SPE Mini Project Report

IMT2020067 Rishi Vakharia

Link to Github repository: https://github.com/rishi2o2o/Calculator

Link to DockerHub repository: https://hub.docker.com/repository/docker/rishi2o2o/calculator/general

Explanation of what the project is

The project is to build a simple menu-driven calculator application accessible via command line. The calculator has the following menu-driven operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit

The code for menu-driven application is very trivial and it is not the focus of this project. The focus of this project is to become familiar with the SDLC processes that happen after development - like testing, compilation, packaging, distributing, deployment, etc. All these processes are seen and done via DevOps approach and thus many DevOps tools like Jenkins, Docker, Ansible, Maven, etc. are used.

Elaboration on the tools used for the project

- **Version control using source code management tools**: Source code management tools like **Git** and **GitHub** help us track the changes made in our projects, revert back to previous working versions, collaborate with a team of developers working on the same project, work on new features using branches, etc.

We have used Git to track versions of our code and see the changes that we made since the start of project. And we have used GitHub to make our code public so that it can be used by various sources like the Jenkins server to pull the code from the GitHub repository.

- **Java application development tools**: The source code for the application is written in **Java**; **JUnit** testing framework is used for the purposes of unit testing and **Maven** build automation tool is used to automate the build process and to manage dependencies.

Maven is a great build automation tool that can simplify the build process of Java projects tremendously. First, it provides a standard structure to all Java projects. Second, it can performs tasks like testing, compilation, build, etc. Third, it provides the facility for dependency management where we can mention the dependencies in pom.xml and Maven can automatically resolve/download them. pom.xml is the main file to interact with Maven and inside it details like plugins and dependencies are mentioned.

We have used JUnit and log4j (optional) in our project and they are mentioned in the pom.xml. Also, we have compiled our project along with all its dependencies into a single jar file using the maven-assembly-plugin.

- **Docker**: Docker is a containerization platform used to package applications and dependencies into portable, lightweight containers which makes the application platform independent.

Steps for how we set up docker, how we use it with Jenkins and Ansible are given in the next section.

- **Ansible**: Ansible is a configuration management tool that automates the deployment and management of infrastructure and applications.

Ansible is based on client-server architecture where the server is called Controller Node and clients are called Managed Hosts. The Controller Node is the only machine where Ansible needs to be installed. And the Controller Node sends 'modules' to the managed hosts, which are listed in the inventory file. The modules are set of instructions that we want to run on each managed host, this could be to configure each of the managed hosts or to deploy a product onto them.

Also, steps to configure Ansible are mentioned in the next section.

- **Jenkins**: Jenkins is a CI/CD tool that automates the test, build and deployment process. With Jenkins, we can make pipelines comprising of many stages and these stages can perform tasks that pertain to CI and CD.

In our Jenkins pipeline, we have the following stages:

1. Pull code from GitHub
2. Build jar file using Maven (includes testing)
3. Build a docker image from the Dockerfile
4. Push the image to Docker Hub
5. Remove dangling images
6. Pull image from Docker Hub and deploy on target hosts (using Ansible)

- **ngrok** and **Github webhooks**: ngrok creates secure tunnels to expose local servers to the public internet and Github webhooks trigger automated actions when specific events occur in a GitHub repository.

Ngrok is used by us to expose our Jenkins server, which by default runs on localhost:8080 url. And, using GitHub webhooks, whenever we make a commit to our GitHub repo, automatically the build of our Jenkins pipeline will start.

