSH_PASSWORD: ${{ secrets.SSH_PASSWORD }}
    SSH_USER: ${{ secrets.SSH_USER }}
```

## 4.1.9 Code walkthrough

### Design Architecture:




- All the building components are stored in /components directory
- All the Services which are responsible for API Calls are stored in /services directory
- All the Routings are configured in app-routing.module.ts file
- All the modules are in app.modules.ts
- Testing is enabled done in *spec.ts files in each component.
- All the assets are stored in /assets folder
- Router Outlet is configured in app.component.html

### Exploring Components:

For creating new Component

```
$ ng g c <path>/<filename>
```

**Following files are created on generating new component:**

*.html -> Component HTML File
*.scss -> Component Styling File
*.spec.ts -> Component testing and spec file
*.ts -> Component Typescript file

**Exploring Service File:**
For creating a service:

```
$ ng g c <path>/<filename>
```

Here “g” stands for generate and “s” stands for service

**Following files are created:**

*.service.ts -> Typescript file for writing services
*.service.spec.ts

**Making the API Calls to the backend:**

```
addUser(userData:any){
  let token = localStorage.getItem("SessionUser");
  let header = new HttpHeaders(
    {
      Authorization  : "Bearer " + token
    }
  );
  return this.http.post(`${this.url}/users/register`, userData,{'headers':header});
}
```

**Calling the API from service file to Component:**

```
import { AdminService } from 'src/app/services/admin.service';
```

```
constructor(private adminService:AdminService,private _snackBar: MatSnackBar) { }
```

```
getEmployeeData(){
  this.adminService.getEmployeeData().subscribe(
    (response:any) => {
      console.log(response)
      this.employeeData = [];
      response.forEach(
        (element:any) => {
          let employeeDetails = {
            id:element.id,
            name:element.name,
            email:element.email,
```

```
            businessTitle:element.businessTitle
          }
          this.employeeData.push(employeeDetails);
        }
      )
      // this.displayManager = true;
      this.employeeDataSource.data = this.employeeData;
      console.log(response);
    },
    (error:any) => {
      this._snackBar.open(error, 'Close', {
        horizontalPosition: this.horizontalPosition,
        verticalPosition: this.verticalPosition,
        duration: 2* 1000,
      });
    }
  )
}
```

**Package Imports in angular.json**

```
          "styles": [
            "./node_modules/@angular/material/prebuilt-themes/deeppurple-amber.css",
            "src/styles.css",
            "./node_modules/bootstrap/dist/css/bootstrap.min.css"
          ],
          "scripts": [
            "node_modules/apexcharts/dist/apexcharts.min.js",
            "node_modules/jquery/dist/jquery.min.js"
          ]
```

# 4.2 Backend

## 4.2.1 Installations

To run node applications, our system has to be set up with node and NPM.

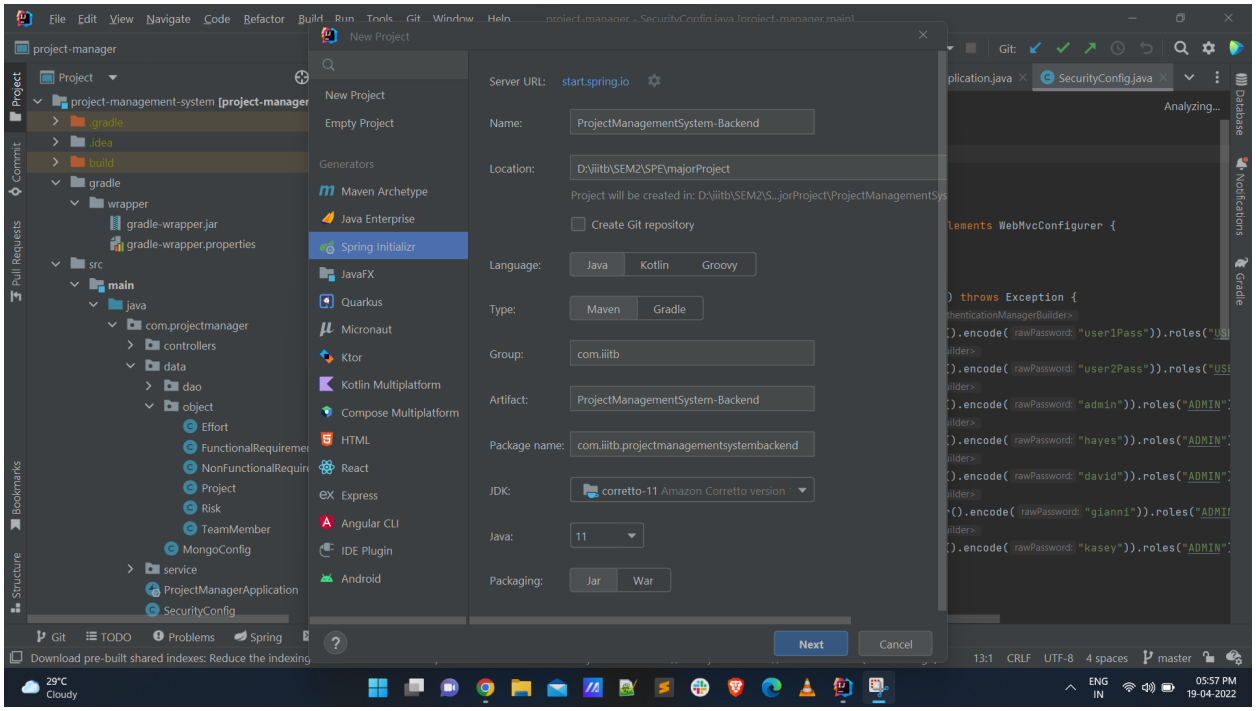
### Install Maven

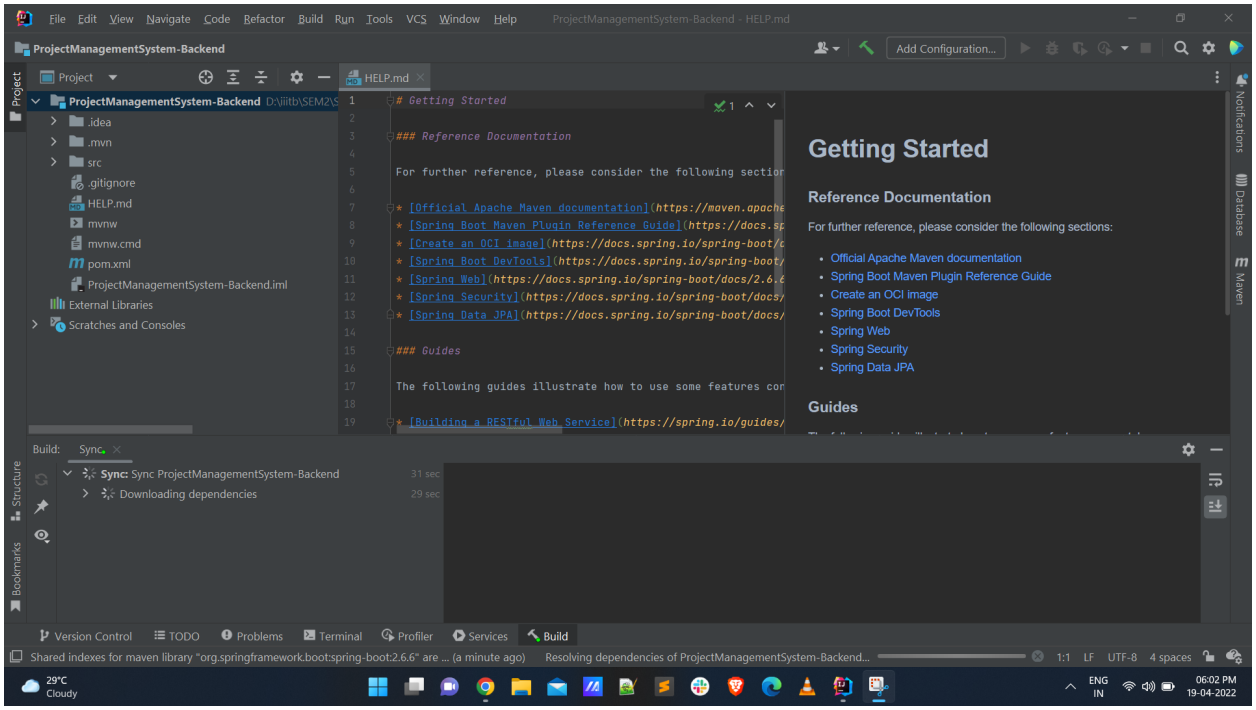
```
$ sudo apt-get install maven
```

### Checking if maven is installed or not

```
PS D:\iiitb\SEM2\SPE\majorProject\ProjectManagementSystem-Backend> mvn -version
Apache Maven 3.8.5 (3599d3414f046de2324203b78ddcf9b5e4388aa0)
Maven home: C:\Program Files\apache-maven-3.8.5
Java version: 11.0.14.1, vendor: Microsoft, runtime: C:\Program Files\Microsoft\jdk-11.0.14.101-hotspot
Default locale: en_IN, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
```

### Initializing the Spring Boot project with default config:

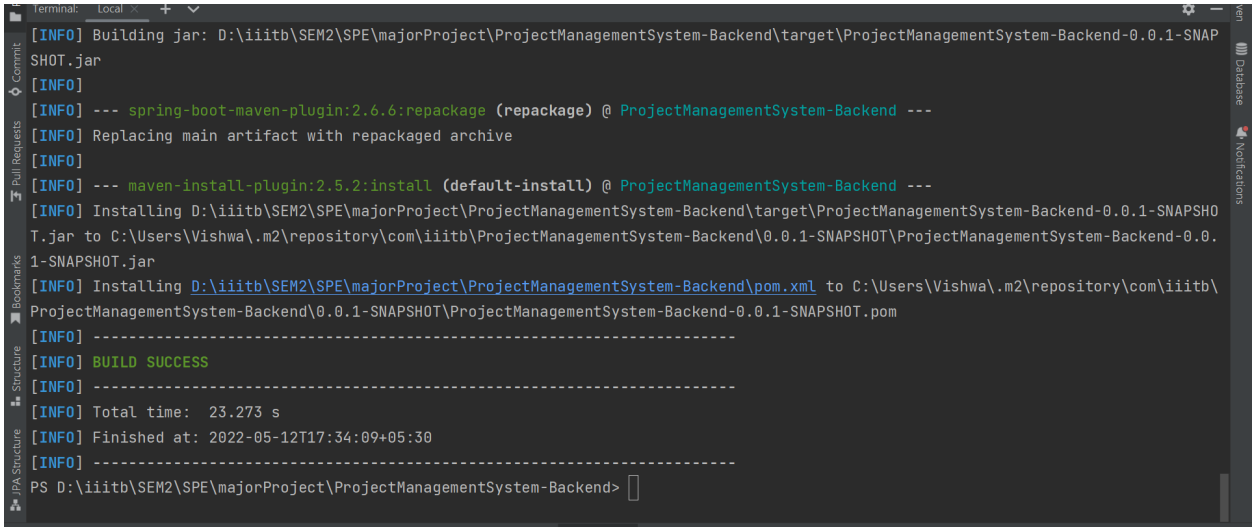





## Running the Spring Boot Application locally:

