SOFTWARE PRODUCTION ENGINEERING

MINI PROJECT REPORT

SCIENTIFIC CALCULATOR

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

in

Computer Science & Engineering

by

Mohit Sharma (MT2024091)

Under the guidance of

Prof. B. Thangaraju

to the

DEPARTMENT OF CSE
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
BANGALORE - 560100, INDIA

March 2025

1 Problem Statement

The objective of this project is to develop a Scientific Calculator that provides users with a menu-driven interface to perform essential mathematical operations. The calculator should support the following functionalities:

1. **Square Root Function** ($\sqrt{x}$) - Computes the square root of a given non-negative number.
2. **Factorial Function** ($x!$) - Calculates the factorial of a given integer, which is the product of all positive integers up to that number.
3. **Natural Logarithm (base e)** ($\ln x$) - Determines the natural logarithm of a positive number, which represents the power to which e must be raised to obtain that number.
4. **Power Function** (a^b) - Computes the result of raising a base number a to the exponent b, where both a and b can be real numbers.

In addition to implementing these mathematical operations, the project follows **DevOps methodologies** to ensure an efficient and automated development lifecycle. This includes integrating a **Continuous Integration and Continuous Deployment (CI/CD) pipeline** to streamline code building, testing, and deployment processes. By leveraging **DevOps principles**, the project aims to enhance automation, scalability, and maintainability, ensuring a robust and efficient scientific calculator application.

2 Abstract

The Scientific Calculator project is a software application that facilitates various mathematical and scientific computations, including square root calculations, logarithms, exponentiation, and factorials. The development process incorporates **DevOps methodologies** to enhance efficiency across different stages, such as coding, testing, and deployment.

To ensure robustness and scalability, the project employs **Continuous Integration and Continuous Deployment (CI/CD)** pipelines, along with automated testing and containerization. By utilizing modern DevOps tools like **GitHub Actions, Docker, Jenkins, and Ansible**, the system optimizes workflow automation, improves maintainability, and enhances collaboration among developers.

This Scientific Calculator serves as a lightweight yet efficient computational tool, designed using contemporary software engineering practices. Through seamless integration of DevOps principles, the project streamlines the software lifecycle, from development to deployment, ensuring reliability and adaptability in dynamic environments.

3 DevOps

3.1 What is DevOps?

DevOps is a modern software development approach that fosters seamless collaboration between development (Dev) and operations (Ops) teams throughout the entire software lifecycle. It aims to enhance automation, efficiency, and communication, ensuring rapid and reliable software delivery. By integrating development and IT operations, DevOps accelerates the deployment process while maintaining high-quality standards.

Key Principles of DevOps:

- **Collaboration and Communication** – Promotes teamwork between developers and operations teams to streamline processes.
- **Automation** – Reduces manual intervention in tasks like code integration, testing, deployment, and infrastructure management.
- **Continuous Integration & Continuous Deployment (CI/CD)** – Enables frequent updates with reliable and seamless releases.
- **Infrastructure as Code (IaC)** – Manages infrastructure through code, ensuring consistency, scalability, and reproducibility.
- **Monitoring and Feedback** – Provides real-time tracking of system performance for continuous improvement.
- **Security (DevSecOps)** – Embeds security practices into every stage of the development pipeline, reducing vulnerabilities.

3.2 Why DevOps?

Implementing DevOps brings numerous advantages to software development and IT operations by enhancing agility, efficiency, and reliability. Below are some of the key benefits:

- **Faster Delivery** – DevOps accelerates the software release cycle, allowing businesses to deploy updates more frequently.
- **Improved Software Quality** – Continuous integration and automated testing help in detecting and resolving errors early in the development process.
- **Enhanced Collaboration** – By bridging the gap between development and operations, DevOps fosters teamwork and communication.
- **Security Integration** – Security is incorporated from the beginning, ensuring robust protection against vulnerabilities.
- **Scalability** – Automation and optimized workflows enable the management of scalable and resilient applications.
- **Better Customer Experience** – Faster and more reliable software updates improve user satisfaction.

- **Cost Efficiency** – Streamlined workflows and automation reduce operational expenses associated with inefficient processes.

3.3 DevOps Tools

To implement DevOps effectively, a variety of tools are used to automate and optimize different phases of the software lifecycle. The table below highlights some key tools used in DevOps:

Purpose	**Tools**
Building CI/CD Pipeline	Jenkins, Travis, CodeShip
Version Control System	Git, Mercurial, Subversion
Build Tools	Maven, Ant, Gradle
Testing Tools	JUnit, Selenium, CppUnit
Deployment of Applications	Docker, Amazon AWS
Infrastructure as Code (IaC)	Ansible, Chef, Kubernetes
Monitoring Services	ELK Stack

Table 1: Commonly Used DevOps Tools

4 Tools Used for Mini-Project

The following tools were utilized in the implementation of this mini-project:

- **Git** – A version control system used for tracking code changes and managing different versions efficiently.
- **GitHub** – A code repository platform integrated with Git. It enables collaboration, webhooks for automation, and remote storage of the codebase.
- **Jenkins** – A CI/CD automation tool used to manage and streamline the deployment pipeline.
- **Maven** – A build automation tool for Java projects, also used in conjunction with JUnit for testing.
- **Docker** – A containerization tool that ensures consistency across different environments by packaging applications into portable containers.
- **Ansible** – An automation tool used for tasks such as pulling and running Docker images without manual intervention.
- **Ngrok** – A tunneling tool used to expose local services to the internet, enabling webhook communication with Jenkins.
- **Apache Log4j** – A logging framework used for tracking and debugging application activity.

5 Source Code Management

Source Code Management (SCM) is essential for tracking modifications in a source code repository. It helps maintain version control, resolve merge conflicts, and manage collaborative development efficiently.

In this project, the code was initially developed on a local machine and later pushed to a remote repository on GitHub. The following Git commands were executed to initialize and manage the repository:

- `git init` – Initializes a new Git repository.
- `git status` – Displays the state of the working directory and staged changes.
- `git add .` – Stages all changes for commit.
- `git commit -m "Commit Message"` – Commits the changes with a message.
- `git remote add origin <repository_URL>` – Links the local repository to a remote one.
- `git push -u origin master` – Pushes the committed code to the master branch on GitHub.

