

# SPE Mini Project 1 Report: Scientific Calculator Program

Siddharth Kothari (IMT2021019)

September 25, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Application Introduction . . . . .	2
1.2	Introduction to Devops . . . . .	2
1.3	Advantages of DevOps . . . . .	2
<b>2</b>	<b>Codebase</b>	<b>2</b>
2.1	Directory Structure . . . . .	3
2.2	Unit Testing . . . . .	3
<b>3</b>	<b>DevOps Tools</b>	<b>4</b>
3.1	Git and Github for Version Control . . . . .	5
3.2	Maven - Build Automation Tool . . . . .	5
3.3	Jenkins for CI/CD . . . . .	7
3.4	Docker for Containerization . . . . .	8
3.5	Ansible for Application Deployment . . . . .	9
3.6	Ngrok for Secure Tunneling . . . . .	10
<b>4</b>	<b>Configuration Steps</b>	<b>11</b>
4.1	Git Installation . . . . .	11
4.1.1	Ubuntu . . . . .	11
4.1.2	Mac . . . . .	11
4.1.3	Windows . . . . .	11
4.2	Ngrok Installation . . . . .	11
4.3	Setup of Webhooks . . . . .	11
4.4	Maven and IntelliJ IDEA Installation . . . . .	11
4.5	Project Setup . . . . .	13
4.6	Jenkins Installation . . . . .	13
4.7	Configuration of Plugins for Jenkins . . . . .	13
4.8	Docker Installation . . . . .	14
4.9	Credentials for Docker and Github in Jenkins . . . . .	14
4.10	Creating a Pipeline Project Jenkins . . . . .	17
4.11	Ansible Installation . . . . .	17
4.12	Running the Application . . . . .	17
<b>5</b>	<b>Results</b>	<b>17</b>
5.1	Outputs of the Run Stages from Jenkins . . . . .	17
5.2	Outputs on Dockerhub . . . . .	19
5.3	Containers and Images Created . . . . .	19
5.4	Outputs of the Application . . . . .	19
<b>6</b>	<b>Relevant Links</b>	<b>21</b>
6.1	Project Github . . . . .	21
6.2	Dockerhub Link . . . . .	21

# 1 Introduction

## 1.1 Application Introduction

I have built a simple scientific calculator app which is capable of performing the following functions -

1. Given an integer, it returns the factorial of the number. If the integer is  $< 0$ , then the function returns -1.
2. Given a decimal number, it can return the square root of the number. If the number is  $< 0$ , it returns Double.NaN.
3. Given a base and an exponent (decimal numbers), it can calculate the base raised to the power exponent.
4. Given a decimal number, it can return the natural logarithm of the number. If the number provided is  $< 0$ , it returns Double.NaN.

This report gives an overview of the application along with the DevOps tools used in the development of this project. The primary tools I have utilised in this project are Jenkins, Github, Docker, Ansible, and Maven.

## 1.2 Introduction to Devops

DevOps is a combination of tools, and methodologies that aim to improve the collaboration between software development (Dev) and IT operations (Ops). It focuses on automating and streamlining the processes of software development, testing, deployment, and infrastructure management, creating a faster, more efficient, and reliable workflow. There are a number of key components of DevOps, such as -

1. **Collaboration:** DevOps fosters strong collaboration between development, operations, and other teams (e.g., security) to break down silos.
2. **Automation:** Automating repetitive tasks such as testing, integration, deployment, and infrastructure provisioning is central to DevOps.
3. **Continuous Integration (CI):** Developers frequently integrate code into a shared repository, where it's automatically tested.
4. **Continuous Delivery/Deployment (CD):** Software is automatically prepared for release and deployed in short cycles.
5. **Monitoring and Logging:** System and application monitoring helps detect issues early and maintain reliability and performance.

## 1.3 Advantages of DevOps

There are a number of reasons to use DevOps, some of which are listed below.

1. **Faster Time to Market:** Automating processes like testing and deployment means you can release new features and bug fixes faster.
2. **Increased Efficiency:** Continuous integration, delivery, and deployment remove manual steps, reducing human error and saving time.
3. **Improved Collaboration:** DevOps practices bring development, operations, and QA teams together, which results in better communication and coordination.
4. **Higher Quality Software:** Automated testing and monitoring help detect and fix issues earlier in the development cycle.
5. **Scalability and Flexibility:** DevOps supports infrastructure management and scaling, making it easier to handle changing business requirements.

# 2 Codebase

The Codebase is written in Java, and uses Maven for project management. Unit Test are written using JUnit.

## 2.1 Directory Structure

The figure below shows the directory structure. There are 2 main directories - src and target. src further contains directories which contain the main code, and the test directory contains the unit tests. Calculator.java contains the main function, and it instantiates a CalculatorService object, which performs all the computations.

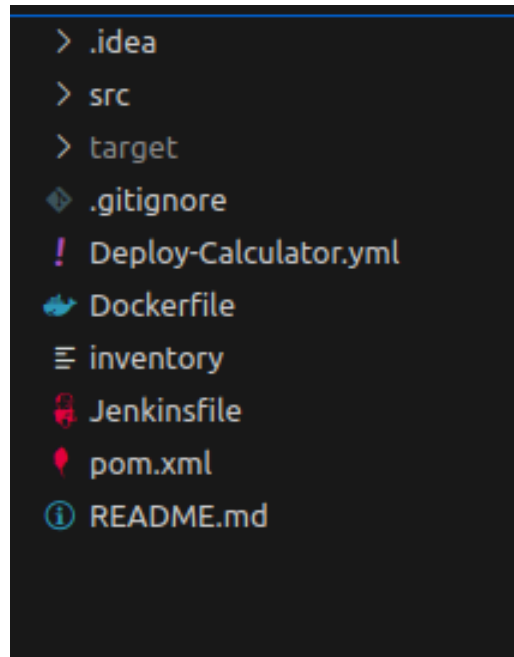


Figure 1: Directory Structure

The code for the CalculatorService.java class is given below -

```
1 public class CalculatorService {
2     public Integer factorial(Integer n){
3         if (n == 1 || n == 0) return 1;
4         else if (n<1) return -1;
5         return n*factorial(n-1);
6     }
7
8     public Double power(Double base, Double exp){
9         return Math.pow(base, exp);
10    }
11
12    public Double logarithm(Double n){
13        if (n<=0) return Double.NaN;
14        return Math.log(n);
15    }
16
17    public Double sqrt(Double n){
18        if (n<0) return Double.NaN;
19        return Math.sqrt(n);
20    }
21 }
```

## 2.2 Unit Testing

The Unit test for the CalculatorService class are written in the test directory., using JUnit. The code for the CalculatorServiceTest.java class is provided below.

```

22 public class CalculatorTest {
23     // Test for square_root method
24     private CalculatorService calculatorService;
25
26     @Before
27     public void setup() {
28         calculatorService = new CalculatorService();
29     }
30
31     @Test
32     public void testSquareRootPositive() {
33         assertEquals(5.0, calculatorService.sqrt(25.0), 0.0001);
34     }
35
36     @Test
37     public void testSquareRootNegative() {
38         assertTrue(Double.isNaN(calculatorService.sqrt(-25.0)));
39     }
40
41     // Test for factorial method
42     @Test
43     public void testFactorialPositive() {
44         assertEquals(120.0, calculatorService.factorial(5), 0.0001);
45     }
46
47     @Test
48     public void testFactorialNegative() {
49         assertEquals(Optional.ofNullable(calculatorService.factorial(-5)), Optional.of
50             (-1));
51     }
52
53     // Test for natural_log method
54     @Test
55     public void testNaturalLogPositive() {
56         assertEquals(Math.log(10), calculatorService.logarithm(10.0), 0.0001);
57     }
58
59     @Test
60     public void testNaturalLogNonPositive() {
61         assertTrue(Double.isNaN(calculatorService.logarithm(0.0)));
62     }
63
64     // Test for exponent method
65     @Test
66     public void testExponentPositive() {
67         assertEquals(8.0, calculatorService.power(2.0, 3.0), 0.0001);
68     }
69
70     @Test
71     public void testExponentZero() {
72         assertEquals(1.0, calculatorService.power(0.0, 0.0), 0.0001);
73     }
74 }

```

### 3 DevOps Tools

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

### 3.1 Git and Github for Version Control

**Git** is a distributed version control system that allows multiple developers to work on a project simultaneously. It tracks changes in the source code, enabling developers to collaborate on a project, manage different versions of files, and revert to earlier versions if needed. **Github** is a web-based platform built around Git that provides hosting for Git repositories. It offers additional tools for collaboration, code review, and project management, making it a key resource for both open-source and private software projects. Figure 2 shows a snapshot of the repo.