```
$ mvn clean install
$ mvn build
```

Output



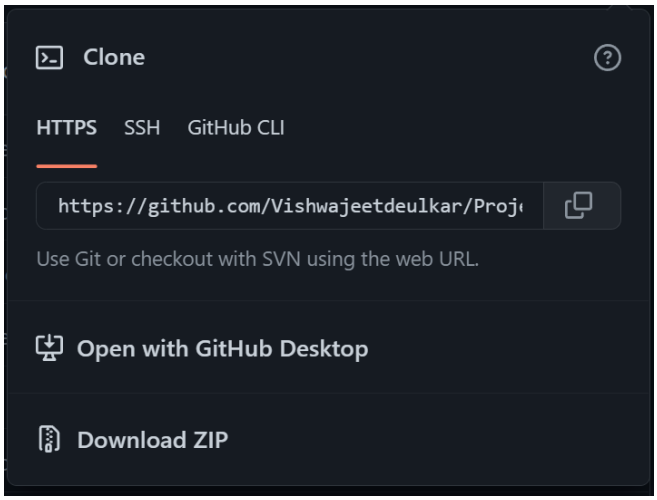
### 4.2.2 Source Control Management

We are using Github for version control in our backend.
**Repository Link:** ActiveCollab - Backend
**Commits:** ActiveCollab - Backend Commits

**Initializing the project with git**

```
$ git init
$ git remote add origin
https://github.com/Vishwajeetdeulkar/ProjectManagementSystem-Backend.git
```

**Cloning the project**




```
$ git clone
https://github.com/Vishwajeetdeulkar/ProjectManagementSystem-Backend.git
```

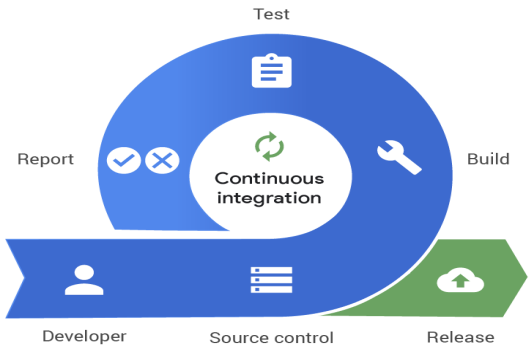
**Workflow**

```
$ git add <files>
$ git commit -m "commit message"
$ git pull origin main
$ git push origin main
```

The code is first pulled before making a push to make sure that our project is in the latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “main” branch.

### 4.2.3 Continuous Integration with Github Actions

Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied. Few Popular tools for CI are Jenkins, Travis CI, GitLab CI, Github actions, and many more.




**GitHub Actions**

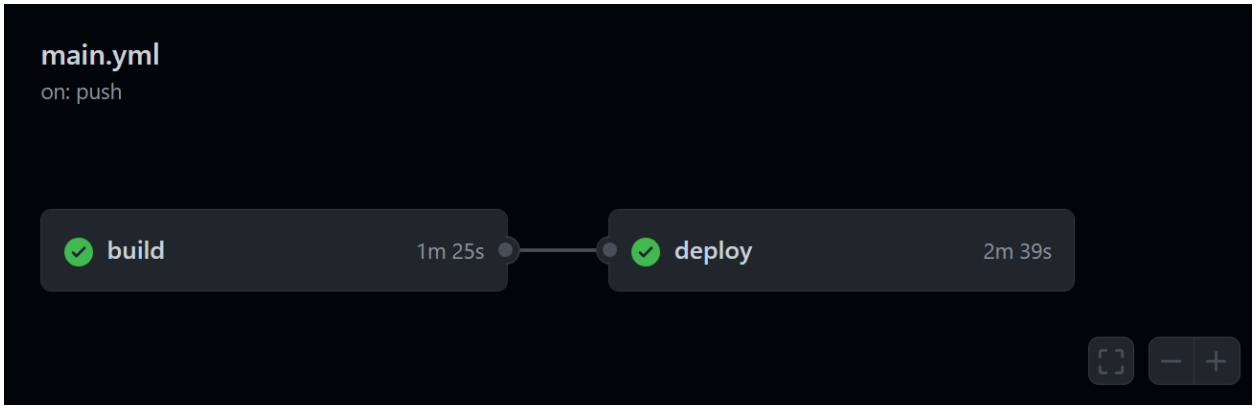
Continuous Integration using GithubActions gives us workflows that can build & test code in the SCM repository. Workflows can run on GitHub-hosted virtual machines, or on machines that you host yourself. Here, we are using GitHub-hosted virtual machines.

**Benefits of GitHub Actions**

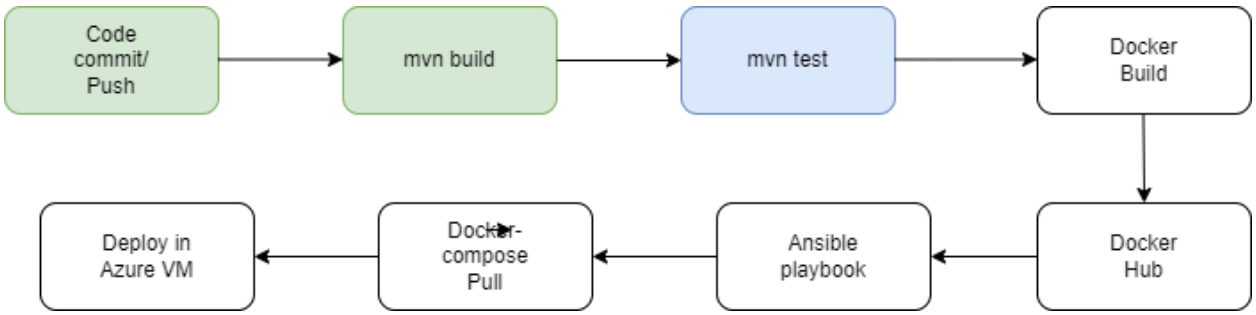
- CI/CD pipeline set-up is simple
- to Respond to any webhook on GitHub
- Community-powered, reusable workflows
- Support for any platform, any language, and any cloud

**Setting up GitHub Actions**

For pipeline scrip we need to create a YAML file that is placed in .github/ directory which will be triggered. Workflow is a collection of jobs and these jobs will run on the trigger of an event, that we are using on the Push event trigger.



### 4.2.4 Testing with Junit




JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. JUnit is linked as a JAR at compile-time.

**Configuring the log4j2.xml file:**

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
   <Appenders>
      <Console name="ConsoleAppender" target="SYSTEM_OUT">
         <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} [%F] [%level]
%logger{36} %msg%n"/>
      </Console>
      <File name="FileAppender" fileName="projectManagementSystem.log"
append="true">
         <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} [%F] [%level]
%logger{36} %msg%n"/>
      </File>
   </Appenders>
   <Loggers>
      <Logger name="ProjectManagementSystem" level="INFO">
         <AppenderRef ref="FileAppender"/>
      </Logger>
      <Root level="info">
         <AppenderRef ref="ConsoleAppender"/>
      </Root>
   </Loggers>
</Configuration>
```

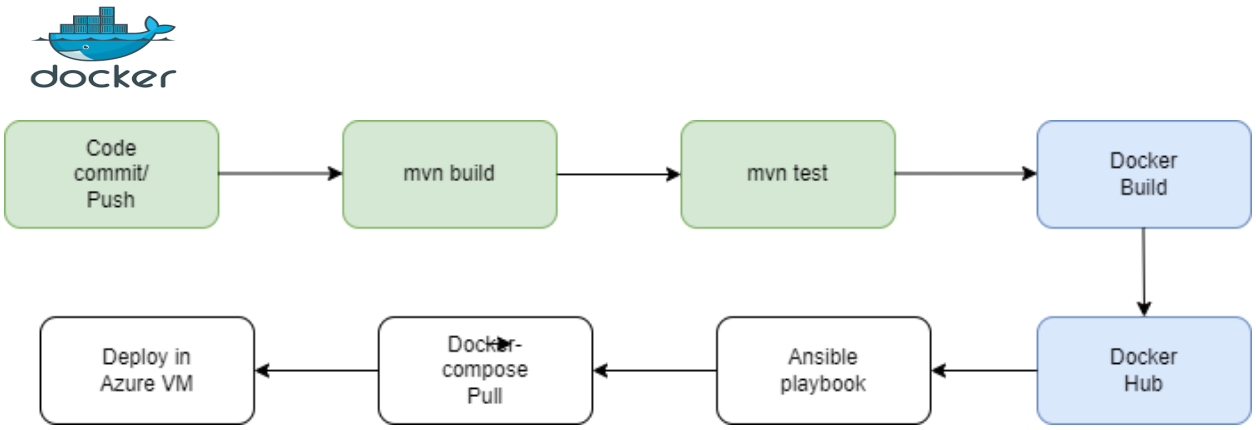
**Sample of a test file:**

```java
package com.iiitb.projectmanagementsystembackend;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = ProjectManagementSystemBackendApplication.class)
@WebAppConfiguration
public abstract class AbstractTest {

    protected String mapToJson(Object obj) throws JsonProcessingException
    {
        ObjectMapper objectMapper = new ObjectMapper();
        return objectMapper.writeValueAsString(obj);
    }
    protected <T> T mapFromJson(String json, Class<T> clazz) throws
JsonParseException, JsonMappingException, IOException
    {
        ObjectMapper objectMapper = new ObjectMapper();
        return objectMapper.readValue(json, clazz);
    }
}
```

**Console Outputs:**

```
2022-05-12 18:05:24 [HikariDataSource.java] [INFO] com.zaxxer.hikari.HikariDataSource HikariPool-1 - Shutdown completed.
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.347 s
[INFO] Finished at: 2022-05-12T18:05:25+05:30
[INFO] ------------------------------------------------------------------------
PS D:\iiitb\SEM2\SPE\majorProject\ProjectManagementSystem-Backend>
```

### 4.2.5 Docker build & push




**Docker setup in Azure VM**
The Docker environment is set up in Azure VM using the commands officially provided by docker.

```
$ sudo apt-get install docker.io
```

**Dockerfile**

```
FROM openjdk:11
EXPOSE 8080
ADD target/ProjectManagementSystem-Backend-0.0.1-SNAPSHOT.jar backend.jar
RUN cat /etc/hosts
ENTRYPOINT ["java","-jar","backend.jar"]
```

**Pushing to DockerHub using Github actions:**
In the CI pipeline once we successfully log in to docker, the image is built and pushed to dockerhub.

