Mini Project

Simple Calculator

**~ Murali Jayam IMT2020035**

# 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*

# 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.

**Link to GitHub Repository: LINK**

# Installing and Setting up tools

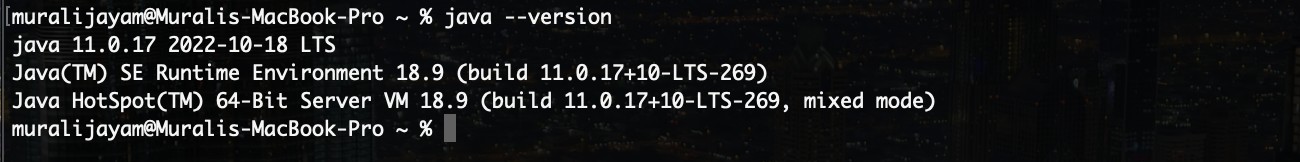
## Java

To install Java, We use the following command;

brew install openjdk

To verify if Java is installed and know the version we use the following command;

java --version



## IntelliJ IDEA Ultimate

Go to the JetBrains website and download the IntelliJ IDEA Community. The website link is: JetBrains IntelliJ IDEA. Once the ‘*.dmg’* file is downloaded,

open the file and drag it to the applications folder and configure initial settings to get started.

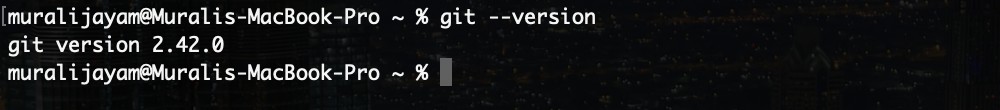
## Git

To install Git, we use the following command;

brew install git

To verify if Git is installed and know the version we use the following command;

git --version



## GitHub

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

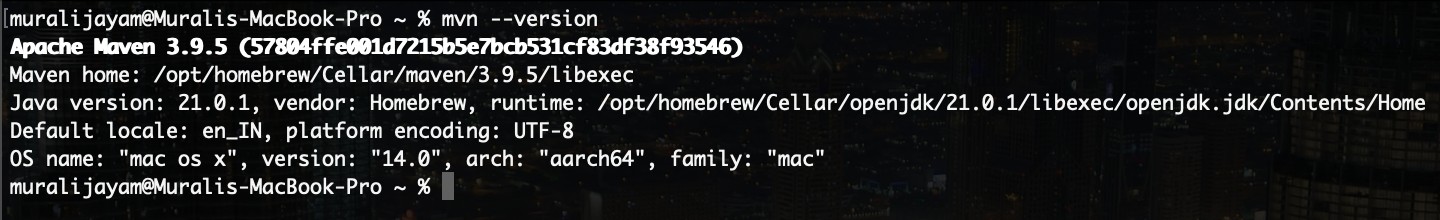
## Maven

To install Maven, we use the following command;

brew install maven

To verify if Maven is installed and know the version we use the following command;

mvn —-version



## Jenkins

To install Jenkins, we use the following command;

brew install jenkins

After installing, you can start the jenkins by executing the following command;

brew services start jenkins

Once installation is done, Access the Jenkins web interface by opening a web browser and navigating to http://localhost:8080/

**|** Note: Jenkins requires Java to be installed on the system.

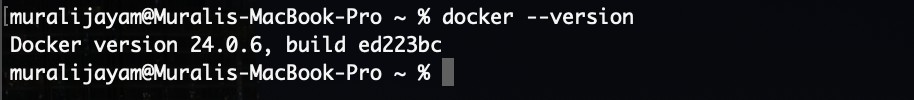
## Docker

Go to the Docker website to download Docker Desktop for Mac. The download link is Docker Desktop for Mac. Once the ‘*.dmg*’ file is downloaded, open the

file and drag it to the applications folder and configure initial settings to get started.

To verify if Docker is installed and know the version we use the following command;

docker --version



## Ansible

To install Ansible, we use the following command;

brew install ansible

To verify if Ansible is installed and know the version we use the following command;

ansible --version



## Ngrok

Visit the Ngrok official website https://ngrok.com/download and sign up for a free account. After signing up, you'll be able to download the Ngrok binary for Mac.

Now, unzip ngrok from a terminal with the following command;

unzip /path/to/ngrok.zip

Move it to a directory that's included in your system's PATH variable to execute it from any location in the Terminal.

sudo mv /path/to/ngrok /usr/local/bin/

Now connect your account with the Authtoken using the following command;

ngrok config add-authtoken <your\_authtoken>

# 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.

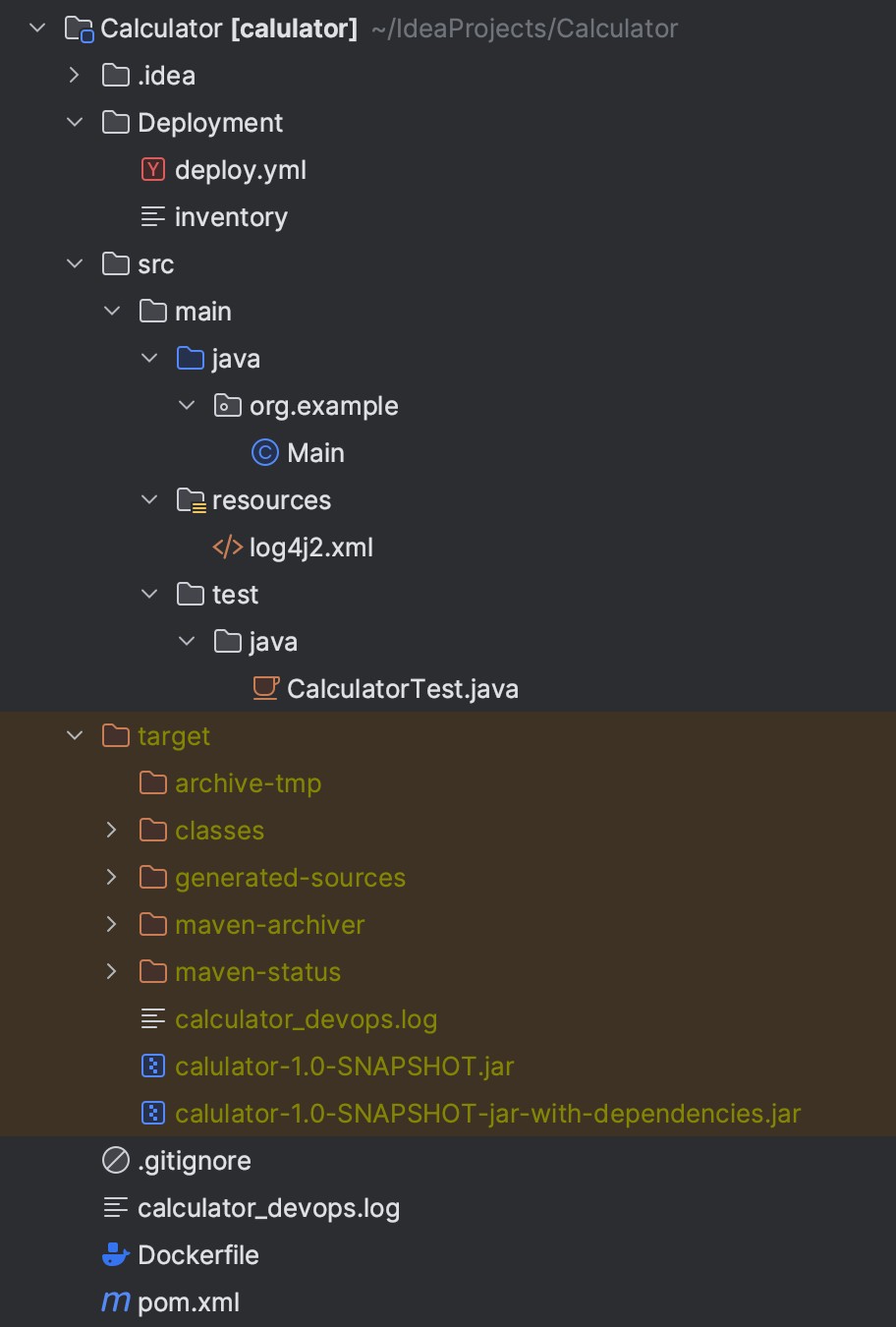
git init git add .

