

Software Production Engineering

# Mini Project - Calculator

Nilay Kamat - IMT2021096

September 2024

# Contents

<b>1</b>	<b>Introduction and Setup</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Objective . . . . .	1
1.3	Tools Used . . . . .	2
1.4	Setup . . . . .	2
1.4.1	Java . . . . .	3
1.4.2	IntelliJ IDEA Ultimate . . . . .	3
1.4.3	Git . . . . .	3
1.4.4	GitHub . . . . .	4
1.4.5	Maven . . . . .	4
1.4.6	Jenkins . . . . .	4
1.4.7	Docker . . . . .	5
1.4.8	Ansible . . . . .	6
1.4.9	Ngrok . . . . .	6
<b>2</b>	<b>Project Implementation</b>	<b>8</b>
2.1	Workflow . . . . .	8
2.2	Code Development and Source Code Management . . . . .	9
2.3	Steps to Build and Run the Project . . . . .	13
2.4	CI/CD using Jenkins . . . . .	14
2.5	Containerisation . . . . .	18
2.6	Configuration Management/Deployment . . . . .	19
2.7	Git SCM Polling and Build Automation . . . . .	21
2.8	Working of Calculator & Links . . . . .	23

# Chapter 1

## Introduction and Setup

### 1.1 Introduction

In this project, we aim to create a Scientific Calculator program to perform the following functions:

1. Square Root -  $\sqrt{x}$
2. Factorial -  $x!$
3. Natural Logarithm(base e) -  $\ln(x)$
4. Power -  $x^h$

The goal of the project is to focus on the DevOps tools and workflows that are used in order to develop, deploy and monitor programs in an application.

### 1.2 Objective

The specific objectives of this project are:

1. To create a local project in an IDE.
2. To create a local repository using version control and push it to a remote version control repository.
3. To create a pipeline that will automatically build the project and run test cases every time changes are pushed to the repository.

4. To automatically containerise the project and push it to a remote container repository.
5. To deploy the container to a target machine using Jenkins.

## 1.3 Tools Used

1. **IntelliJ IDEA Ultimate:** A *powerful IDE and Code Editor* for Java and other languages, offering advanced development, debugging, and integration features for seamless coding and deployment.
2. **Git & GitHub:** Git is a *version control system and source code management* tool that allows developers to collaborate on code and track changes. GitHub is a web-based platform for *version control and collaboration*, allowing developers to host, review, and manage code repositories using Git.
3. **Apache Maven:** A *build automation* tool that helps manage dependencies and build Java-based projects.
4. **Jenkins:** a continuous integration and continuous delivery (CI/CD) tool that automates the build, test, and deployment processes.
5. **Docker:** a *containerization platform* that enables developers to package applications and dependencies into portable, lightweight containers.
6. **Ansible:** a *configuration management tool* that automates the deployment and management of infrastructure and applications.
7. **GitHub Webhooks:** a tool that *triggers automated actions* when specific events occur in a GitHub repository.
8. **ngrok:** a tool that *creates secure tunnels* to expose local servers to the public internet, useful for testing and development purposes.
9. **JUnit:** an open-source *testing framework* for Java that enables developers to write and run unit tests.

## 1.4 Setup

We need to install and setup some tools and frameworks before getting started on the workflow.

### 1.4.1 Java

To install Java and check the version, we use the following commands:

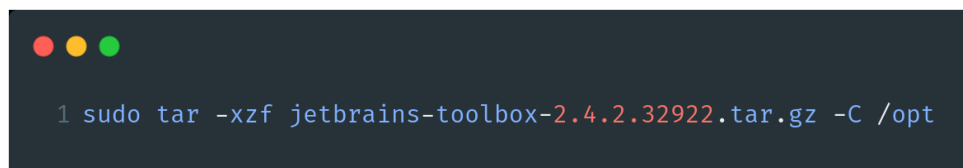


```
1 sudo apt install default-jre
2
3 java --version

nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ java --version
openjdk 17.0.12 2024-07-16
OpenJDK Runtime Environment (build 17.0.12+7-Ubuntu-1ubuntu222.04)
OpenJDK 64-Bit Server VM (build 17.0.12+7-Ubuntu-1ubuntu222.04, mixed mode, sharing)
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```

### 1.4.2 IntelliJ IDEA Ultimate

To install the code editor, go to the JetBrains website([Website Link : ToolBox Download](#)) and download the ToolBox App in tarball(.tar.gz)form. Then, extract and run it using the command shown in the image. Then, open the ToolBox App and download IntelliJ IDEA Ultimate.



