

By 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.

# 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: *Ngrok, GitHub webhooks(Optional Step, you can also poll periodically in which case this is not needed)*

# 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:** Employ Ngrok to create a secure tunnel to GitHub, enabling GitHub webhooks to trigger Jenkins builds upon code updates.

# Installations and setups:

**1. Install Java**

* **Download and Install**: Download the JDK from [Oracle](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html).
* **Set Environment Variables**:
  + Go to System Properties → Environment Variables.
  + Add JAVA\_HOME and set it to the path where JDK is installed.
  + Add JAVA\_HOME\bin to the Path variable.

**2. Install Maven**

* **Download and Install**: Download Maven from [here](https://maven.apache.org/download.cgi) and extract it.
* **Set Environment Variables**:
  + Add MAVEN\_HOME and set it to the extracted Maven folder path.
  + Add MAVEN\_HOME\bin to the Path.

**3. Install Docker**

* **Download and Install**: Download Docker Desktop for Windows from Docker Hub.
* **Enable WSL Integration**: In Docker settings, enable "Use WSL 2 based engine" and integrate it with your preferred Linux distribution.
* **Note that this Docker instance need to run as long as CI/CD is required to run.**

**4. Set Up WSL (Windows Subsystem for Linux)**

* **Enable WSL**:
  + In PowerShell as Admin, run:

**wsl --install**

* + Install Ubuntu or another Linux distribution from the Microsoft Store.
  + Restart your system.
  + After reboot, set up a Linux distribution from the Microsoft Store (e.g., Ubuntu).

This setup will allow you to run these tools seamlessly on Windows while using WSL for Linux-based tools like Ansible.

**5. Install Ansible**

* + In WSL (Ubuntu), run:

**sudo apt update**

**sudo apt install ansible**

**6.Download Jenkins**

* **Download Jenkins: Go to the Jenkins official website and download the Windows MSI installer. Double-click the MSI file and follow the installation wizard.**
* **Default Settings: Use default ports (8080 for HTTP) and select the default installation folder. Do the necessary setup and Configurations.**
* **After installation, open a browser and go to** [**http://localhost:8080**](http://localhost:8080)**. Note that this Jenkins server need to run as long as CI/CD is required to run.**
* **We also need to make sure that this Jenkins has got all the necessary permission to execute its tasks. To do this it is better to lauch it using an admin terminal. Press windows + X and chose admin terminal. Go to C:\Program Files\Jenkins and run the command java -jar Jenkins.war.**

**7. Git Installation and Setup**

* **Download: Get Git from** [**git-scm.com**](https://git-scm.com/download/win)**.**
* **Install: Run the installer with default settings.**
* **Setup:**

**git config --global user.name "Your Name"**

**git config --global user.email "you@example.com"**

**8. GitHub Setup**

* **Create Account: Sign up at** [**GitHub**](https://github.com)**.**
* **Create Repository:**
  + **Go to your GitHub profile → Repositories → New → Fill in details → Create.**
* **Push to GitHub:**

**git init**

**git remote add origin https://github.com/username/repo.git**

**git push -u origin main**

**9. Docker Hub Setup**

* **Create Account: Sign up at Docker Hub.**
* **Push Image:**

**docker login**

**docker tag local-image username/repository:tag**

**docker push username/repository:tag**

**10. IntelliJ IDEA Installation and Setup (Windows)**

* **Download: Get IntelliJ IDEA from** [**JetBrains**](https://www.jetbrains.com/idea/download/)**, choose the Community (free) or Ultimate version.**
* **Install: Run the installer and follow the prompts, using default settings.**
* **Set Up Java/Maven:**
  + **Open IntelliJ, create a New Project.**
  + **Select Java and Maven (if needed).**
  + **Choose the JDK path (you can link your existing JDK installation).**
* **Version Control:**
  + **Enable Git under File > Settings > Version Control > Git.**
  + **Link to your GitHub under Settings > Version Control > GitHub by signing in.**

**You can verify git,maven,docker,jdk,java,ansible using ‘<ServiceName> --version’ command in terminal.**

# 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.

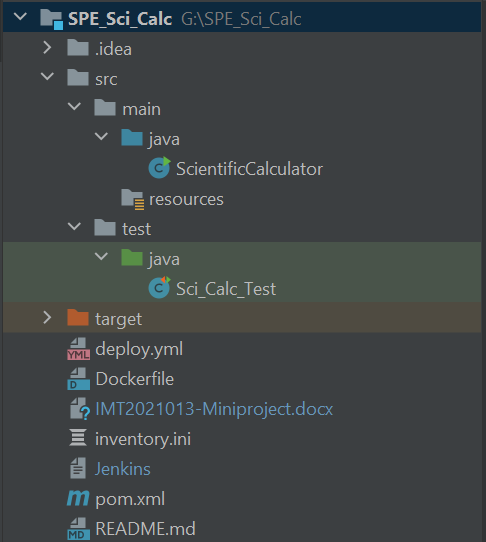
## Git Workflow

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.