git commit -m “<message>” git status

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

Enter username and PAT

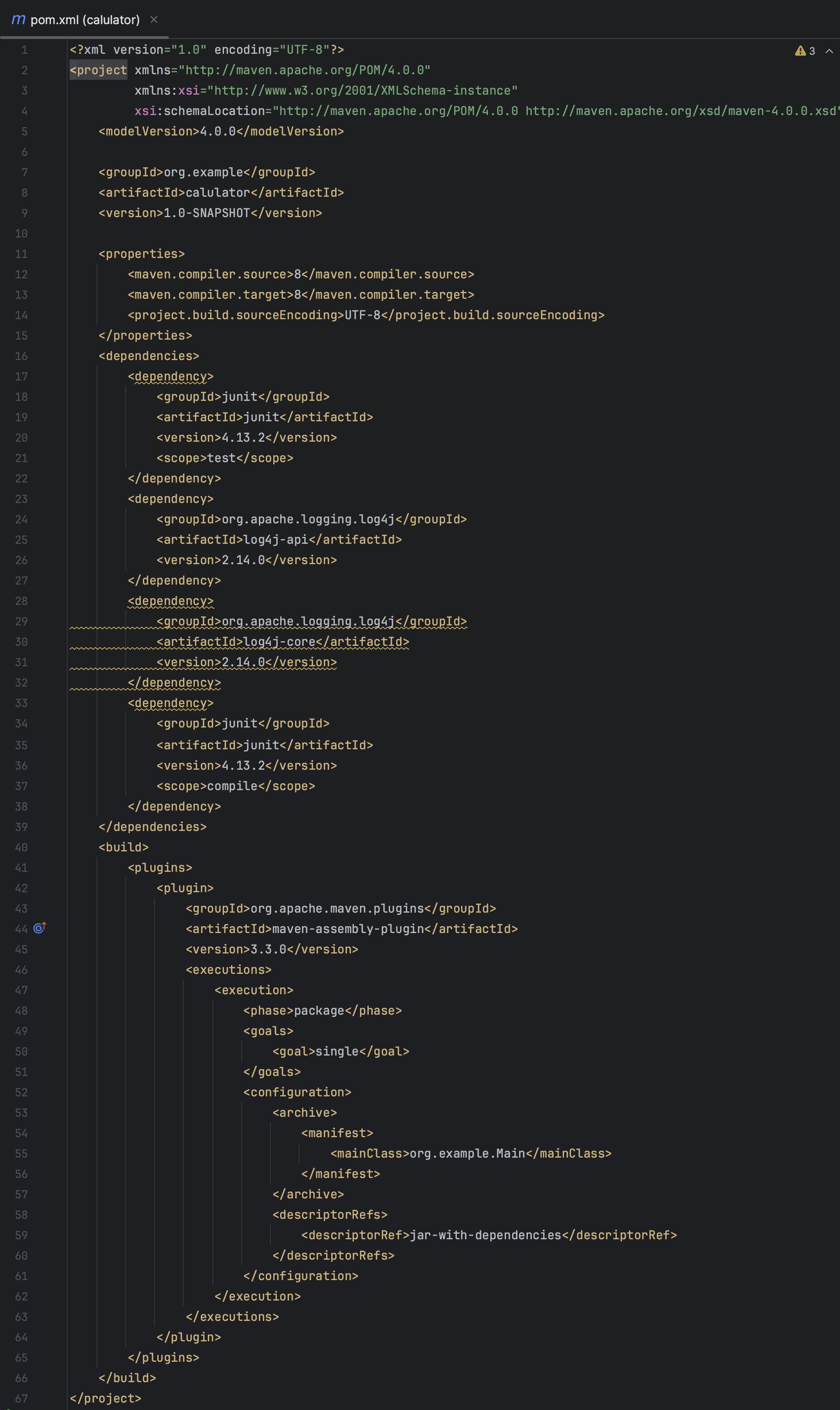
## Project Structure



Folders: Files:

|  |  |  |  |
| --- | --- | --- | --- |
| .idea | Contains configuration files for IntelliJ IDEA | .gitignore | Contains file patterns to be ignored by Git |
| Deployment | Files for ansible | calculator\_devops.log | Log file for the project |
| 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

- pox.xml file

mvn clean

Removes the 'target' folder,

ensuring a fresh start for the

subsequent

compilation. This

step eliminates any previous build

artifacts.

mvn compile

Compiles the project and its associated test

cases. This phase ensures the code is

error-free and ready for the subsequent steps.

mvn install

Generates the JAR file. This final step packages the

project, creating

the desired output artifact (JAR file) once the

compilation is successful.

Now, navigate to the target folder using the command;