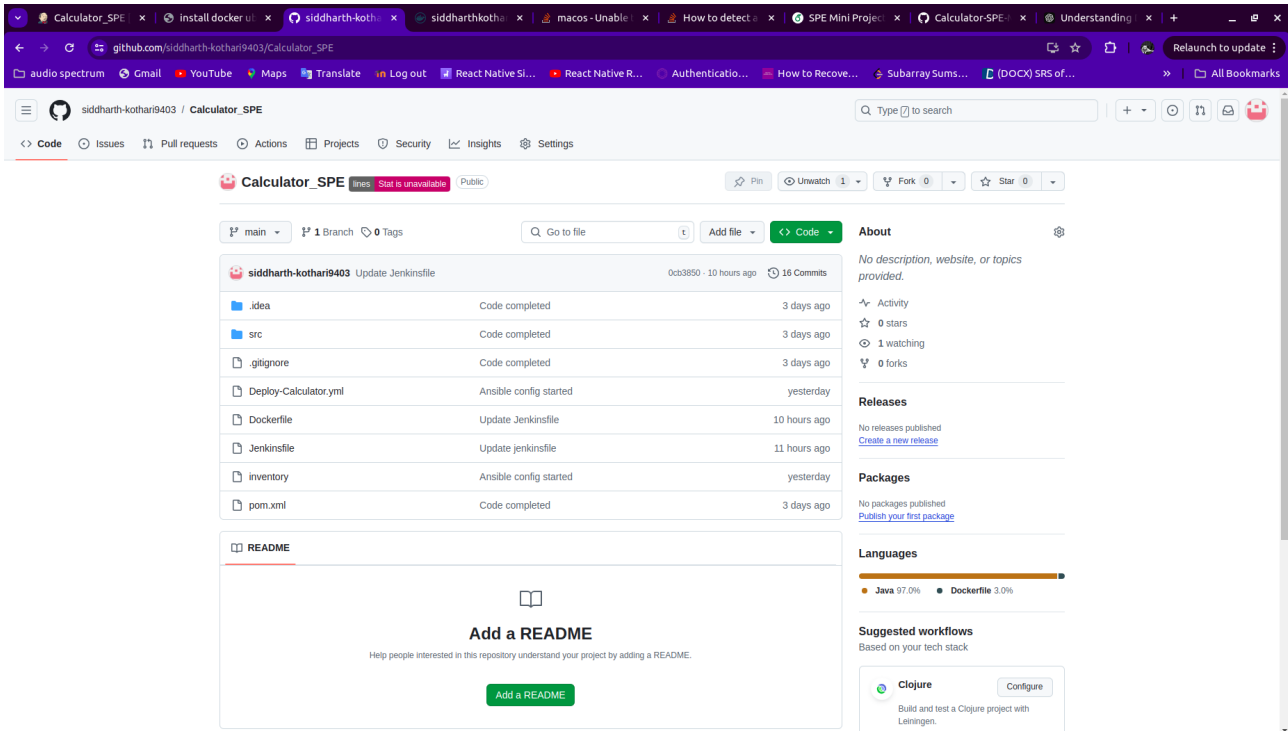


Figure 2: Screenshot of Repository

### 3.2 Maven - Build Automation Tool

**Maven** is a build automation and project management tool primarily used for Java projects, though it can be used with other programming languages as well. It simplifies the process of building, managing dependencies, and handling project lifecycle tasks, such as compiling code, running tests, packaging applications, and deploying them. Maven uses a pom.xml file that contains:

1. Project Information: Basic project details such as name, version, and description.
2. Dependencies: A list of external libraries the project needs.
3. Build Plugins: Tools for tasks like compiling code, running tests, and packaging.
4. Build Phases: The phases of the build lifecycle, such as compile, test, package, and deploy.

When Maven runs, it reads the pom.xml file and executes tasks according to the build lifecycle. Below is the code for this project's pom.xml file.

Listing 1: pom.xml file

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
      /maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>org.example</groupId>
8     <artifactId>Calculator_SPE</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>17</maven.compiler.source>
13         <maven.compiler.target>17</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <dependencies>
18         <dependency>
19             <groupId>junit</groupId>
20             <artifactId>junit</artifactId>
21             <version>4.12</version>
22             <scope>test</scope>
23         </dependency>
24     </dependencies>
25
26     <build>
27         <plugins>
28             <plugin>
29                 <!-- Build an executable JAR -->
30                 <groupId>org.apache.maven.plugins</groupId>
31                 <artifactId>maven-jar-plugin</artifactId>
32                 <version>3.1.0</version>
33                 <configuration>
34                     <archive>
35                         <manifest>
36                             <mainClass>Calculator</mainClass>
37                         </manifest>
38                     </archive>
39                 </configuration>
40             </plugin>
41         </plugins>
42     </build>
43
44 </project>

```

We use Maven for a number of reasons, including the following -

1. **Simplified Dependency Management:** Instead of manually downloading and managing library dependencies, Maven handles this automatically. It resolves transitive dependencies, meaning it can also fetch dependencies that your dependencies require.
2. **Standardized Project Structure:** Maven enforces a standard directory layout (like placing source files in `src/main/java` and tests in `src/test/java`), which improves consistency across projects and reduces configuration effort.
3. **Build Lifecycle Management:** Maven provides a clear and automated build lifecycle that ensures tasks like compiling, testing, and packaging are handled consistently.
4. **Extensibility:** With a rich ecosystem of plugins, Maven can be extended to support a variety of tasks, from code analysis to deployment.
5. **Integration with CI/CD Tools:** Maven integrates easily with continuous integration/continuous delivery (CI/CD) tools like Jenkins, Bamboo, and GitLab CI to automate testing, building, and deploying projects.

### 3.3 Jenkins for CI/CD

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

1. **Automation of Build and Testing:** Jenkins automates the process of compiling code, running tests, and building artifacts, ensuring consistency and minimizing manual effort.
2. **Plugins:** Jenkins has a rich ecosystem of plugins that allow integration with version control systems (e.g., Git), build tools (e.g., Maven), test frameworks, and deployment systems.
3. **Continuous Integration (CI):** Jenkins automatically builds and tests the code whenever changes are pushed to the version control repository, helping catch bugs early.
4. **Continuous Delivery (CD):** Jenkins facilitates the deployment of code to production or staging environments in an automated fashion.
5. **Pipeline as Code:** Jenkins supports "pipeline as code" using Jenkinsfiles, allowing developers to define build pipelines using simple code.
6. **Extensibility:** Jenkins can be integrated with popular DevOps tools such as Docker, Kubernetes, Ansible, and AWS, making it highly adaptable to different workflows.

The Jenkinsfile I have used for this project is shown below -

Listing 2: Jenkinsfile

```
1 pipeline{
2     environment{
3         DOCKERHUB_CRED = credentials("Dockerhub-Credentials-ID")
4         GITHUB_REPO_URL = 'https://github.com/siddharth-kothari9403/Calculator_SPE.git'
5     }
6     agent any
7     stages{
8
9         stage("Stage 1 : Git Clone"){
10             steps{
11                 script{
12                     git branch : 'main', url : "${GITHUB_REPO_URL}"
13                 }
14             }
15         }
16
17         stage("Stage 2 : Maven Build"){
18             steps{
19                 sh 'mvn clean install'
20             }
21         }
22
23         stage("Stage 3 : Build Docker Image"){
24             steps{
25                 sh "docker build -t siddharthkothari9403/calculator:latest ."
26             }
27         }
28
29         stage("Stage 4 : Push Docker Image to Dockerhub"){
30             steps{
31                 sh 'echo $DOCKERHUB_CRED_PSW | docker login -u $DOCKERHUB_CRED_USR --password-stdin'
32                 sh "docker push siddharthkothari9403/calculator:latest"
33             }
34         }
35
36         stage("Stage 5 : Clean Unwanted Docker Images"){
37             steps{
38                 sh "docker container prune -f"
39                 sh "docker image prune -a -f"
40             }
41         }
42     }
43 }
```