**Docker composes**
For a fully functional backend, we need to spin 3 docker images. (project management backend, MYSQL, Angular frontend). Docker-compose is used for defining and running multiple containers at once.
File: docker-compose.yaml

```
version: "3.7"
services:
  clientnode:
    image: vishwajeet1321/project-management-system-frontend:latest
    ports:
      - "8085:80"
    networks:
      net_ubuntu:
        ipv4_address: 172.28.1.5
  servernode:
    image: vishwajeet1321/project-management-system-backend:latest
    deploy:
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 5
        window: 120s
    ports:
      - "8086:8086"
    depends_on:
      - "dbnode"
    environment:
      MYSQL_HOST: "dbnode"
      MYSQL_USER: "root"
      MYSQL_PASSWORD: "password"
      MYSQL_PORT: "3306"
    networks:
      net_ubuntu:
        ipv4_address: 172.28.1.6
  dbnode:
    image: mysql
    ports:
     - "3306:3306"
    volumes:
```

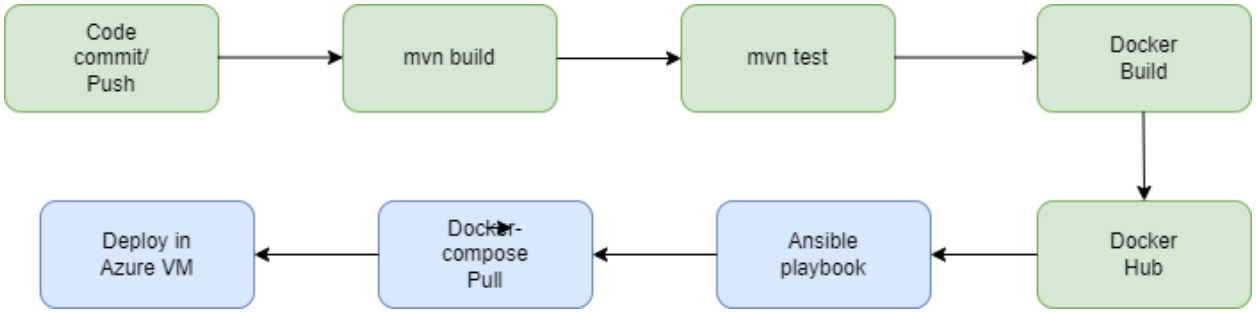
```
    - /home/vishwa/Documents/projects/DockerCompose/dbnode:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: "password"
    MYSQL_DATABASE: "projectmanagementsystem"
    MYSQL_PASSWORD: "password"
  networks:
    net_ubuntu:
      ipv4_address: 172.28.1.7

networks:
  net_ubuntu:
    ipam:
      driver: default
      config:
      - subnet: 172.28.1.0/24
```

- The file compose is set to version 3.
- We are running 3 containers (frontend, backend & MYSQL database) to run our application.
- The frontend container is run and processed by the docker file (path is mentioned). Port 80 is exposed to talk to the container from outside the docker region.
- Backend container is run and processed by the docker file (path is mentioned). The port 8086 is exposed to talk to the container from outside the docker region.
- Backend depends on the MYSQL container. I,e the database is first set up then the backend starts running.
- Both the containers are run within the same network i.e docker_net
- MYSQL container is using a volume for data perseverance so, even when the server is down, user data is not lost.

**Installing and Running docker-compose:**

```
$ sudo apt-get install docker-compose
$ docker-compose up
```

### 4.2.6 Continuous Deployment with Ansible




Ansible is an automation tool which helps in automating application deployment, config management and many more. As github marketplace does not provide any plugin which supports complete automation with ansible, we build our own ansible plugin to deploy our application in Azure VM. We will spin a simple lightweight linux docker image to run ansible commands in it.

This Above code from the main.yml file will trigger the ansible.yml file in the ansible directory.

```
name: "Ansible"
description: "Runs an Ansible playbook"
runs:
  using: "docker"
  image: "Dockerfile"
```

When the above file is triggered, a simple Dockerfile is spinned which is present within the same directory i.e /.github/actions/ansible folder.

```
FROM alpine
ENV ANSIBLE_HOST_KEY_CHECKING=False
RUN apk add ansible gcc python3-dev libc-dev libffi-dev openssl-dev
RUN apk add python3 py3-pip openssh-client sshpass
RUN pip3 install --upgrade paramiko
RUN pip3 install docker
COPY hosts /hosts
COPY ansible.cfg /etc/ansible/ansible.cfg
COPY entrypoint.sh /entrypoint.sh
CMD ["sh", "/entrypoint.sh"]
```

The above Dockerfile is built on taking alpine as the base image. All the required modules and packages are installed to set up ansible. Once the environment is set up, All the required files are copied from the base to docker environment.

**Exploring ansible.cfg:**

```
[defaults]
inventory          = /hosts
remote_tmp         = /tmp/.ansible-${USER}/tmp
```

Parameters and paths of inventory file are setup in ansible.cfg file.

Here we are entering all the required details of the slave server by leveraging the use of echo and pipe. All the secrets information is taken from the github secrets i.e(SSH username and SSH Password).

```
#!/bin/sh
echo "Ansible Entrypoint"
echo "[azure]" >> /hosts
echo "20.83.212.97" >> /hosts
echo "[all:vars]" >> /hosts
echo "ansible_connection=ssh" >> /hosts
echo "ansible_user= $SSH_USER" >> /hosts
echo "ansible_ssh_user= $SSH_USER" >> /hosts
echo "ansible_python_interpreter=/usr/bin/python3.8" >> /hosts
echo "ansible_ssh_pass=$SSH_PASSWORD" >> /hosts
echo "ansible_become_pass=$SSH_PASSWORD" >> /hosts
echo "Entering the ansible using ansible-playbook"
ansible-playbook ansible/playbook.yml --user $SSH_USER
```

Exploring the playbook.yml:

```
---
- hosts: all
  become: true
  tasks:

    - name: remove crontab jobs if any
```

```
    shell: "crontab -u vishwa -r"
  - name: remove any containers if exists
    shell: "docker rm -vf $(docker ps -aq) || true"
  - name: remove any docker images if exists
    shell: "docker rmi -f $(docker images -aq) || true"
  - name: docker compose up
    shell: "docker-compose up -d"
  - name: sleep command
    shell: "sleep 30s"
  - name: kill previous logstash instance
    shell: "ps aux | grep -ie /usr/share/logstash/jdk/bin/java | awk '{print $2}' | xargs kill
>kill.log 2>&1 &"
  - name: set a crontab job to copy of file from container
    shell: "sh /home/vishwa/job.sh &"
  - name: run logstash
    shell: "/usr/share/logstash/bin/logstash -f /home/vishwa/pipe.conf >output.log
2>&1 &"
```

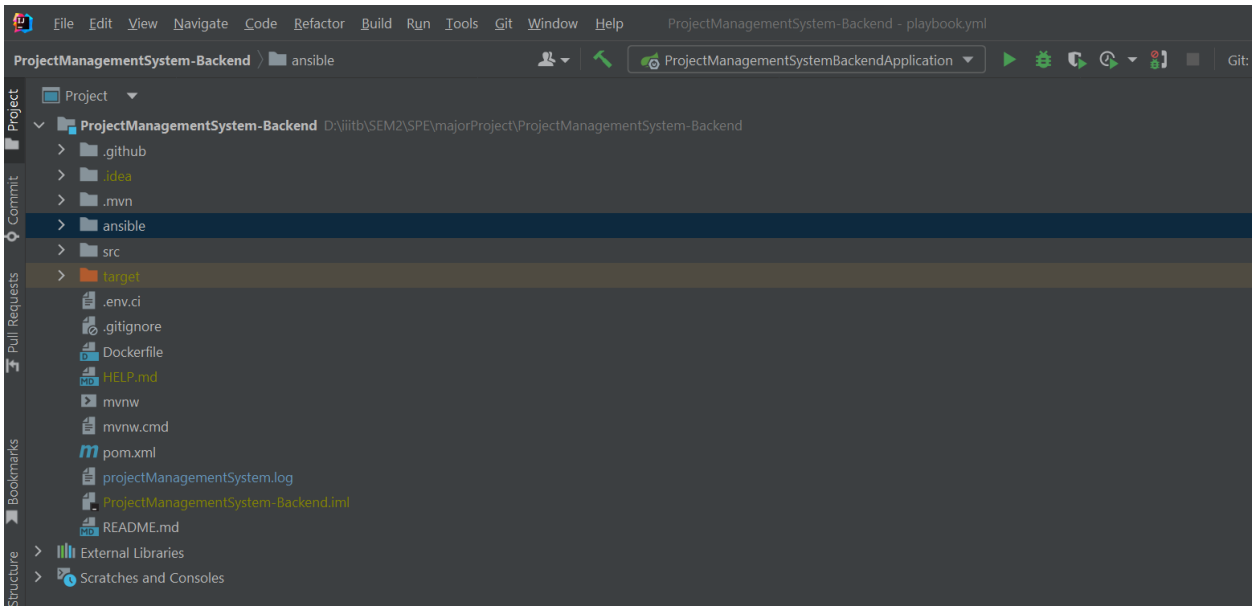
After successful SSH into the slave machine, our playbook triggers and runs a modules defined in it.

1. crontab -u vishwa -r :  it will remove any crontab job present in Azure VM.
2. docker rm -vf $(docker ps -aq) || true : remove any containers if exists
3. docker rmi -f $(docker images -aq) || true : remove any docker images if exists
4. docker-compose up -d : docker compose up
5. sleep 30s : sleeps cli for containers to become up & serving
6. ps aux | grep -ie /usr/share/logstash/jdk/bin/java | awk '{print $2}' | xargs kill >kill.log 2>&1 & : kill previous logstash instance
7. sh /home/vishwa/job.sh :  set a crontab job to copy log file from container
8. /usr/share/logstash/bin/logstash -f /home/vishwa/pipe.conf >output.log 2>&1 & run logstash to send latest logs to Elasticsearch & Kibana automatically.

