CSE 816 - Software Production Engineering September 26, 2024

Miniproject: Calculator

Instructor: Prof. B. Thangaraju Sunny Kaushik - IMT2021007

1 Problem Statement

The objective of this project is to develop a scientific calculator program that includes the following functions:

- Square root function: $\sqrt{x}$
- Factorial function: $x!$
- Natural logarithm (base e): $\ln(x)$
- Power function: x^b

The development process follows DevOps tools and workflows with a focus on development, deployment, and monitoring of the program.

2 Objective

The specific objectives of this project are:

1. To create a local project in the system for developing a scientific calculator.
2. To initialize a local Git repository for version control and maintain the history of changes.
3. To push the local repository to a remote version control platform (e.g., GitHub).
4. To create a Jenkins pipeline that will automatically build the project and execute test cases whenever changes are pushed to the repository.
5. To containerize the project using Docker and push the container image to a remote container repository (e.g., DockerHub).
6. To deploy the Docker container to a target machine using Jenkins for continuous deployment.

3 Tools Used

- **Code Editor: Visual Studio Code**
  A lightweight and powerful source code editor with support for development operations like debugging, task running, and version control.
- **Version Control System: Git**
  A distributed version control system that allows developers to track changes in their codebase, collaborate with others, and revert to previous versions if needed.
- **Remote Repository: GitHub**
  A cloud-based platform for hosting Git repositories, enabling collaboration and sharing of code with remote teams.
- **Build Tool: Apache Maven**
  A project management and build automation tool that manages project dependencies, compiles code, and packages it into deployable formats.

- **CI/CD Pipeline: Jenkins**
  An open-source automation server that enables continuous integration and continuous delivery (CI/CD) by automating the building, testing, and deployment of code.
- **Containerisation: Docker**
  A platform that automates the deployment of applications inside lightweight, portable containers that can run consistently across different computing environments.
- **Configuration Management: Ansible**
  An open-source tool used for IT automation, configuration management, and application deployment. It enables the automated configuration of servers and applications.
- **Git SCM Polling and Build Automation: Ngrok, GitHub Webhooks**
  Ngrok allows secure tunneling for local development environments, and GitHub Webhooks trigger Jenkins builds automatically when changes are pushed to the Git repository, ensuring continuous integration.

4 Workflow

1. **Code Development:** Visual Studio Code was used to develop the scientific calculator program. The code editor provides a lightweight environment with rich features.
2. **Version Control:** Git was employed to manage the version control of the project. The code was stored in a local Git repository, and changes were periodically pushed to a remote repository hosted on GitHub.
3. **Building:** Apache Maven was used as the build automation tool. It simplified the management of project dependencies, compiled the source code, and packaged the calculator program into a deployable format (e.g., a JAR file).
4. **Continuous Integration and Continuous Delivery (CI/CD):** Jenkins, an automation server, was set up to handle continuous integration and continuous delivery. It was configured to automatically pull the latest changes from the GitHub repository, run unit tests, and deploy the program to production environments.
5. **Containerisation:** Docker was used to containerize the calculator program. By packaging the program into a Docker image, the application can run consistently across different environments, eliminating the "works on my machine" problem.
6. **Configuration Management:** Ansible was employed to automate the setup and configuration of the infrastructure needed to run the application. It ensured that the environment, including all dependencies and settings, was consistent across different machines.
7. **Git hook trigger for GITScm Polling and Build Automation:** Ngrok was used to create a secure tunnel between Jenkins and GitHub, enabling GitHub webhooks to communicate with the local Jenkins server. This setup allowed Jenkins to automatically trigger builds whenever new code was pushed to the GitHub repository. The integration of GitHub webhooks with Jenkins ensures that the pipeline is automatically executed upon code updates, providing continuous integration and faster feedback on code changes.

Link to the Github Repository

SPE Mini-Project Calculator

5 Setting up the Environment and required Tools

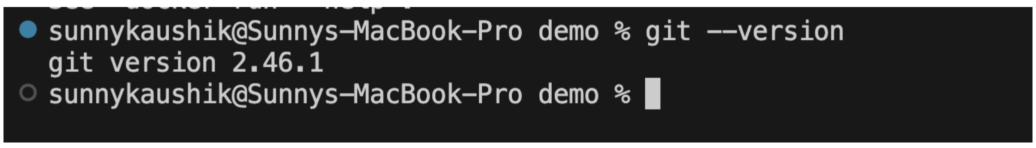
5.1 Git

To install Git, use the following command:

```
brew install git
```

To verify if Git is installed and to check the version, use the following command:

```
git --version
```



```
sunnykaushik@Sunnys-MacBook-Pro demo % git --version
git version 2.46.1
sunnykaushik@Sunnys-MacBook-Pro demo %
```

Figure 1: Git version

5.2 GitHub

Visit www.github.com to create an account.

5.3 Visual Studio Code

To install Visual Studio Code on a Mac, follow these steps:

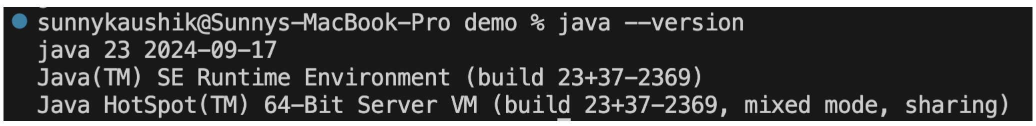
1. Open your web browser and go to the official Visual Studio Code website: https://code.visualstudio.com/.
2. Click on the **Download for macOS** button on the homepage.
3. Once the download is complete, locate the downloaded file in your `Downloads` folder. The file will be named something like `VSCode-darwin-stable.zip`.

5.4 Java JDK

To install the Java JDK on a Mac, follow these steps:

1. Open your web browser and go to the official Oracle JDK download page: https://www.oracle.com/java/technologies/javase-jdk11-downloads.html (or the appropriate version page).
2. Scroll down to the section labeled **Download** and locate the macOS Installer.
3. Click on the **Download** button next to the macOS installer package.
4. To verify that the Java JDK is installed, open the Terminal application and enter the following command:

   ```
   java -version
   ```



```
sunnykaushik@Sunnys-MacBook-Pro demo % java --version
java 23 2024-09-17
Java(TM) SE Runtime Environment (build 23+37-2369)
Java HotSpot(TM) 64-Bit Server VM (build 23+37-2369, mixed mode, sharing)
```

Figure 2: Java version

You should see the installed version of Java displayed in the terminal.