```

42
43     stage( 'Stage 6 : Ansible Deployment' ) {
44         steps {
45             ansiblePlaybook colorized: true ,
46                 credentialsId: 'localhost' ,
47                 installation: 'Ansible' ,
48                 inventory: 'inventory' ,
49                 playbook: 'Deploy-Calculator.yml'
50         }
51     }
52 }
53 }

```

There are 6 stages in this pipeline, which are described below -

#### 1. Stage 1: Git Clone

In this stage, we simply go to the public Github Repository, and clone it, to perform the remaining commands.

- **Purpose:** To ensure cloning of the Github Repository to carry out later stages smoothly.

#### 2. Stage 2: Maven Build

In this stage, the pipeline runs a Maven command. This compiles the source code, runs tests, and packages the code into a deployable artifact (e.g., JAR or WAR file). Maven also resolves dependencies and ensures that the project is in a clean state before building.

- **Purpose:** To ensure that the project compiles, tests pass, and the code is packaged properly.

#### 3. Stage 3: Build Docker Image

This stage builds a Docker image using the Dockerfile present in the project's root directory. The command creates a Docker image.

- **Purpose:** To package the application into a Docker container, allowing for consistent deployment across environments.

#### 4. Stage 4: Push Docker Image to Dockerhub

In this stage, Jenkins pushes the built Docker image to DockerHub, a container registry. It first logs into DockerHub using credentials stored in Jenkins. It then pushes the image built in the previous step to the DockerHub repository.

- **Purpose:** To store the Docker image in a central registry (DockerHub) so it can be pulled from other environments (e.g., production).

#### 5. Stage 5: Clean Unwanted Docker Images

This stage cleans up unused or unnecessary Docker containers and images.

- **Purpose:** To free up disk space and ensure the server doesn't accumulate unnecessary Docker images and containers.

#### 6. Stage 6: Ansible Deployment

The final stage deploys the application using Ansible, an automation tool for configuration management and application deployment. The command runs an Ansible playbook that handles the deployment of the Docker image to the target servers.

- **Purpose:** To automate the deployment of the Dockerized application using Ansible, ensuring it is deployed correctly and consistently.

### 3.4 Docker for Containerization

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



1. **Containerization:** Docker packages applications and their dependencies into containers, ensuring that the application runs consistently in different environments.
2. **Portability:** Containers can run on any machine that has Docker installed, regardless of the underlying infrastructure, whether it's a developer's laptop, a cloud environment, or a production server.
3. **Isolation:** Each container is isolated, meaning applications in separate containers do not interfere with each other. This leads to better security and resource management.
4. **Version Control:** Docker allows developers to version their container images, making it easy to roll back to previous versions if needed.

The dockerfile for this application is provided below. It simply fetches the openjdk:20 image from dockerhub, the jar files created by Maven, and runs a command to make the image.

Listing 3: Dockerfile

```
1 FROM openjdk:20
2
3 COPY ./target/Calculator.SPE-1.0-SNAPSHOT.jar ./
4
5 WORKDIR ./
6
7 CMD ["java", "-jar", "Calculator.SPE-1.0-SNAPSHOT.jar"]
```

Docker is used for a number of reasons, including the following -

1. **Consistency Across Environments:** Docker ensures that an application runs the same way in any environment (development, testing, or production), avoiding issues like "it works on my machine."
2. **Rapid Deployment:** Docker containers can be started quickly, reducing the time it takes to deploy and scale applications.
3. **Microservices Architecture:** Docker is ideal for deploying microservices, where each microservice runs in its own container, and services can be managed independently.
4. **Isolation and Security:** By running each application in its own container, Docker provides process and resource isolation, improving security and avoiding conflicts between applications.
5. **Simplified CI/CD Pipelines:** Docker can be integrated into Continuous Integration (CI) and Continuous Delivery (CD) pipelines, allowing automated testing and deployment of applications in a consistent environment.
6. **Resource Efficiency:** Docker containers use fewer resources compared to traditional virtual machines, as they share the OS kernel of the host system.

### 3.5 Ansible for Application Deployment

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

1. **Automation:** Ansible automates repetitive IT tasks such as server provisioning, application deployment, and configuration management, reducing manual intervention.
2. **Ease of Use:** With a simple declarative syntax (YAML), Ansible is easy to learn and use, making it accessible for developers and system administrators alike.
3. **Consistency and Reliability:** Ansible playbooks ensure that tasks are executed in the same way every time, reducing the chances of errors and inconsistencies.
4. **Infrastructure as Code (IaC):** Ansible follows the IaC model, allowing system configurations to be version-controlled and treated as code.

5. **Cross-Platform:** Ansible can manage systems across different platforms (Linux, Windows, macOS) using a single playbook, making it highly versatile.
6. **Flexibility:** It can handle both small-scale and enterprise-level infrastructures, supporting multiple cloud platforms, virtual machines, and physical servers.

The .yml file for this project is provided below. It pulls the relevant image from dockerhub, and creates a container from it.

Listing 4: Deploy-Calculator.yml file

```
1 ---
2 - name: Pull Docker Image of Calculator
3   hosts: all
4
5   tasks:
6     - name: Start Docker Service
7       service:
8         name: docker
9         state: started
10
11    - name: Pull Image
12      shell: docker pull siddharthkothari9403/calculator:latest
13
14    - name: Run the container
15      shell: docker create -it --name Calculator siddharthkothari9403/calculator
```

### 3.6 Ngrok for Secure Tunneling

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

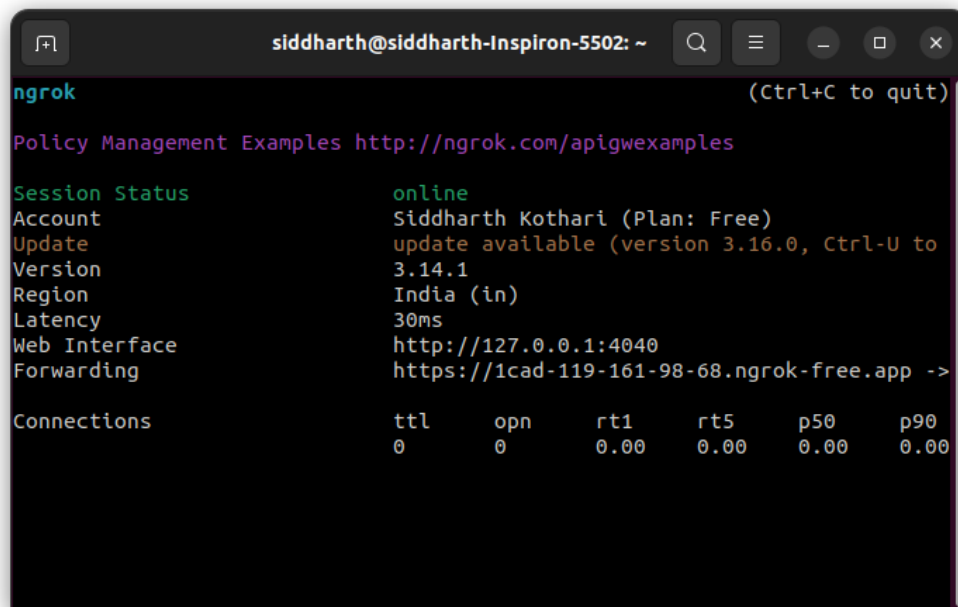


Figure 3: Screenshot of Ngrok Running

It is primarily used in this application for configuring webhooks. We create a webhook in the Github repository, which sends all updates to the provided URL. The URL given is a temporary one created by Ngrok, which provides a tunnel to localhost port 8080, which is where Jenkins is running. By providing relevant Github credentials to Jenkins, it can then listen to any changes in the repository, which it is notified of via the Webhook. A snapshot of the same is shown in figure 3.

## 4 Configuration Steps

### 4.1 Git Installation

#### 4.1.1 Ubuntu

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

Listing 5: Command for Git Version

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

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

Listing 6: Command for Git Installation

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

#### 4.1.2 Mac

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

Listing 7: Command for Git Installation

```
1 $ brew install git
```

#### 4.1.3 Windows

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

### 4.2 Ngrok Installation

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

### 4.3 Setup of Webhooks

Setting Up of Webhooks involves the following steps -

1. Open your Github Profile and go to Settings → Developer Settings → Personal Access Tokens → Tokens (Classic). You can create a new Personal Access Token from here. We should ensure to provide it the required permissions. This will be used later.
2. Open the terminal, and run the command below. This will provide us with a temporary url TEMPORARY\_URL which we will use in this step.

Listing 8: Command for Ngrok Tunneling