cd target

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

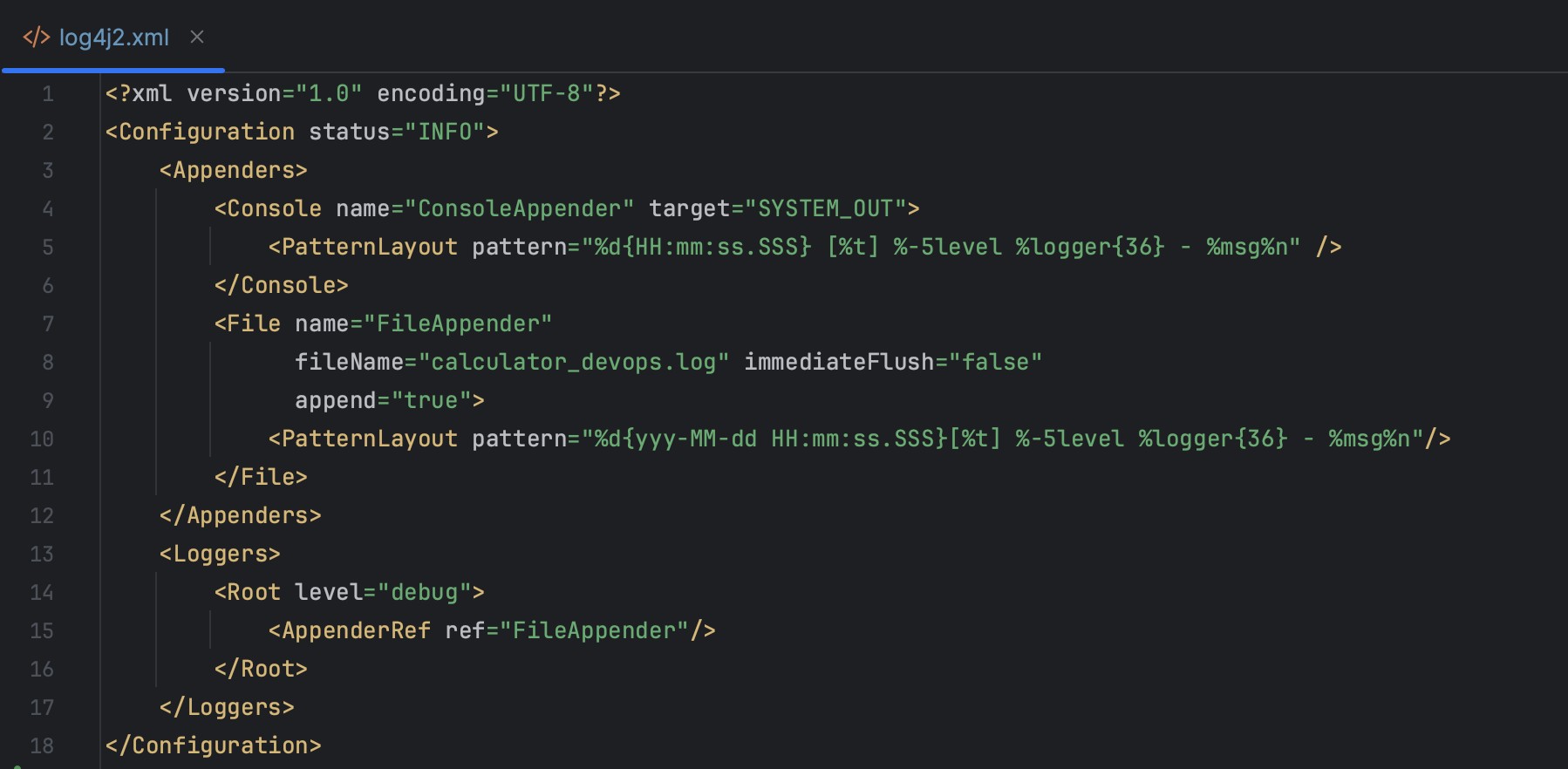
Within the Maven configuration file (pom.xml), the <mainClass> tag specifies

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.

- Log4j2.xml file



The specified logging format includes the following components:

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

**%d{yyy-MM-dd HH:mm:ss.SSS}**: Represents the timestamp with millisecond precision.

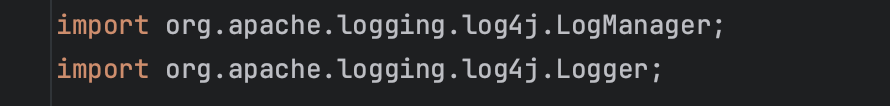
**%t**: Indicates the running thread's name.