**Job.sh file in azure server :**

```
#!/bin/bash
(crontab -u vishwa -l ; echo "*/1 * * * * docker cp
vishwa_servernode_1:projectManagementSystem.log /home/vishwa/activecollab.log")
crontab -u vishwa -
```

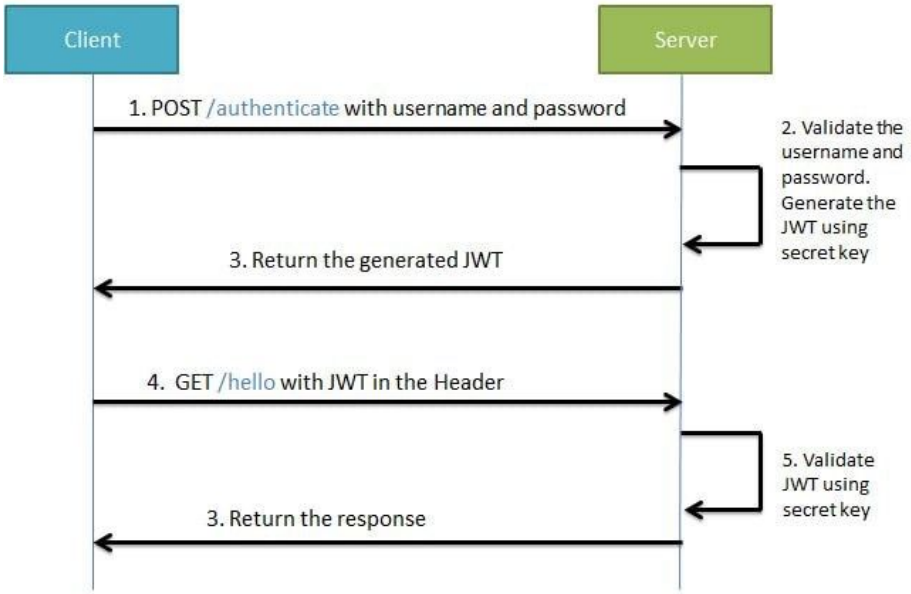
## 4.2.7 Code walkthrough Backend




**Code walkthrough:**
- All API calls are stored in /controllers directory.
- All the business logic is stored in /service directory.
- All the DB schemas are stored in /model directory
- All the JPA repositories access the database /repository directory.
- All the test files are stored in /test directory
- All the JWT configuration details are stored in the /config directory.
- All project configuration details are stored in the application.properties file.
- All data initializations are stored in data.sql file.
- All the workflow files are stored in the .github directory.

## 4.2.8 Login through JWT

JWT, or JSON Web Tokens, is a standard that is mostly used for securing REST APIs. Despite being a relatively new technology, it is gaining rapid popularity. In the JWT auth process, the front end irstly sends some credentials to authenticate itself. The server then checks those credentials, and if they are valid, it generates a JWT and returns it. After this step client has to provide this token in the request's Authorization header in the "Bearer TOKEN" form. The back end will check the validity of this token and authorize or reject requests. The token may also store user roles and authorize the requests based on the given authorities.




JWT configurations are written in /config directory.

# Chapter 5 Continuous Monitoring

The proactive method of observing systems with goal of preventing downtime and outages, is monitoring. It involves measuring current behavior against predetermined baselines. Some of the commonly observed devices are CPU usage, storage capacities, request hits i.e network traffic, etc. by which we can find the root cause of the failure. Here, We used ELK stack monitoring.
Elasticsearch is a modern analytics and search engine which is based on Apache Lucene. Using this we can draw meaningful information from index logs created by applications. We can use elastic to manage, store and search data which can come in handy in Logs, Metrics, Application monitoring, Endpoint security, and Search backend. We can use ELK stack locally or kibana & elasticsearch can be run on the cloud and we can send logs using logstash installed on the host machine. Here, we have used cloud approaches as for installing ELK stack RAM requirements were high.

## 5.1 Using elastic cloud

First we need to create an account on Elasticsearch cloud.

- Go to https://www.elastic.co/ URL and signup.
- Create the first deployment.
- Go with the default configuration.
- Save username & password after the creation of deployment as we will need them hereafter and they will not be shown again.

## 5.2 Parse log file using logstash

We have used Apache log4j2 for log creation, when the application runs on any host respective calculator.log file is created. So after the Deployment of a docker image to a container inside an Ubuntu server, we need to copy the file from container to server and then the server to host for monitoring and feeding to Elastic and Kibana clouds.

1. Get the docker container id

```
$ docker ps -a
```

2. Copy file from docker container to server

```
$ docker cp <container_id>:<source_location>  <destination_location>
```

3. Copy file from server to host machine i.e. Ubuntu server to Ubuntu in my case

```
$ scp <remote_user@ip_address:source_location>  <destination_location>
```

4. To parse a log file via logstash use the following command or you can upload it manually.

```
$ bin/logstash -f /home/vishwa/documents/calculatorproject/calculatorlogger.conf
```

**logger.conf** is a configuration file, it contains the grok pattern , timestamp, cloud link , cloud_id and cloud_auth of users.

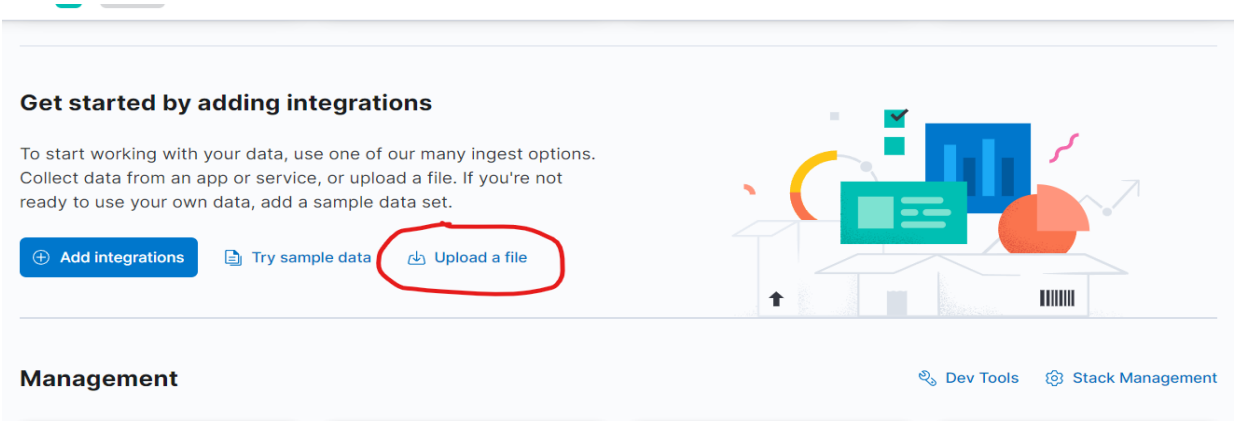
```
input {
  file {
    path => "/home/vishwa/activecollab.log"
    start_position => "beginning"
  }
}

filter {
  grok {
    match => [
      "message", "%{TIMESTAMP_ISO8601:timestamp} \[.*?\] \[%{LOGLEVEL:level}\]
%{GREEDYDATA:logger} \[%{GREEDYDATA:controller}\] \-
\[%{GREEDYDATA:message}\]"
    ]
  }

  date {
    match => ["timestamp", "yyyy-MM-dd HH:mm:ss"]
  }

  mutate {
    remove_field => [timestamp]
  }
}

output {
   elasticsearch {
```

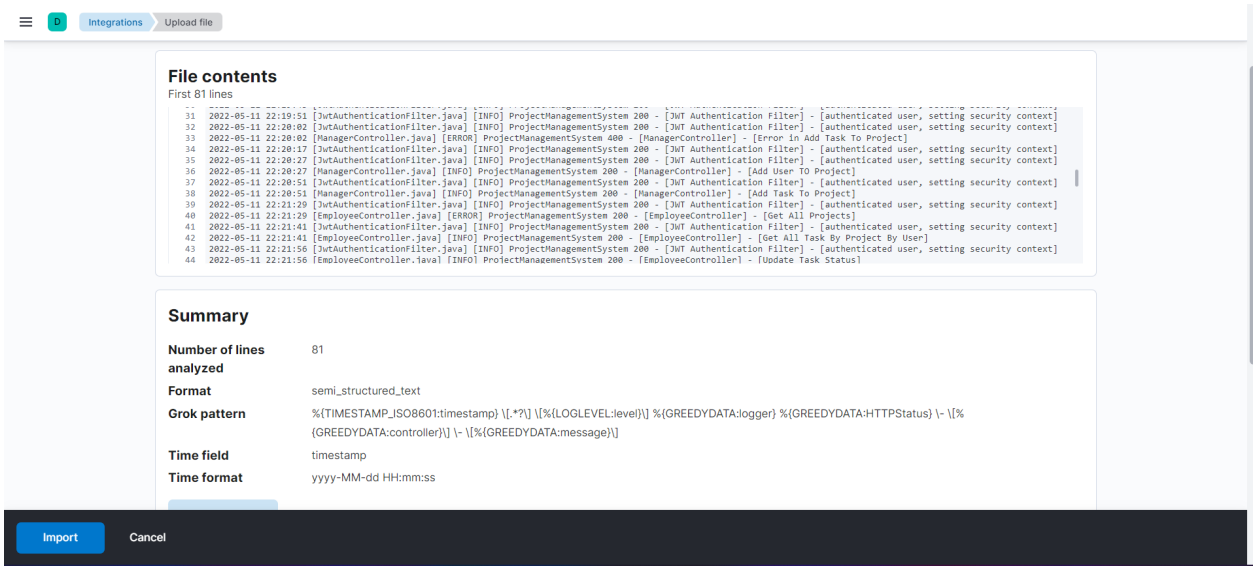
```
    index => "activecollab"
    cloud_id =>
"activecollab:dXMtY2VudHJhbDEuZ2NwLmNsb3VkLmVzLmlvJGFhZjk4YzIzZmY4NDQwODlhN2YzZmUyYzAyN2I2ZDJlJDY0YWRlNjA5NjlkNjRiY2ViYzEwMzk2NmJiYzkwY2Qx"
    cloud_auth =>  "elastic:Q7YSpOKTkjlXPn0TrRmLM5Tu"
  }

 stdout {
  codec => rubydebug
 }
}
```