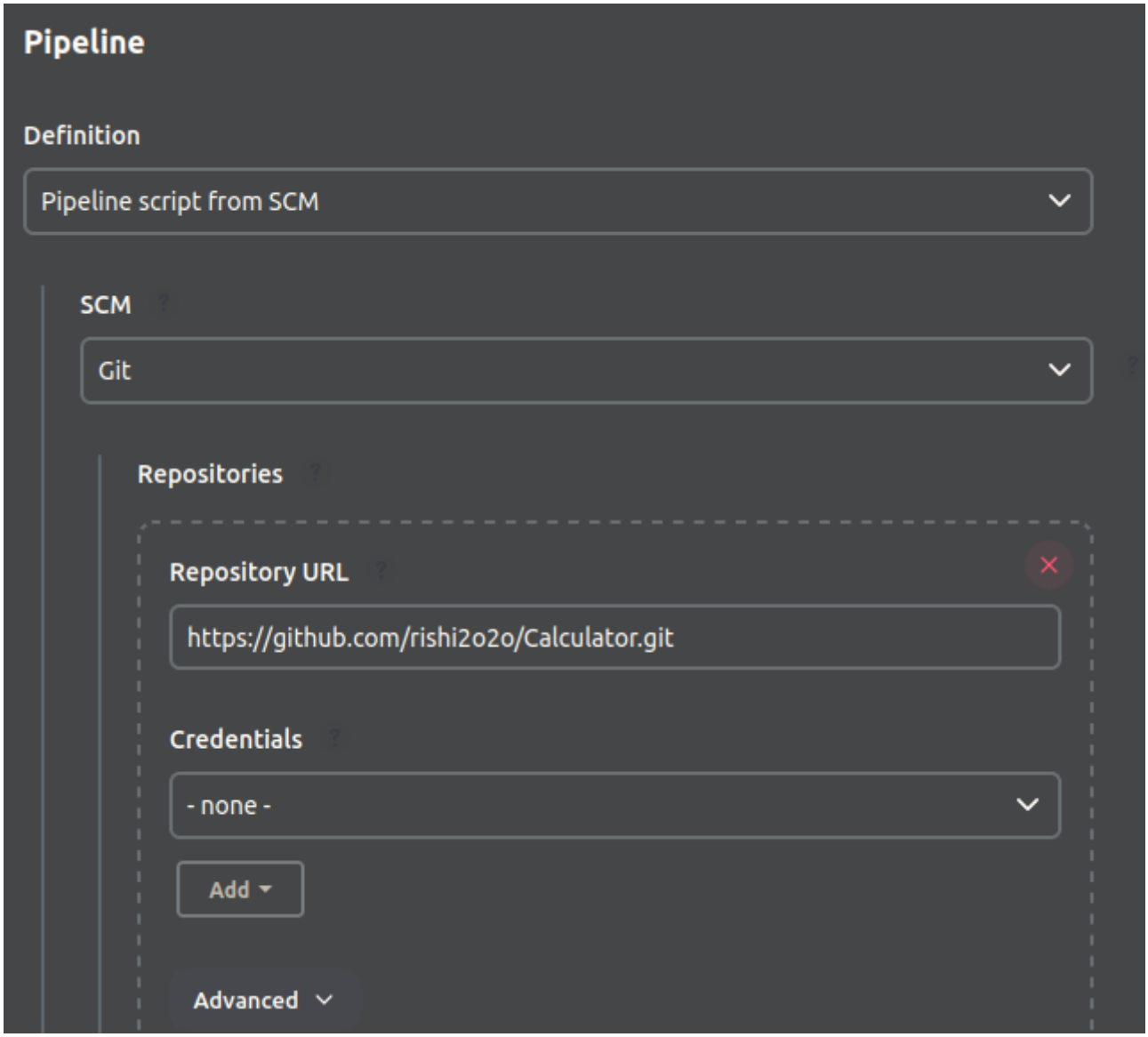
Explanation of the steps taken to achieve the end result

First the following tools will need to be installed: Git, Java, Maven, Jenkins, Docker, Ansible, ngrok

Then, we need to make a repository for our project on GitHub.

Jenkins

Then, we need to setup Jenkins for the first time use. After this, we can make a new pipeline project in Jenkins and in its Configure option, specify the GitHub repository of our project. I have added my Jenkins pipeline script in a Jenkinsfile and added it to the Github repo.



Also, since we are working with Git, GitHub, Docker and Ansible in our Jenkinsfile, we need to install those plugins in Jenkins. (Dashboard > Manage Jenkins > Plugins)

Also, since the "credentials for Docker Hub" and the "credentials for target user in Ansible" are being used in our Jenkinsfile, we need to add those credentials in Jenkins and set their visibility to Global. (Dashboard > Manage Jenkins > Credentials).