**%-5level**: Denotes the log level with left alignment and a maximum width of 5 characters.

**%logger{36}**: Refers to the logger's name within a maximum of 36 characters.

**%msg**: Represents the user-written message contained in the source code.

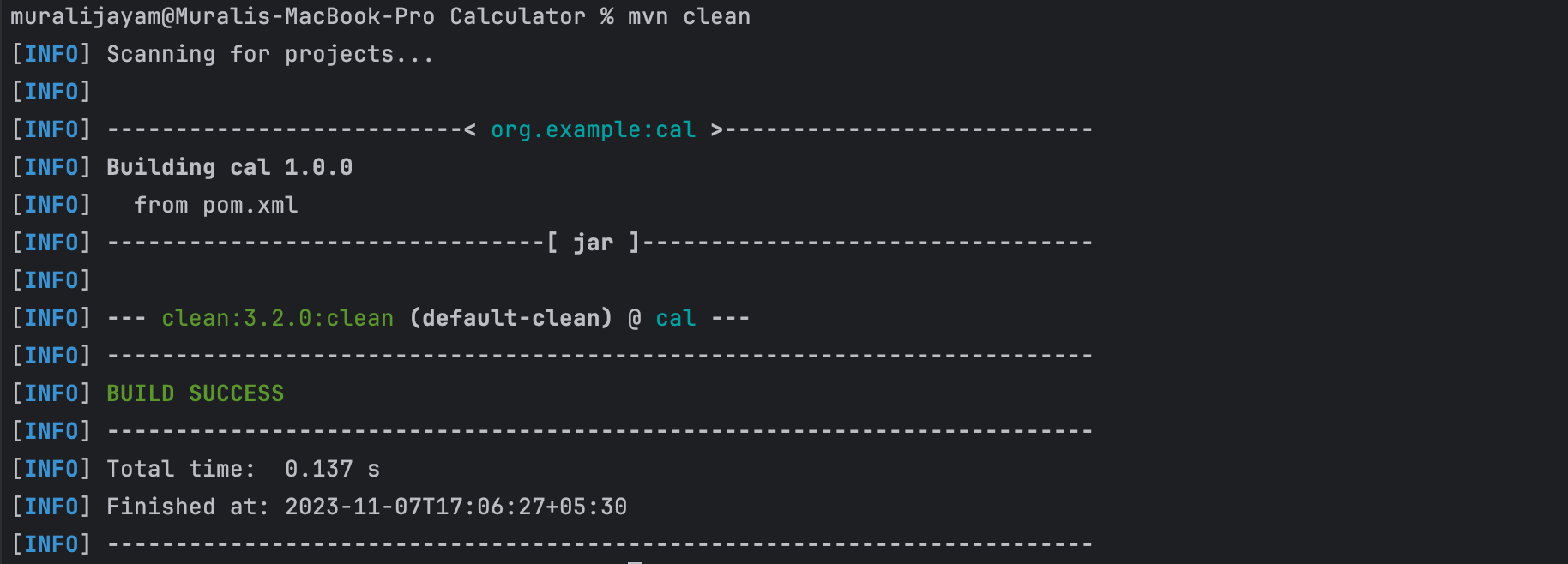
The necessary imports for Log4j



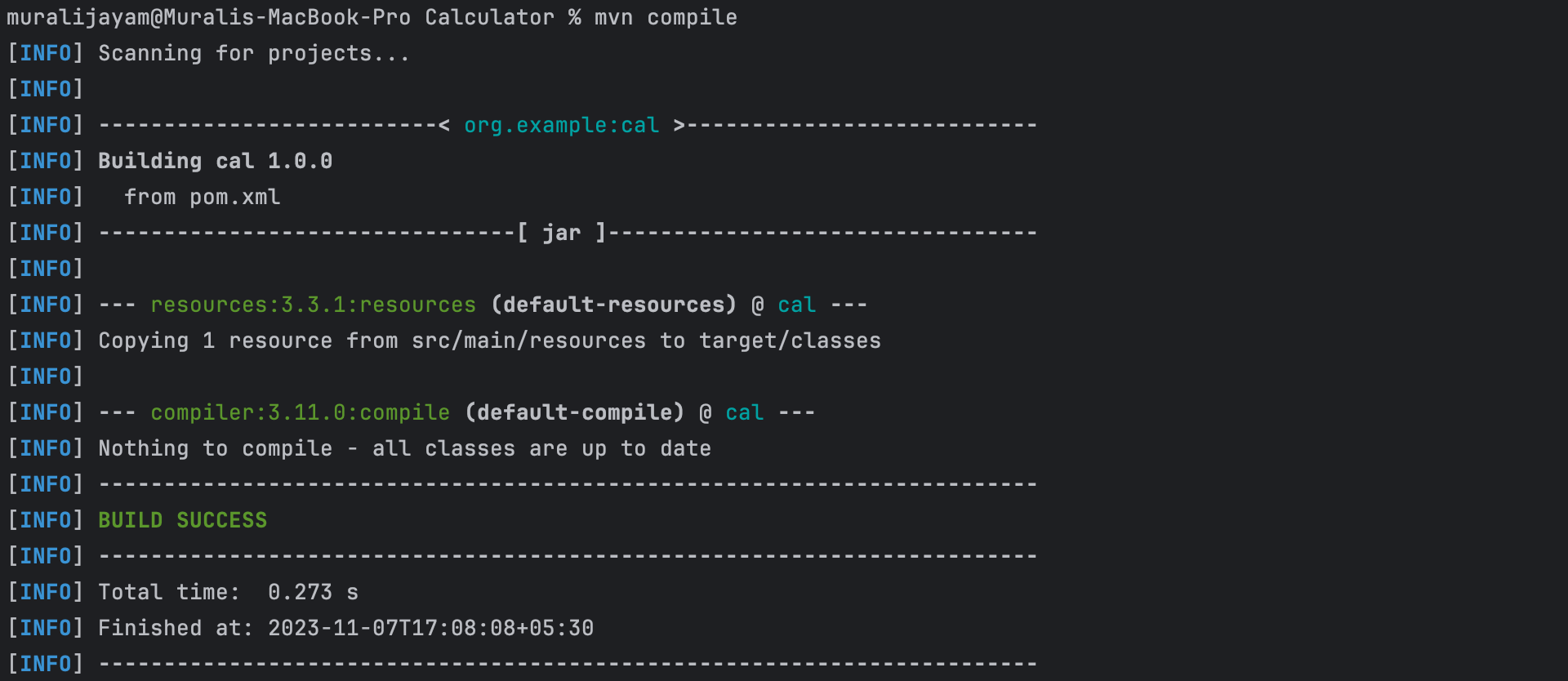
Creating an instance of the Logger in Log4j involves using a statement to initialise a Logger object within the code.