## 5.3 Visualizing using Kibana

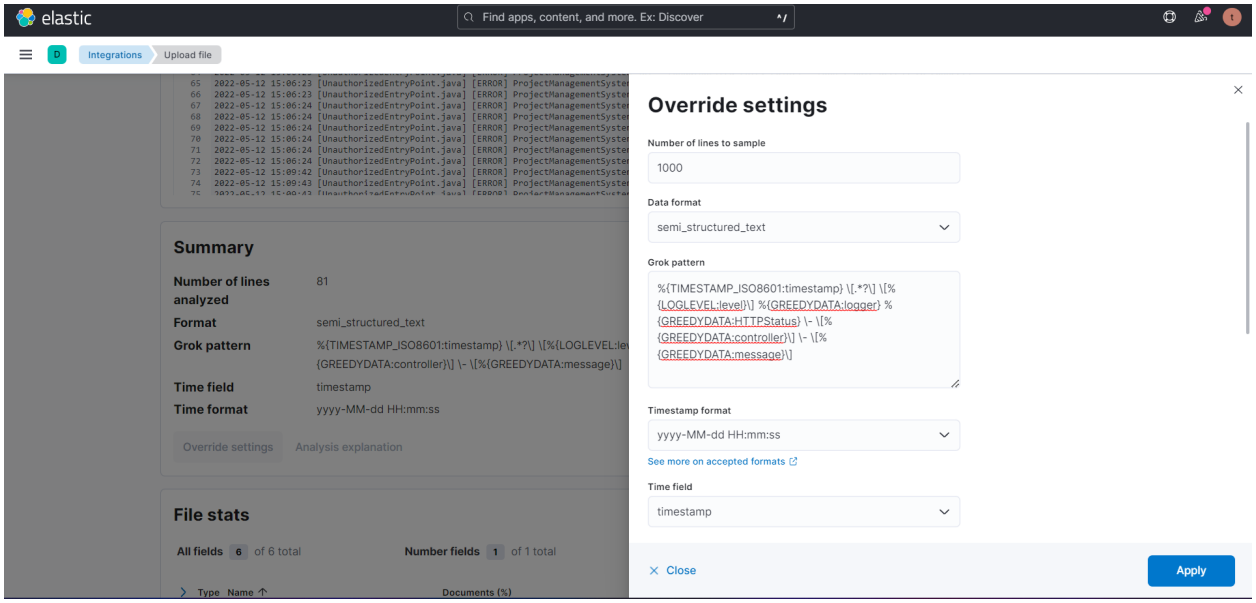
For visualizing logs on Kibana we are  following the following steps:

1. On deployment, in the Home window, press on the upload data TAB.

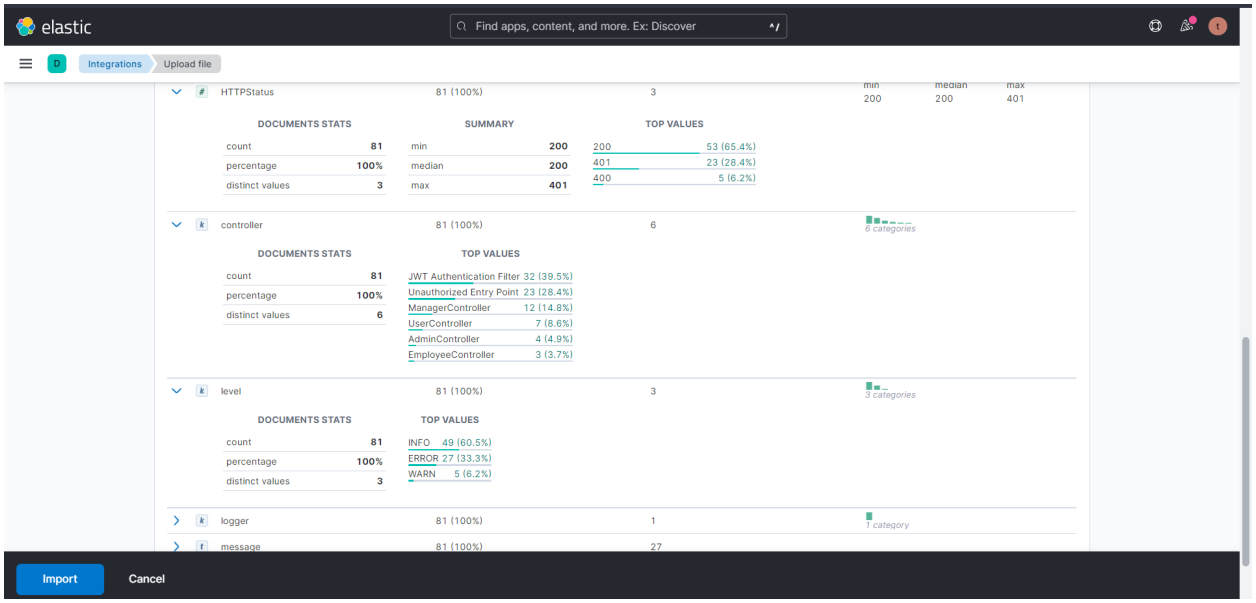




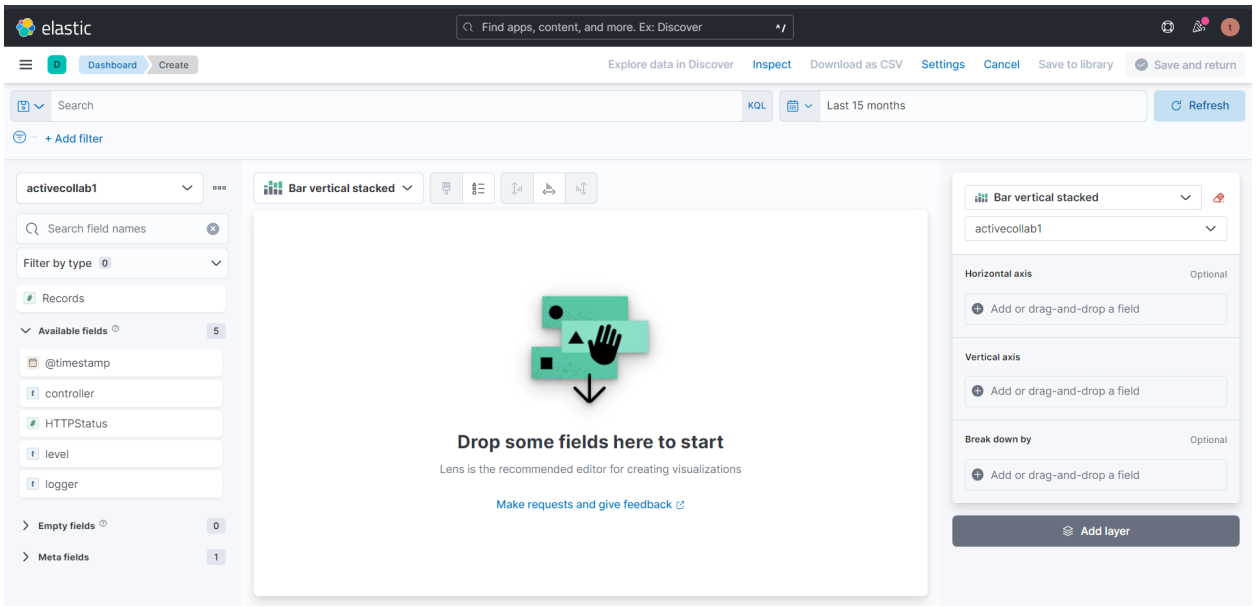

2. Go on override settings options and apply a custom grok pattern and timestamp.



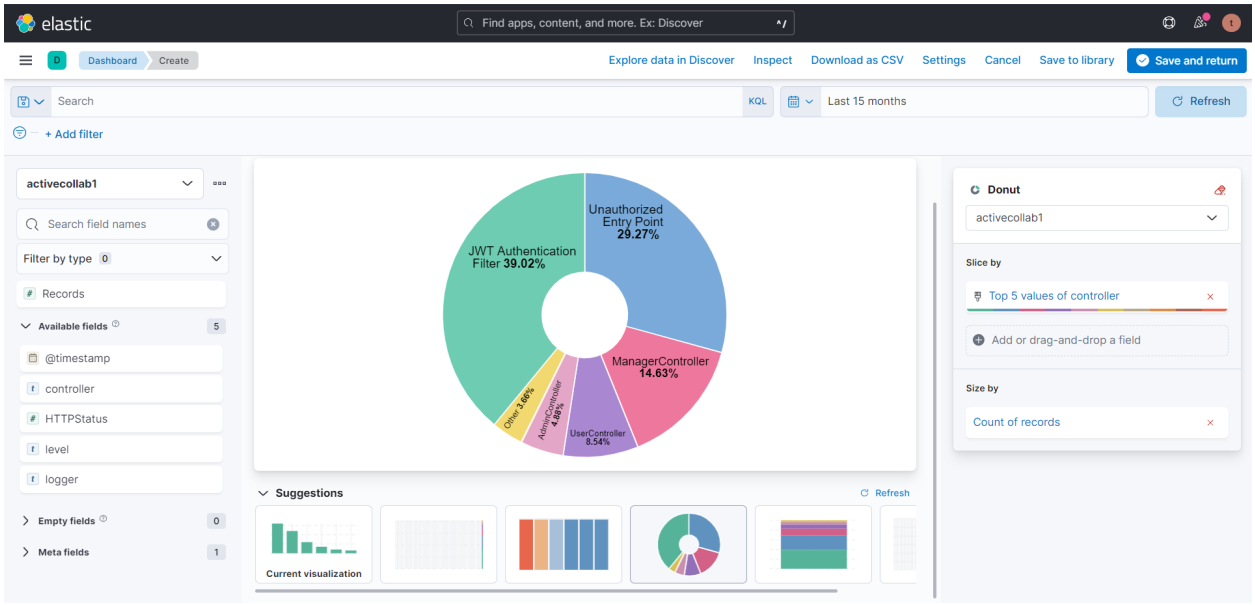
3. controller field will be created according to controllers:



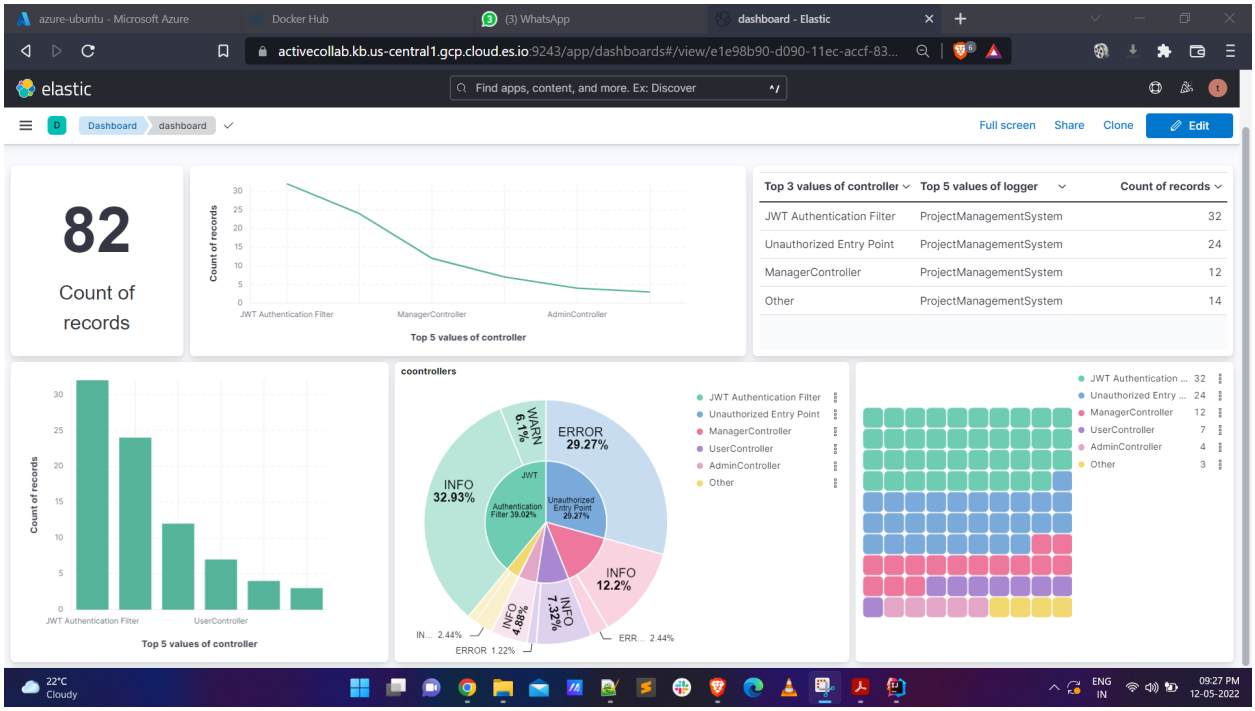

4. Go to the create dashboard optio and choose the time period of logs using calendar option.



5. Create visualizations for dashboards.




6. To create a full dashboard, first, create visualizations and save them by option save & return. After creating & adjusting visualizations, save to create a dashboard.



