Mini Project

Simple Calculator

**~Samarjeet Wankhade(IMT2021013)**

# Introduction

The project is to develop a simple calculator program using DevOps tools and workflows. The goal is completely focused on DevOps tools and workflows that are used to develop, deploy, and monitor the program.

# 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.
3. To push the local repository to a remote version control repository.
4. To create a pipeline that will automatically build the project and run test cases every time changes are pushed to the repository.
5. To automatically containerise the project and push it to a remote container repository.
6. To deploy the container to a target machine using Jenkins.
7. All of the above steps to be done in windows.

# Tools Used

* Code Editor: *IntelliJ IDEA Ultimate*
* Version Control System: *Git*
* Remote Repository: *GitHub*
* Build Tool: *Apache Maven*
* CI/CD Pipeline: *Jenkins*
* Containerisation: *Docker*
* Configuration Management: *Ansible*
* Git SCM Polling and Build Automation(optional): *Ngrok, GitHub webhooks (I am building periodically so not compulsary)*

# Workflow

1. **Code Development:** Utilised IntelliJ IDEA Ultimate to write the calculator program's code.
2. **Version Control:** 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.

1. **Git SCM Polling and Build Automation(optional):** Employ Ngrok to create a secure tunnel to GitHub, enabling GitHub webhooks to trigger Jenkins builds upon code updates.

**Link to GitHub Repository:** [**Link**](https://github.com/SamNotLazy/SPE_Sci_Calc.git)

# **Installing and Setting up tools**

# **Java Development kit**

1. Install Java on Windows:

* Step 1: Download Java Development Kit (JDK)
  + Go to the official Oracle website: [Java SE Downloads](https://www.oracle.com/java/technologies/javase-downloads.html).
  + Download the latest version of the JDK installer for Windows.
* Step 2: Install Java
  + Run the downloaded JDK installer.
  + Follow the installation prompts and select the desired location for the JDK.
  + By default, Java will be installed in C:\Program Files\Java\jdk-<version>.

2. Set up Environment Variables:

* Step 1: Open Environment Variables
  + Right-click on This PC (or My Computer) and select Properties.
  + Click on Advanced system settings on the left.
  + In the System Properties window, click on Environment Variables.
* Step 2: Add JAVA\_HOME
  + In the Environment Variables window, click New under the System variables section.
  + Set the variable name to JAVA\_HOME.
  + Set the variable value to the path where the JDK is installed (e.g., C:\Program Files\Java\jdk-<version>).
  + Click OK.
* Step 3: Update the PATH Variable
  + In the System variables section, find and select the Path variable and click Edit.
  + Click New and add the following path: **%JAVA\_HOME%\bin**
  + Click OK to close all windows.

3. Verify Java Installation:

* Open the Command Prompt and type the following command: **java --version**
* This will show the installed Java version and confirm the setup.

# **Git**

# 1. Download and Install Git: Go to the official Git website and download the Git for Windows installer (an .exe file). Run the Installer

# 2. Verify Git Installation: In the Command Prompt (or Git Bash), run the following command to verify the installation:

# **git –version**

# 

# **GitHub**

Visit [www.github.com](http://www.github.com/) to create an account.

# **Maven**

# 1: Go to the official Apache Maven website: Maven Download. Under the Files section, download the binary zip archive (apache-maven-<version>-bin.zip).

# 2. Install Maven: Extract the Archive. After downloading the Maven .zip file, extract it to a location of your choice, for example, C:\Program Files\Apache\Maven\.

# 3: Set up Environment Variables

# 3.1: Set MAVEN\_HOME Variable and path to it similar to JDKs steps. Right-click on This PC (or My Computer) and select Properties. In the System Properties window, click on Environment Variables. Under System Variables, click New and add: Variable name: MAVEN\_HOME, Variable value: The path to where you extracted Maven (e.g., C:\Program Files\Apache\Maven\apache-maven-<version>).

# 3.2: Add Maven to the Path. In the System variables section, find the Path variable and click Edit. Click New and add the following path: **%MAVEN\_HOME%\bin**. Click OK to close all windows.

# 4. Verify Maven Installation: Run the following code to check if it is installed properly or not.

# **mvn -version**

# If Maven is installed and configured correctly, this will display the Maven version, Java version, and operating system information.

# 

# **Jenkins**

# 1: Download Jenkins for Windows: Go to the official Jenkins website: Jenkins Download. Scroll down to the Windows section and download the Jenkins .msi installer.