5.5 Jenkins

To install Jenkins on a Mac using Homebrew, follow these steps:

1. Open the Terminal application.
2. Use the following command to install Jenkins:

   ```
   brew install jenkins-lts
   ```

3. Once the installation is complete, start Jenkins-lts (Jenkins- Long Term Support) using:

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

4. Open your web browser and go to http://localhost:8080 to access the Jenkins dashboard.

5.6 Apache Maven

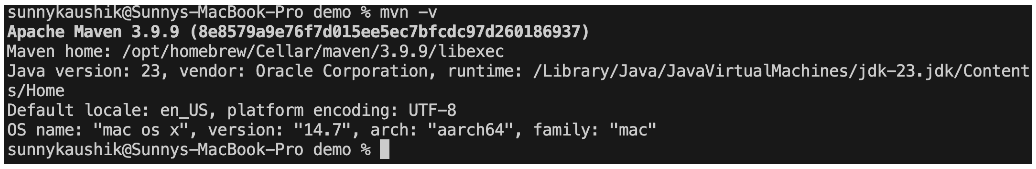
To install Apache Maven on a Mac using Homebrew, follow these steps:

1. Open the Terminal application.
2. Use the following command to install Maven:

   ```
   brew install maven
   ```

3. To verify the installation, check the Maven version by running:

   ```
   mvn -v
   ```



```
sunnykaushik@Sunnys-MacBook-Pro demo % mvn -v
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: /opt/homebrew/Cellar/maven/3.9.9/libexec
Java version: 23, vendor: Oracle Corporation, runtime: /Library/Java/JavaVirtualMachines/jdk-23.jdk/Content
s/Home
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "14.7", arch: "aarch64", family: "mac"
sunnykaushik@Sunnys-MacBook-Pro demo %
```

Figure 3: Maven version

5.7 Docker

To install Docker Desktop on a Mac using Homebrew, follow these steps:

1. Open the Terminal application.
2. Use the following command to install Docker:

   ```
   brew install --cask docker
   ```

3. Once the installation is complete, open Docker from the Applications folder.
4. After launching Docker, wait for it to start up, and you will see the Docker icon in the menu bar.
5. Sign In or create an account to get started with the docker desktop.

5.8 Ansible

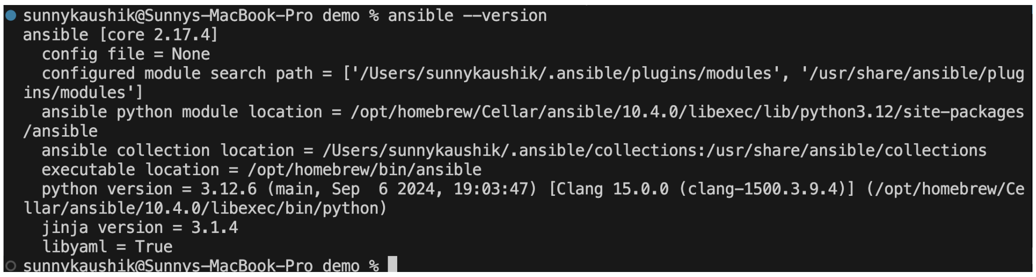
To install Ansible on a Mac using Homebrew, follow these steps:

1. Open the Terminal application.
2. Use the following command to install Ansible:

```
brew install ansible
```

3. To verify the installation, check the Ansible version by running:

```
ansible --version
```



```
sunnykaushik@Sunnys-MacBook-Pro demo % ansible --version
ansible [core 2.17.4]
  config file = None
  configured module search path = ['/Users/sunnykaushik/.ansible/plugins/modules', '/usr/share/ansible/plug
ins/modules']
  ansible python module location = /opt/homebrew/Cellar/ansible/10.4.0/libexec/lib/python3.12/site-packages
/ansible
  ansible collection location = /Users/sunnykaushik/.ansible/collections:/usr/share/ansible/collections
  executable location = /opt/homebrew/bin/ansible
  python version = 3.12.6 (main, Sep  6 2024, 19:03:47) [Clang 15.0.0 (clang-1500.3.9.4)] (/opt/homebrew/Ce
llar/ansible/10.4.0/libexec/bin/python)
  jinja version = 3.1.4
  libyaml = True
sunnykaushik@Sunnys-MacBook-Pro demo %
```

Figure 4: Ansible version

5.9 Installing Ngrok on Mac

Follow the steps below to install Ngrok on your Mac:

1. **Download Ngro:** Open your web browser and go to the Ngrok website: `https://ngrok.com/download`. Click on the **Download** button for macOS after the signup.
   Alternatively, you can directly open your terminal and type in the following command.

```
brew install ngrok/ngrok/ngrok
```

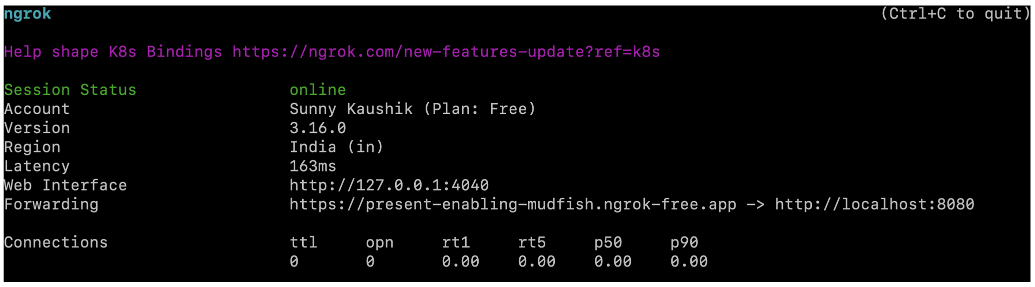
2. **Authenticate Ngrok:** To use Ngrok, you need to authenticate with your account. First, sign up on the Ngrok website if you don't have an account. After signing in, you will find your authentication token under the **Auth** section of the dashboard.
3. **Set Authentication Token:** Copy your authentication token and run the following command in the Terminal, replacing `YOUR_AUTH_TOKEN` with the actual token:

```
ngrok config add-authtoken {YOUR_AUTH_TOKEN}
```