```
1 sudo tar -xzf jetbrains-toolbox-2.4.2.32922.tar.gz -C /opt
```

### 1.4.3 Git

To install Git and check the version, we use the following commands:

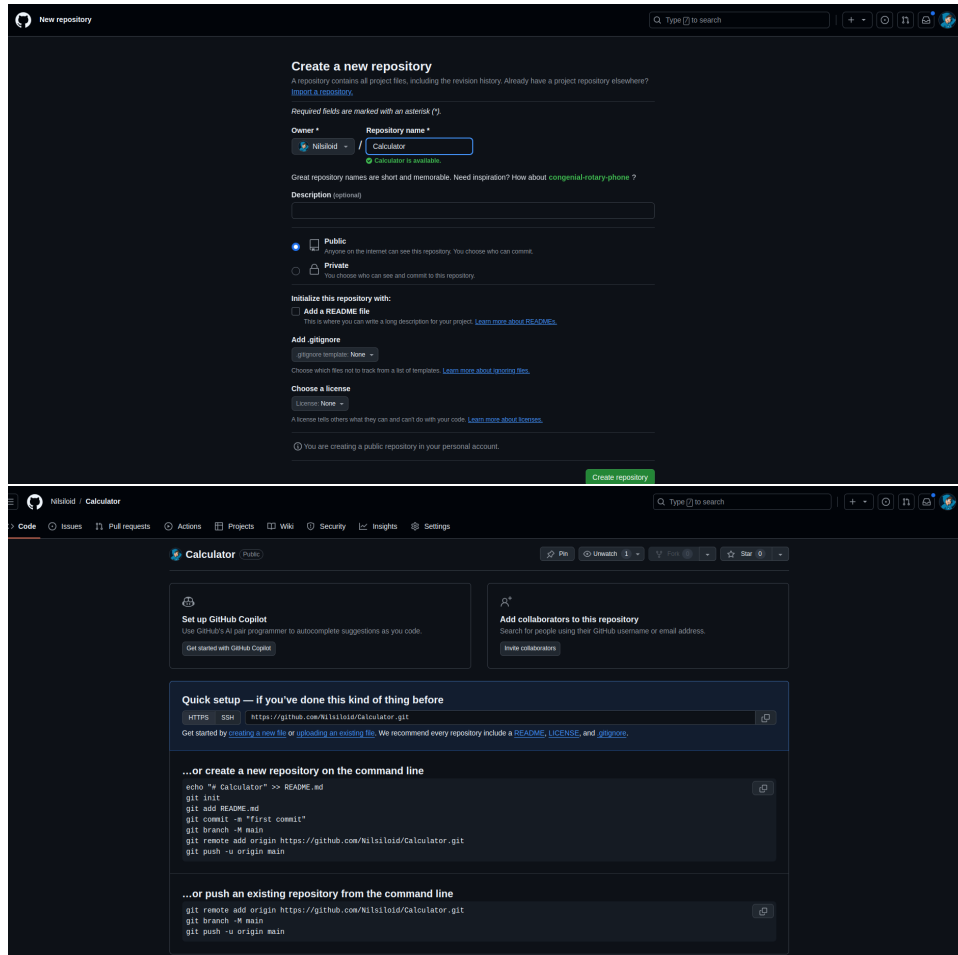


```
1 sudo apt install git
2
3 git --version

nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ git --version
git version 2.34.1
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```

### 1.4.4 GitHub

Visit GitHub and create an account if you don't have one. Then create a new repository named "Calculator" or "SPE-Mini Project".



### 1.4.5 Maven

To install Maven and check the version, we use the following commands:

```
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ mvn --version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcd97d260186937)
Maven home: /home/nilay/IIITB/SEM7/SPE/apache-maven-3.9.9
Java version: 17.0.12, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
Default locale: en_IN, platform encoding: UTF-8
OS name: "linux", version: "6.8.0-40-generic", arch: "amd64", family: "unix"
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```

### 1.4.6 Jenkins

To install Jenkins and check the version, we use the following commands:

```
1 sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
2   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
3
4 echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
5   https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
6   /etc/apt/sources.list.d/jenkins.list > /dev/null
7
8 sudo apt-get update
9
10 sudo apt-get install jenkins
```


```
1 sudo systemctl start jenkins
2
3 sudo systemctl status jenkins
```

### 1.4.7 Docker

To install Docker, check the version and ensure that both Jenkins and Docker are in the same user group, we use the following commands:

```
1 sudo apt install curl
2 curl -fsSL https://get.docker.com -o get-docker.sh
3 sh get-docker.sh
4 sudo docker --version


nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker --version
[sudo] password for nilay:
Docker version 27.3.1, build ce12230
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```



```
1 sudo tail /etc/gshadow
2 sudo usermod -aG docker jenkins
3 sudo systemctl restart jenkins
```

### 1.4.8 Ansible

To install Ansible and check the version, we use the following commands:



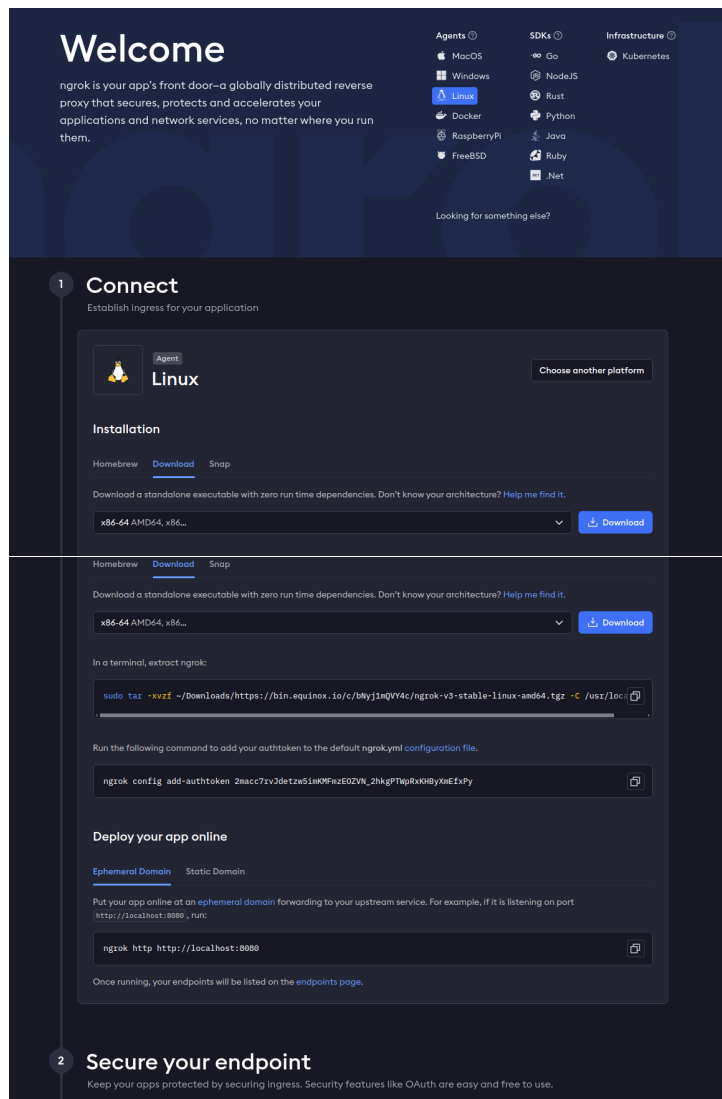
```
1 sudo apt-add-repository ppa:ansible/ansible
2 sudo apt update
3 sudo apt install ansible
```

```
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo ansible --version
ansible 2.10.8
  config file = None
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```

### 1.4.9 Ngrok

Visit the Ngrok official [website](#) and sign up for a free account. Then download the Ngrok binary executable for your OS.





Once you have downloaded ngrok, you run the following command in the same directory that you have downloaded ngrok, to extract it to the destination:

```
1 sudo tar xvzf ~/Downloads/ngrok-v3-stable-linux-amd64.tgz -C /usr/local/bin
```

Ensure you replace `ngrok-v3-stable-linux-amd64.tgz` with the appropriate version of the zip you downloaded. Once you do that, you run the following command:

```
1 ngrok config add-authtoken 2macc7rvJdetzw5imKMFmzEOZVN_2hkgPTWpRxKHByXmEfXPy
```

Replace the auth token with what is shown in your account dashboard.

## Chapter 2

# Project Implementation

### 2.1 Workflow

1. **Code Development:** Utilised IntelliJ IDEA Ultimate to code the program for a Scientific Calculator.
2. **Version Control/Source Code Management:** Stored the code in a Git repository hosted on GitHub, enabling collaborative development and tracking changes.
3. **Building:** Employed Apache Maven as a build tool to automate the compilation and packaging of the calculator program.
4. **Continuous Integration and Continuous Delivery (CI/CD):** Used Jenkins as a CI/CD tool to automatically pull code from GitHub, perform unit tests, and deploy the program to production environments.
5. **Containerisation:** Package the calculator program into a Docker image, ensuring consistent deployment across different environments.
6. **Configuration Management:** Utilise Ansible to automate the configuration of infrastructure resources, ensuring consistency and streamlined deployment.
7. **Git SCM Polling and Build Automation:** Employ Ngrok to create a secure tunnel to GitHub, enabling GitHub webhooks to trigger Jenkins builds upon code updates.

## 2.2 Code Development and Source Code Management

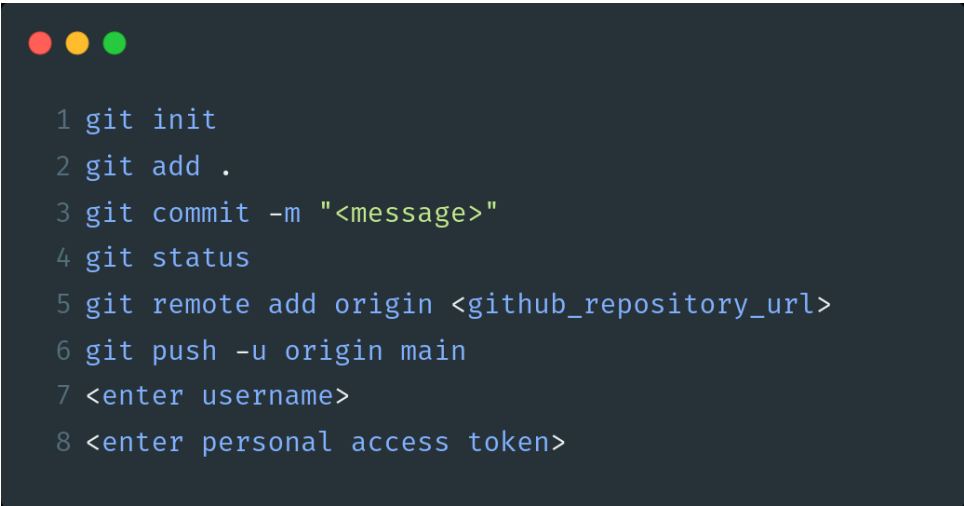
We start working on the implementation by creating a Project in our code editor(here, IntelliJ IDEA). Create a new project in IntelliJ with the name "Calculator" and choose "Maven" as the build tool. From the dropdown menu, we also select the default Java JDK.

Source Code Management tools allow developers to collaborate on code, track changes and maintain version control of their codebase. They provide features such as branching and merging, which allow developers to work on different versions of the codebase and merge changes together.

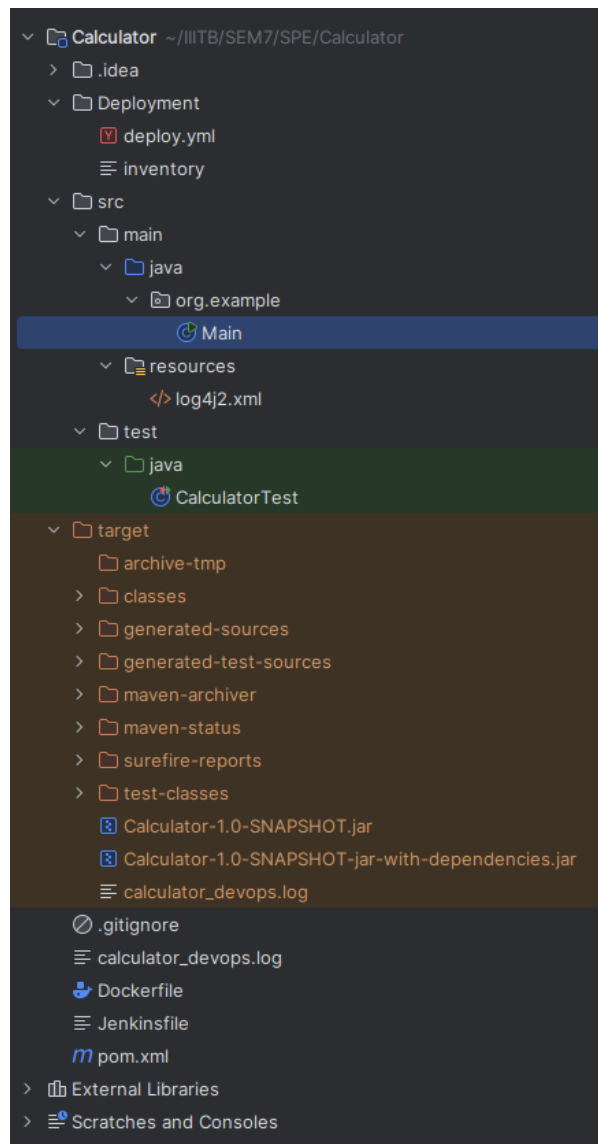
### Tools used:

1. Programming Language - Java(version 11)
2. Logging - Log4j
3. Testing - JUnit
4. Build - Maven
5. SCM - Git and GitHub

**Git Workflow:** The following are the commands used to initialise a new Git repository, add files to the staging area, commit changes, set up a remote repository on GitHub, and push the changes to the remote repository.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a list of eight Git commands, each preceded by a number from 1 to 8.