```
1 $ ngrok http 8080
```

3. Open the repository which we want to attach webhooks to. Go to Settings → Webhooks → Add Webhook. Here we can specify the URL to be TEMPORARY\_URL/github-webhooks/ This allows us to notify the port 8080 (where jenkins will be running) about any changes in the Repository. The page for the same is shown in Figure 4.

### 4.4 Maven and IntelliJ IDEA Installation

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

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

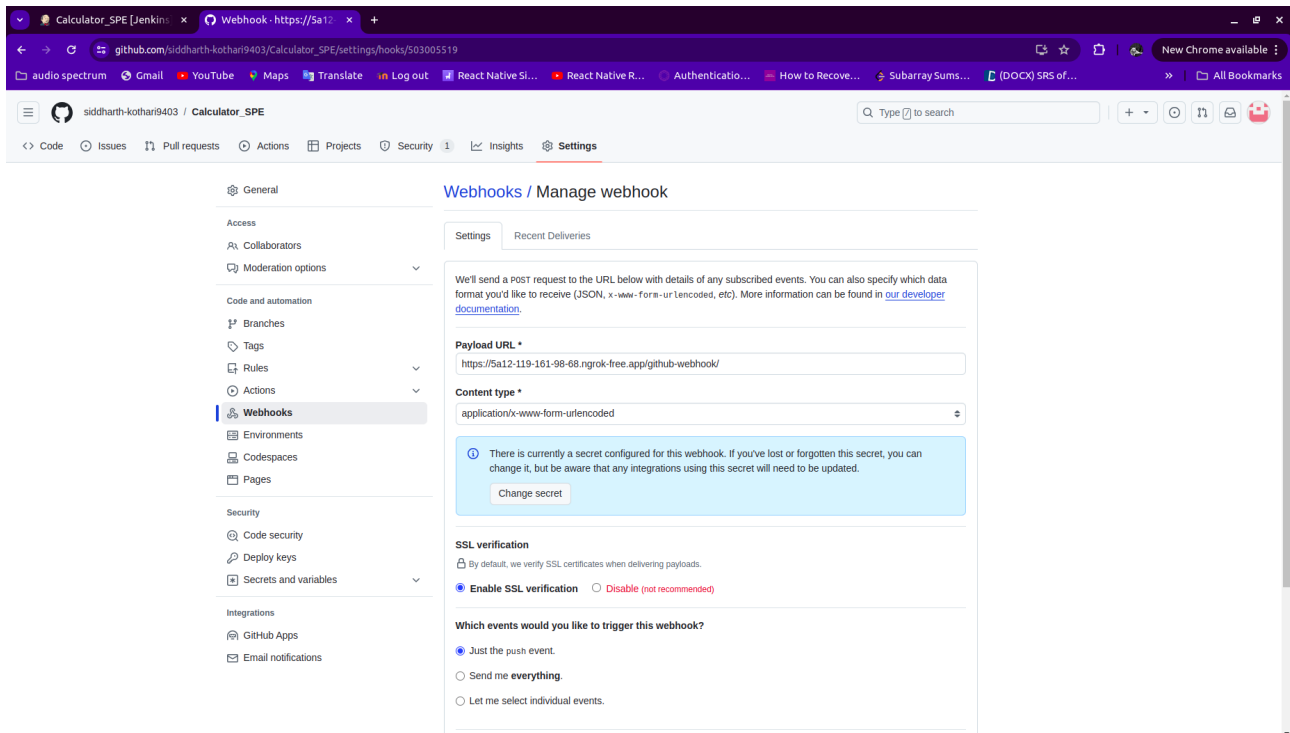


Figure 4: Webhook configuration page of Github

```
(base) siddharth@siddharth-Inspiron-5502:~/IdeaProjects/Calculator_SPE$ mvn clean install
[INFO] Scanning for projects...
[INFO]
-----< org.example:Calculator_SPE >-----
[INFO] Building Calculator_SPE 1.0-SNAPSHOT
[INFO]
-----[ jar ]-----
[INFO]
--- maven-clean-plugin:2.5:clean (default-clean) @ Calculator_SPE ---
[INFO] Deleting /home/siddharth/IdeaProjects/Calculator_SPE/target
[INFO]
--- maven-resources-plugin:2.6:resources (default-resources) @ Calculator_SPE ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
--- maven-compiler-plugin:3.1:compile (default-compile) @ Calculator_SPE ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to /home/siddharth/IdeaProjects/Calculator_SPE/target/classes
[INFO]
--- maven-resources-plugin:2.6:testResources (default-testResources) @ Calculator_SPE ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/siddharth/IdeaProjects/Calculator_SPE/src/test/resources
[INFO]
--- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ Calculator_SPE ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/siddharth/IdeaProjects/Calculator_SPE/target/test-classes
[INFO]
--- maven-surefire-plugin:2.12.4:test (default-test) @ Calculator_SPE ---
[INFO] Surefire report directory: /home/siddharth/IdeaProjects/Calculator_SPE/target/surefire-reports

-----
T E S T S
-----
Running CalculatorTest
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.044 sec

Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

[INFO]
--- maven-jar-plugin:3.1.0:jar (default-jar) @ Calculator_SPE ---
[INFO] Building jar: /home/siddharth/IdeaProjects/Calculator_SPE/target/Calculator_SPE-1.0-SNAPSHOT.jar
[INFO]
--- maven-install-plugin:2.4:install (default-install) @ Calculator_SPE ---
[INFO] Installing /home/siddharth/IdeaProjects/Calculator_SPE/target/Calculator_SPE-1.0-SNAPSHOT.jar to /home/siddharth/.m2/repository/org/example/Calculator_SPE/1.0-SNAPSHOT/Calculator_SPE-1.0-SNAPSHOT.jar
[INFO] Installing /home/siddharth/IdeaProjects/Calculator_SPE/pom.xml to /home/siddharth/.m2/repository/org/example/Calculator_SPE/1.0-SNAPSHOT/Calculator_SPE-1.0-SNAPSHOT.pom
[INFO]
BUILD SUCCESS
-----
[INFO] Total time: 2.320 s
[INFO] Finished at: 2024-09-25T21:27:52+05:30
[INFO]
-----
(base) siddharth@siddharth-Inspiron-5502:~/IdeaProjects/Calculator_SPE$
```

Figure 5: Maven Build Output

## 4.5 Project Setup

The code structure and pom.xml file have been shown above. To create a maven project, we can create a new project in IntelliJ IDEA, and select Maven as the build tool from there. Each of the files (Calculator.java and CalculatorTests.java) can be run using the Run button in IntelliJ. The project can be built using the following command -

Listing 9: Command for Building Using Maven