4. **Run Ngrok:** You can now run Ngrok. For example, to expose a local web server running on port 8080, use the command:

```
ngrok http 8080
```

Ngrok will provide a public URL that allows external access to your local server.



```
ngrok                                                                                       (Ctrl+C to quit)

Help shape K8s Bindings https://ngrok.com/new-features-update?ref=k8s

Session Status                online
Account                       Sunny Kaushik (Plan: Free)
Version                       3.16.0
Region                        India (in)
Latency                       163ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://present-enabling-mudfish.ngrok-free.app -> http://localhost:8080

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

Figure 5: NGROK

6 Project Implementation - DevOps Solution

6.1 Project Setup in Visual Studio Code

To begin, we'll create the project in **Visual Studio Code**. Follow these steps to create the project and set it up with Maven as the build tool:

1. **Launch Visual Studio Code:** Open VS Code on your local machine.
2. **Create a New Folder for the Project:** Navigate to `File` − > `Open Folder`, and create or select an empty folder to serve as the project folder. Name the folder `Calculator`.
3. **Create a New Maven Project:**
   (a) Open the Command Palette by pressing `Cmd + Shift + P`.
   (b) In the Command Palette, type `Maven: Generate from Maven Archetype` and select it.
   (c) Choose the archetype (e.g., `maven-archetype-quickstart`) from the list.

   This will create a basic Java project with Maven.
4. **Configure Java JDK:** Ensure that the Java JDK is properly installed and configured in VS Code. You can configure the JDK in `Settings` or ensure that `JAVA_HOME` is set in your environment.

6.2 Git Workflow

The following commands are used to initialize 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
```

The username will be the github username and personal access token needs to be created and authenticated. To authenticate with GitHub using a Personal Access Token (PAT), follow these steps:

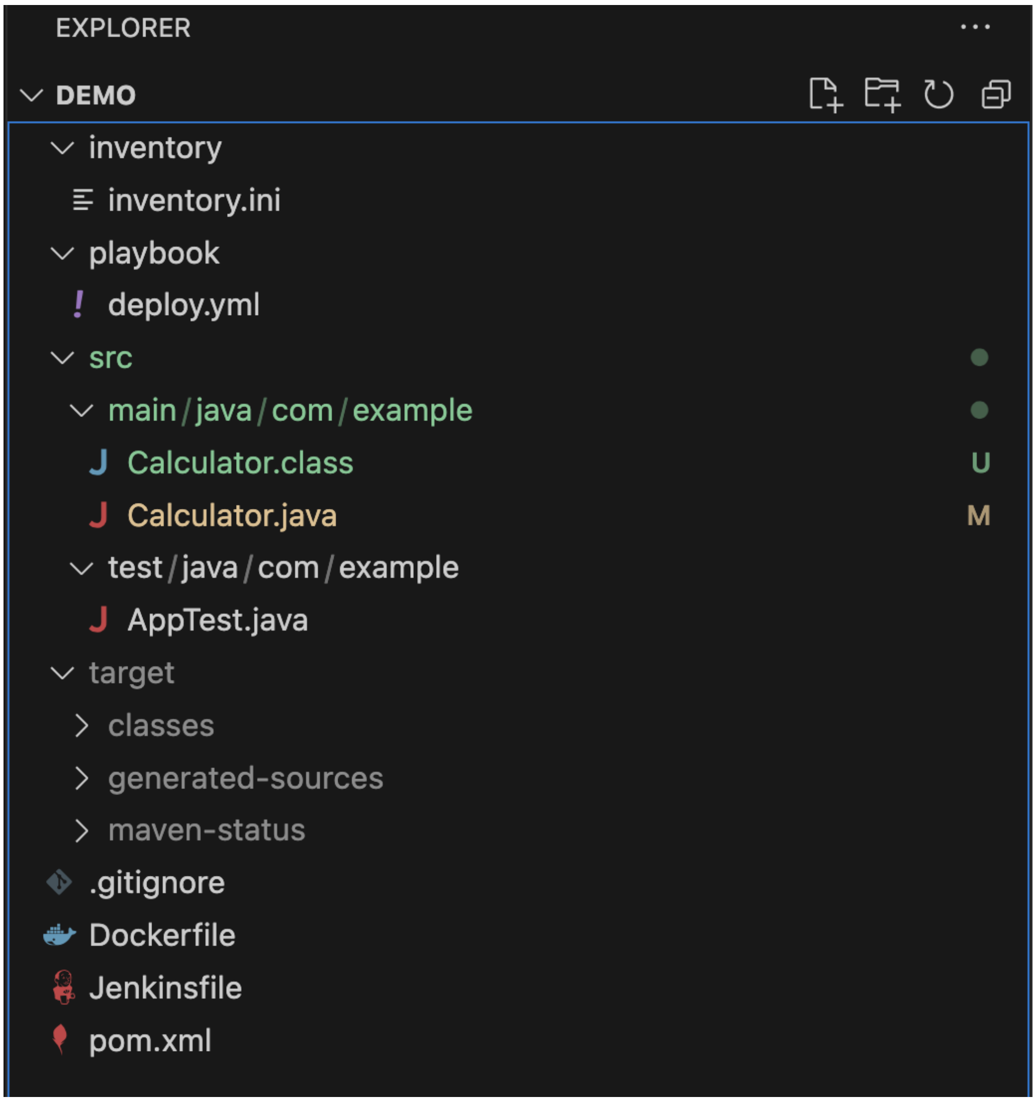
1. **Navigate to Developer Settings:**
   - In the top-right corner of the page, click on your profile picture and select `Settings`.
   - On the left sidebar, scroll down and click on `Developer settings`.