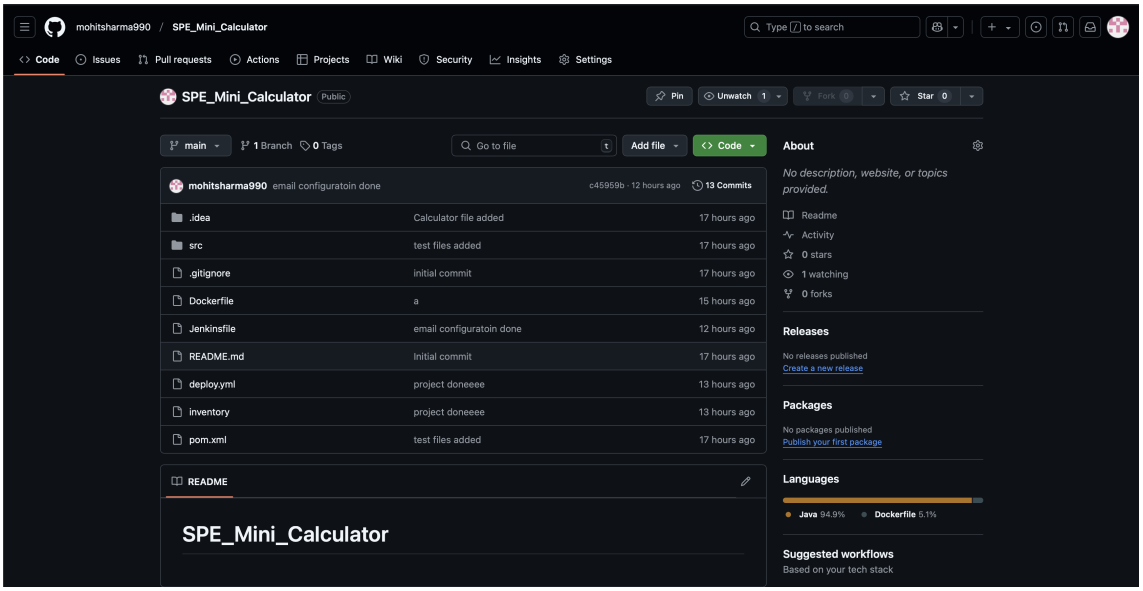

Figure 1: Source Code Management Workflow

6 Docker

Docker is a **containerization tool** that allows packaging software with its dependencies to ensure it runs consistently across different environments. It uses **kernel-level virtualization** to create lightweight, portable containers.

In this project, Docker was used to **containerize the application** by creating a Docker image that runs the JAR file generated through Maven. This image is then **pushed to Docker Hub**, making it accessible for other machines to pull and run as a container.

6.1 Building and Pushing a Docker Image

To create a Docker image, the following command is used:

```
docker build -t <USERNAME>/<IMAGE_NAME>:<TAG>
```

Once the image is built, it can be pushed to Docker Hub using:

```
docker push <USERNAME>/<REPOSITORY_NAME>:<TAG>
```

6.2 Integration with CI/CD Pipeline

- The **Jenkins pipeline** automates the Docker **build and push** process after a successful build stage.
- The image is later pulled and deployed through **Ansible jobs** on target machines.
- The image can be **executed as a container** using:

```
docker run -it <IMAGE_NAME>
```

After pushing the image to Docker Hub, it can be accessed through the user's **Docker Hub profile**. The **local image is deleted** once it is successfully pushed to optimize storage.

6.3 Dockerfile

The following image represents a sample **Dockerfile** used for containerization:

```
FROM maven as build

WORKDIR /app

COPY pom.xml .
COPY src ./src

RUN mvn clean package


FROM openjdk:17

WORKDIR /app

COPY --from=build /app/target/SPE_MINI_CALCULATOR-1.0-SNAPSHOT.jar .

CMD ["java", "-cp", "/app/SPE_MINI_CALCULATOR-1.0-SNAPSHOT.jar", "com.mohit.Calculator"]
```

Figure 2: Sample Dockerfile

6.4 Docker Hub Repository

Once the image is pushed, it appears in the Docker Hub repository, as shown below:

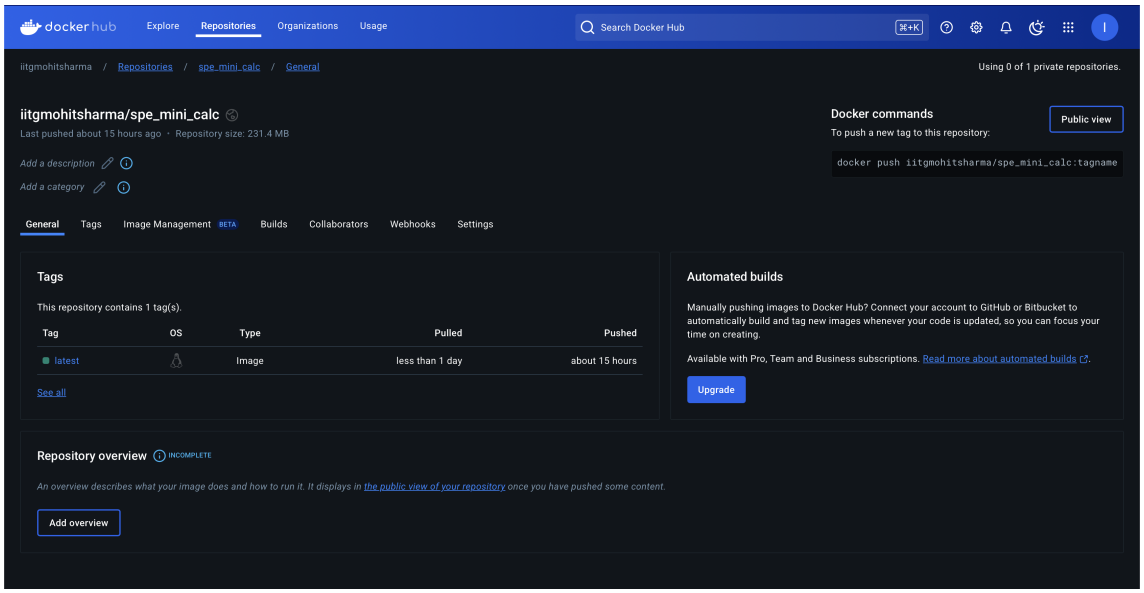

Figure 3: Docker Hub Repository

7 Jenkins: Automating CI/CD

Jenkins is an open-source automation tool that facilitates **Continuous Integration (CI)** and **Continuous Deployment (CD)** for software projects across different platforms. It supports seamless integration with multiple testing and deployment technologies, making it a robust solution for automating software pipelines.

7.1 Jenkins Pipeline Workflow

The pipeline is designed to automatically trigger upon detecting any new commit in the GitHub repository. Once activated, Jenkins executes a sequence of predefined steps to streamline the software delivery process.

A Jenkins pipeline enables a structured approach to software deployment by systematically progressing each code change through different phases such as **building, testing, containerization, and deployment**. This ensures reliability and repeatability in software development.

7.2 CI/CD Pipeline Stages

The Jenkins pipeline consists of the following automated stages:

- **Checkout**: Retrieves the latest changes from the GitHub repository.
- **Build**: Uses Maven to compile and package the project.
- **Docker Image Creation**: Builds a Docker image containing the application.
- **Push to Docker Hub**: Uploads the Docker image to a remote repository.
- **Deploy via Ansible**: Executes an Ansible playbook to deploy the application.
- **Post Action**: Sends an email notification to acknowledge the pipeline status.