## 5.4 Continuous monitoring using logstash

For continuous monitoring i.e visualizations should be updated as new logs will get created for that we are using logstash and passing a log file through the .conf file.

```
$ bin/logstash -f logger.conf
```

For copying logfile from container to host we have used crontab

```
#!/bin/bash
(crontab -u vishwa -l ; echo "*/1 * * * * docker cp
vishwa_servernode_1:projectManagementSystem.log /home/vishwa/activecollab.log")
| crontab -u vishwa -
```

# Chapter 6 API Documentation

## 5.1 Description of Modules

All the modules described here require appropriate credentials to login and use the mentioned features which leads to access to the system by authorized and authenticated users only.

1. **Admin**
   In this module, we provide all the administrator rights to the administrator of the system. The administrator can add new employees or managers and can remove them.
2. **Manager**
   In this module, the Project Managers can create new projects, update the status of ongoing projects, assign various tasks to employees, assign employees to projects and solve the queries faced by employees while trying to accomplish a particular task assigned to them.
3. **Employee**
   In this module, the employees can see their tasks, and can update the status of the tasks being performed by them for a particular project.

## 6.2 API Documentation

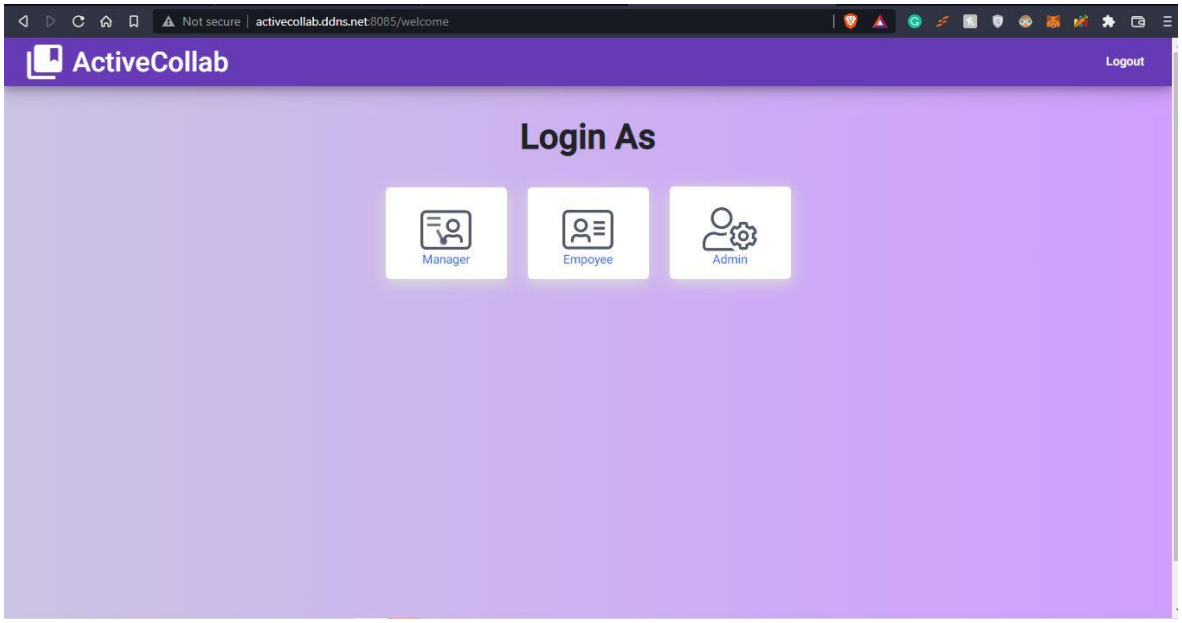
As described above module, We create API documentation for each feature for the respective module. We use the Postman API platform for API testing and documentation. In this document, we have each API call with their example.
**Postman API document**: ActiceCollab API Documentation

# Chapter 7 Application Screenshot

## 7.1 User UI ScreenShot

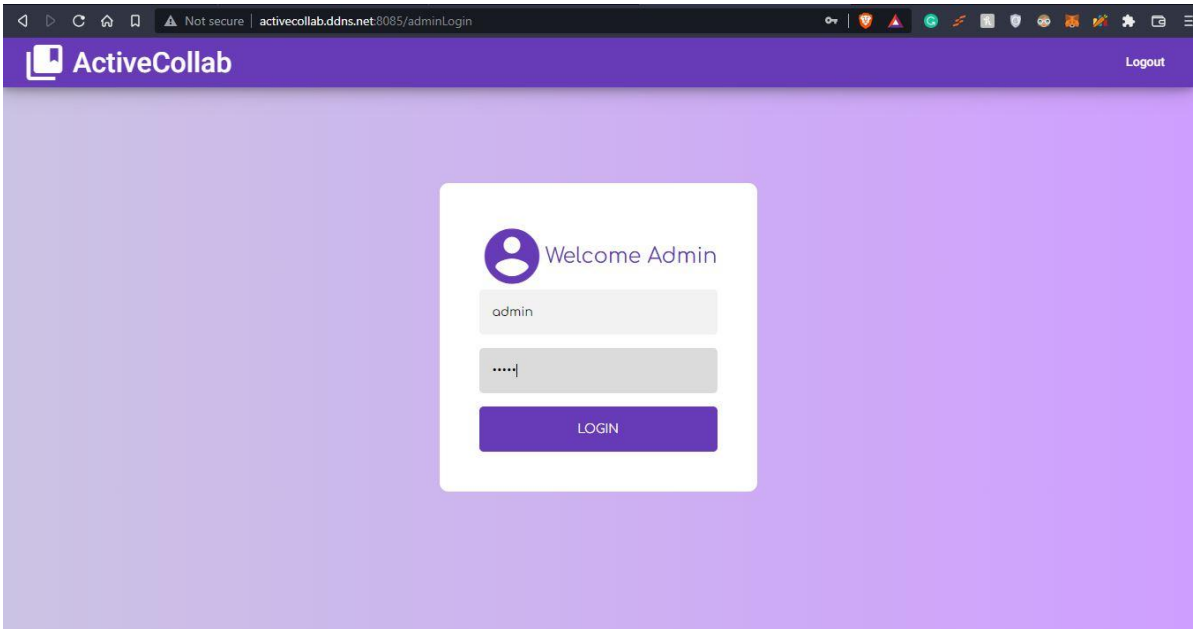
### 7.1.1 Welcome Page

Initially when we run the application it will display a welcome page for login as Manager, Employee, Admin.




### 7.1.2 Login Page

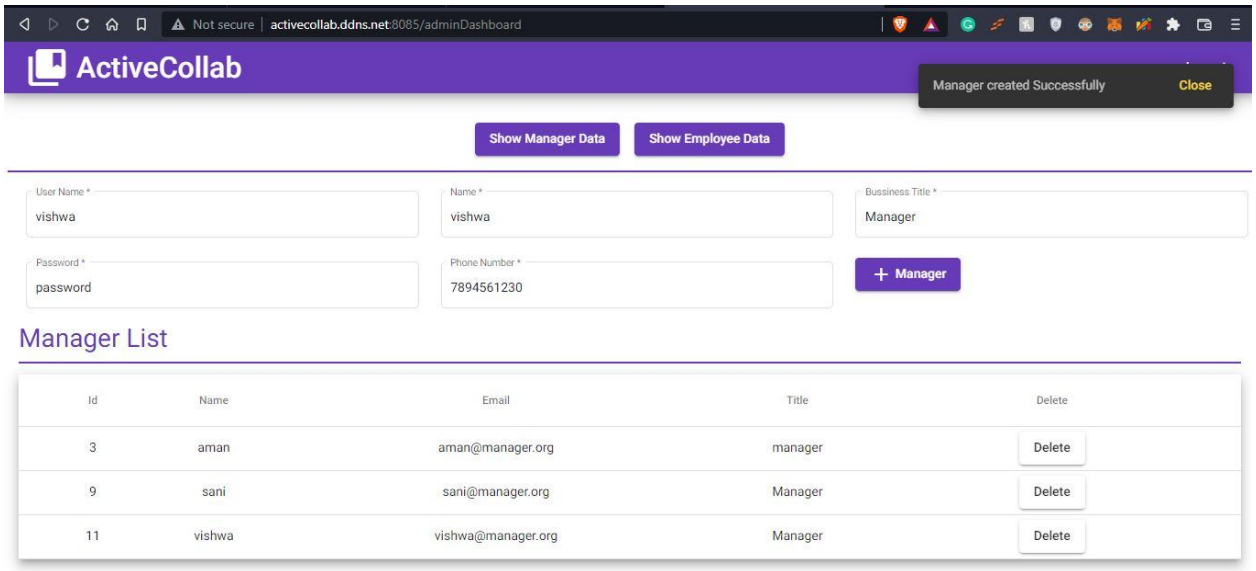
This page will display the login page for Users as per their role.