2. **Create a Personal Access Token:**
   - Under `Developer settings`, click on `Personal access tokens`.
   - Click `Generate new token`.
3. **Select Scopes:**
   - Provide a name for the token, like `VS Code Access Token`.
   - Under the `Select scopes` section, choose the necessary permissions. For repository access, you will typically select:
     - `repo` - Full control of private repositories
     - `workflow` - Update GitHub Action workflows
4. **Generate the Token:**
   - Scroll to the bottom and click `Generate token`.
   - Copy the token and store it in a safe place. **Note:** You won't be able to see this token again, so make sure to copy it.
5. **Use the Token in VS Code:**
   - When pushing your code to GitHub from Visual Studio Code using Git, it will prompt for a username and password.
   - Enter your GitHub username as the username.
   - Use the Personal Access Token (PAT) as the password when prompted.

6.3 Project Structure

File/Folder	**Description**
`inventory/`	Directory containing the Ansible inventory files for managing hosts.
`inventory.ini`	Ansible inventory file specifying the hosts and groups for deployment.
`playbook/`	Directory containing the Ansible playbooks for automating deployment processes.
`deploy.yml`	Ansible playbook for automating the deployment of the project.
`src/`	Source directory containing the Java project files.
`main/java/com/example/`	Directory containing the main Java source code for the calculator program.
`Calculator.java`	Main source file containing the calculator's business logic.
`test/java/com/example/`	Directory containing test files for the Java project.
`AppTest.java`	Unit test class for testing the functionality of the calculator.
`target/`	Directory where compiled files and build artifacts are stored after the build.
`.gitignore`	Specifies files and directories to be ignored by Git version control.
`Dockerfile`	Defines the instructions to containerize the application using Docker.
`Jenkinsfile`	Script for automating the CI/CD pipeline using Jenkins.
`pom.xml`	Maven configuration file for managing dependencies and project build settings.

Table 1: Explanation of Project Files and Folders



```
EXPLORER                                   ...
DEMO
  inventory
    inventory.ini
  playbook
    deploy.yml
  src
    main / java / com / example
      Calculator.class                     U
      Calculator.java                      M
    test / java / com / example
      AppTest.java
  target
    classes
    generated-sources
    maven-status
  .gitignore
  Dockerfile
  Jenkinsfile
  pom.xml
```

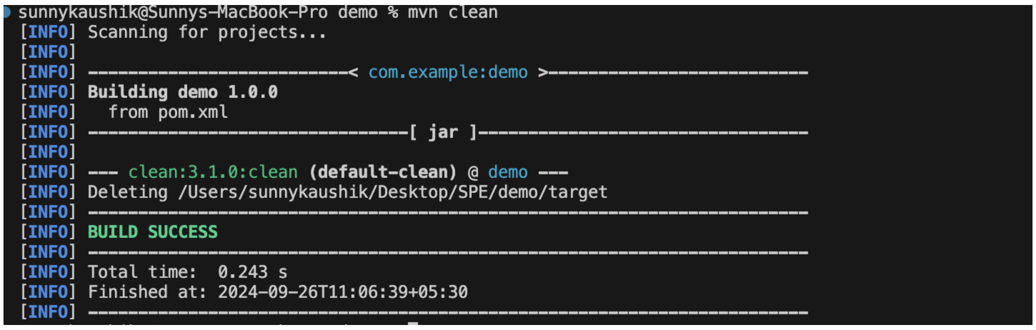
Figure 6: Structure

6.4 Steps to Build and Run the Project

To build and run the project using Maven, the following steps are executed:

1. **Clean the Project:**

```
mvn clean
```



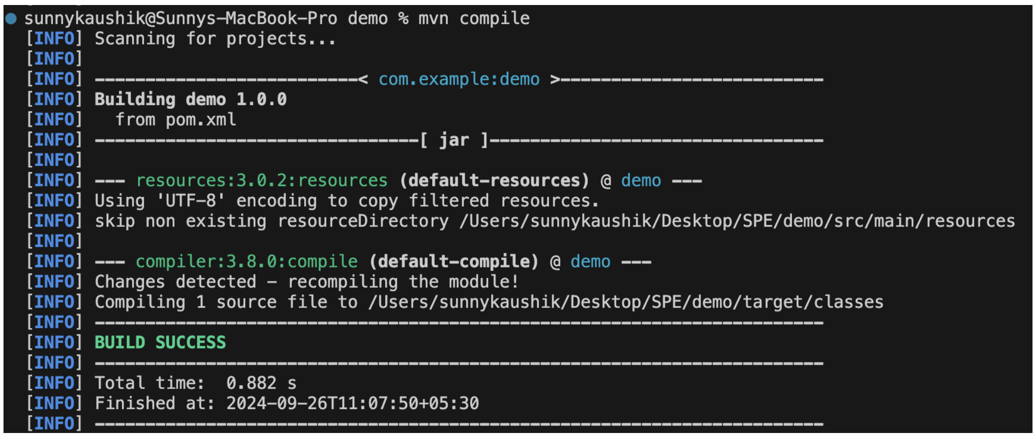
```
sunnykaushik@Sunnys-MacBook-Pro demo % mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------------< com.example:demo >---------------------------
[INFO] Building demo 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.1.0:clean (default-clean) @ demo ---
[INFO] Deleting /Users/sunnykaushik/Desktop/SPE/demo/target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.243 s
[INFO] Finished at: 2024-09-26T11:06:39+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 7: mvn clean

This command removes the `target` folder, ensuring a fresh start for the subsequent compilation. It eliminates any previous build artifacts that might interfere with the new build.

2. **Compile the Project:**