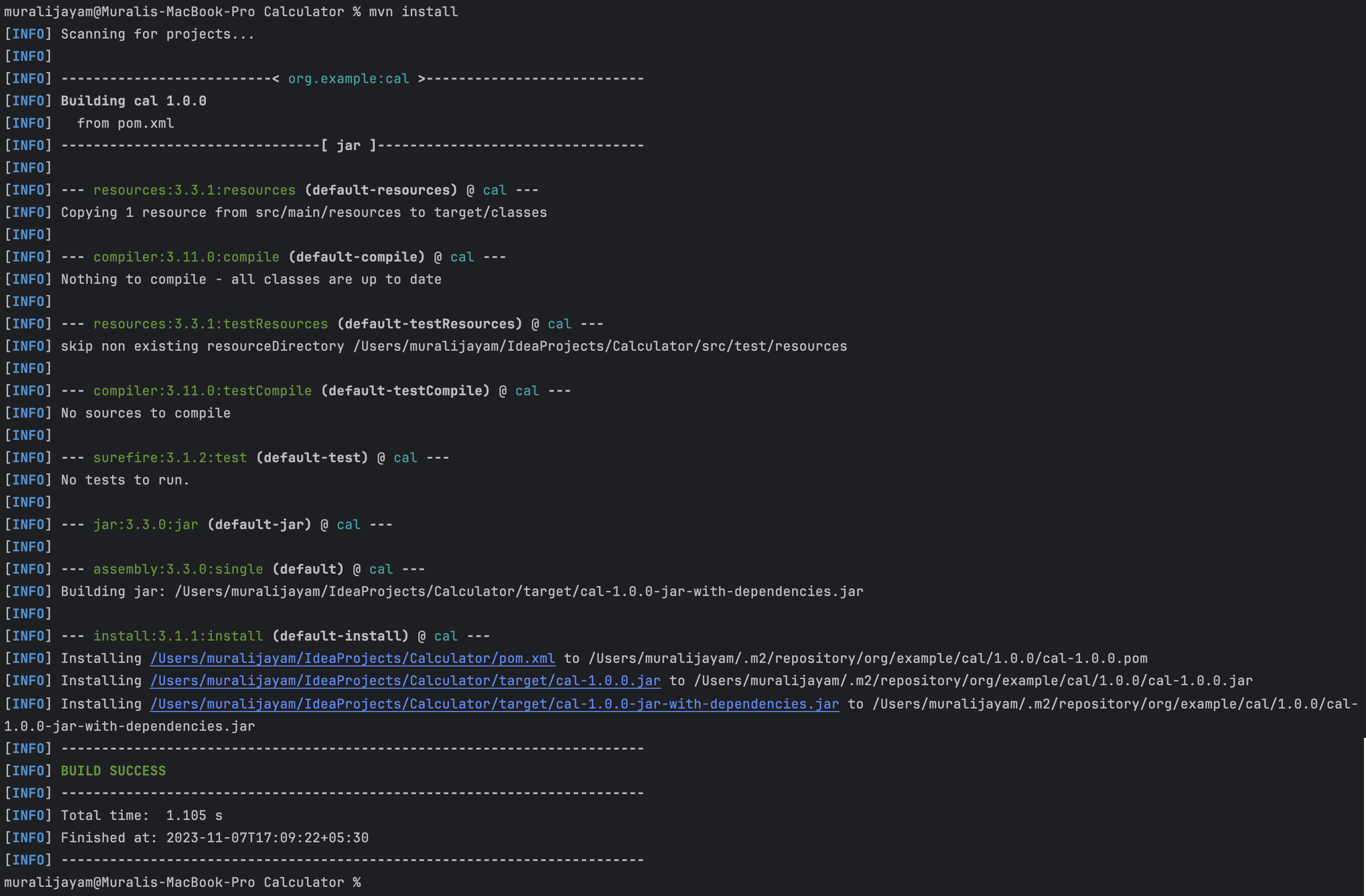
Clean the project using mvn clean



Compile the code using mvn compile



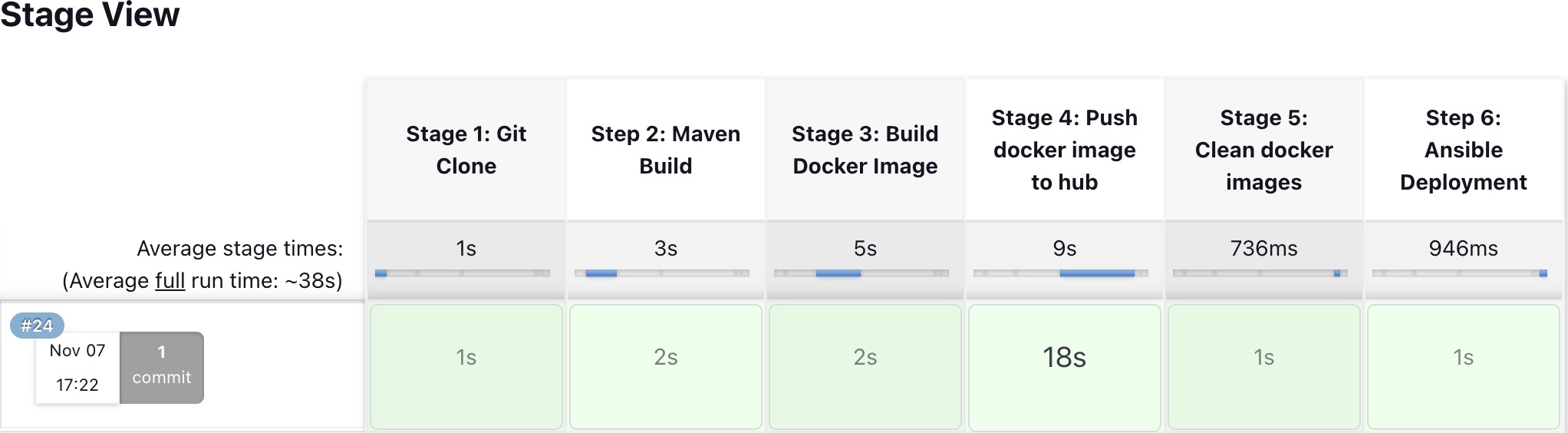
Run tests and create jar files using mvn install

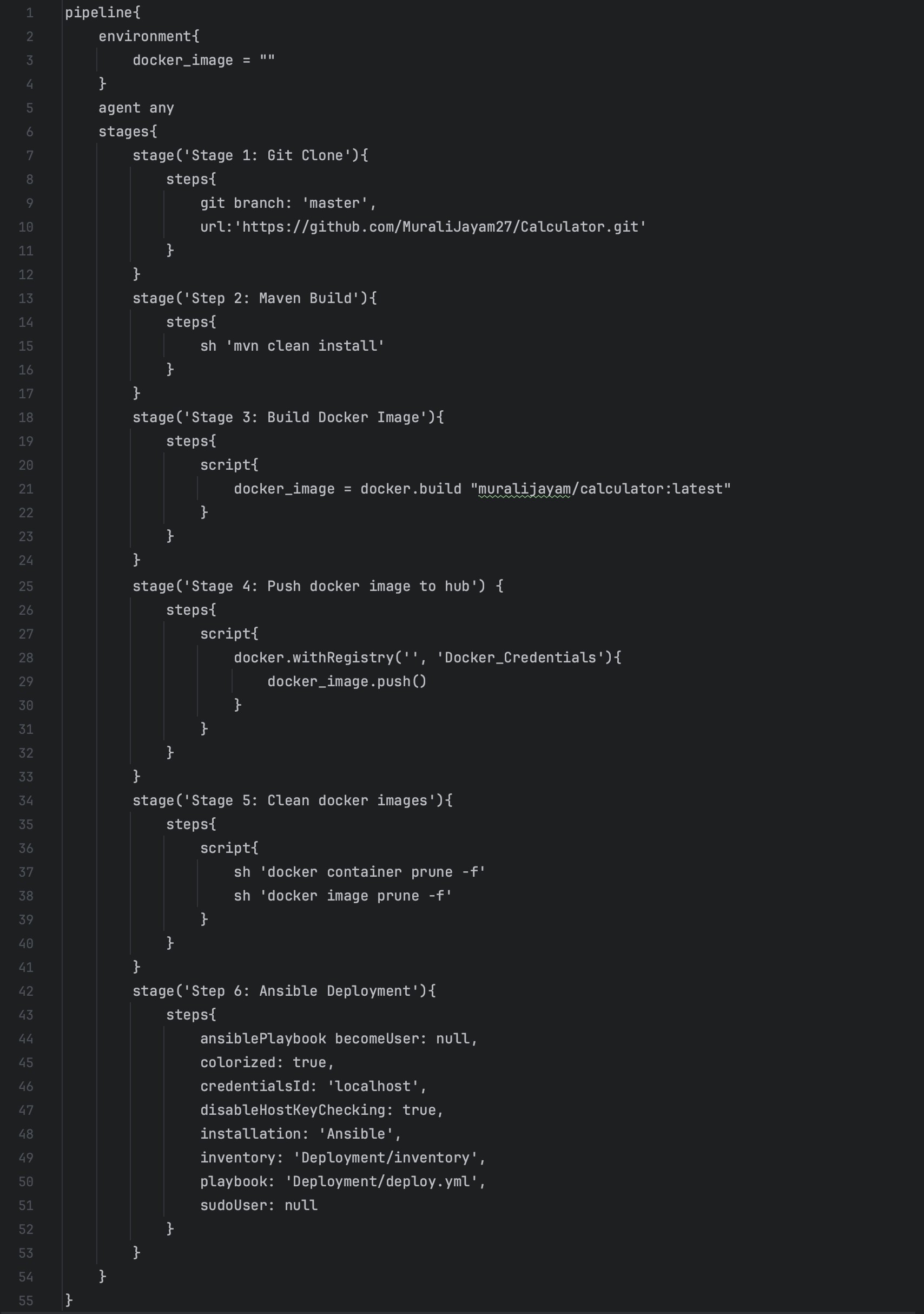


## Jenkins

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.

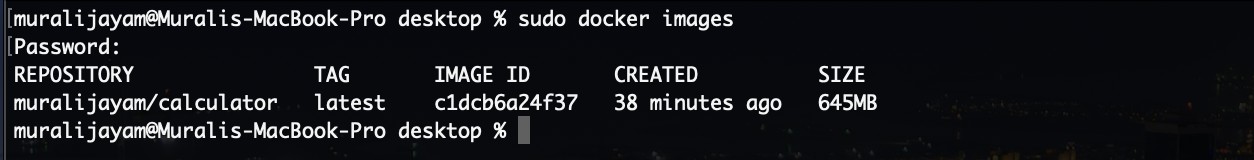
## Docker

Now, we will create a docker container for our project. For this, we need a Dockerfile.