The ansible Controller Node in our case is the jenkins user @ localhost machine and managed host is rishi user @ localhost machine. Ansible works via ssh, and the Controller needs to ssh into hosts to execute the modules, so the password for rishi@localhost needs to be specifed to the Controller Node jenkins@localhost. This is done by making a Jenkins Credential "LocalhostUserCredentials" for our user 'rishi' and referencing it in our Jenkins pipeline:

```
age('Stage 6: Pull image from Docker Hub and deploy on hosts u
  steps {
      ansiblePlaybook installation: 'Ansible',
      playbook: 'Deployment/deploy.yml',
      inventory: 'Deployment/inventory',
      credentialsId: 'LocalhostUserCredentials'
  }
```

Jenkins stores the files for its projects in the /var/lib/jenkins/workspace folder. You can check the folder named ‘Calculator’, it will contain all the code that Jenkins pulls.

Jenkins, docker, ansible (some caveats)

To run the docker commands without sudo from 'jenkins' user and local user ('rishi' in our case), we need to add those users to the docker user group. The 'jenkins' user will run docker commands in the Stages 3, 4 and 5 of pipeline:

```
    }
}
stage('Stage 3: Build a docker image from the Dockerfile') {
    steps {
        script {
            docker_image = docker.build "rishi2o2o/calculator:latest"
        }
    }
}
stage('Stage 4: Push the image to Docker Hub') {
    steps {
        script {
            docker.withRegistry('', 'DockerHubCred') {
                docker_image.push()
            }
        }
    }
}
stage('Stage 5: Remove dangling images') {
    steps {
        script {
            sh 'docker image prune -f'
        }
    }
```

And 'rishi' user will run docker commands when it will be invoked by Ansible (as that user is mentioned in the inventory file).

- To see groups and its members in linux, you can do “sudo cat /etc/gshadow"
- To add a user to a group, you can do “sudo usermod -aG ”
- After doing this you should see:

```
snap_daemon:!::
docker:!::jenkins,rishi
jenkins:!::
```

--> users 'jenkins' and 'rishi' are added to docker group and can run docker commands without sudo.

Ansible

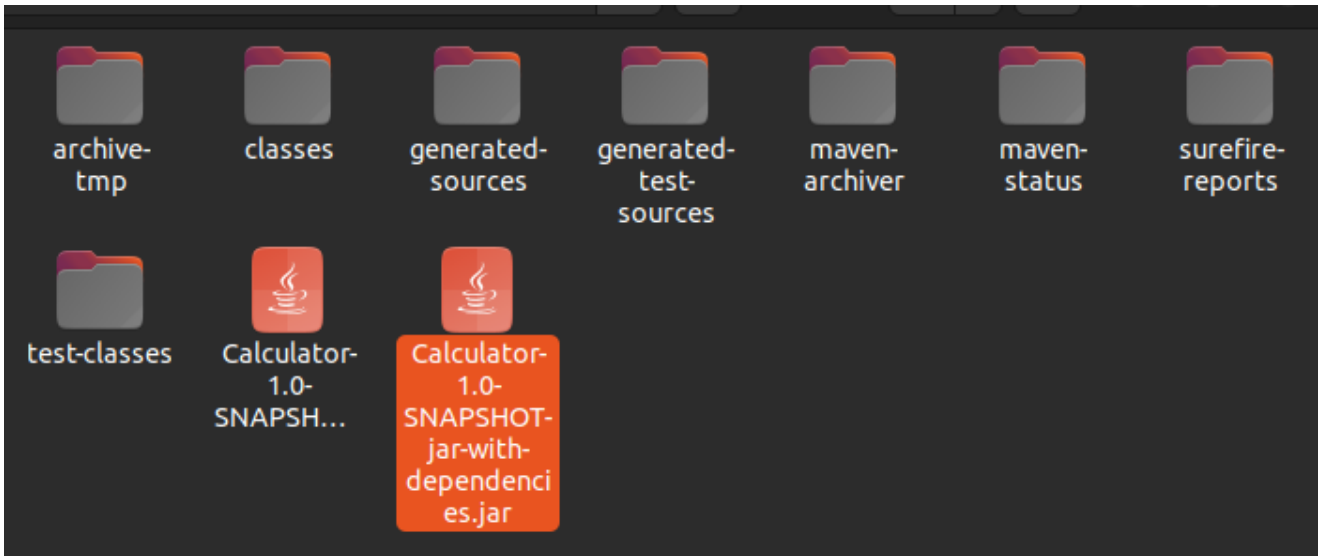
- In our case in Ansible, we pull our docker image from Docker Hub and create a container named “calc_container” on each of our managed hosts.
- We only have 1 managed host, which also resides on our same machine (localhost).