```
mvn compile
```



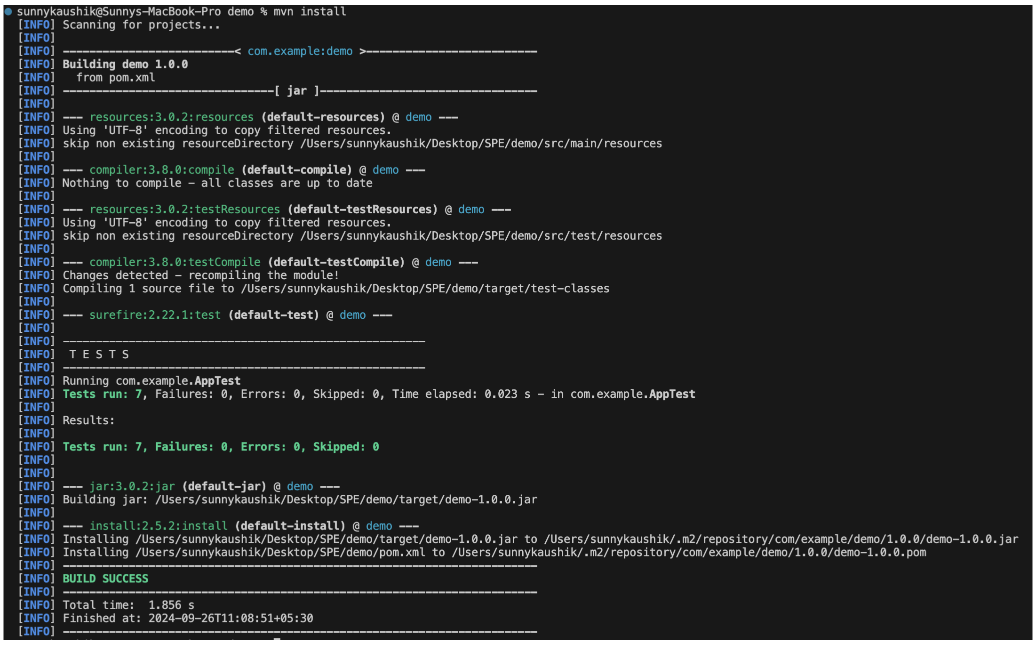
```
sunnykaushik@Sunnys-MacBook-Pro demo % mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------------< com.example:demo >---------------------------
[INFO] Building demo 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.0.2:resources (default-resources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/sunnykaushik/Desktop/SPE/demo/src/main/resources
[INFO]
[INFO] --- compiler:3.8.0:compile (default-compile) @ demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /Users/sunnykaushik/Desktop/SPE/demo/target/classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.882 s
[INFO] Finished at: 2024-09-26T11:07:50+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 8: mvn compile

This command compiles the project and associated test cases. It ensures that the code is free from compilation errors before proceeding to packaging and running tests.

3. **Install the Project:**

```
mvn install
```



```
sunnykaushik@Sunnys-MacBook-Pro demo % mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.example:demo >--------------------------
[INFO] Building demo 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.0.2:resources (default-resources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/sunnykaushik/Desktop/SPE/demo/src/main/resources
[INFO]
[INFO] --- compiler:3.8.0:compile (default-compile) @ demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.0.2:testResources (default-testResources) @ demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/sunnykaushik/Desktop/SPE/demo/src/test/resources
[INFO]
[INFO] --- compiler:3.8.0:testCompile (default-testCompile) @ demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /Users/sunnykaushik/Desktop/SPE/demo/target/test-classes
[INFO]
[INFO] --- surefire:2.22.1:test (default-test) @ demo ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.AppTest
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.023 s - in com.example.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.0.2:jar (default-jar) @ demo ---
[INFO] Building jar: /Users/sunnykaushik/Desktop/SPE/demo/target/demo-1.0.0.jar
[INFO]
[INFO] --- install:2.5.2:install (default-install) @ demo ---
[INFO] Installing /Users/sunnykaushik/Desktop/SPE/demo/target/demo-1.0.0.jar to /Users/sunnykaushik/.m2/repository/com/example/demo/1.0.0/demo-1.0.0.jar
[INFO] Installing /Users/sunnykaushik/Desktop/SPE/demo/pom.xml to /Users/sunnykaushik/.m2/repository/com/example/demo/1.0.0/demo-1.0.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.856 s
[INFO] Finished at: 2024-09-26T11:08:51+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 9: mvn install

This command generates the JAR file, runs the test cases, packaging the project into a final output artifact once the compilation is successful.

6.5 Maven Configuration

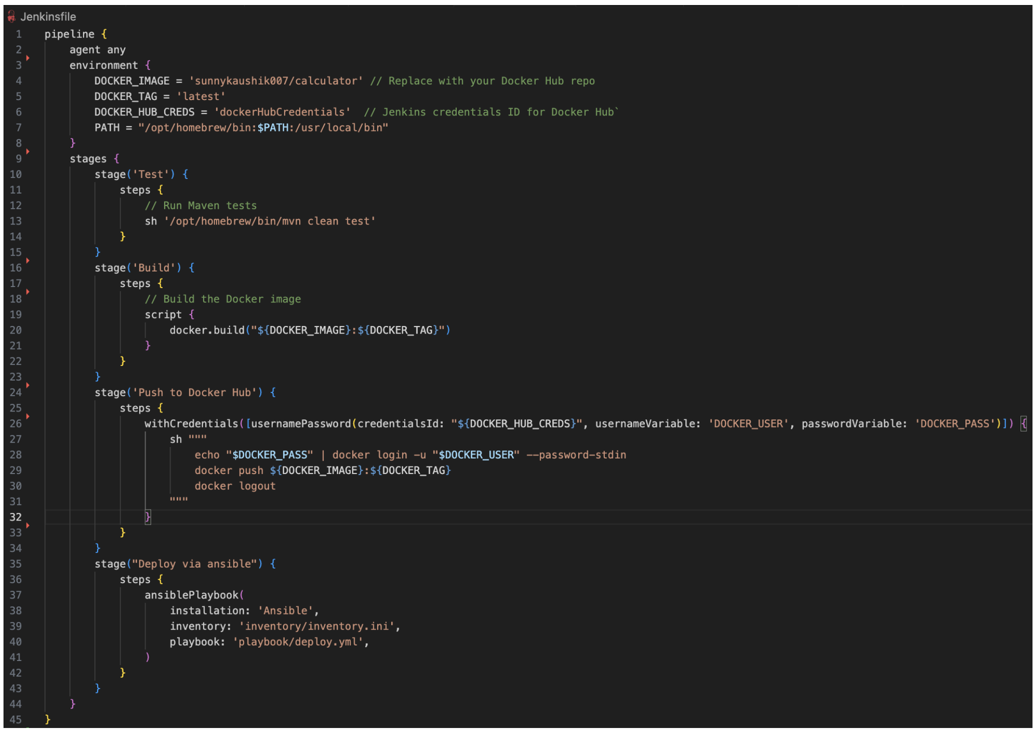
Within the Maven configuration file (`pom.xml`), certain key elements define the structure and behavior of the project build:

- The `<mainClass>` tag specifies the path to the main Java file following the package structure. In this case, the main class is `com.example.Calculator`.
- The `<dependencies>` tag includes the project dependencies. For this project, we added the JUnit library for running unit tests, defined with the groupId `junit`, artifactId `junit`, and version 4.11.
- The `<build>` section specifies various Maven plugins used in the project:
  - The `maven-compiler-plugin` compiles the project using Java 1.8, as set in the project properties.
  - The `maven-jar-plugin` is responsible for creating the final JAR file, including the main class manifest information.
  - The `maven-surefire-plugin` is used to run unit tests.

6.6 Jenkins

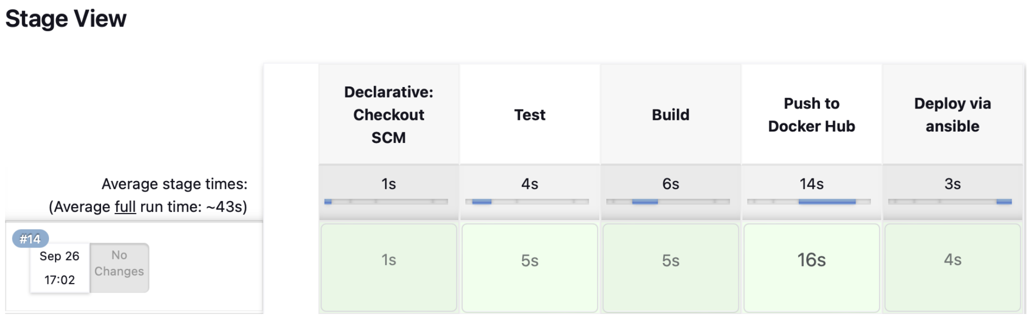
Go to `http://localhost:8080` and install the necessary plugins. We will establish a Jenkins pipeline comprising six distinct stages, as outlined below:

- **Stage 1: Test**
  This stage runs Maven tests to ensure that the code is functioning correctly before building the Docker image. It uses the command `/opt/homebrew/bin/mvn clean test` to execute the tests.
- **Stage 2: Build**
  In this stage, the Docker image is built using the specified Dockerfile. The command creates a new Docker image based on the source code.