7.3 Visual Representation of CI Pipeline

The following diagram illustrates the CI/CD pipeline workflow:

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE_NAME = 'spe_mini_calc'
        GITHUB_REPO_URL = 'https://github.com/mohitsharma990/SPE_Mini_Calculator.git'
    }
```

Figure 4: Jenkins CI/CD Pipeline

7.4 Stages of Jenkins Pipeline

The pipeline is broken into multiple stages to ensure smooth execution. Below are the key stages:

```
    stages {
        stage('Checkout') {
            steps {
                script {
                    // Checkout the code from the GitHub repository
                    git branch: 'main', url: "${GITHUB_REPO_URL}"
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    // Build Docker image
                    docker.build("${DOCKER_IMAGE_NAME}", '.')
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('', 'DockerHubCred') {
                        sh 'docker tag spe_mini_calc iitgmohitsharma/spe_mini_calc:latest'
                        sh 'docker push iitgmohitsharma/spe_mini_calc:latest'
                    }
                }
            }
        }
```

Figure 5: Pipeline Stages - Part 1

Each stage plays a crucial role in ensuring that the application is properly built, tested, and deployed while maintaining automation and efficiency.

```
        stage('Run Ansible Playbook') {
            steps {
                script {
                    withEnv(["ANSIBLE_HOST_KEY_CHECKING=False"]) {
                        ansiblePlaybook(
                            playbook: 'deploy.yml',
                            inventory: 'inventory'
                        )
                    }
                }
            }
        }
    }

    post {
        success {
            mail to: 'Mohit.Sharma@iiitb.ac.in',
                 subject: "Application Deployment SUCCESS: Build ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                 body: "The build was successful!"
        }
        failure {
            mail to: 'Mohit.Sharma@iiitb.ac.in',
                 subject: "Application Deployment FAILURE: Build ${env.JOB_NAME} #${env.BUILD_NUMBER}",
                 body: "The build failed."
        }
        always {
            cleanWs()
        }
    }
}
```

Figure 6: Pipeline Stages - Part 2

8 Configuration of Project in Jenkins

Jenkins is a widely used automation server that enables **Continuous Integration (CI)** and **Continuous Deployment (CD)**. To integrate our project with Jenkins, we need to configure Jenkins on a local machine, link it with the GitHub repository, and set up credentials for Docker Hub and Ansible.

8.1 Starting Jenkins on macOS

To run Jenkins on a local machine using Homebrew, execute the following command:

```
brew services start jenkins-lts
```

To verify if Jenkins is running, use:

```
brew services list
```

If Jenkins is running successfully, it should appear as an active service in the list.

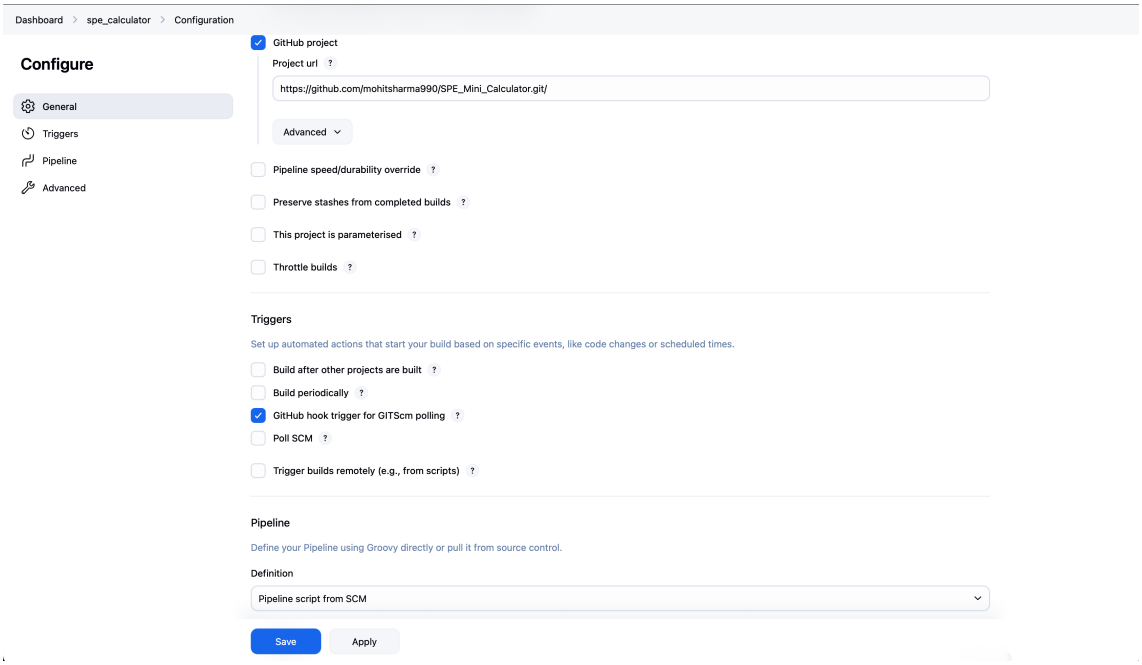

Figure 7: Starting Jenkins on macOS

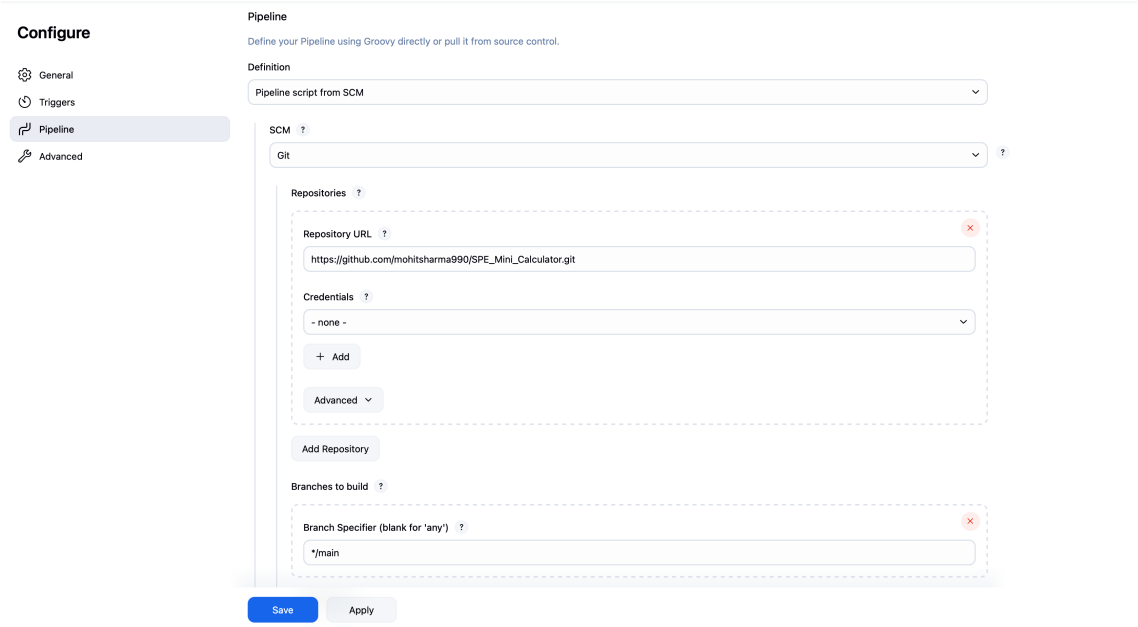

Figure 8: Checking Jenkins Service Status

8.2 Creating a GitHub Project in Jenkins

8.3 Setting Up System Credentials for Docker Hub and Local Machine

Jenkins requires credentials for **Docker Hub** and the **local machine** to manage Docker images and execute Ansible playbooks.

Adding Docker Hub Credentials

To authenticate with Docker Hub, follow these steps:

1. In Jenkins, navigate to **Manage Jenkins** → **Manage Credentials**.
2. Select **Global credentials (unrestricted)**.
3. Click **Add Credentials**.
4. Choose **Username with password** and enter your **Docker Hub username** and **access token**.
5. Save the credentials.

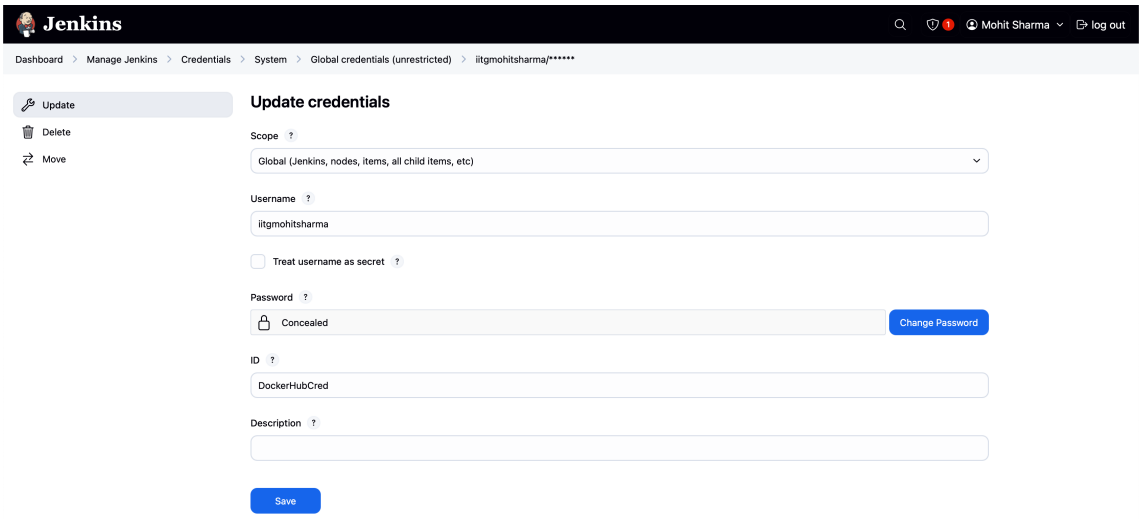

Figure 9: Adding Docker Hub Credentials in Jenkins

Adding Local Machine Credentials for Ansible

For Jenkins to securely access the local machine and execute Ansible playbooks, we need to add SSH credentials:

1. Go to **Manage Jenkins** → **Manage Credentials**.
2. Click **Add Credentials**.
3. Choose **SSH Username with Private Key**.
4. Enter the **local system username** and upload the **private SSH key**.
5. Save the credentials.

Once these configurations are set up, Jenkins can securely authenticate with both Docker Hub and the local system, enabling a smooth CI/CD workflow.

9 Ansible: Automating Deployment

Ansible is a powerful automation tool used for **continuous deployment** and **configuration management**. It operates using a **control node**, which manages the deployment process, and **managed nodes**, where the actual deployment takes place.

Unlike many other automation tools, **Ansible does not require any agent** to be installed on the managed nodes. It communicates with them securely over **SSH**. The deployment steps, including configurations and tasks, are defined in a YAML file known as a **playbook**. Additionally, an **inventory file** specifies the target machines where the playbook should be executed.

9.1 Installing Ansible on macOS

To set up Ansible on a Mac, it can be installed easily using Homebrew:

```
brew install ansible
```

To verify the installation, run:

```
ansible --version
```

9.2 Ansible Inventory File

The **inventory file** is a critical component of Ansible that defines the target hosts (managed nodes) and their groupings. The file can be written in various formats, depending on the environment configuration.

Below is an example of a basic inventory file:

```
[webservers]
server1.example.com
server2.example.com

[databases]
db1.example.com
```

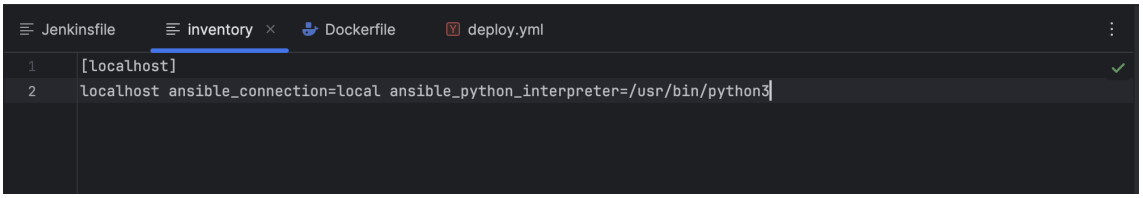

Figure 10: Sample Ansible Inventory File

9.3 Ansible Playbooks

Ansible playbooks define the configuration and deployment steps required to automate tasks across managed nodes. Playbooks are written in YAML format and contain a series of **tasks** that Ansible executes sequentially.

A sample playbook to install required dependencies and deploy a Docker container might look like this:

```
---
- name: Deploy Application
  hosts: webservers
  become: yes
  tasks:
    - name: Install Python and Docker
      homebrew:
        name: python3
        state: present
```

```
- name: Pull Docker Image
  command: docker pull <USERNAME>/<IMAGE_NAME>

- name: Run Docker Container
  command: docker run -d -p 8080:8080 <USERNAME>/<IMAGE_NAME>
```

```yaml
---
- name: Pull Docker Image from Docker Hub
  hosts: localhost
  remote_user: mohitsharma
  become: false
  vars:
    ansible_python_interpreter: /Users/mohitsharma/.ansible-env/bin/python3
  tasks:
    - name: Install required Ansible collections
      ansible.builtin.command: ansible-galaxy collection install community.docker

    - name: Pull Docker Image
      community.docker.docker_image:
        name: "iitgmohitsharma/spe_mini_calc:latest"
        source: pull
```

Figure 11: Example Ansible Playbook (Part 1)