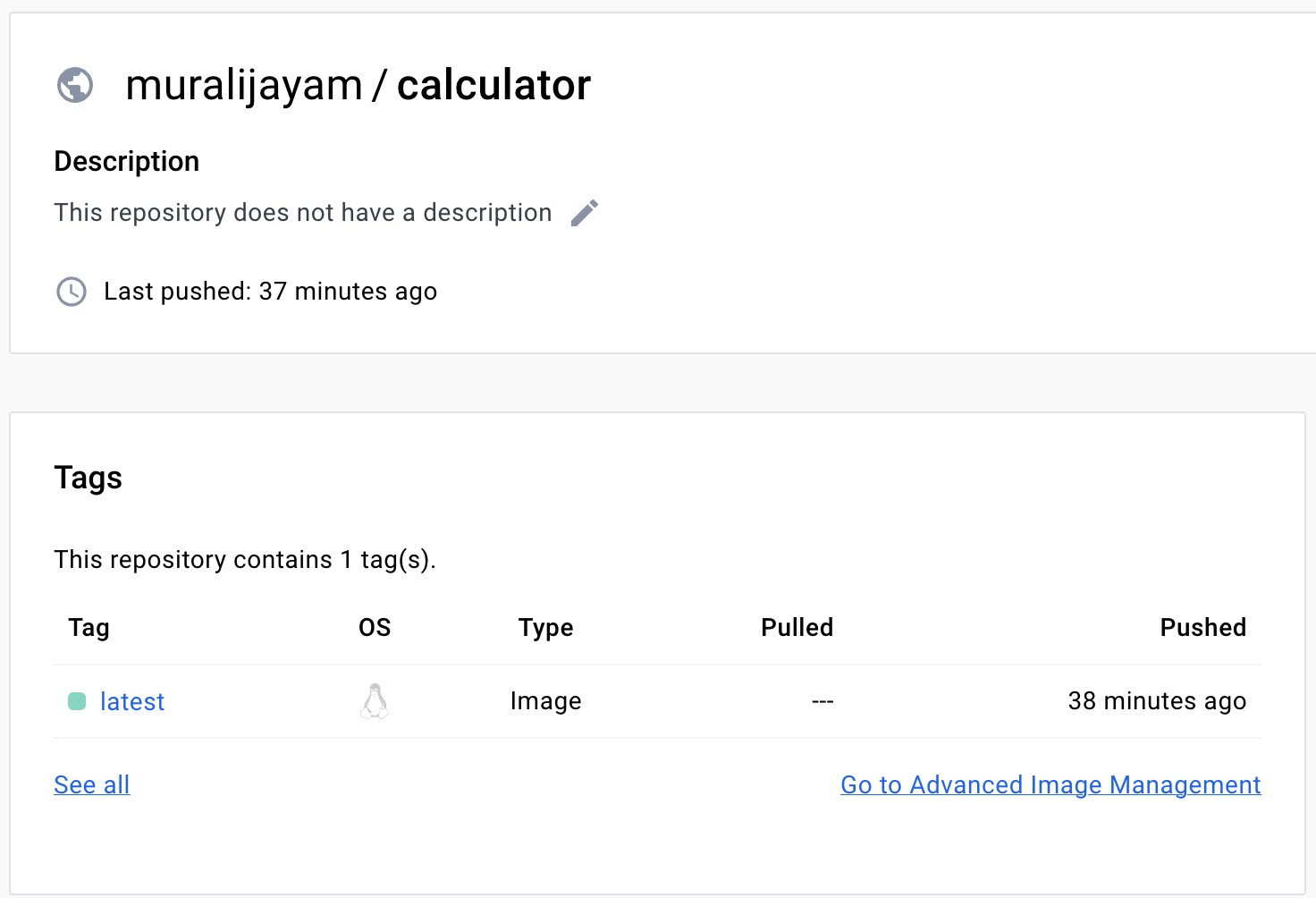
The purpose of the above code is to generate a Docker container with OpenJDK version 11 as the base image which acts like a JVM. The final container can be run independently without any dependencies.

In the pipeline script, docker\_image = docker.build “muralijayam/calculator” builds the docker image.



The provided command allows verification of the successful creation of a Docker image.

**Step 4:** Focuses on pushing the created Docker image to Docker Hub, making the image accessible for other users. To achieve this, a credential is set up in Jenkins, containing the Jenkins username and password for authentication.



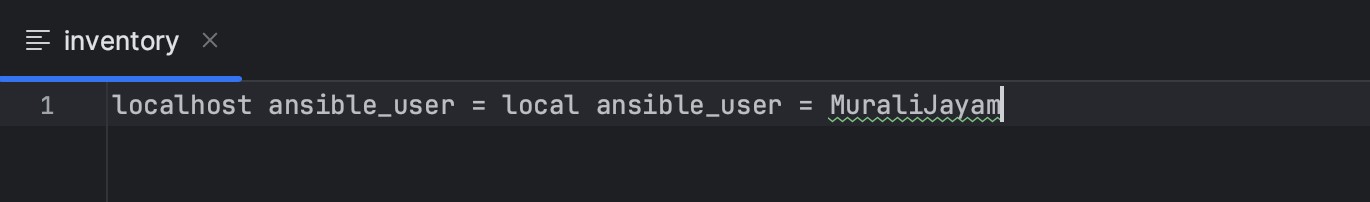
We can a overview of the repository at https://hub.docker.com/

**Step 5:** To free up space and we’ll delete the unused docker containers and images.

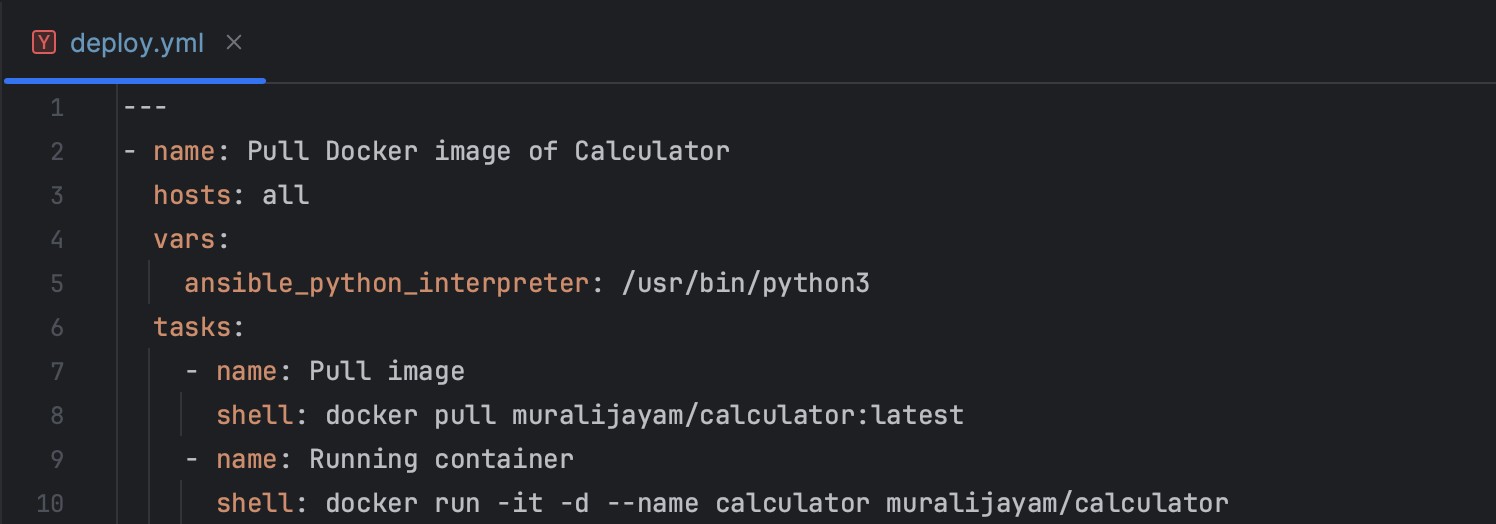
docker container prune -f docker image prune -f