Java development using Maven

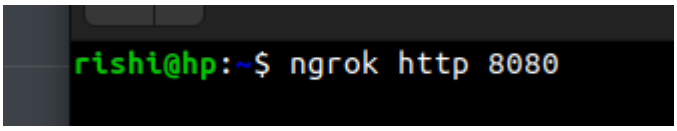
Create an empty Maven project using an IDE (or we can even use maven command line). In it, the code for calculator is written in src/main/java/org/example/Main.java and the code for testing is written in src/test/java/org/example/MainTest.java and maven project configurations are in pom.xml. The code for calculator and testing are really trivial, you can see it on GitHub. The layout of project structure is given below:

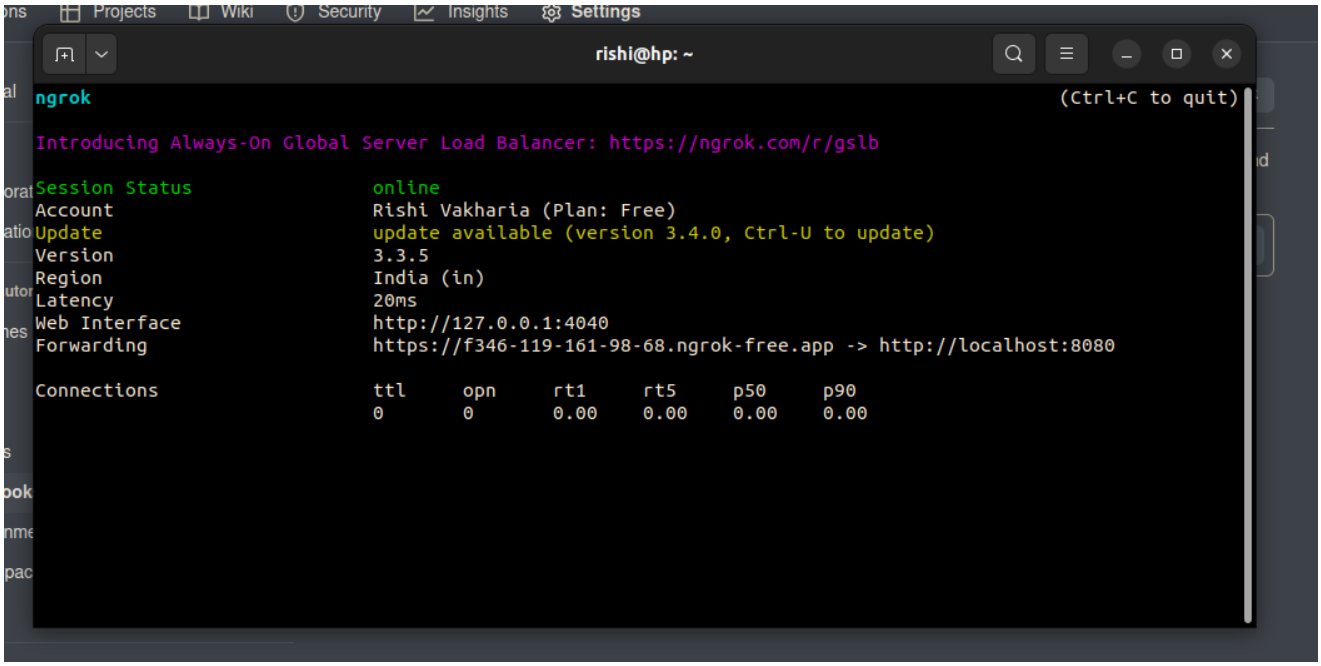
```
ry .
├── calculator.log
├── Deployment
│   ├── deploy.yml
│   └── inventory
├── Dockerfile
├── Jenkinsfile
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   └── org
│   │   │       └── example
│   │   │           └── Main.java
│   │   └── resources
│   │       └── log4j2.xml
│   └── test
│       └── java
│           └── org
│               └── example
│                   └── MainTest.java
```