# 2: Install Jenkins: Run the downloaded .msi installer and follow the installation prompts. Choose the installation path for Jenkins (e.g., C:\Program Files\Jenkins).

# During installation, Jenkins will ask you to choose a port to run Jenkins on (default is 8080).Jenkins will also create a Windows service that allows it to run in the background. You will need to have Jenkins server running as long as the CI/CD is needed that too with administrative privilages.

# 3:Running the Jenkins Server: Open a terminal with administrative privilages by pressing windows + X. Go to ‘**C:\Program Files\Jenkins’**. Run this command to run it: ‘ **java -jar jenkins.war**‘(you may need to close the instance of already running Jenkins which does not have admin privilages from task maneger).

# **WSL**

Open any teminal in windows and type the command ‘wsl --install’.

# **Docker**

# 1: Download Docker Desktop for Windows from Docker's website. Run the .exe installer and follow the prompts. Enable WSL 2 integration if prompted.

# 2. Start Docker and Verify Installation. Launch Docker Desktop (it may start automatically) .Open Command Prompt or PowerShell and type **docker --version**

# Test by running:

# docker run hello-world

# Note you need to keep the docker Desktop open like the Jenkins server till the CI/CD is needed.

# 

# **Ansible**

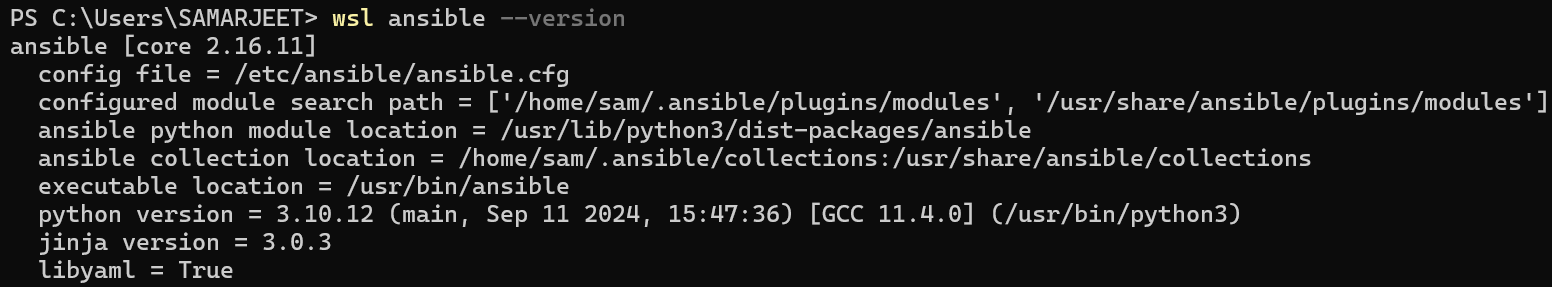
1: Open WSL and run:

**sudo apt update**

**sudo apt install ansible**

2. Verify Ansible Installation: In WSL, check the version:

**ansible –version**

****

# **DevOps Solution - Project Implementation**

To begin our journey, we'll create the project in IntelliJ IDEA. Launch IntelliJ IDEA and navigate to the New Project window. Provide the project name

"Calculator" and select Maven as the build tool. Choose the default Java JDK from the dropdown menu. Also chose the git version Control and configure it with your repository.

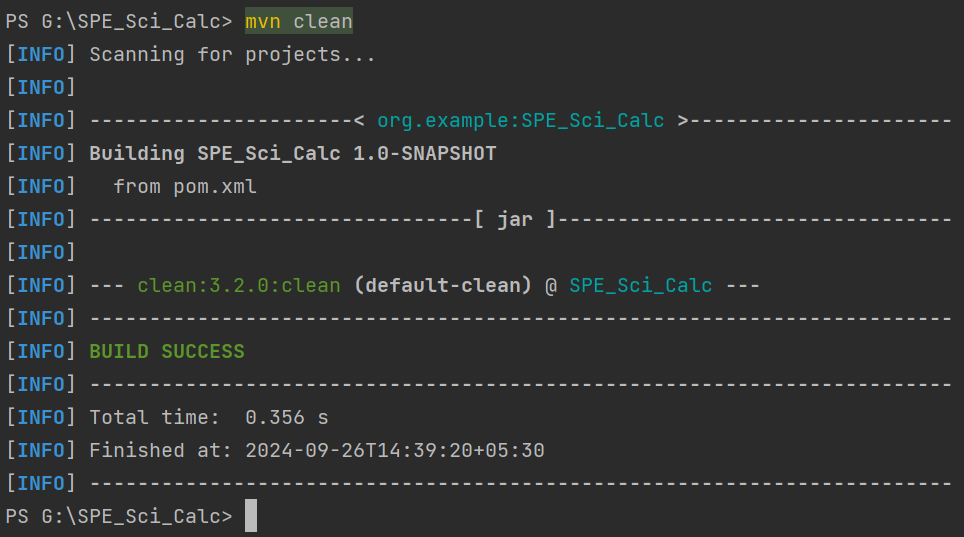
## Project Structure

Folders: Files:

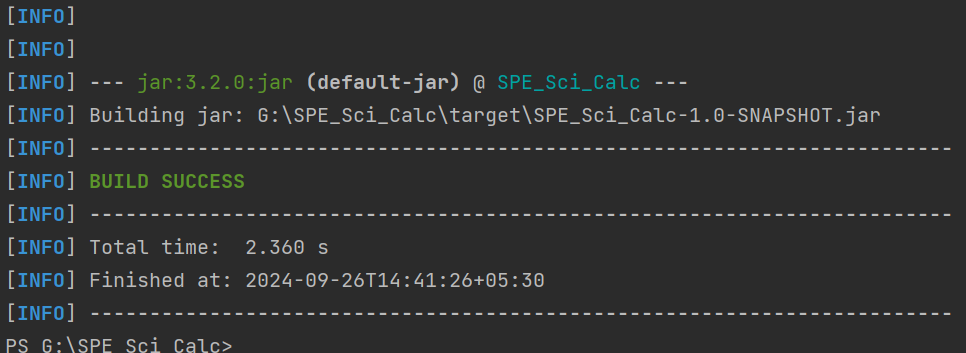
|  |  |  |  |
| --- | --- | --- | --- |
| .idea | Contains configuration files for IntelliJ IDEA | .gitignore | Contains file patterns to be ignored by Git |
| src | Contains Java source code | Dockerfile | Configuration file for docker |
| main | Contains the Java code for Calculator program | pom.xml | Maven configuration file |
| resources | Holds configuration files for the log4j2.xml dependency | — | — |
| test | Holds Unit tests for the source code | — | — |
| target | Stores output JAR and class files | — | — |

## Steps to build and run the project

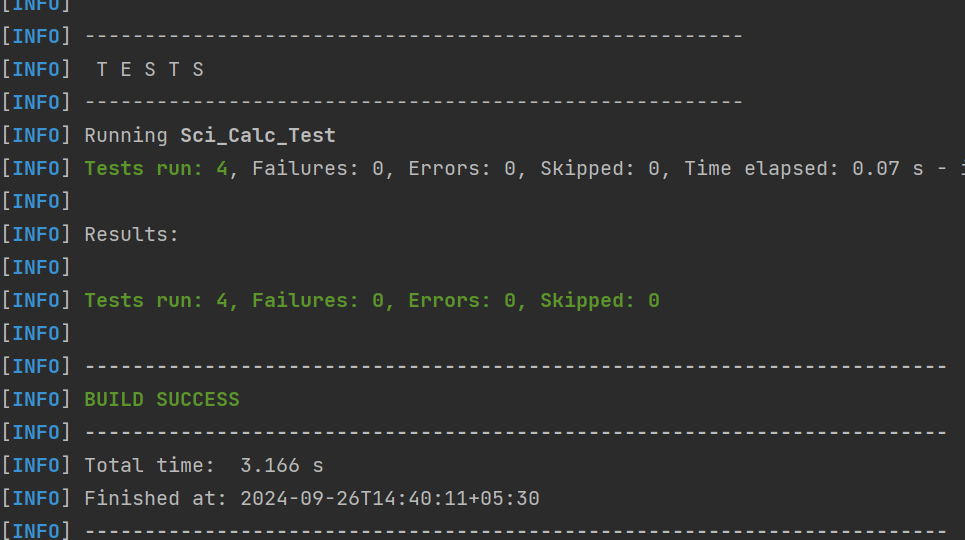
1. **mvn clean**:Deletes the target directory, cleaning up any previously compiled files or build artifacts. This ensures that the next build starts from a clean state without any leftover files.



1. **mvn package**:Compiles the source code, runs tests (if any), and packages the application into a distributable format (like JAR or WAR) in the target directory.



1. **mvn test:**Runs all the unit tests defined in the project using the specified testing framework (e.g., JUnit) without building or packaging the application.