```
1 $ mvn clean install
```

The output of the command is shown in Figure 5.

## 4.6 Jenkins Installation

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

Listing 10: Jenkins Service Start

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

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

## 4.7 Configuration of Plugins for Jenkins

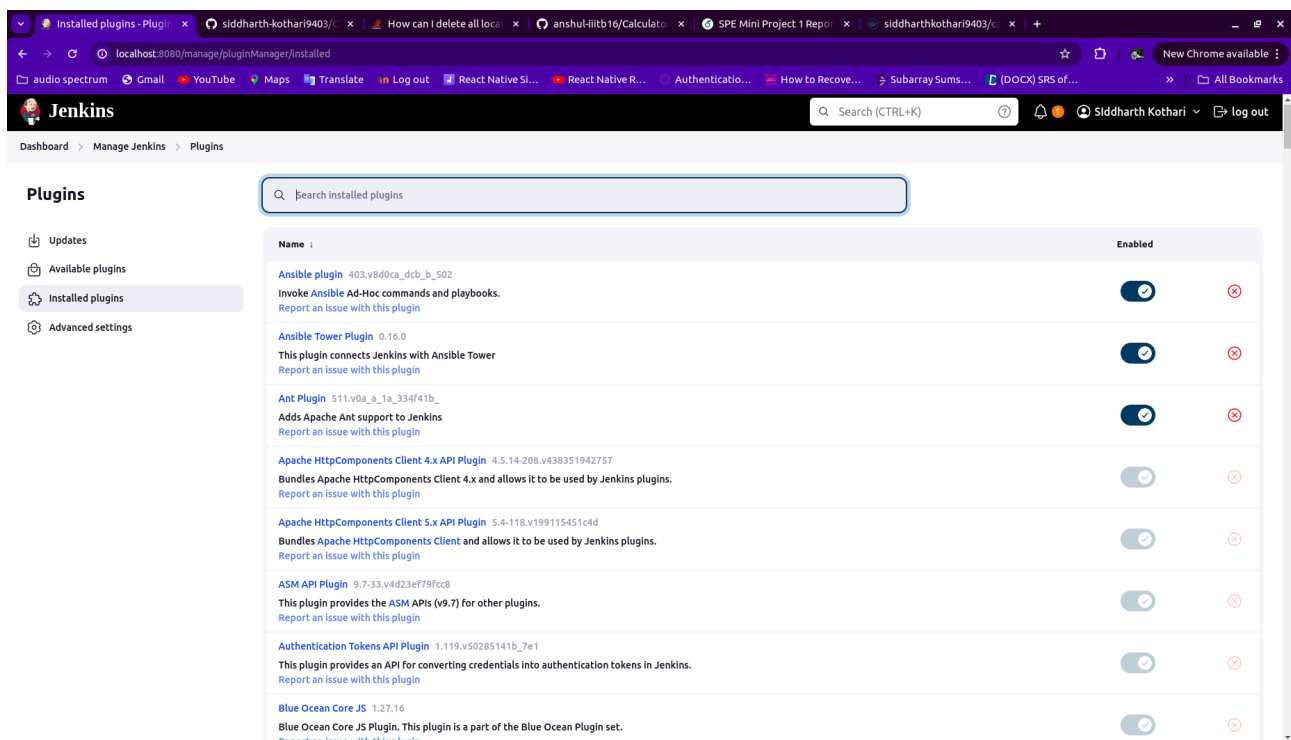


Figure 6: Plugins in Jenkins

Once Jenkins Setup is completed, we can open the Jenkins Dashboard. Go to Manage Jenkins → Plugins. From here we can install the required plugins. Figure 6 shows where to add plugins in Jenkins.

For this project, we need the following plugins - Ansible Plugin, Credentials Plugin, Docker API, Docker Pipeline, Docker Commons and Docker Plugin, Git Plugin, Github API Plugin, Maven Integration Plugin, Pipeline, Pipeline Maven Integration Plugin, Pipeline Stage View Plugin. Installing these will also install the required other plugins for their installation.

Once they are installed, restart Jenkins using the following command -

#### Listing 11: Jenkins Service Restart

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

## 4.8 Docker Installation

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

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

#### Listing 12: Docker Login Command

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

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

#### Listing 13: Add Jenkins to Docker

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