```
Jenkinsfile
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'sunnykaushik007/calculator' // Replace with your Docker Hub repo
        DOCKER_TAG = 'latest'
        DOCKER_HUB_CREDS = 'dockerHubCredentials'  // Jenkins credentials ID for Docker Hub`
        PATH = "/opt/homebrew/bin:$PATH:/usr/local/bin"
    }
    stages {
        stage('Test') {
            steps {
                // Run Maven tests
                sh '/opt/homebrew/bin/mvn clean test'
            }
        }
        stage('Build') {
            steps {
                // Build the Docker image
                script {
                    docker.build("${DOCKER_IMAGE}:${DOCKER_TAG}")
                }
            }
        }
        stage('Push to Docker Hub') {
            steps {
                withCredentials([usernamePassword(credentialsId: "${DOCKER_HUB_CREDS}", usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                    sh """
                        echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER" --password-stdin
                        docker push ${DOCKER_IMAGE}:${DOCKER_TAG}
                        docker logout
                    """
                }
            }
        }
        stage("Deploy via ansible") {
            steps {
                ansiblePlaybook(
                    installation: 'Ansible',
                    inventory: 'inventory/inventory.ini',
                    playbook: 'playbook/deploy.yml',
                )
            }
        }
    }
}
```

Figure 10: Jenkins



Stage View

	Declarative: Checkout SCM	Test	Build	Push to Docker Hub	Deploy via ansible
Average stage times: (Average full run time: ~43s)	1s	4s	6s	14s	3s
#14 Sep 26 17:02 No Changes	1s	5s	5s	16s	4s

Figure 11: Stage View

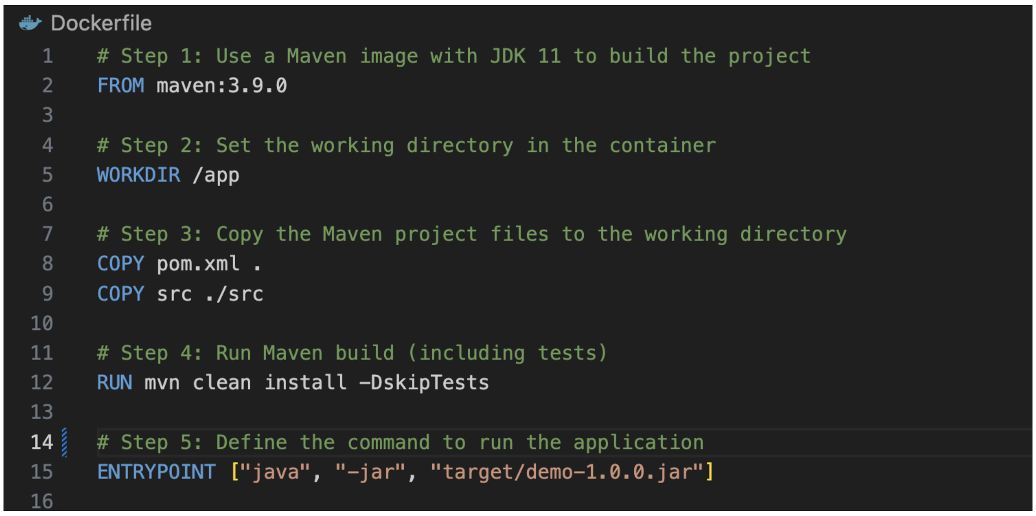
In the Declarative: Checkout SCM stage the following things are being checked–

1. **Git Installation:** Jenkins checks for the Git installation and determines which version of Git is available. In this case, it is using `git version 2.39.5 (Apple Git-154)`.
2. **Git Fetch:** Jenkins fetches the latest changes from the remote Git repository (`https://github.com/Sunny-Kaushik/SPE_Mini_Project.git`). It retrieves the branches and updates the references to match the remote repository.
3. **Git Checkout:** Jenkins checks out the specific commit (`2f0d1564781034b7af71439367130aa04206cf61`) for the pipeline job to ensure that the workspace contains the correct version of the code. This commit corresponds to the latest version in the repository at that point.
4. **Why is it appearing?** This step is automatically included because Jenkins pipelines, by default, include the `checkout scm` step at the beginning of the pipeline to ensure that the code from the repository is fetched.
5. **How to remove it:** If you do not want the automatic checkout of the repository, you can explicitly disable it in your pipeline script by adding the following code before your stages:

```
options {
    skipDefaultCheckout true
    }
```

This will prevent Jenkins from automatically checking out the code before the pipeline stages begin.

6.7 Docker


Dockerfile

```
# Step 1: Use a Maven image with JDK 11 to build the project
FROM maven:3.9.0

# Step 2: Set the working directory in the container
WORKDIR /app

# Step 3: Copy the Maven project files to the working directory
COPY pom.xml .
COPY src ./src

# Step 4: Run Maven build (including tests)
RUN mvn clean install -DskipTests

# Step 5: Define the command to run the application
ENTRYPOINT ["java", "-jar", "target/demo-1.0.0.jar"]

```

Figure 12: Docker

Now, we need to create the docker container for our project. In the project structure this goes under the dockerfile.

This Dockerfile uses a multi-step process to build and run a Maven-based Java application. In `Step 1`, the base image is set to Maven version 3.9.0 with JDK 11 to ensure compatibility with the project. `Step 2` sets the working directory to `/app` within the container. `Step 3` copies the project files, including `pom.xml` and the source code in the `src` directory, to the container. In `Step 4`, the `mvn clean install` command is used to compile the project and package it, while skipping tests. Finally, in `Step 5`, the entry point is defined to run the JAR file located at `target/demo-1.0.0.jar` when the container starts. The final container can run independently without any dependencies.

- **Stage 3: Push to Docker Hub**
  This stage logs into Docker Hub using the provided credentials and pushes the built Docker image to the specified repository on Docker Hub. It uses the `docker push` command to upload the image. We can a overview of the repository at `https://hub.docker.com`

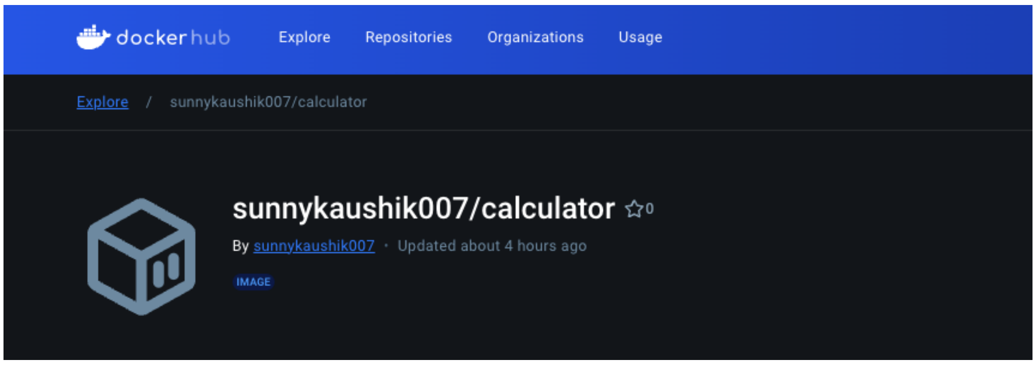
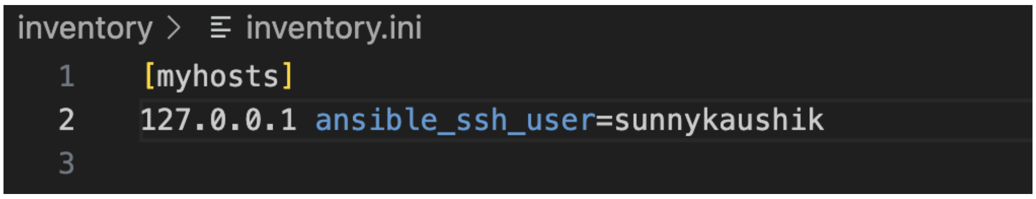

docker hub
Explore
Repositories
Organizations
Usage
Explore / sunnykaushik007/calculator
sunnykaushik007/calculator ☆0
By sunnykaushik007 · Updated about 4 hours ago
IMAGE

Figure 13: Docker Hub

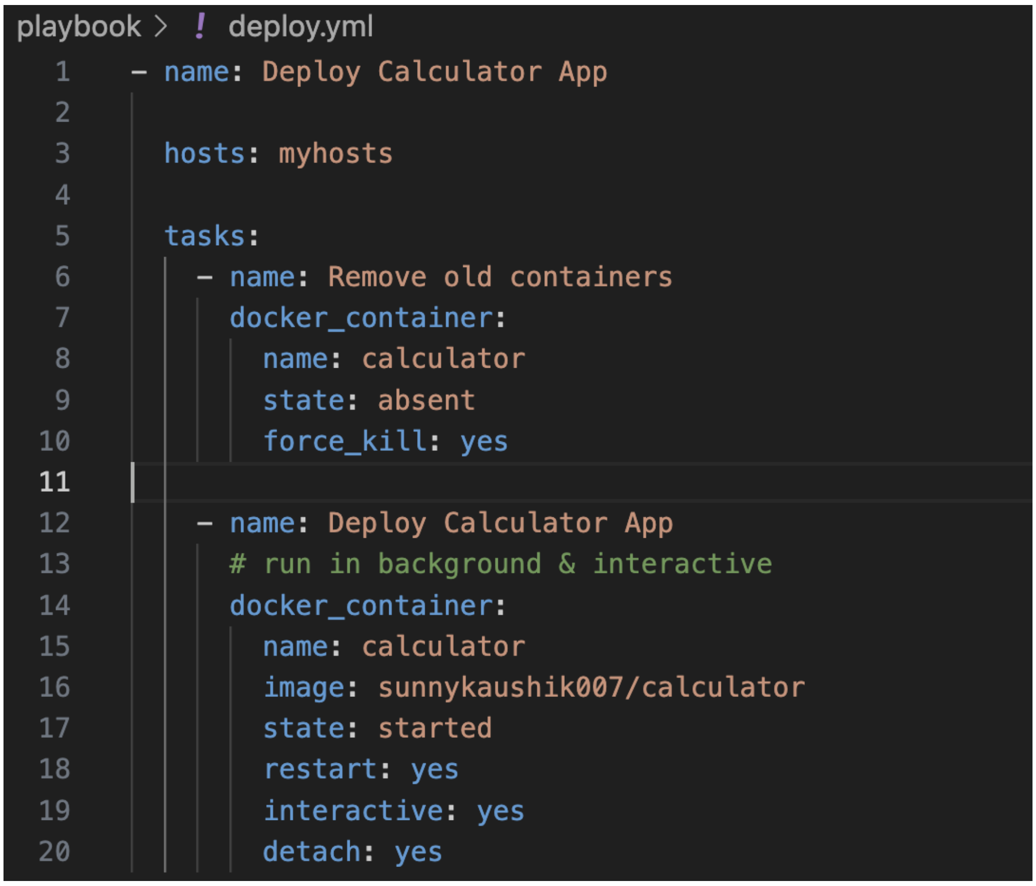
- **Stage 4: Deploy via Ansible**
  We are going to pull images from the DockerHub and create container using Ansible.In this stage, the deployment is handled using Ansible. It specifies the installation to use, the inventory file, and the playbook to run for deployment.
  The above steps will be contained in the folder inventory and playbook containing inventory.ini and deploy.yml files respectively.