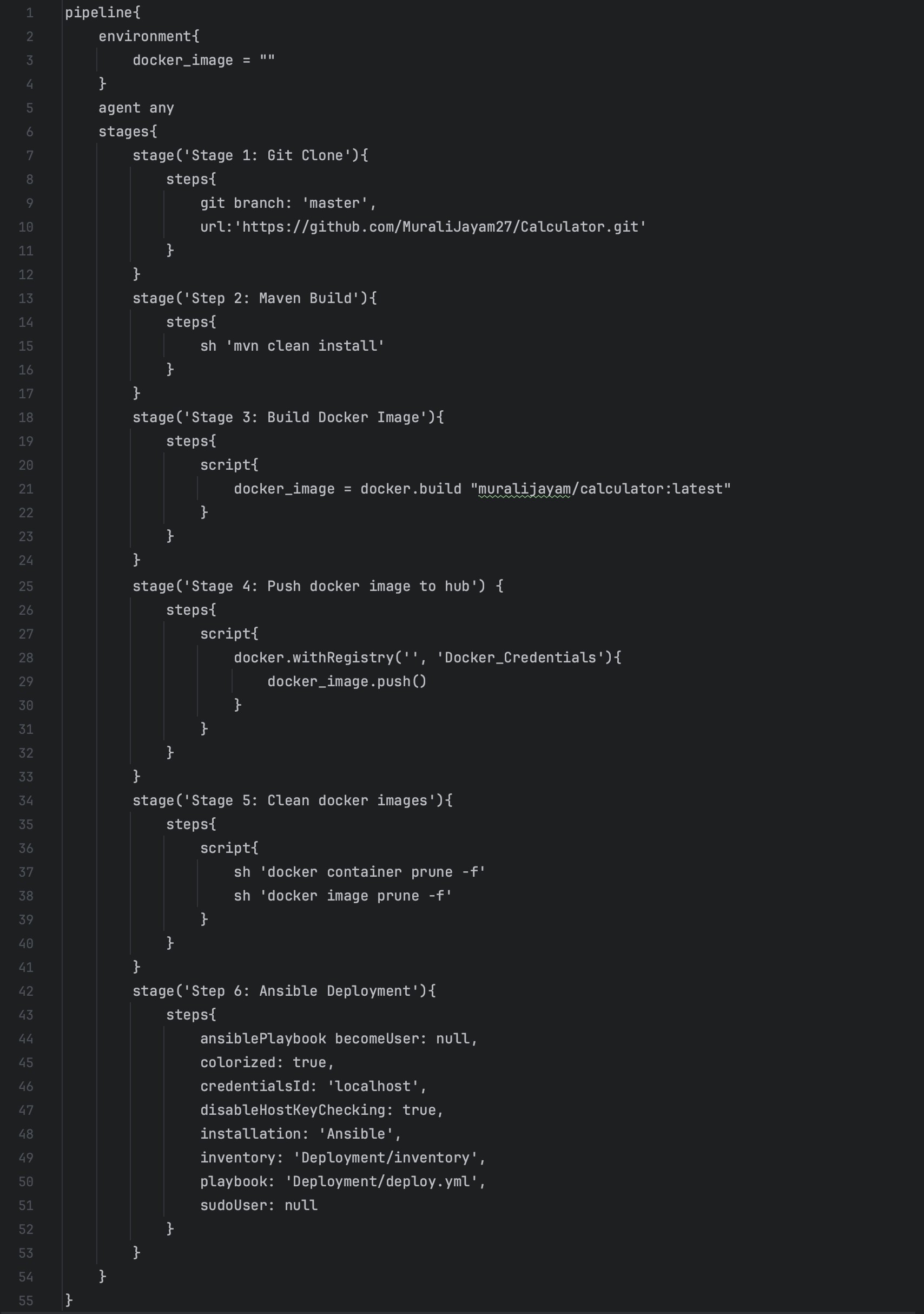
To run the JAR file use the command; **java -jar target\<filename>.jar.**

Generally the jar file created will be named <project-name>-1.0-SNAPSHOT.jar

## Jenkins

Go to https://localhost:8080 and install the necessary plugins.

We will establish a Jenkins pipeline comprising six distinct stages, as outlined below:



The Jenkins pipeline script consists of multiple stage blocks, each representing a distinct phase in the workflow:

**Stage 1:** Primarily involves a git clone operation to fetch the latest code from the designated Git repository.

**Stage 2:** Performs project cleanup by removing .class and .jar files, runs all tests, proceeds with code compilation, and generates updated JAR files.

**Stage 3:** Focuses on integrating Docker within the pipeline.