## 4.9 Credentials for Docker and Github in Jenkins

We now need to add Github and Docker Credentials to Jenkins, to ensure that it has the necessary permissions to perform relevant tasks. We also need to add our sudo user credentials to ensure that the ansible playbook can run smoothly. For this, go to Manage Jenkins → Credentials → System → Global credentials (unrestricted). Here we can add new credentials. Figures 7 and 8 show where and how to add credentials to Jenkins.

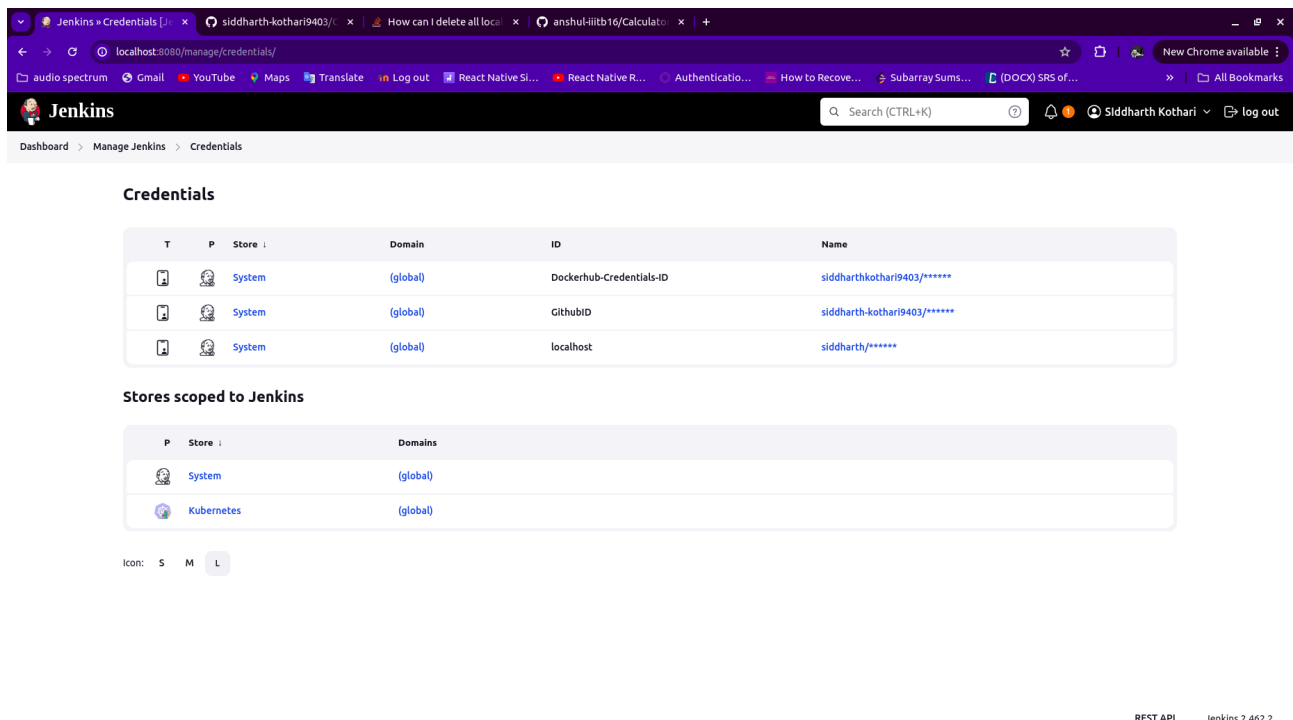


Figure 7: Credentials in Jenkins

For Github, we can select the Secret Kind to be Username and Password. Here, the username is the Github Username, and Password is the Access Token created previously. Any ID can be provided.

For Docker, the steps are similar. The username is the Docker Username, and password is the Docker password. The ID should be kept as "Dockerhub-Credentials-ID", as it has been configured accordingly in the Jenkinsfile provided previously.

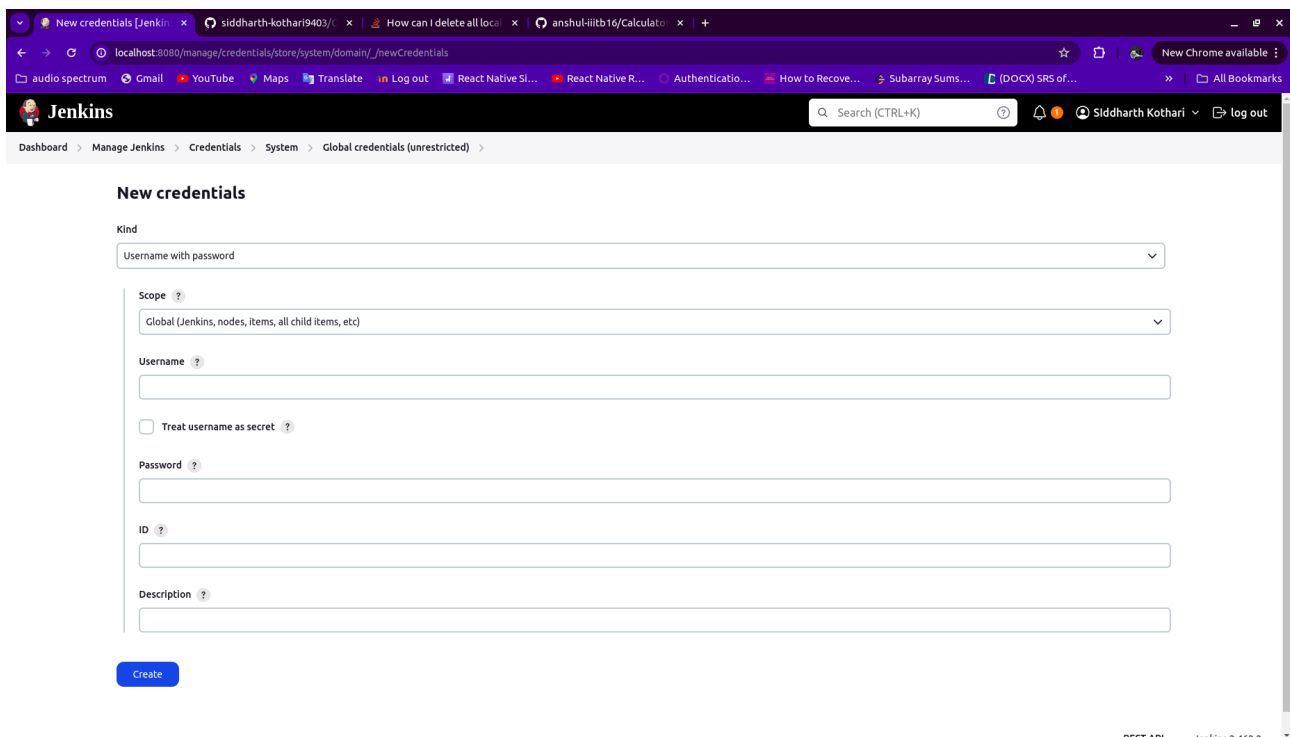


Figure 8: Adding Credentials in Jenkins

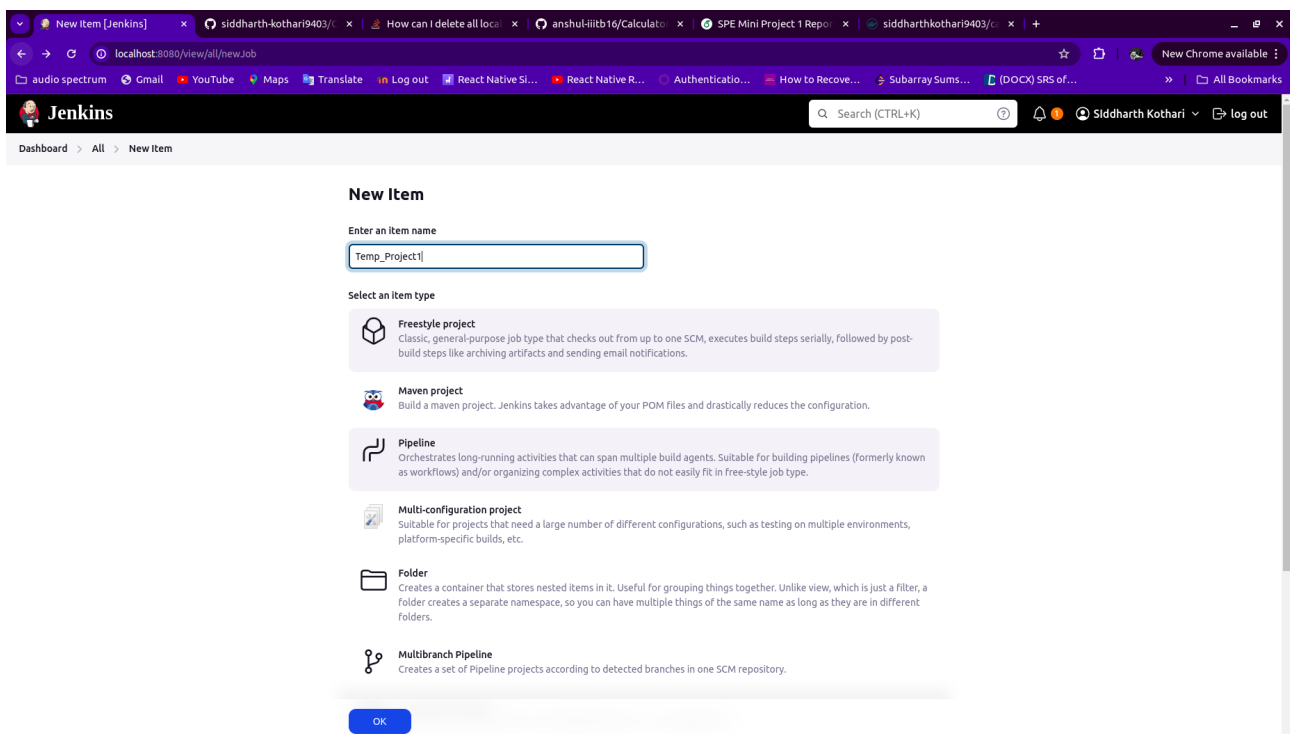


Figure 9: Creating Pipeline Project

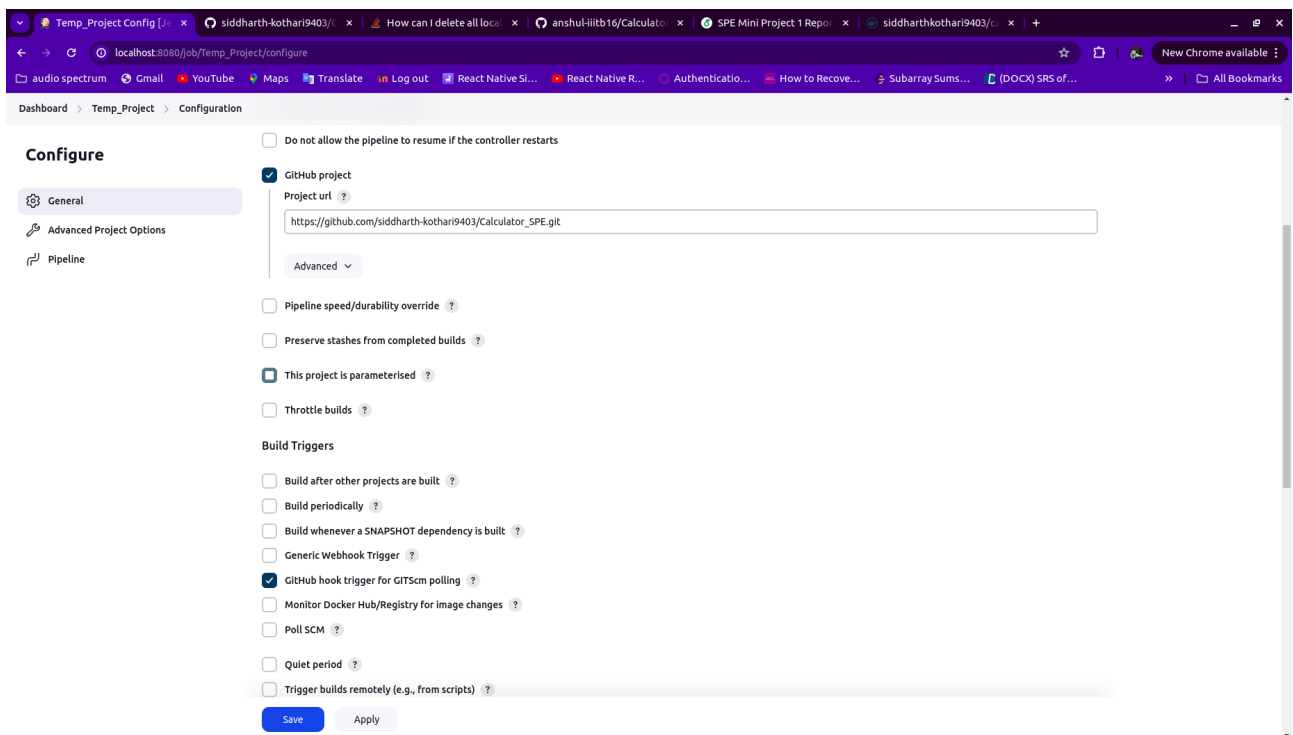


Figure 10: Specifying Git Polling SCM

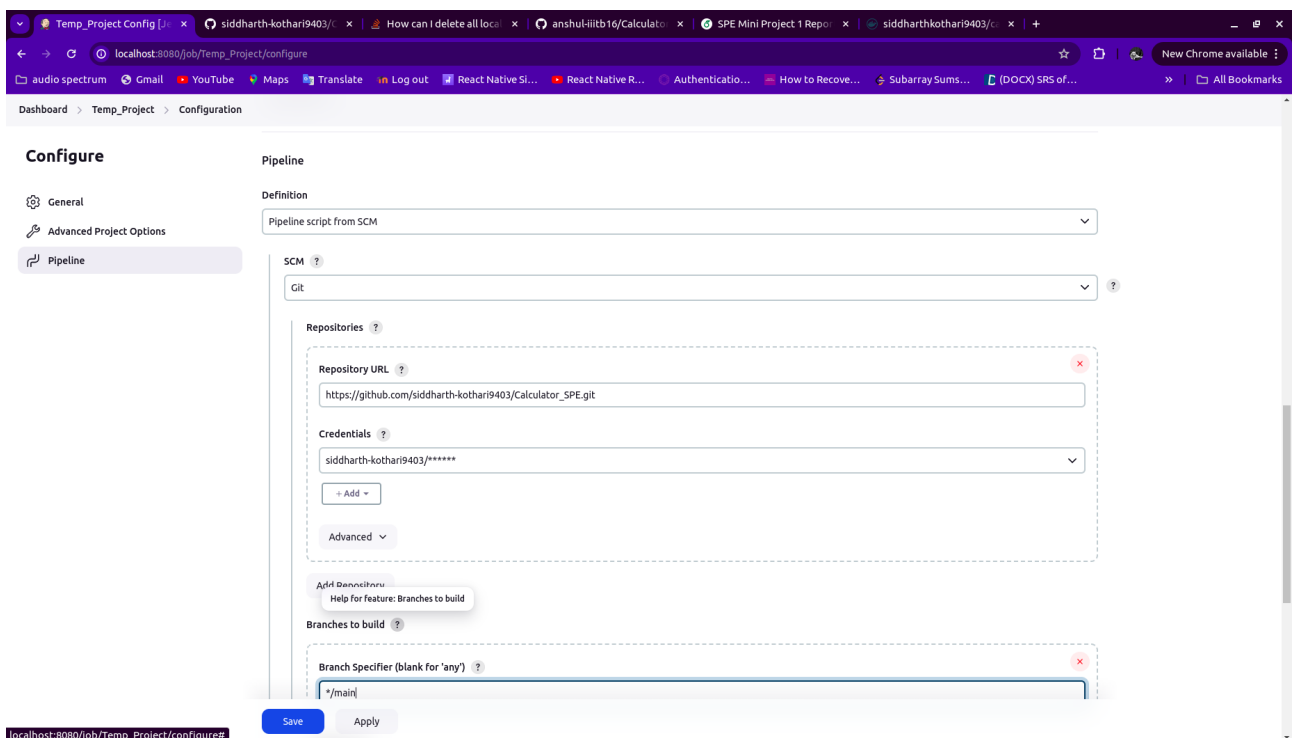


Figure 11: Configuring Pipeline Script from Git Repository



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

## 4.10 Creating a Pipeline Project Jenkins

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

Figures 9, 10, 11 and 12 show the steps for creating and configuring the pipeline project.

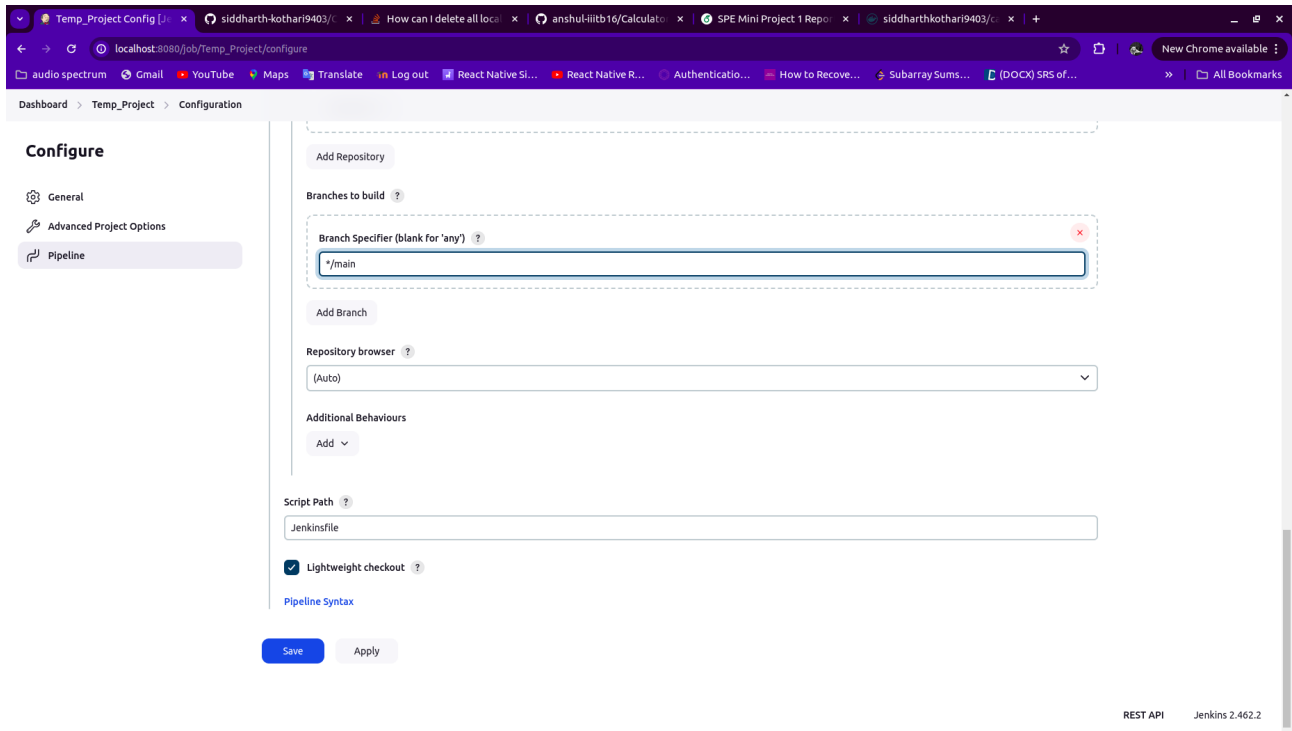


Figure 12: Configuring Pipeline Script from Git Repository