```
inventory > inventory.ini
[myhosts]
127.0.0.1 ansible_ssh_user=sunnykaushik

```

Figure 14: Inventory



```
playbook > deploy.yml
- name: Deploy Calculator App

  hosts: myhosts

  tasks:
    - name: Remove old containers
      docker_container:
        name: calculator
        state: absent
        force_kill: yes

    - name: Deploy Calculator App
      # run in background & interactive
      docker_container:
        name: calculator
        image: sunnykaushik007/calculator
        state: started
        restart: yes
        interactive: yes
        detach: yes
```

Figure 15: Deploy

Before moving ahead let's first check the already existing images using the command:

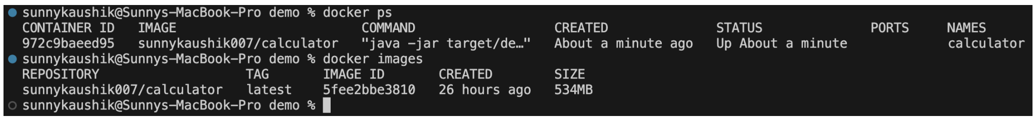
```
docker images
```

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

```
docker run -it -d --name calculator sunnykaushik007/calculator:latest
```

To check the containers run the following command:

```
docker ps
```



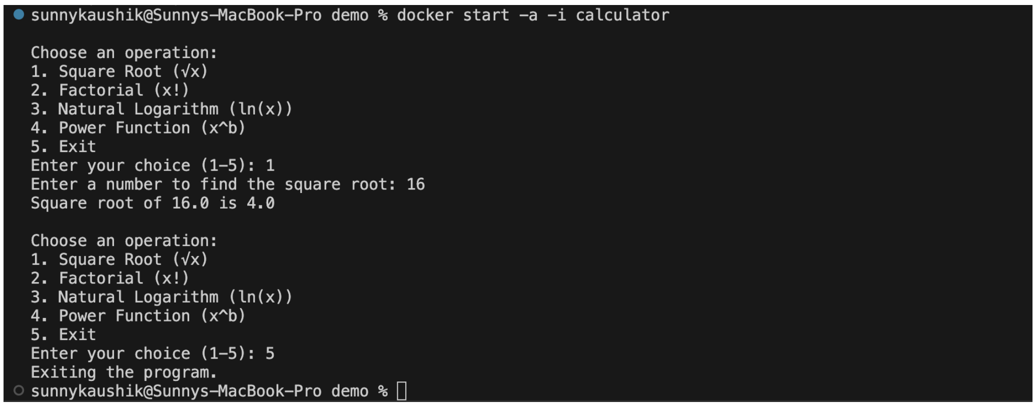
```
● sunnykaushik@Sunnys-MacBook-Pro demo % docker ps
  CONTAINER ID   IMAGE                        COMMAND                  CREATED              STATUS              PORTS     NAMES
  972c9baeed95   sunnykaushik007/calculator   "java -jar target/de…"   About a minute ago   Up About a minute             calculator
● sunnykaushik@Sunnys-MacBook-Pro demo % docker images
  REPOSITORY                   TAG       IMAGE ID       CREATED        SIZE
  sunnykaushik007/calculator   latest    5fee2bbe3810   26 hours ago   534MB
○ sunnykaushik@Sunnys-MacBook-Pro demo %
```

Figure 16: Deploy

Now finally to execute the container, run the following command(the calculator here is the name of the conatiner):

```
docker start -a -i calculator
```

The output will be the following and the desired operations will be performed, which in our case is a simple calculator:



```
● sunnykaushik@Sunnys-MacBook-Pro demo % docker start -a -i calculator

  Choose an operation:
  1. Square Root (√x)
  2. Factorial (x!)
  3. Natural Logarithm (ln(x))
  4. Power Function (x^b)
  5. Exit
  Enter your choice (1-5): 1
  Enter a number to find the square root: 16
  Square root of 16.0 is 4.0

  Choose an operation:
  1. Square Root (√x)
  2. Factorial (x!)
  3. Natural Logarithm (ln(x))
  4. Power Function (x^b)
  5. Exit
  Enter your choice (1-5): 5
  Exiting the program.
○ sunnykaushik@Sunnys-MacBook-Pro demo %
```

Figure 17: Calculator