**git init git add .**

**git commit -m “<message>” git status**

**git remote add origin <github\_repo\_url> git push -u origin main**



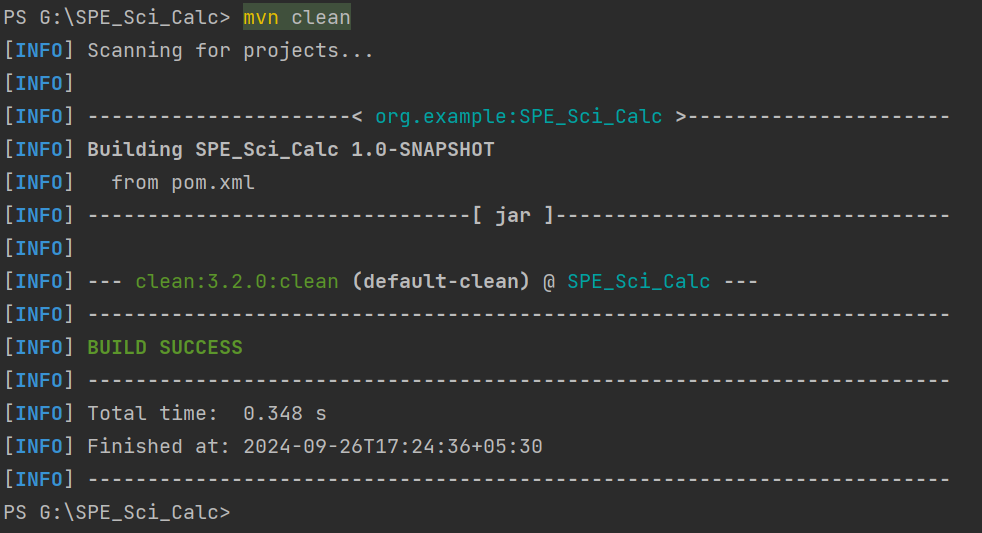
Folders: Files:

|  |  |  |  |
| --- | --- | --- | --- |
| .idea | Contains configuration files for IntelliJ IDEA | Deploy.yml | Ansible playbook file |
| 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 | Inventory.ini | Host list file |
| test | Holds Unit tests for the source code | Jenkins | Jenkins pipeline script for this project |
| target | Stores output JAR and class files | — | — |

**Key Maven Commands**

1. **mvn clean**  
   Removes all files generated by the previous build.

**mvn clean**

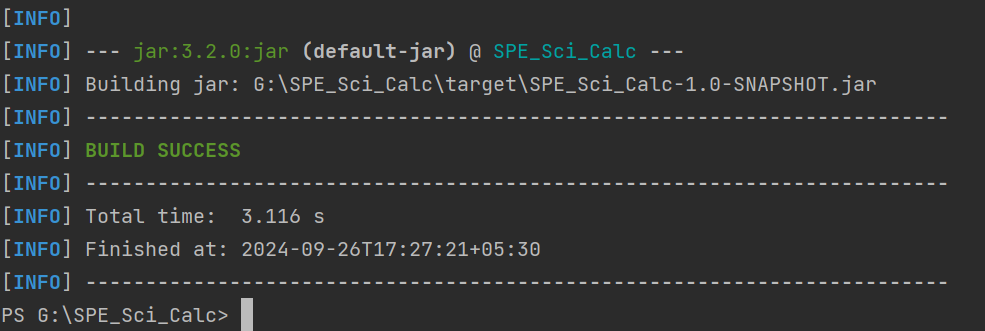
****

1. **mvn compile**  
   Compiles the project source code.

**mvn compile**

1. **mvn package**  
   Packages the compiled code into a JAR or WAR file (depending on your project setup).

**mvn package**

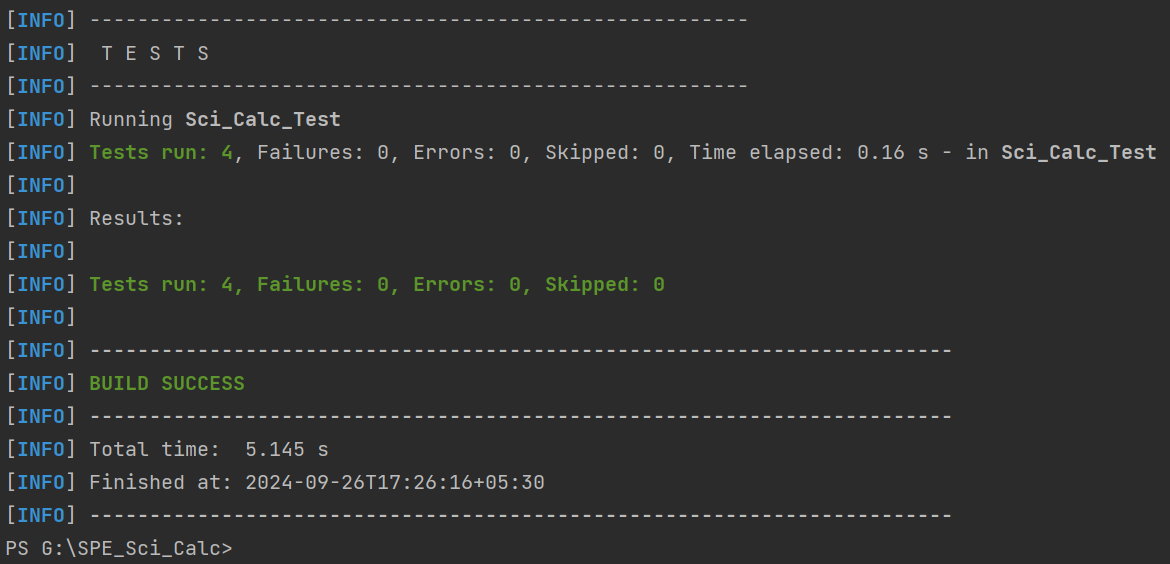
****

1. **mvn install**  
   Installs the package into your local repository, making it available for other projects.

**mvn install**

1. **mvn test**  
   Runs the tests written in the project.

**mvn test**

****

1. **mvn deploy**  
   Deploys the final package to a remote repository.

**mvn deploy**

Important one to be used here are mvn clean, mvn test, mvn package.

Within the Maven configuration file (pom.xml), the <mainClass> tag specifiesthe 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. This is important for maven to build the project properly.

You can run the jar file created by maven which is present in target folder. Use

**java -jar <Name>.jar**