## 4.11 Ansible Installation

Guides for Ansible Installation can be found at the following link - [https://docs.ansible.com/ansible/latest/installation\\_guide/installation\\_distros.html](https://docs.ansible.com/ansible/latest/installation_guide/installation_distros.html).

## 4.12 Running the Application

We can now run the application by either pushing to the Github Repository, or by manually building the entire application. We have already provided the relevant files for the same (Deploy-Calculator.yml, Dockerfile, Jenkinsfile and inventory). The results are shown in the next section.

# 5 Results

## 5.1 Outputs of the Run Stages from Jenkins

Figure 13 shows the outputs of the various pipeline stages. As we observe, the outputs of all the stages are green, indicating that all stages ran without any issues. We will now confirm the various changes this script was supposed to carry out, and verify whether everything works.

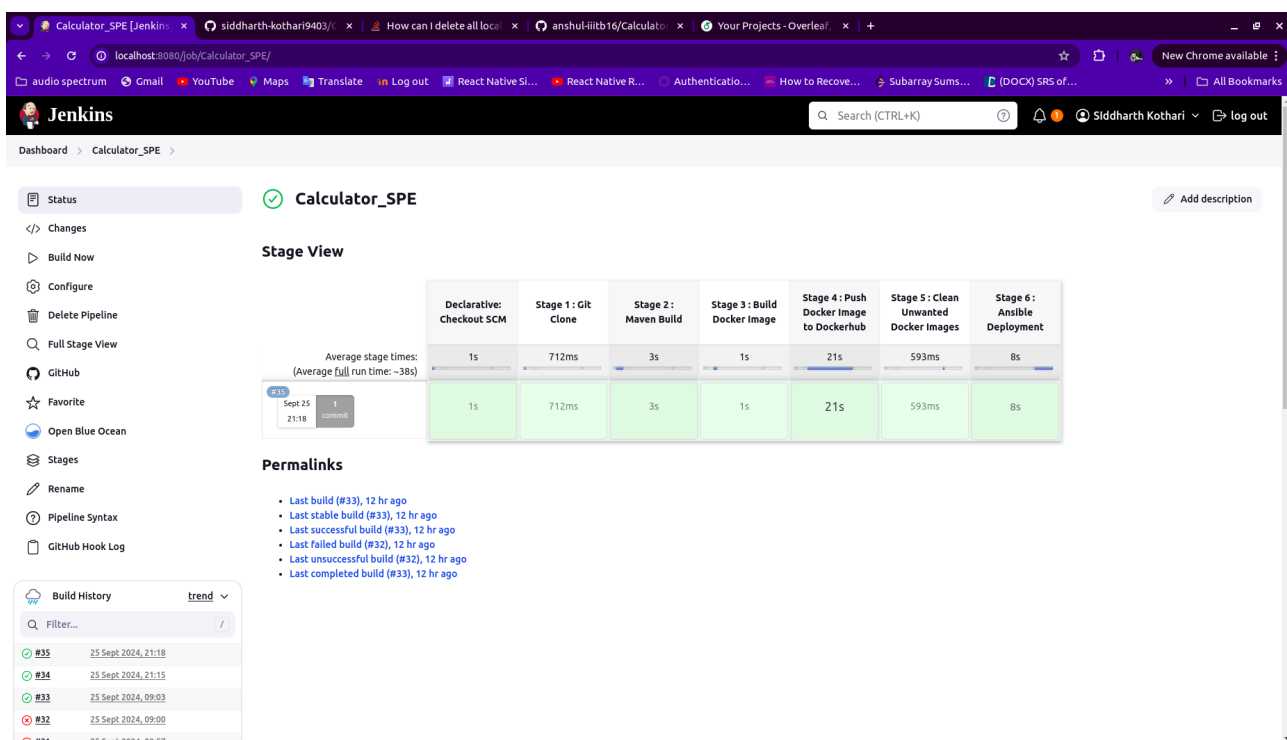


Figure 13: Outputs in Pipeline View

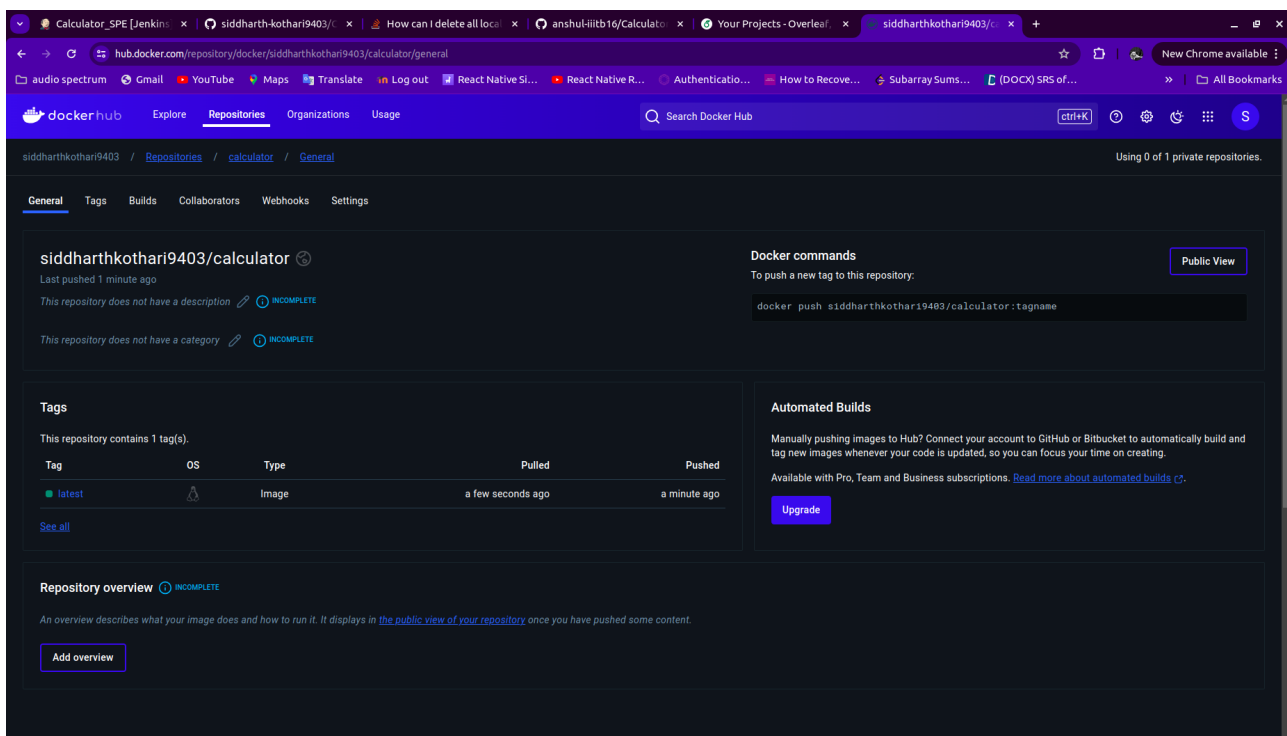


Figure 14: Outputs on Dockerhub

## 5.2 Outputs on Dockerhub

The 4th stage in the Jenkins pipeline requires us to push the created image to dockerhub, while the 6th stage requires us to pull the image back on the system, after the previous image has been pruned.

As we observe from Figure 14 both the pull and the push occur successfully, and we observe the terminal output for the same in the next section.

## 5.3 Containers and Images Created

Figure 15 contains the outputs of the following 2 commands when executed via the terminal. The first command can be used to view all containers, while the second command is used to view all images.

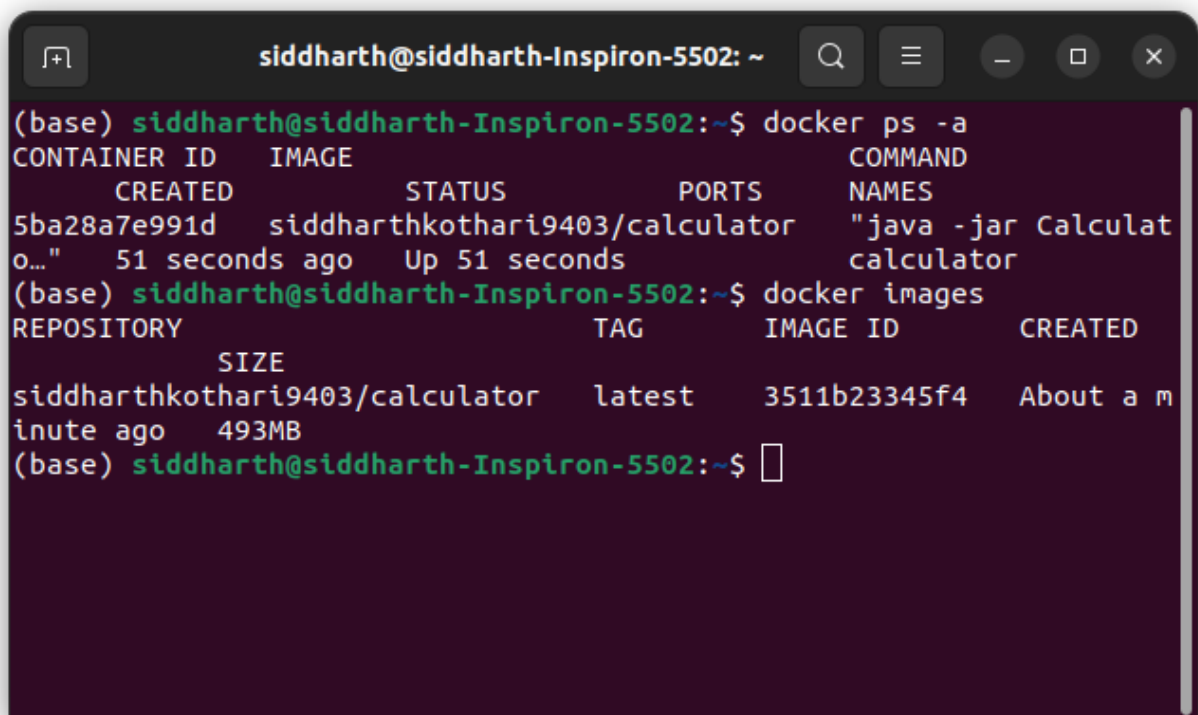
Listing 14: Docker Command to View All Containers