```
1 git init
2 git add .
3 git commit -m "<message>"
4 git status
5 git remote add origin <github_repository_url>
6 git push -u origin main
7 <enter username>
8 <enter personal access token>
```

**Directory Structure:****Description of Folders/Files:****1. Folders:**

- (a) **Deployment:** Contains files for Ansible.
- (b) **src:** Contains Java source code.
- (c) **main:** Contains the Java code for the Scientific Calculator Program.
- (d) **resources:** Holds the configuration files for the *log4j2.xml* dependency
- (e) **test:** Holds Unit tests for the source code.
- (f) **target:** Stores output JAR and class files.

## 2. Files:

- (a) **Calculator.java:** Contains the main function including the menu and the required functions(as follows):
  - i. Square Root -  $\sqrt{x}$
  - ii. Factorial -  $x!$
  - iii. Natural Logarithm(base e) -  $\ln(x)$
  - iv. Power -  $x^h$
- (b) **CalculatorTest.java:** Contains JUnit test cases for the functions described above. Each function has multiple test cases.
- (c) **log4j2.xml:** Includes basic configurations for log4j, such as which appender to use (e.g., console appender, file appender) and other settings like pattern and log level.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
    </Console>
    <File name="FileAppender" fileName="calculator_devops.log" immediateFlush="false" append="true">
      <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS}[%t] %-5level %logger{36} - %msg%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug">
      <AppenderRef ref="FileAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

- (d) **pom.xml:** Contains information about the project and configuration details used by Maven to build the project.

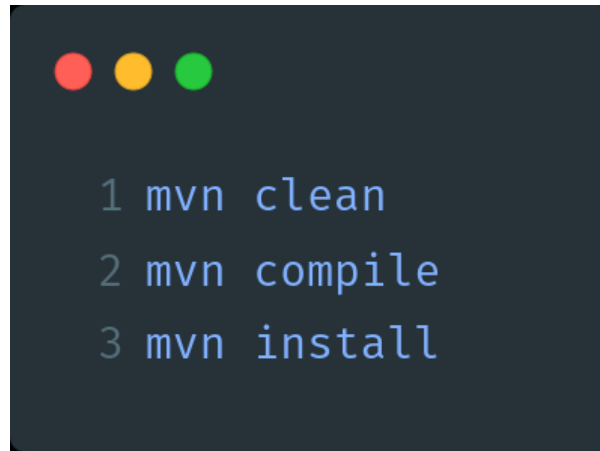
```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
  <!--https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
    <scope>compile</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.3.0</version>
      <executions>
        <execution>
          <phase>package</phase><goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <archive>
              <manifest>
                <mainClass>org.example.Main</mainClass>
              </manifest>
            </archive>
            <descriptorRefs>
              <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

- (e) **.gitignore:** Contains files patterns to be ignored by Git.
- (f) **calculator\_devops.log:** Log files for the project.
- (g) **Dockerfile:** Configuration file for docker.