On doing "mvn clean install", we get our executable jar file in target folder:

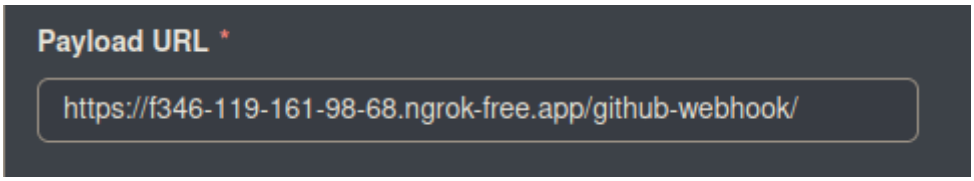


Setting up ngrok and Github webhook

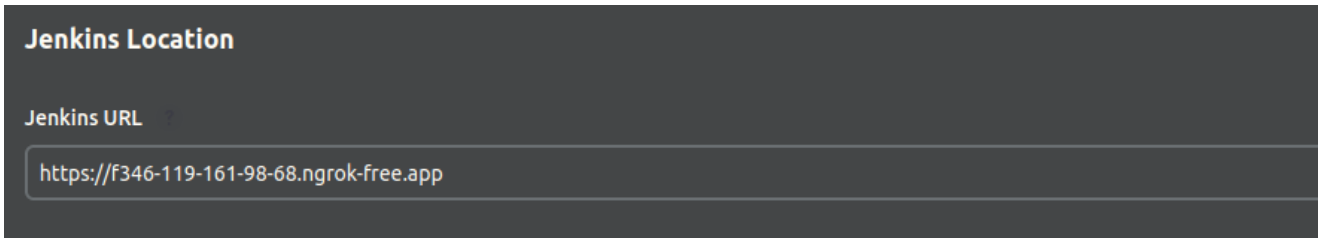




Now, Go to your GitHub repo > Settings > Webhooks, edit or create a webhook and change the payload URL like this:



In Jenkins, we need to change the system configuration, which can be done by going to Jenkins Dashboard > Manage Jenkins > System Configuration > System, and there we can change the Jenkins URL like this:



Now in our Jenkins project, we can specify our GitHub URL and select the build trigger as ‘GitHub hook trigger for GITScm polling’.

Screenshots (for the Java application development parts)

1. Sample code for addition operation of calculator (other functions have similar code)

```java
public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    int choice;
    double a, b;

    do {

        System.out.println("MENU:");
        System.out.println("1. Addition");
        System.out.println("2. Subtraction");
        System.out.println("3. Multiplication");
        System.out.println("4. Division");
        System.out.println("5. Exit");
        System.out.println();

        System.out.printf("Enter choice: ");
        choice = scanner.nextInt();

        switch(choice) {
            case 1:
                System.out.printf("Enter a: ");
                a = scanner.nextDouble();
                System.out.printf("Enter b: ");
                b = scanner.nextDouble();
                System.out.printf("Result: %.2f + %.2f is %.2f \n", a, b, addition(a,b));
                break;
            case 2:
```

--> The addition function has nothing fancy (we have added logging to the function which is optional)

```java
public static double addition(double a, double b) {

    logger.info("[ADDITION OPERATION] [START] " + a + " " + b);

    double res = a+b;

    logger.info("[ADDITION OPERATION] [END] " + res);

    return res;
}
```

2. Code for unit testing the addition operation (other operations are tested similarly)

MainTest.java

```java
package org.example;

import org.example.Main;
import org.junit.Assert;
import org.junit.Test;

public class MainTest {

    @Test
    public void test_addition() {

        double a = 33.3;
        double b = 22.2;
        double expec_res = 55.5;
    double res = Main.addition(a,b);
        Assert.assertEquals(expec_res, res, 0.00001);
    }