# 7.2 Admin UI ScreenShot

## 7.2.1 Admin Dashboard Page

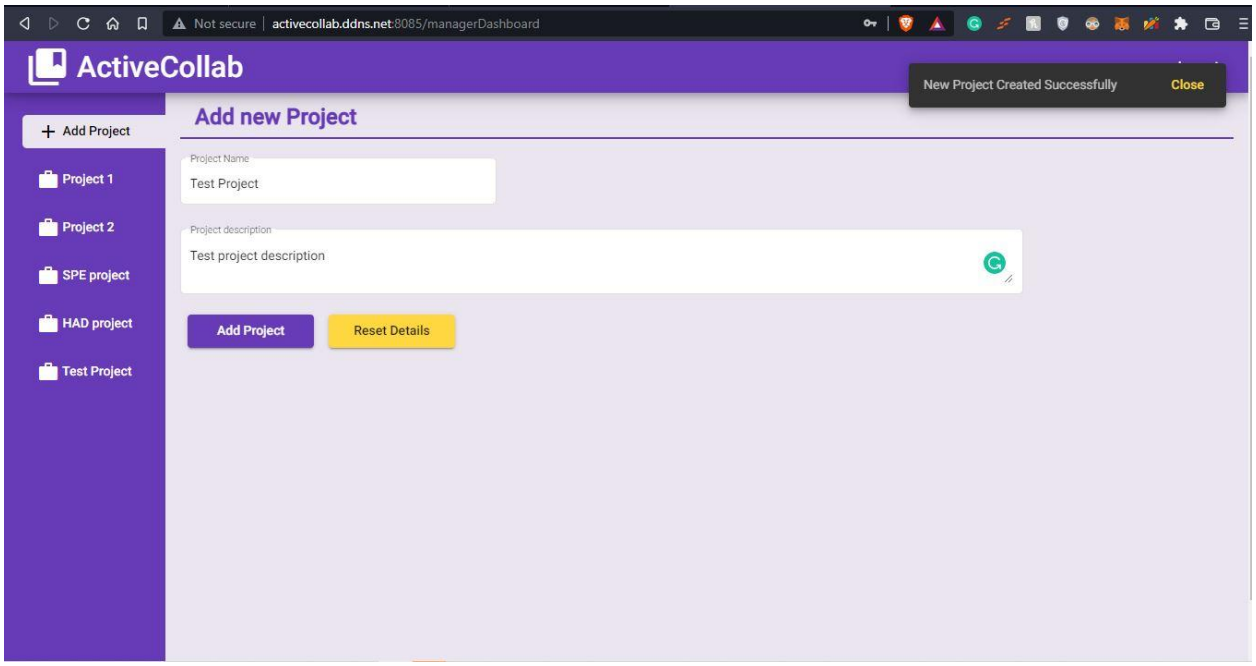
This page shows the admin dashboard with a manager user list and employee user list.




# 7.3 Manager UI ScreenShot

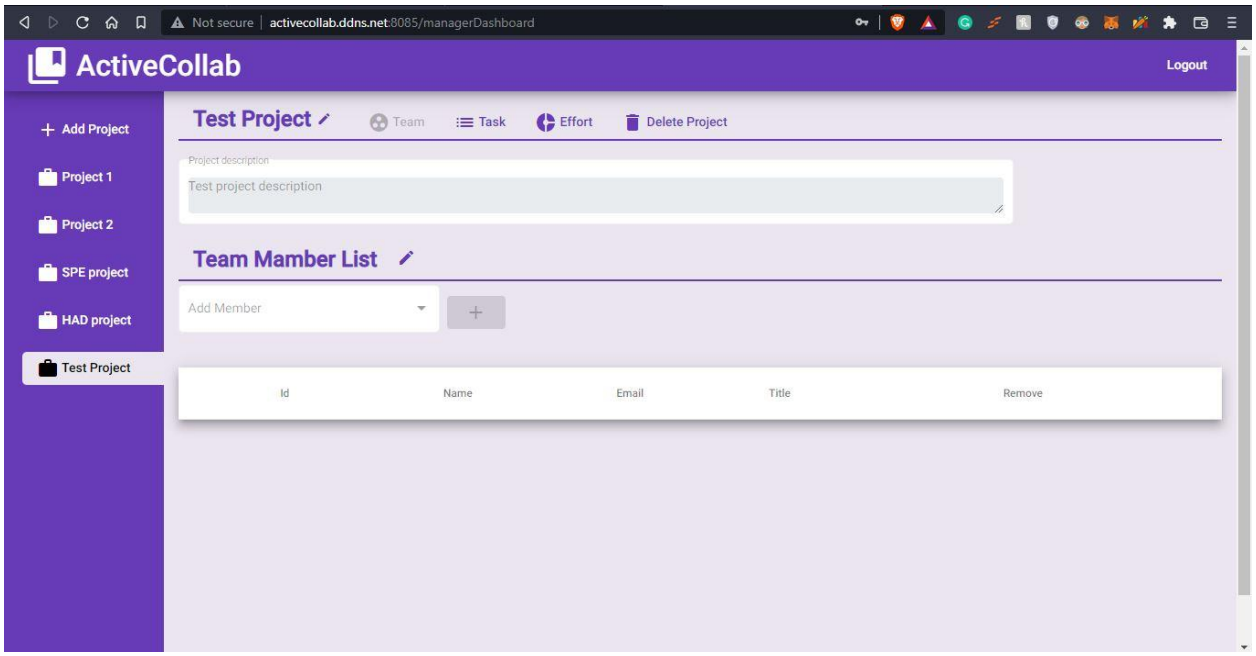
## 7.3.1 Manager Dashboard Page

This page show the manager dashboard with the list of project and add project functionality.



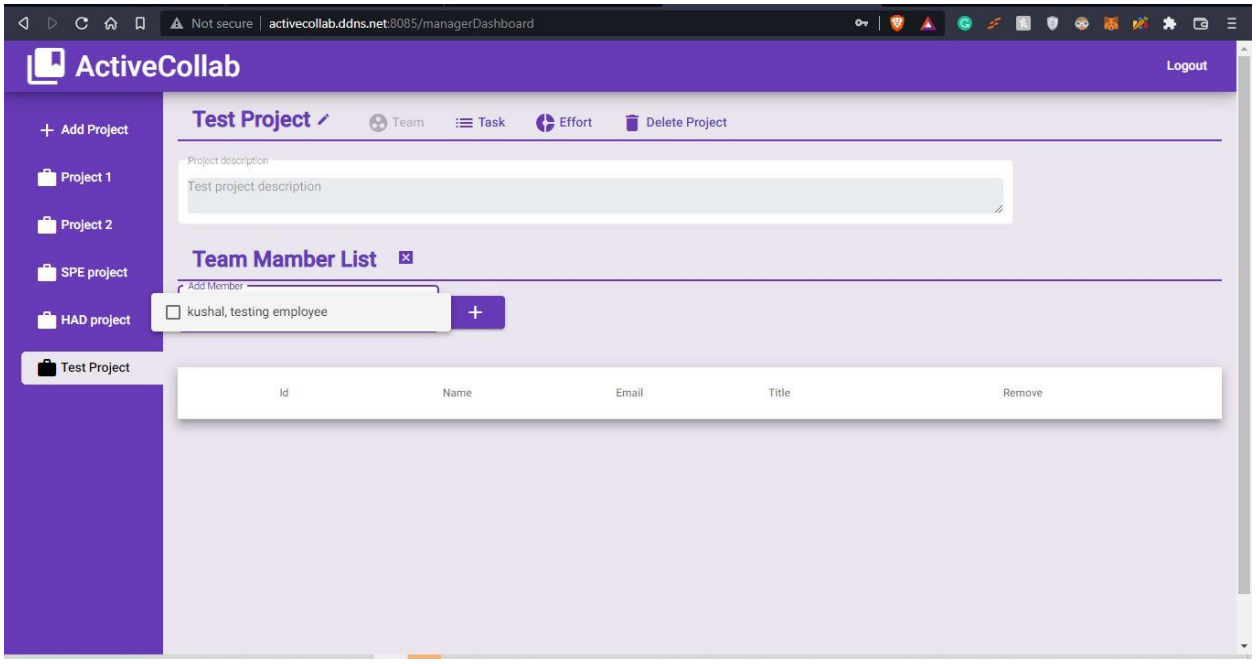
## 7.3.2 Project Dashboard Page

This page shows the project dashboard for the selected project



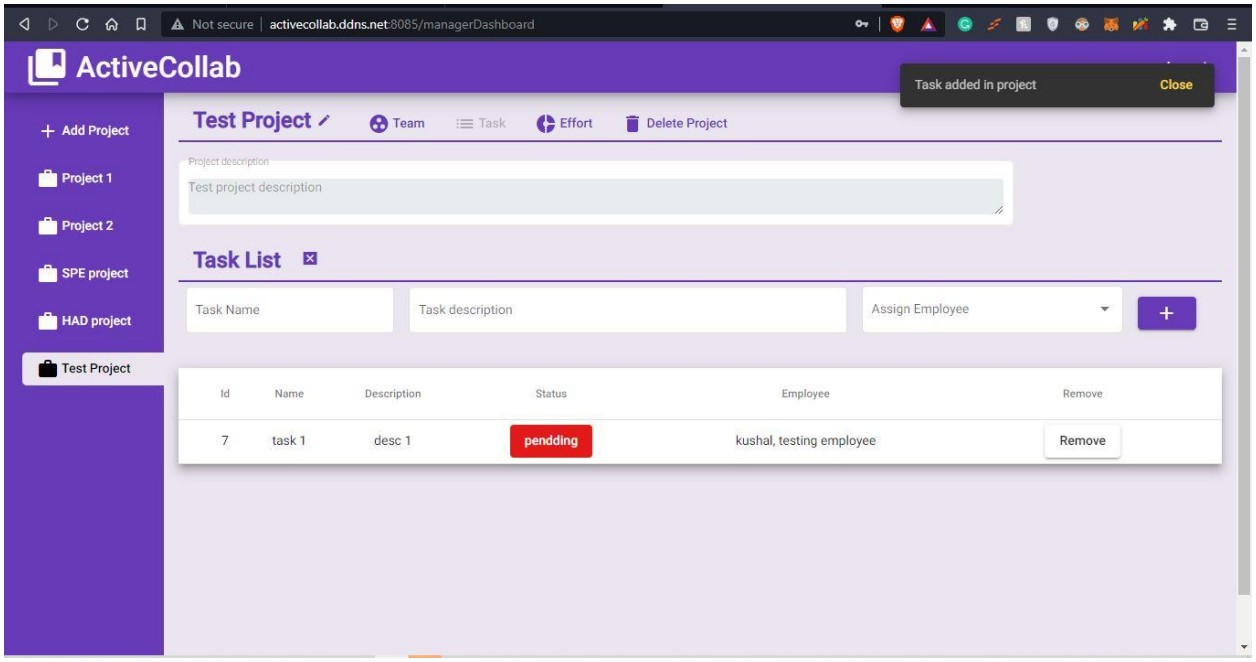

## 7.3.3 Add Employee Page

This page show add employee functionality.