## 2.3 Steps to Build and Run the Project

To generate a JAR file with dependencies, we run the following commands:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays three numbered commands in a light blue monospaced font: 1 mvn clean, 2 mvn compile, and 3 mvn install.

```
1 mvn clean
2 mvn compile
3 mvn install
```

1. **mvn clean:** Removes the "target" folder, ensuring a fresh start for the subsequent compilation. This step eliminates any previous build artifacts.
2. **mvn compile:** Compiles the project and its associated test cases. This phase ensures the code is error-free and ready for the subsequent steps.
3. **mvn install:** Generates the JAR file. This final step packages the project, creating the desired output artifact (JAR file) once the compilation is successful.

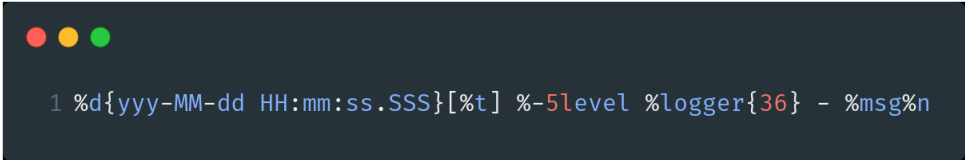
Maven will build the project and check all test cases. Once completed, a "target" directory will be created in the current directory, which will contain the JAR file.

Now, we navigate to the "target" folder using the command: *cd target.*

Then, we run the JAR file using the command: *java -jar filename.jar*

Within the Maven configuration file(pom.xml), the *mainClass* tag specifies the path to the main Java file following the package structure. Additionally, the *descriptorRef* tag is employed to modify the default output JAR file name. To include project dependencies, the *dependencies* tag is utilised, enabling the addition of external libraries. In our scenario, we've incorporated dependencies such as log4j, utilised for logging functionality, and JUnit, employed for testing purposes.

**Log4j.xml file:**

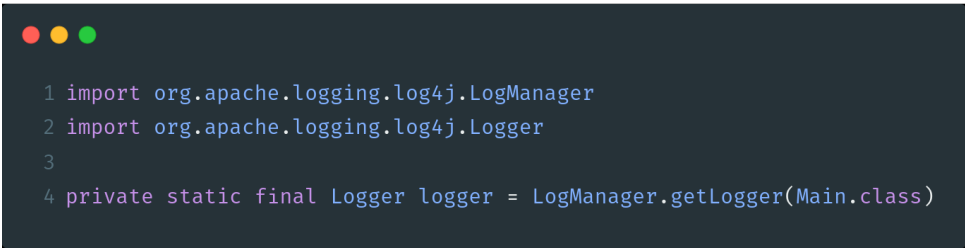


```
1 %d{yyy-MM-dd HH:mm:ss.SSS}[%t] %-5level %logger{36} - %msg%n
```

The specified logging format shown above includes the following components:

1. **%d{yyy-MM-dd HH:mm:ss.SSS}**: Represents the timestamp with millisecond precision.
2. **%t**: Indicates the running thread's name.
3. **%-5level**: Denotes the log level with left alignment and a maximum width of 5 characters.
4. **%logger36**: Refers to the logger's name within a maximum of 36 characters.
5. **%msg**: Represents the user-written message contained in the source code.

The necessary imports for Log4j is shown in below image along with creation of an instance of the Logger in Log4j which involves using a statement to initialise a Logger object within the code.



```
1 import org.apache.logging.log4j.LogManager
2 import org.apache.logging.log4j.Logger
3
4 private static final Logger logger = LogManager.getLogger(Main.class)
```

## 2.4 CI/CD using Jenkins

- Jenkins is an open-source automation tool written in Java with plugins built for continuous integration.
- Jenkins is utilized to continuously build and test software projects, simplifying the process for developers to integrate changes, and allowing users to obtain up-to-date builds with ease.
- After installing Jenkins using the procedure mentioned in the previous chapter, go to <https://localhost:8080>, then browse to **Manage Jenkins** → **Plugin Manager** → **Available Plugins** and install the following necessary plugins.
  - Git plugins & GitHub plugins



- Maven Integration
- Docker plugin & Docker pipeline
- Ansible plugin
- JUnit plugin

Following this, browse to *Manage Jenkins* → *Credentials* → *System* → *Global Credentials* and create 2 credentials as shown in image below:

The screenshot displays the Jenkins 'Global Credentials' configuration page. It contains two identical forms for creating credentials. Each form has the following fields and options:

- Scope:** A dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'.
- Username:** A text input field. The first entry has 'nilay95' and the second has 'nilay'.
- Treat username as secret:** An unchecked checkbox.
- Password:** A field with a lock icon and the text 'Concealed'. A blue 'Change Password' button is to the right.
- ID:** A text input field. The first entry has 'DockerHubCred' and the second has 'localhost'.
- Description:** A text input field. The first entry has 'Docker Hub Credential' and the second has 'Localhost User Login Credentials'.
- Save:** A blue button at the bottom of each form.