```yaml
    - name: Stop and remove existing container if running
      ansible.builtin.shell: |
        docker stop spe_mini_calc || true
        docker rm spe_mini_calc || true
      ignore_errors: yes

    - name: Run container
      community.docker.docker_container:
        name: spe_mini_calc
        image: iitgmohitsharma/spe_mini_calc:latest
        state: started
        detach: true
        interactive: true
        tty: true
```

Figure 12: Example Ansible Playbook (Part 2)

9.4 Executing the Ansible Playbook

Once the inventory and playbook files are ready, we can execute the playbook using the following command:

```
ansible-playbook deploy.yml -i inventory
```

Here, `deploy.yml` is the Ansible playbook, and `inventory` is the file that contains information about the managed nodes.

9.5 Running the Jenkins Pipeline

After configuring Ansible for automated deployment, the project can be executed using a **Jenkins pipeline**. This allows us to automate the build, test, and deployment stages.

To trigger the pipeline execution:

1. Open Jenkins and navigate to the configured pipeline.
2. Click on **"Build Now"** to start the process.
3. Monitor each stage of the pipeline as it executes the configured tasks.

Figure 13: Jenkins Pipeline Stages

10 Scientific Calculator Source Code

The core functionality of this project is implemented in **Java**, utilizing object-oriented programming principles. The scientific calculator is designed to perform essential mathematical computations, including square root, factorial, logarithm, and power functions.

The program follows a modular approach, ensuring **scalability, maintainability, and efficiency**. Exception handling is integrated to handle invalid inputs gracefully. Additionally, the code is structured to allow easy expansion with new mathematical functions in the future.

Below are the main components of the scientific calculator:

- **Main Class:** Initializes the calculator and provides a menu-driven interface for user interaction.
- **Mathematical Functions:** Implements core mathematical operations such as square root, factorial, logarithm, and power.

- **Error Handling:** Ensures the program can handle incorrect user input and prevent runtime errors.
- **Modular Design:** The code is structured into separate methods for better readability and reusability.

10.1 Source Code Implementation

The following images display the implementation of the scientific calculator in Java:

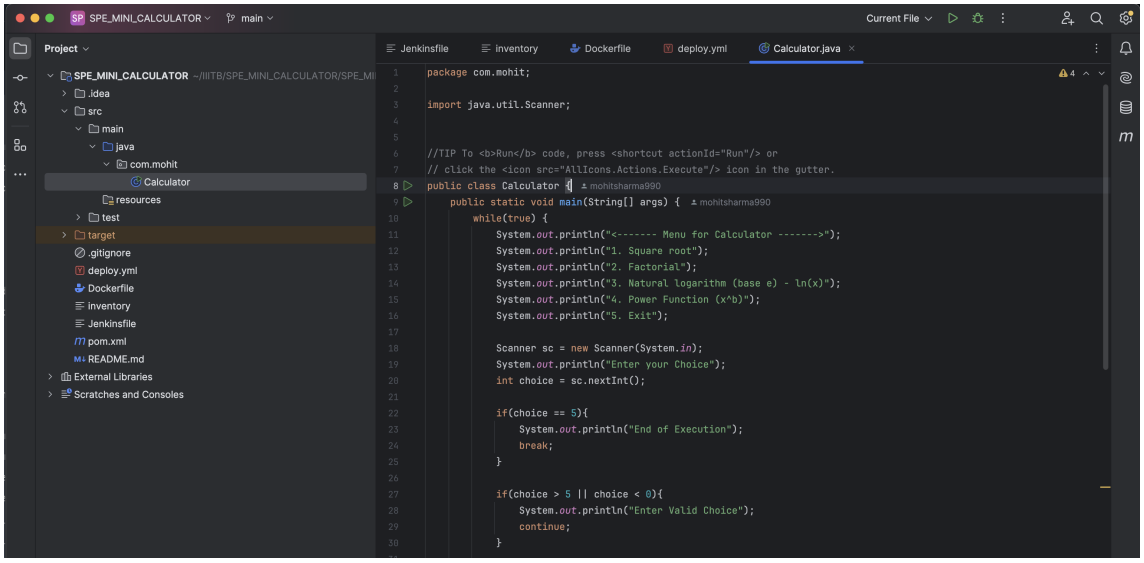

Figure 14: Scientific Calculator Source Code - Part 1

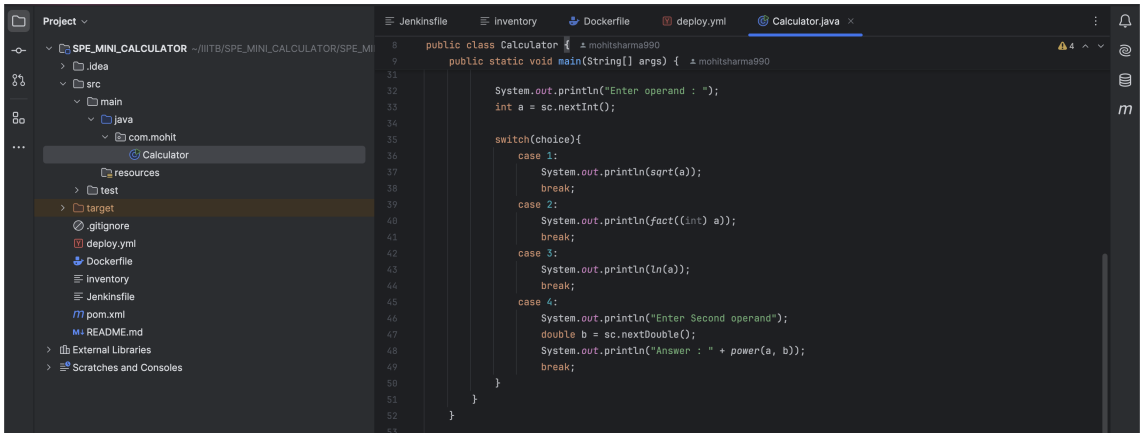

Figure 15: Scientific Calculator Source Code - Part 2

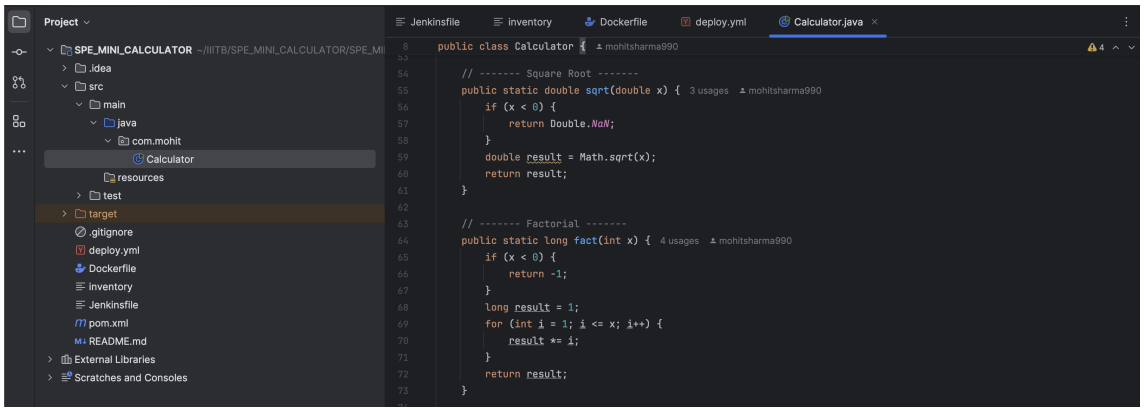

Figure 16: Scientific Calculator Source Code - Part 3

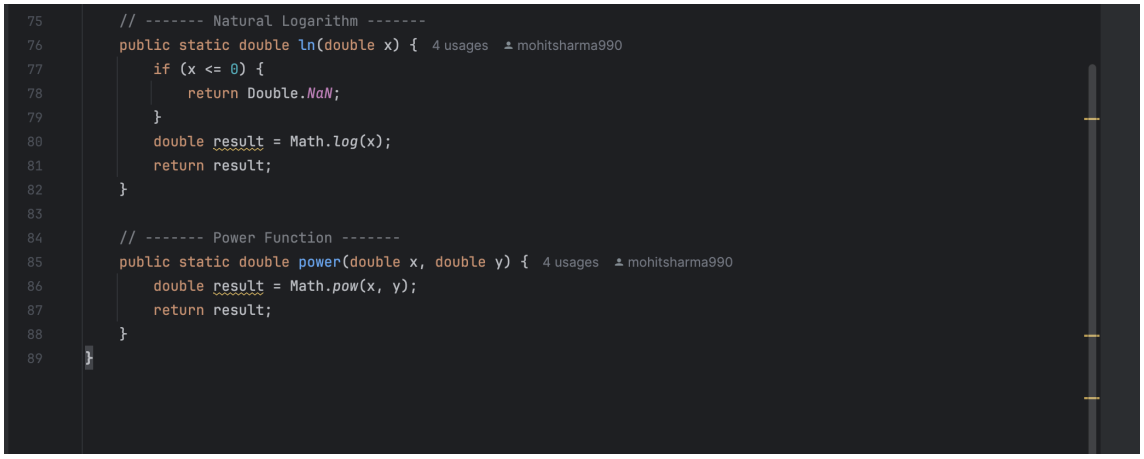

Figure 17: Scientific Calculator Source Code - Part 4

10.2 Key Features of the Implementation

- **User Input Handling:** The program takes user input and processes it based on the selected operation.
- **Mathematical Computations:** Uses built-in Java functions and custom implementations for precise calculations.
- **Looping for Continuous Execution:** Allows users to perform multiple calculations without restarting the program.
- **Exception Handling:** Ensures smooth execution by handling invalid inputs such as negative numbers for square root or incorrect format for power functions.
- **Code Optimization:** Structured efficiently for readability and future enhancements.

10.3 Future Enhancements