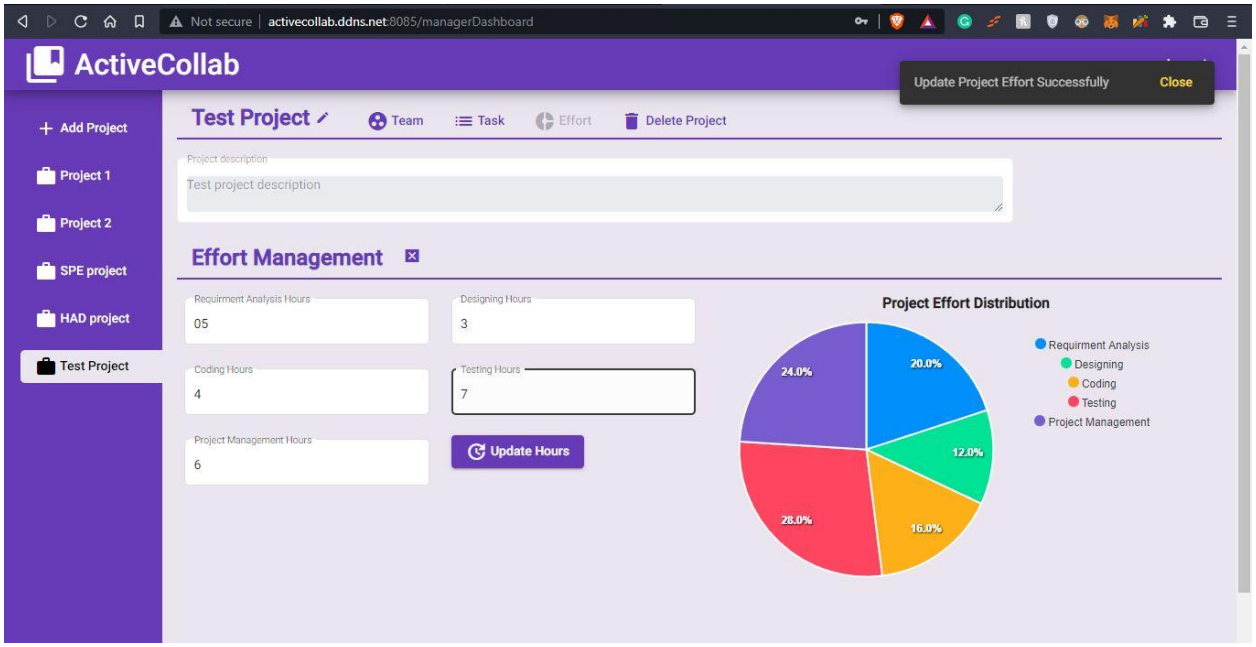

## 7.3.4 Add Task Page

This page shows add task functionality.




## 7.3.5 Update Effort table Page

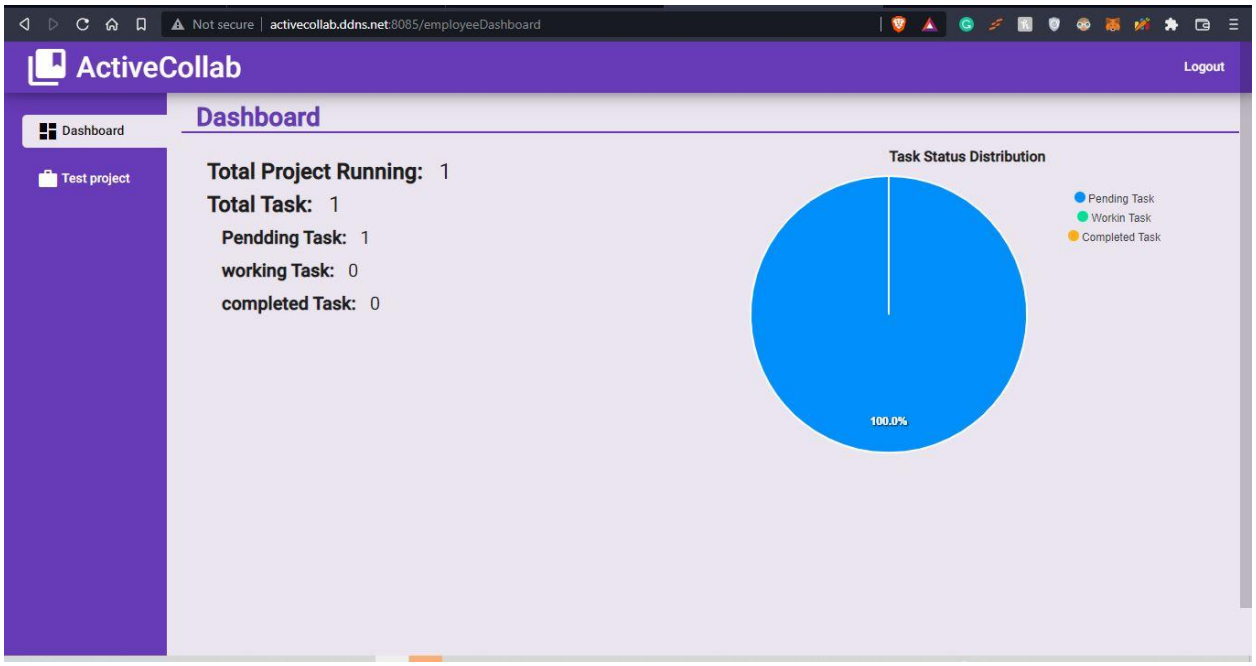
This page show update effort table functionality



# 7.4 Employee UI ScreenShot

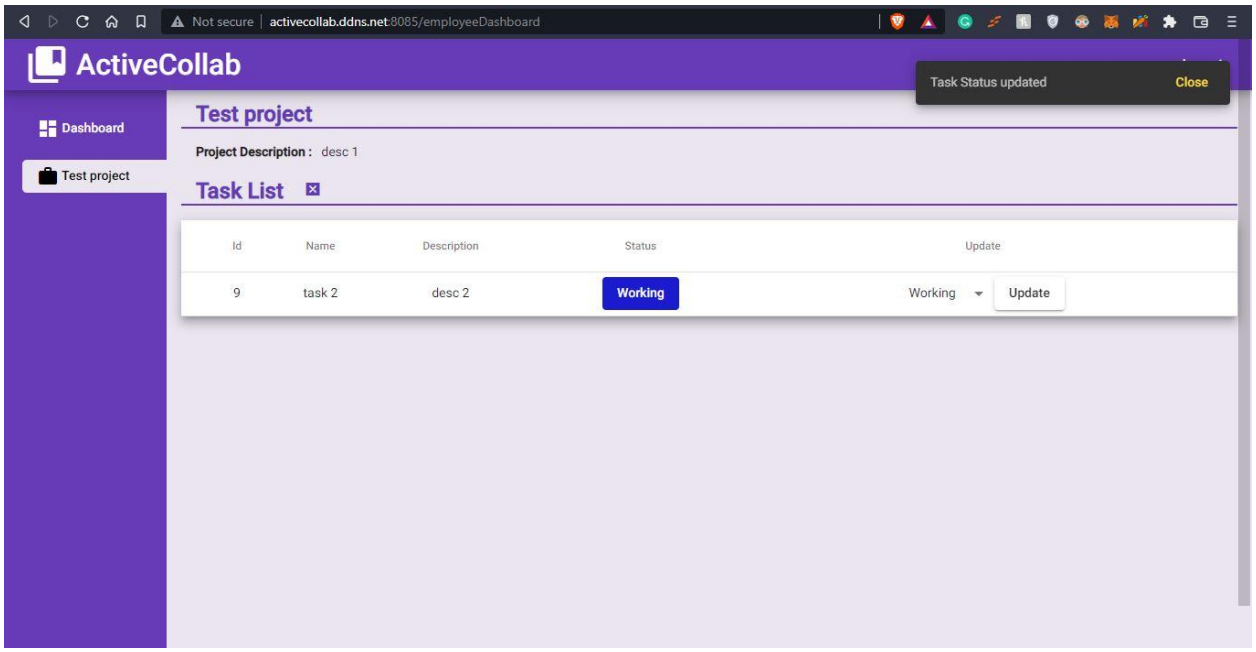
## 7.4.1 Employee Dashboard Page

This page show the employee dashboard with a total number of project and task statistics.




## 7.4.2 Update Task Status  Page

This page show task of the selected projects for the employee.



# Chapter 8 Limitation and Future Work

## 8.1 Limitation

- Admin is required to handle the application
- No Communication functionality between manager and employee.
- Tasks are not divided into further sub-tasks.
- Not any mechanism for rising tickets/queries by users.

## 8.2 Future Work

- Remove admin dependencies
- Add the User information tab and update functionality.
- Email and OTP-based authentication.
- Chat facility for easy and fast communications of employees/managers.
- Improve User experience.
- Add user performance metrics.

# Chapter 9 Conclusion

The functionality implemented in the system was done after understanding all the system modules according to their particular requirements.

**Functionalities that are successfully implemented are:**

- Login
- Registration
- Manage Employees/Managers
- Manage on Project
- Manage Tasks
- Assign Task
- Assign Employee to Project
- Update Task status
- Update Project effort analysis
- Show Employee Statistics

After the successful implementation of the system along with all the modules and functionalities, comprehensive testing was performed to determine the loopholes and possible flaws/errors in the system.

This entire application was automated using DevOps tools like **Github Actions** for continuous integration, **Ansible** for configuration management and deployment, **Docker** for containerization, **Junit** and **Jasmine** for testing, **Microsoft Azure Ubuntu-server VM** for deployment, and finally **ELK stack** for monitoring.

# Bibliography


- **Apache Maven:** https://maven.apache.org/
- **Spring Framework:** https://spring.io/projects/spring-framework
- **Angular Framework:** https://angular.io/
- **Angular Material:** https://material.angular.io/
- **Docker:** https://www.docker.com/
- **Ansible:** https://www.ansible.com/
- **ELK Stack:** https://www.elastic.co/cloud/
- **Stack Overflow:** https://stackoverflow.com/
- https://stackoverflow.com/questions/64199452/logstash-not-showing-any-output-in-ubuntu
- https://www.reddit.com/r/angular/comments/ha4kek/404_not_found_nginx_angular_docker/
- https://blog.bitsrc.io/https-medium-com-adhasmana-how-to-do-ci-and-cd-of-node-js-application-using-github-actions-860007bebae6
- https://stackoverflow.com/questions/66989383/could-not-resolve-dependency-npm-err-peer-angular-compiler11-2-8
- https://stackoverflow.com/questions/71014098/docker-compose-fails-with-copy-failed-stat-usr-src-app-dist-file-does-not-exi
- https://blog.devgenius.io/how-to-build-and-run-a-nodejs-app-with-docker-github-actions-59eb264dfef5