We will then establish a Jenkins pipeline comprising of 6 distinct stages:

1. **Stage 1 : Git Clone** - This stage clones the repository from the main branch of the provided GitHub URL.

```
1 stage('Stage 1: Git Clone'){
2     steps{
3         git branch: 'main',
4         url:'https://github.com/Nilsiloid/SPE-MiniProject'
5     }
6 }
```

2. **Stage 2 : Maven Build** - Executes the mvn clean install command to build the project and resolve dependencies using Maven.

```
1 stage('Step 2: Maven Build'){
2     steps{
3         sh 'mvn clean install'
4     }
5 }
```

3. **Stage 3 : Build Docker Image** - Uses the Docker build process to create a Docker image named nilay95/calculator:latest from the project.

```
1 stage('Stage 3: Build Docker Image'){
2     steps{
3         script{
4             docker_image = docker.build "nilay95/calculator:latest"
5         }
6     }
7 }
```

4. **Stage 4 : Push Docker image to hub** - Authenticates using DockerHub credentials (DockerHubCred) and pushes the created image to Docker Hub.

```
1 stage('Stage 4: Push docker image to hub') {
2     steps{
3         script{
4             docker.withRegistry('', 'DockerHubCred'){
5                 docker_image.push()
6             }
7         }
8     }
9 }
```

5. **Stage 5 : Clean Docker images** - Removes any stopped Docker containers and unused Docker images to free up space.

```
1 stage('Stage 5: Clean docker images'){
2     steps{
3         script{
4             sh 'docker container prune -f'
5             sh 'docker image prune -f'
6         }
7     }
8 }
```

6. **Stage 6 : Ansible Deployment** - Executes an Ansible playbook (Deployment/deploy.yml) to automate the deployment of the application using the specified inventory file and settings.



```
1 stage('Step 6: Ansible Deployment'){
2     steps{
3         ansiblePlaybook becomeUser: null,
4         colorized: true,
5         credentialsId: 'localhost',
6         disableHostKeyChecking: true,
7         installation: 'Ansible',
8         inventory: 'Deployment/inventory',
9         playbook: 'Deployment/deploy.yml',
10        sudoUser: null
11    }
12 }
```

## 2.5 Containerisation

- Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure.
- Containers are lightweight, portable, and self-contained environments that enable developers to package an application with all its dependencies, libraries, and configuration files, ensuring that it runs consistently across different environments.
- We will use Docker to create containers. Docker is an open-source tool that enables developers to build, package, and deploy applications in a containerized environment.

We will now create a Docker container for our project by creating a Dockerfile.

- **FROM:** We use the OpenJDK 11 base image to build our image.
- **COPY:** Copy jar file from source on the host machine into the container's file system.
- **WORKDIR:** Changes the current working directory.
- **ENTRYPOINT:** Specify the command that should be run when a container based on the image is started.

```
1 FROM openjdk:11
2 COPY ./target/Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar ./
3 WORKDIR ./
4 CMD ["java", "-cp", "Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar", "org.example.Main"]
```

The purpose of the above code is to generate a Docker container with OpenJDK version 11 as the base image which acts like a JVM. The final container can be run independently without any dependencies. The Jenkins Pipeline script has been configured to include the Dockerfile and Docker commands, automating the process of building and pushing the Docker Image to DockerHub. In the pipeline script, `docker_image = docker.build "nilay95/calculator"` builds the docker image. The following command allows verification of the successful creation of a Docker image.

The image is a composite of two parts. The top part is a terminal window with a dark background and three colored circles (red, yellow, green) in the top left corner. It displays the command `1 sudo docker images` in a light blue font. The bottom part is a screenshot of the Docker Hub page for the repository `nilay95/calculator`. The page shows the repository name, a last push time of about 13 hours ago, and a table of images. The table has columns for Tag, OS, Type, Pulled, and Pushed. The 'latest' tag is listed with an 'Image' type, pushed 13 hours ago. To the right of the table, there are sections for 'Docker commands' (showing `docker push nilay95/calculator:tagname`) and 'Automated Builds' (with a note about connecting to GitHub or Bitbucket).

```
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker images
[sudo] password for nilay:
REPOSITORY                TAG                IMAGE ID           CREATED           SIZE
nilay95/calculator        latest            b47ed0429f8e      14 hours ago     654MB
openquantumsafe/curl      latest            99e4a58e76ed      14 months ago    47.1MB
hello-world               latest            9c7a54a9a43c      16 months ago    13.3kB
nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$
```

**nilay95/calculator**

Last pushed about 13 hours ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

**Tags**

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	13 hours ago	13 hours ago

[See all](#)

**Docker commands**

To push a new tag to this repository:

```
docker push nilay95/calculator:tagname
```

[Public View](#)

**Automated Builds**

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#) .

[Upgrade](#)

## 2.6 Configuration Management/Deployment

- We will use Ansible for local deployment. Ansible is a suite of software tools that enables infrastructure as code.
- In Ansible, managed hosts or servers which are controlled by the Ansible control node are defined in a host inventory file. The Ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate.

- We are going to pull the images from the DockerHub and create containers using Ansible. We will create a Deployment folder and create two files named inventory and deploy.yml.

```
1 localhost ansible_user = local ansible_user = nilay
```

- Ansible Playbooks offer a repeatable, re-usable, simple configuration management.
- Playbooks consist of one or more plays run in a particular order. A play is an ordered set of tasks run against hosts chosen from your inventory. Plays define the work to be done. Each play contains a set of hosts to configure, and a list of tasks to be executed.

```
1 ---
2 - name: Pull Docker Image of Calculator
3   hosts: all
4   connection: local
5   vars:
6     ansible_python_interpreter: /usr/bin/python3.8
7   tasks:
8     - name: Pull image
9       docker_image:
10        name: nilay95/calculator:latest
11        source: pull
12     - name: Start docker service
13       service:
14        name: docker
15        state: started
16     - name: Running container
17       shell: docker run -it -d --name Calculator nilay95/calculator
```

- The Jenkins Pipeline script has been configured to execute the Ansible Playbook automatically.
- Now, run the following commands to view all docker images and then to create a container from a docker image.

```
1 sudo docker images
2 sudo docker run -it -d --name calc nilay95/calculator:latest
```

- Run the following commands to view all existing containers and to execute required container.