```

3. You can view the pom.xml from GitHub repo to see how plugins and dependencies are specified in Maven.

Screenshots (for the DevOps parts)

1. Code for the Jenkins pipeline script

Jenkinsfile

```
pipeline {
    environment {
        docker_image = ""
    }
    agent any
    stages {
        stage('Stage 1: Pull code from Github') {
            steps {
                git branch: 'main', url: 'https://github.com/rishi2o2o/Calculator.git'
            }
        }
        stage('Stage 2: Build jar file using Maven') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Stage 3: Build a docker image from the Dockerfile') {
            steps {
                script {
                    docker_image = docker.build "rishi2o2o/calculator:latest"
                }
            }
        }
        stage('Stage 4: Push the image to Docker Hub') {
            steps {
                script {
                    docker.withRegistry('', 'DockerHubCred') {
                        docker_image.push()
                    }
                }
            }
        }
        stage('Stage 5: Remove dangling images') {
            steps {
                script {
                    sh 'docker image prune -f'
                }
            }
        }
        stage('Stage 6: Pull image from Docker Hub and deploy on hosts using Ansible') {
            steps {
                ansiblePlaybook installation: 'Ansible',
                playbook: 'Deployment/deploy.yml',
                inventory: 'Deployment/inventory',
                credentialsId: 'LocalhostUserCredentials'
            }
        }
    }
}
```

2. Code for the Dockerfile (used to create a docker image)

Dockerfile

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

3. Code for Ansible inventory file (used to specify the hosts on which Ansible modules need to be run)

inventory

```
[localhost]
127.0.0.1 ansible_connection=local ansible_user=rishi
```

4. Code for Ansible Playbook (used for specifying the modules that need to be run on each host machine)

```yaml
---
- name: On target machines, pull image from Docker Hub and create a container
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3
  tasks:
    - name: Pull image from docker hub
      docker_image:
        name: rishi2o2o/calculator:latest
        source: pull
    - name: Start docker service
      service:
        name: docker
        state: started
    - name: Remove container from target machine if it exists already
      shell:
        cmd: |
          if [ "$(docker ps -a -q -f name=calc_container)" ]; then
            docker rm calc_container
          fi
    - name: Finally, create container on target machine
      shell: docker create -i -t --name calc_container rishi2o2o/calculator:latest
```

Screenshots of the result

1. On pushing a commit to Github:

```
rishi@hp: ~/Desktop/Code/SPE/Calculator
rishi@hp:~/Desktop/Code/SPE/Calculator$ git add .
rishi@hp:~/Desktop/Code/SPE/Calculator$ git commit -m "new commit"
[main c8971fa] new commit
 1 file changed, 1 insertion(+)
rishi@hp:~/Desktop/Code/SPE/Calculator$ git push -u origin main
Username for 'https://github.com': rishi2o2o
Password for 'https://rishi2o2o@github.com':
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 556 bytes | 185.00 KiB/s, done.
Total 8 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/rishi2o2o/Calculator.git
   c3482de..c8971fa  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
rishi@hp:~/Desktop/Code/SPE/Calculator$
```

2. The pipeline successfully executes:

	Declarative: Checkout SCM	Stage 1: Pull code from Github	Stage 2: Build jar file using Maven	Stage 3: Build a docker image from the Dockerfile	Stage 4: Push the image to Docker Hub	Stage 5: Remove dangling images	Stage 6: Pull image from Docker Hub and deploy on hosts using Ansible
Average stage times: (Average full run time: ~49s)	2s	3s	8s	3s	20s	517ms	55s
#41 Nov 06 05:05 1 commit	1s	14s	11s	4s	20s	562ms	9s

3. The container is deployed on the target machine:

```
rishi@hp: ~
rishi@hp:~$ docker ps -a
CONTAINER ID   IMAGE                        COMMAND                  CREATED             STATUS    PORTS     NAMES
3712daf19b44   rishi2o2o/calculator:latest  "java -cp Calculator…"   About a minute ago  Created             calc_container
rishi@hp:~$
```

4. Starting the container, we can access our menu-driven calc. application ! (😊)

```
rishi@hp: ~
rishi@hp:~$ docker start -i -a calc_container
MENU:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Exit

Enter choice:
```