**Step 6:** 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.



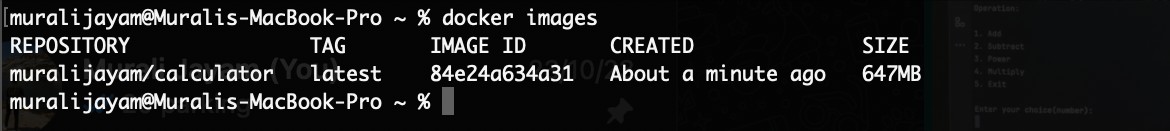
*Inventory file*



*Deploy.yml file*

Run the following command to view the list of images;

docker images

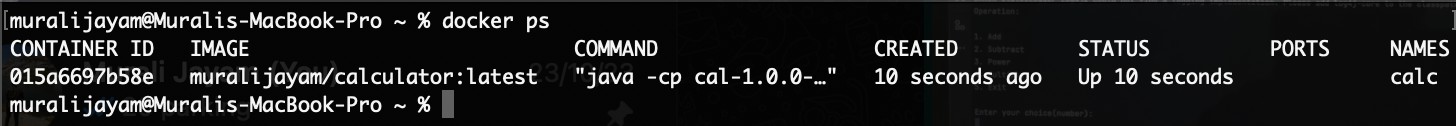


Now, run the following command to create a container from a image;

docker run -it -d --name calc muralijayam/calculator:latest

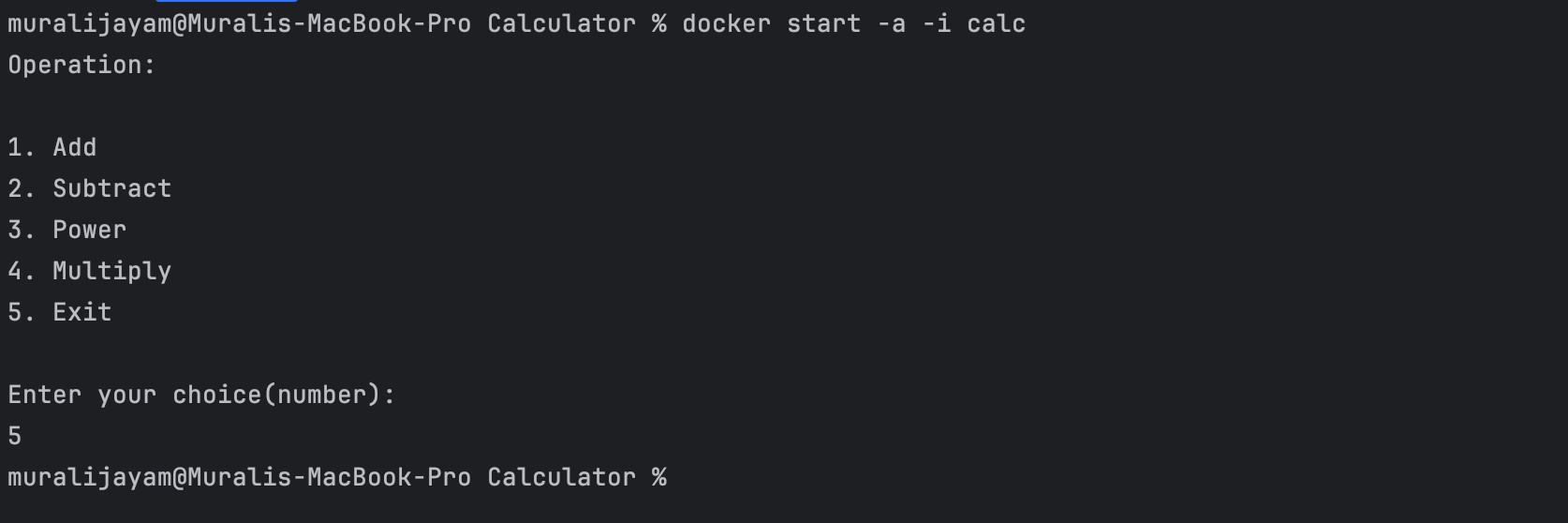
To view the containers run the following command;

docker ps



Now, to execute the container;

docker start -a -i <name of container>



## Automating Pipeline Execution with Git SCM Polling and Ngrok

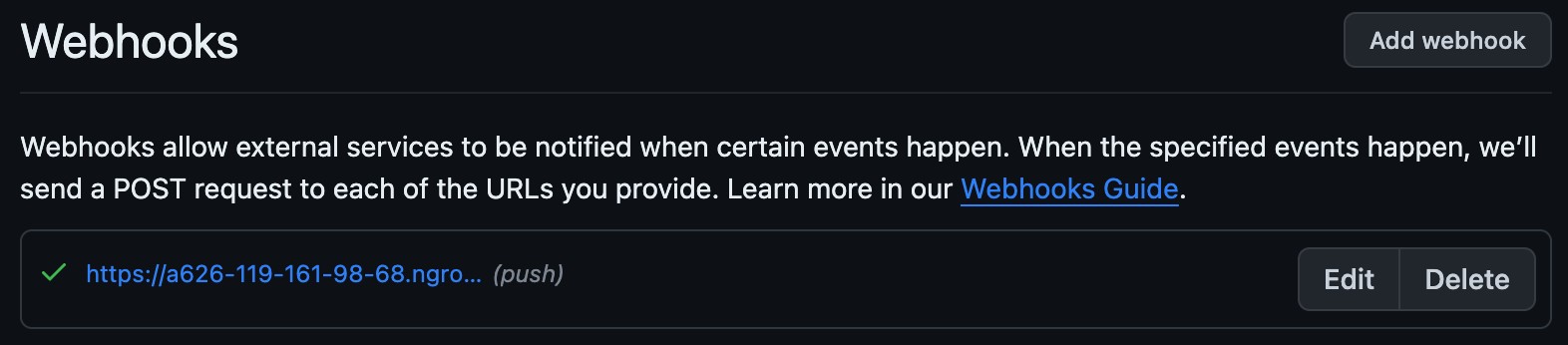
Open a terminal window and enter the command ngrok http 8080. This

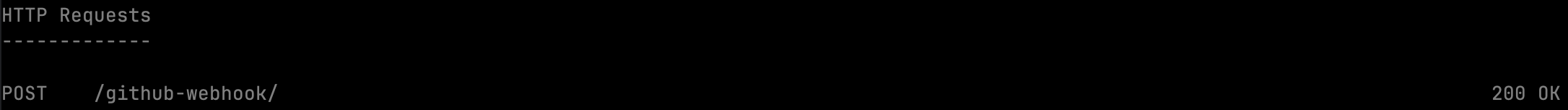
command will establish an HTTP tunnel using Ngrok, exposing the local server running on port 8080 to the internet.