```
1 $ docker ps -a
```

Listing 15: Docker Command to View All Images

```
1 $ docker images
```

As we observe from the figure, there is one running container and one image pulled from dockerhub after running the ansible-playbook. The previous image which we pushed to Dockerhub is no longer existent, as it was pruned in Stage 5 of the pipeline, hence only one image.



```
siddharth@siddharth-Inspiron-5502: ~  
(base) siddharth@siddharth-Inspiron-5502:~$ docker ps -a  
CONTAINER ID   IMAGE                                COMMAND  
5ba28a7e991d   siddharthkothari9403/calculator    "java -jar Calculat  
o..." 51 seconds ago Up 51 seconds  
siddharth@siddharth-Inspiron-5502:~$ docker images  
REPOSITORY          TAG         IMAGE ID      CREATED  
siddharthkothari9403/calculator latest      3511b23345f4  About a m  
inute ago 493MB  
(base) siddharth@siddharth-Inspiron-5502:~$
```

Figure 15: Checking Docker Containers and Images on Terminal

## 5.4 Outputs of the Application

We now run the container using the command given below. This is to start the container which had been created in the previous step, and observe the outputs.

As we observe from Figures 16 and 17, all the testcases provided by the user provide the correct output. This completes the application demo.

```
siddharth@siddharth-Inspiron-5502: ~  
(base) siddharth@siddharth-Inspiron-5502:~$ docker start -l 5ba28a7e991d  
Welcome to Calculator SPE Project!  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
1  
-----  
Enter Positive Real Number for Square Root  
4  
2.0  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
2  
-----  
Enter Positive Integer for Factorial  
4  
24  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
3  
-----  
Enter Positive Real Number for Logarithm  
2.71  
0.9969486348916096  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
4  
-----  
Enter Base:  
4  
Enter Exponent:  
3  
64.0
```

Figure 16: Execution of the Docker Container

```
siddharth@siddharth-Inspiron-5502: ~  
-----  
Enter Positive Integer for Factorial  
4  
24  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
3  
-----  
Enter Positive Real Number for Logarithm  
2.71  
0.9969486348916096  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
4  
-----  
Enter Base:  
4  
Enter Exponent:  
3  
64.0  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
6  
-----  
Invalid Choice  
-----  
Enter choice:  
1. Square Root of a Decimal Number  
2. Factorial of an Integer  
3. Natural Logarithm of a Positive Real Number  
4. Power of a Number  
5. Exit Menu  
Enter your choice:  
5  
-----  
Exiting  
(base) siddharth@siddharth-Inspiron-5502:~$
```

Figure 17: Execution of the Docker Container

Listing 16: Docker Command to Start a Created Container

```
1 $ docker start -i ${CONTAINER_ID}
```

## 6 Relevant Links

### 6.1 Project Github

The Github Link for the Project is - [https://github.com/siddharth-kothari9403/Calculator\\_SPE](https://github.com/siddharth-kothari9403/Calculator_SPE).

### 6.2 Dockerhub Link

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