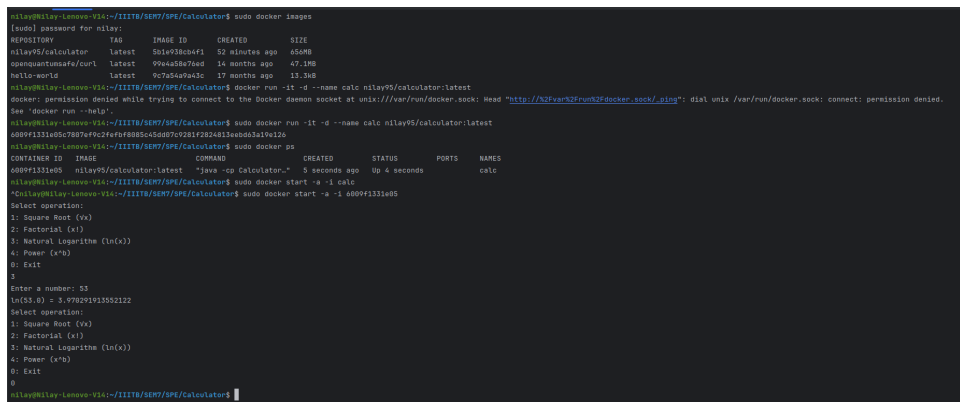


```

1 sudo docker ps
2 sudo docker start -a -i 6009f1331e05

```

- The container executes the program as shown below:

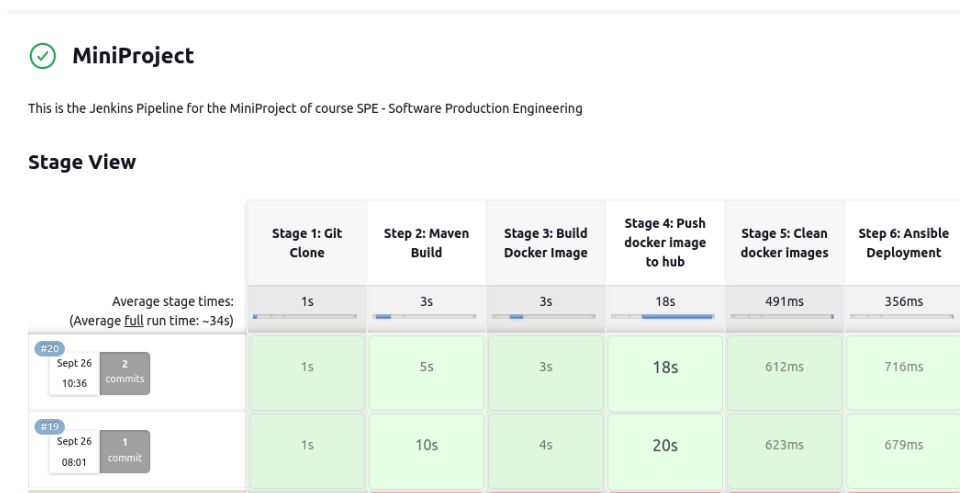


```

nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
nllay95/calculator  latest         5d1e93dc04f1    52 minutes ago  46MB
openquantumsafe/curl latest         09e4a58a70ed    14 months ago  47.1MB
hello-world         latest         7c7864e9a43c    17 months ago  11.3kB
nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ docker run -it -d --name calc nllay95/calculator:latest
docker: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Head "http://N2Fvar92Frun92Fdocker.sock/_ping": dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker run -it -d --name calc nllay95/calculator:latest
6009f1331e057807ef6c2efef8085cd5d087c281f2824813e0d03a10v12b
nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS   NAMES
6009f1331e05   nllay95/calculator:latest           "java -cp Calculator-"   5 seconds ago Up 4 seconds   calc
nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker start -a -i calc
^Cnllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$ sudo docker start -a -i 6009f1331e05
Select operation:
1: Square Root (√x)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^y)
0: Exit
3
Enter a number: 53
ln(53.0) = 3.970291913552122
Select operation:
1: Square Root (√x)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^y)
0: Exit
0
nllay@nllay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator$

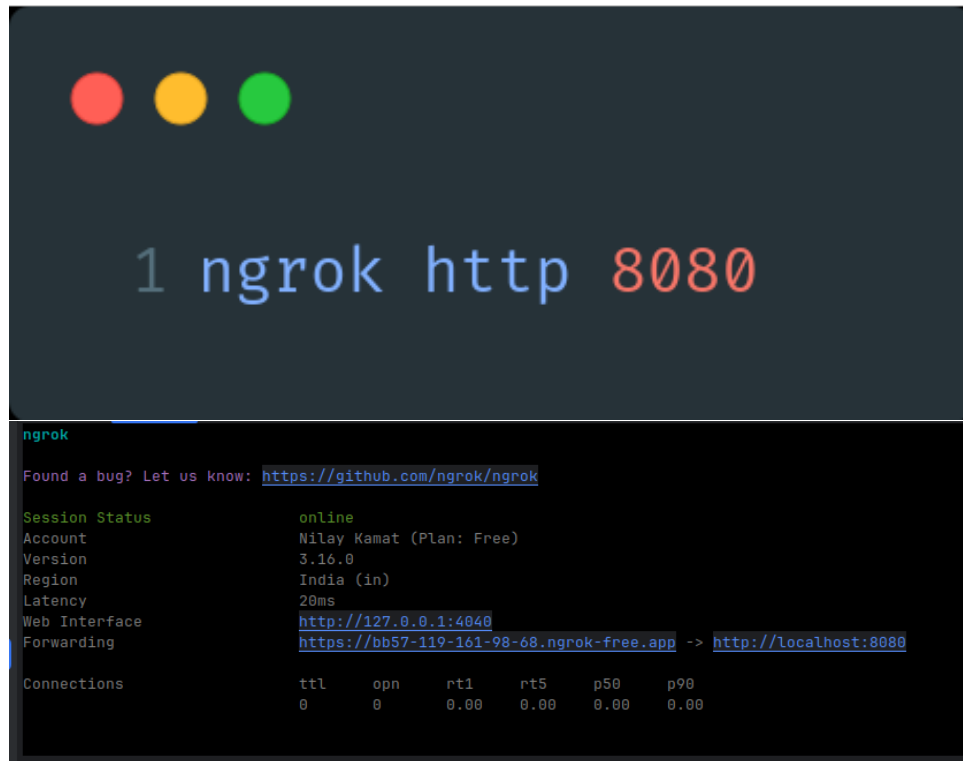
```