Copy the forwarding URL provided by Ngrok. Subsequently, create a GitHub webhook and utilise this URL as the payload URL for the webhook

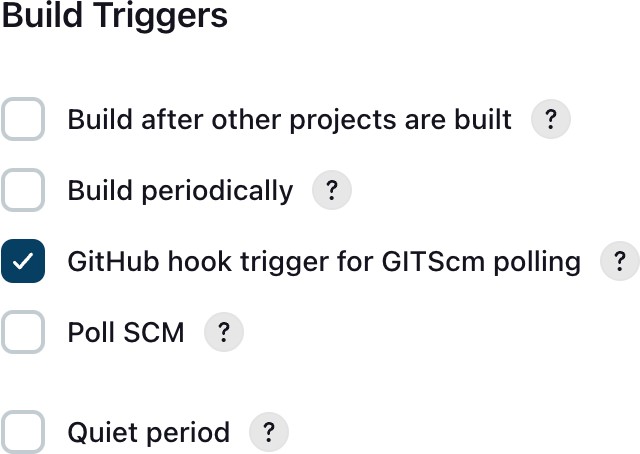
configuration. GitHub initiates a test connection, and upon successful configuration, a '200 OK' message confirms the proper setup.

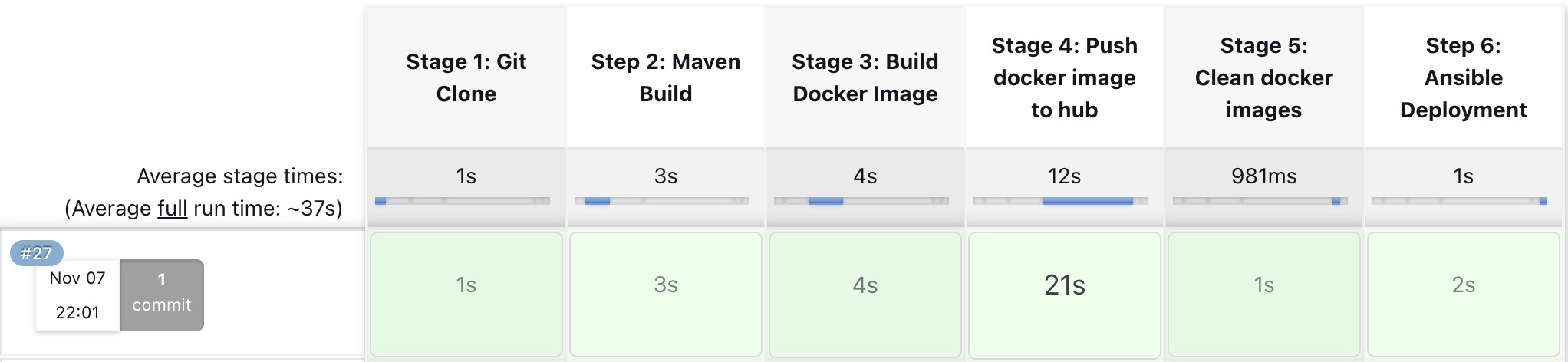
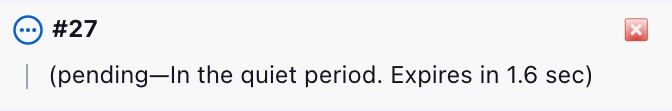




Now, we'll update the Jenkins URL to the forwarding URL obtained and configure a build trigger for Git SCM polling. This setup ensures that our

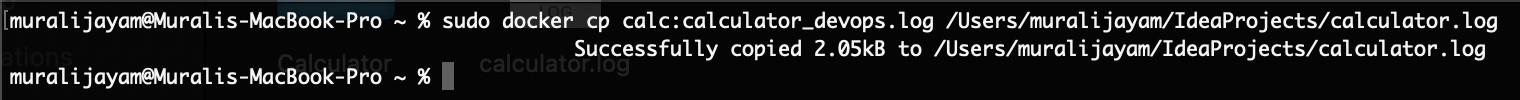
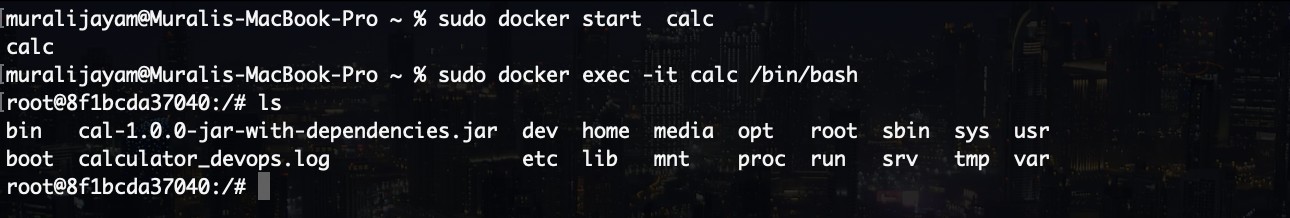
pipeline automatically initiates the build process whenever Jenkins detects a new commit made to the associated GitHub repository.

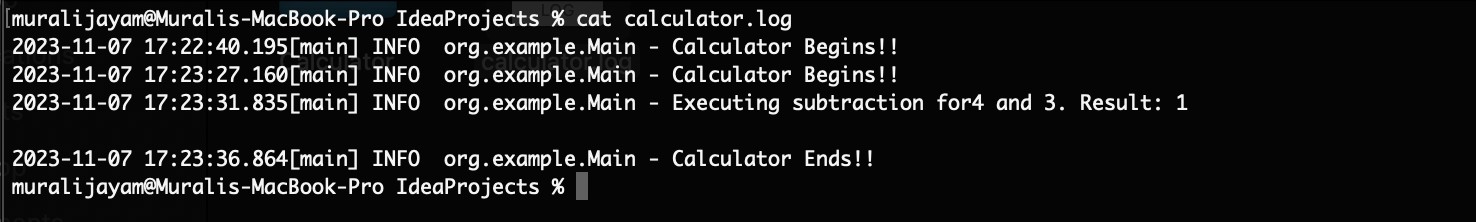




## Accessing the log file

Accessing the log file involves starting the container and treating it as a file directory, enabling exploration and interaction with its contents.





Now, we can use the log file to analyse the data using ELK stack.