The current implementation provides fundamental mathematical operations, but future versions could include:

- Support for **trigonometric functions** such as sine, cosine, and tangent.
- A **graphical user interface (GUI)** for an improved user experience.
- Additional **error-handling mechanisms** to prevent invalid operations.
- **Integration with a web-based interface** for online usage.

This modular and efficient approach ensures that the scientific calculator remains extensible for further improvements while maintaining a solid foundation.

11 Adding Test Cases

11.1 Understanding Unit Testing

Unit testing is a fundamental practice in software development that involves testing individual components of a program to verify their correctness. A **unit** refers to the smallest testable part of the software, such as a function, method, or class. The goal of unit testing is to identify errors at an early stage, ensuring that each component operates correctly in isolation before integrating it into the larger system.

Key benefits of unit testing include:

- **Early Bug Detection:** Identifies issues in the development phase, reducing debugging time.
- **Code Refactoring Support:** Allows developers to make changes without worrying about breaking existing functionality.
- **Faster Development Cycle:** Automating test execution speeds up development and deployment.
- **Improved Code Quality:** Ensures each function behaves as expected, leading to more reliable software.

11.2 JUnit: Unit Testing in Java

JUnit is a widely used **open-source testing framework** for Java that allows developers to write and execute automated unit tests efficiently. Since Java applications often undergo frequent modifications, JUnit ensures that newly added code does not break existing functionality by re-executing test cases automatically.

JUnit provides a structured environment for running tests, offering the following features:

- **Annotations:** Simplifies test execution using annotations like `@Test`, `@Before`, and `@After`.
- **Assertions:** Compares expected and actual outputs using methods such as `assertEquals()` and `assertTrue()`.
- **Test Reports:** Generates visual test progress reports using a green (successful) and red (failed) status indicator.
- **Automation:** Supports automated test execution, reducing manual effort and increasing efficiency.

11.3 JUnit Test Cases for Scientific Calculator

In this project, JUnit test cases are written to validate the core functionalities of the scientific calculator, ensuring accurate mathematical computations. The following key operations are tested:

- **Square Root Function:** Tests if the correct square root value is returned.
- **Factorial Function:** Ensures accurate calculation of factorial values for different inputs.
- **Logarithm Function:** Validates the logarithm calculation, checking for domain errors.
- **Power Function:** Confirms that the exponentiation function produces expected results.

11.4 Source Code for JUnit Test Cases

The images below showcase the JUnit test cases implemented for the scientific calculator:

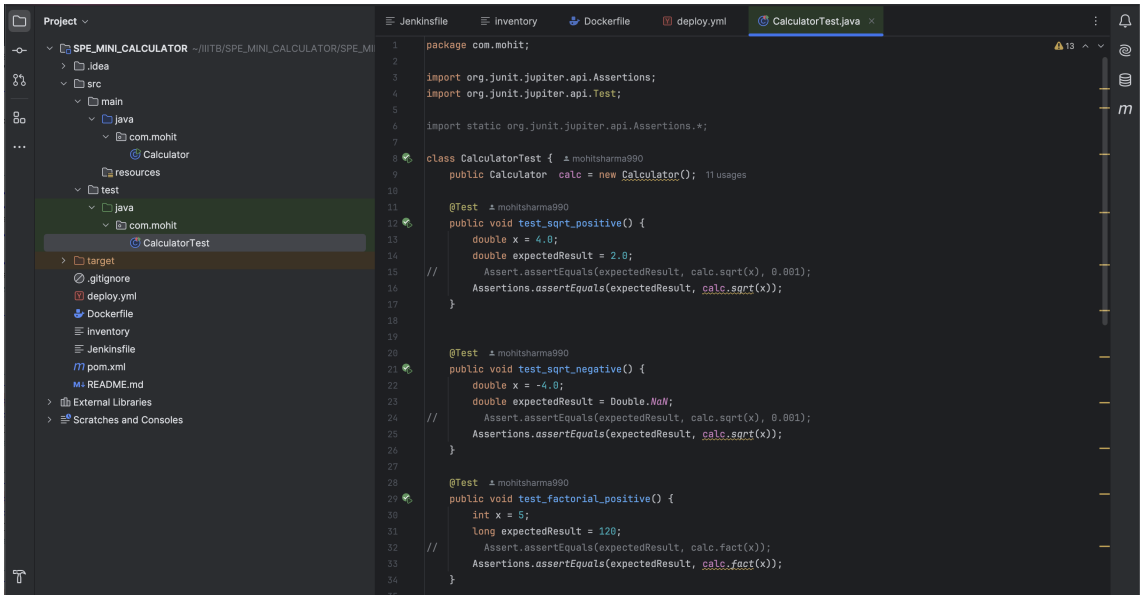

Figure 18: JUnit Test Cases - Part 1

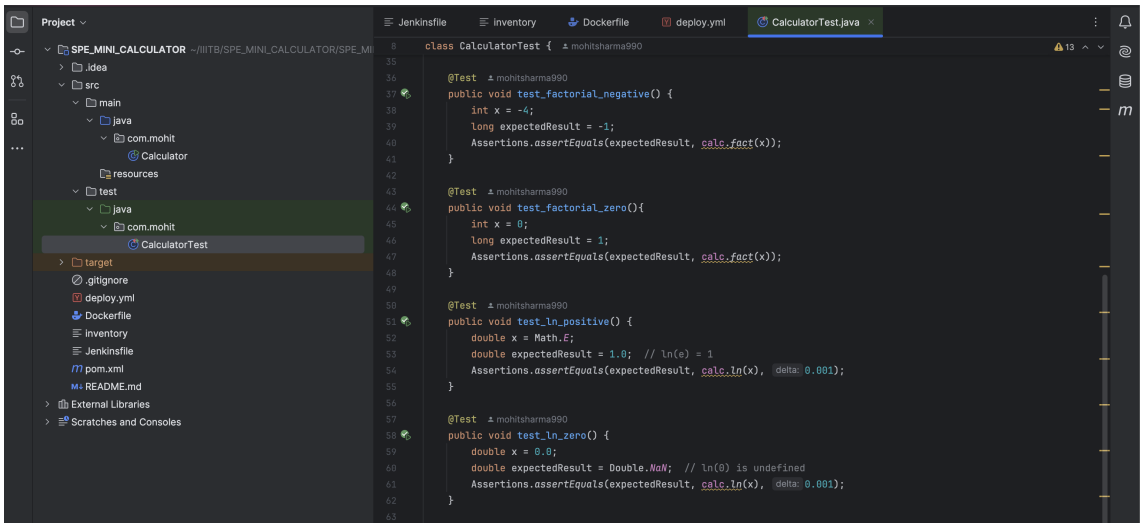

Figure 19: JUnit Test Cases - Part 2

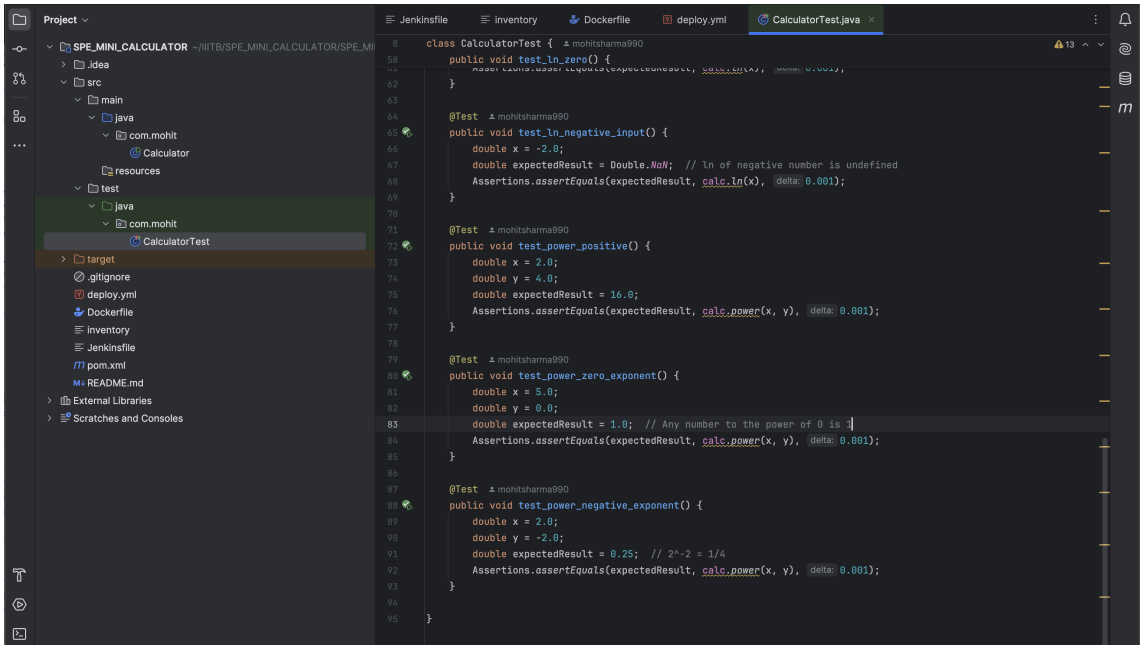

Figure 20: JUnit Test Cases - Part 3

11.5 Running JUnit Tests

JUnit tests can be executed using an Integrated Development Environment (IDE) such as **IntelliJ IDEA**, **Eclipse**, or through the command line using Maven:

```
mvn test
```

Upon execution, JUnit displays a **green progress bar** for successful test cases and a **red progress bar** if any test fails. If an assertion fails, JUnit provides detailed information, helping developers quickly identify and fix the issue.

11.6 Conclusion

By integrating JUnit into the project, we ensure that the scientific calculator functions as expected under various input conditions. Automated unit testing significantly improves software reliability, reduces debugging effort, and streamlines the development process.

12 Executing the Project

After successfully configuring the CI/CD pipeline, the next step is to execute the project and verify its functionality. This involves running the Jenkins pipeline, deploying the Docker container, and validating the application's behavior through test case execution.

12.1 Triggering the Jenkins Pipeline

The Jenkins pipeline automates the build, testing, and deployment of the application. Once triggered, it pulls the latest code from the GitHub repository, builds the project, creates a Docker image, and pushes it to Docker Hub.

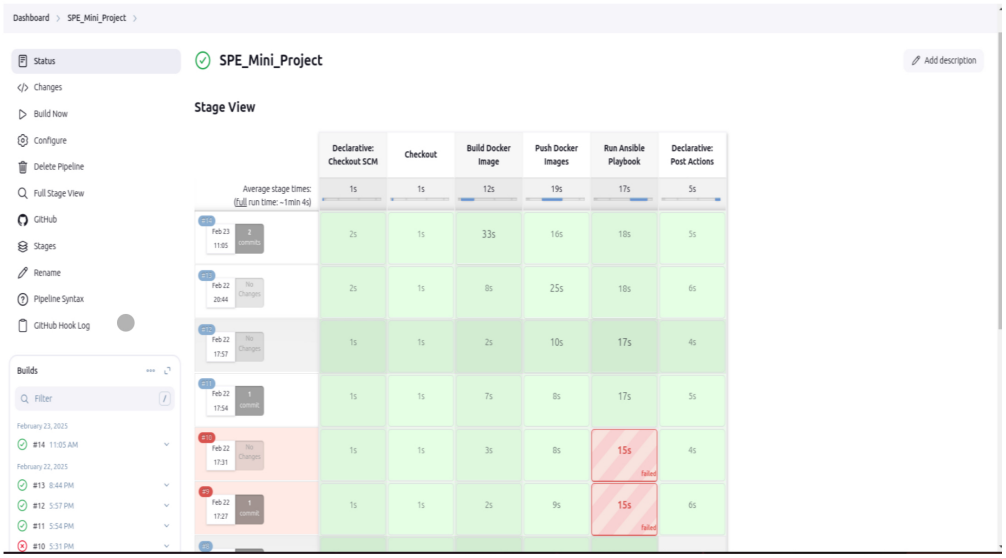

Figure 21: Jenkins Pipeline Execution

12.2 Verifying Running Containers

After deployment, we can check whether the Docker container is running using:

```
docker ps
```

The output lists active containers along with their container ID, image name, and status.

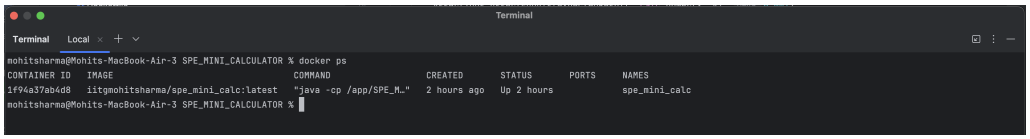

Figure 22: Verifying Running Docker Containers

12.3 Executing the Application in the Container

To run the scientific calculator application inside the Docker container, use:

```
docker exec -it spe_mini_calc \
    java -cp /app/SPE_MINI_CALCULATOR-1.0-SNAPSHOT.jar \
    com.mohit.Calculator
```

This ensures that the calculator runs inside the containerized environment.

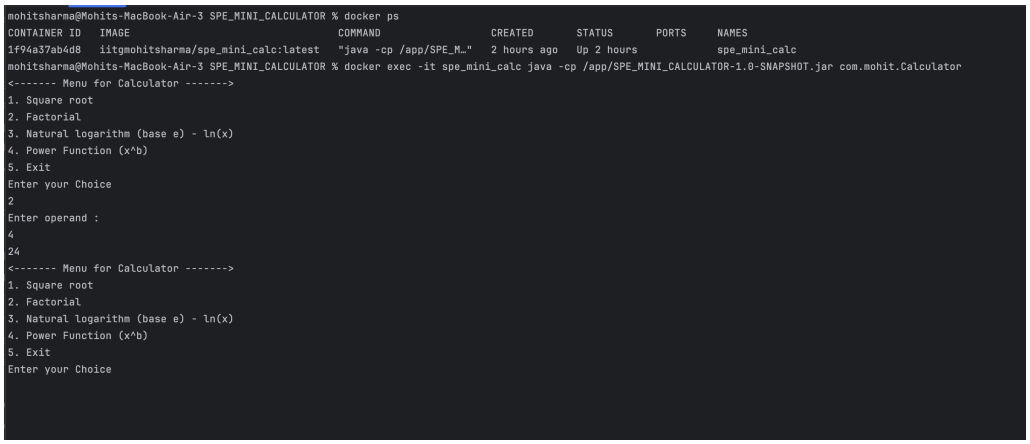

Figure 23: Executing the Scientific Calculator inside the Docker Container

12.4 Validating Test Case Execution

To confirm the correctness of the implementation, predefined test cases are executed within the container. The images below showcase the successful execution of test cases:

```
<------- Menu for Calculator ------->
1. Square root
2. Factorial
3. Natural logarithm (base e) - ln(x)
4. Power Function (x^b)
5. Exit
Enter your Choice
2
Enter operand :
5
120
<------- Menu for Calculator ------->
1. Square root
2. Factorial
3. Natural logarithm (base e) - ln(x)
4. Power Function (x^b)
5. Exit
Enter your Choice
3
Enter operand :
25
3.2188758248682006
<------- Menu for Calculator ------->
1. Square root
2. Factorial
3. Natural logarithm (base e) - ln(x)
4. Power Function (x^b)
5. Exit
Enter your Choice
```

Figure 24: Verifying Test Cases Execution - Part 1

```
<------- Menu for Calculator ------->
1. Square root
2. Factorial
3. Natural logarithm (base e) - ln(x)
4. Power Function (x^b)
5. Exit
Enter your Choice
4
Enter operand :
2
Enter Second operand
5
Answer : 32.0
<------- Menu for Calculator ------->
1. Square root
2. Factorial
3. Natural logarithm (base e) - ln(x)
4. Power Function (x^b)
5. Exit
Enter your Choice
5
End of Execution
mohitsharma@Mohits-MacBook-Air-3 SPE_MINI_CALCULATOR %
```

Figure 25: Verifying Test Cases Execution - Part 2

12.5 Conclusion

With the project successfully running inside a Docker container, the deployment pipeline has been verified. The integration of **Jenkins, Docker, and Ansible** has enabled automated testing, streamlined deployment, and ensured the scientific calculator functions as expected.

13 Project Repositories

The project's source code and Docker image are hosted on GitHub and DockerHub, ensuring easy access, version control, and containerized deployment.

13.1 GitHub Repository

The complete source code, including implementation, test cases, and CI/CD configurations, is available on GitHub.

- **URL:** GitHub Repository
- Contains:
  - Full source code and JUnit test cases
  - Jenkins pipeline configuration
  - Dockerfile and Ansible playbook

GitHub Repository Link

13.2 DockerHub Repository

A pre-built Docker image is available on DockerHub, enabling quick deployment.

- **URL:** DockerHub Repository
- Features:
  - Ready-to-use Docker image
  - Ensures consistency across environments
  - Easy to pull and run

DockerHub Repository Link

13.3 Cloning and Running the Project

To clone the repository and run the application:

```
git clone https://github.com/mohitsharma990/SPE_Mini_Calculator.git
cd SPE_Mini_Calculator
docker pull <USERNAME>/<IMAGE_NAME>:<TAG>
docker run -d -p 8080:8080 <USERNAME>/<IMAGE_NAME>
```