- At the end of these 6 steps of the workflow, the Jenkins Pipeline is completed and the stage view should be as follows:



## 2.7 Git SCM Polling and Build Automation

- Open a terminal window and enter the below command. This command will establish an HTTP tunnel using Ngrok, exposing the local server running on port 8080 to the internet.



```

1 ngrok http 8080

ngrok
Found a bug? Let us know: https://github.com/ngrok/ngrok

Session Status      online
Account             Nilay Kamat (Plan: Free)
Version             3.16.0
Region              India (in)
Latency              20ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://bb57-119-161-98-68.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      0      0.00   0.00   0.00   0.00

```

- Copy the forwarding URL provided by Ngrok. Subsequently, create a GitHub webhook and utilise this URL as the payload URL for the webhook configuration. GitHub initiates a test connection, and upon successful configuration, a '200 OK' message confirms the proper setup.



```

ngrok
Sign up to try new private endpoints https://ngrok.com/new-features-update?ref=private

Session Status      online
Account             Nilay Kamat (Plan: Free)
Version             3.16.0
Region              India (in)
Latency              26ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://6b94-103-156-19-229.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      1      0.00   0.00   0.00   0.00

HTTP Requests
-----
11:34:03.344 IST POST /github-webhook/      200 OK

```

- Now, we'll update the Jenkins Pipeline script to a Jenkinsfile configure a build trigger for Git SCM polling. This setup ensures that our pipeline automatically initiates the build process whenever Jenkins detects a new commit made to the associated GitHub repository.



Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Nilsiloid/SPE-MiniProject.git

Credentials ?

- none -

+ Add ▾

Advanced ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Add Branch

Repository browser ?

(Auto)

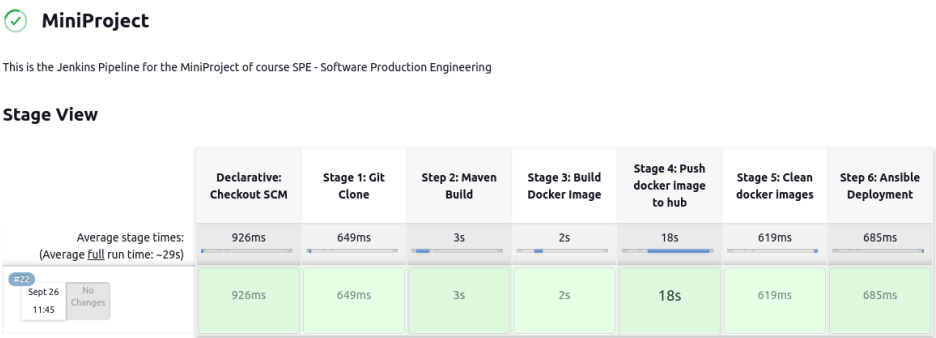
Additional Behaviours

Add ▾

Script Path ?

Jenkinsfile

- Upon making any commits, the Jenkins Pipeline automatically initiates the build process and the final pipeline+build is as follows:



## 2.8 Working of Calculator & Links

The following images show the working of the Calculator in the terminal.

```

nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator/target$ java -jar Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
1
Enter a number: 16
√16.0 = 4.0
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
2
Enter an integer: 12
12! = 479001600
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
3
Enter a number: 25
ln(25.0) = 3.2188758248682006
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
4
Enter base: 4
Enter exponent: 3
4.0^3.0 = 64.0
Select operation:
1: Square Root (vx)
2: Factorial (x!)

nilay@Nilay-Lenovo-V14:~/IIITB/SEM7/SPE/Calculator/target$ java -jar Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
1
Enter a number: -4
Error: Cannot take square root of negative number.
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
2
Enter an integer: -1
Error: Cannot calculate factorial of negative number.
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
3
Enter a number: 0.2
ln(0.2) = -1.6094379124341003
Select operation:
1: Square Root (vx)
2: Factorial (x!)
3: Natural Logarithm (ln(x))
4: Power (x^b)
0: Exit
4
Enter base: -3
Enter exponent: 7
-3.0^7.0 = -2187.0
Select operation:
1: Square Root (vx)
2: Factorial (x!)

```

1. GitHub repository - [SPE-MiniProject Repository](#)

2. Docker repository - [Calculator](#)