International Institute of Information Technology Bangalore

CSE 816 - Software Production Engineering, 2025

Mini Project Report

Scientific Calculator with DevOps

Course Instructor:
Professor Thangaraju B
IIITB

Author:
Prabhav Pandey
MT2024115

Contents

List of Figures

1 Introduction

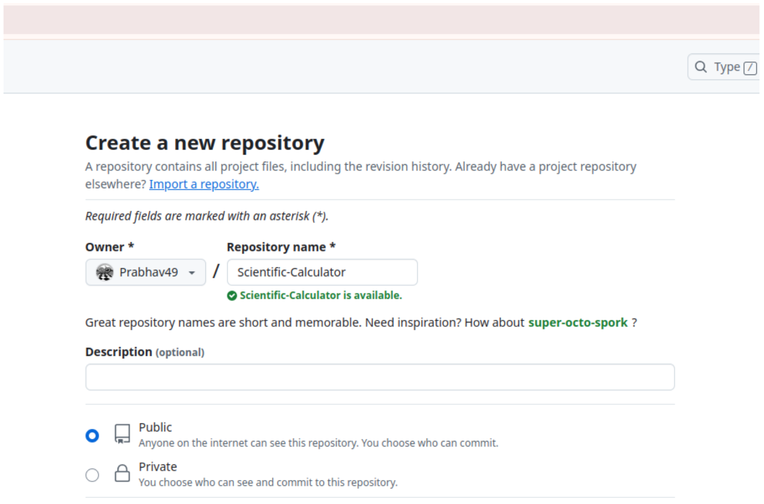
DevOps is a software engineering approach that bridges the gap between development and operations. It enables automation, collaboration, and continuous delivery, reducing manual effort and deployment failures.

2 Tools Used

- **Git and GitHub** - Git is a distributed version control system that tracks changes in source code during software development. GitHub is a web-based hosting service for Git repositories, providing collaboration features and a platform for managing and sharing code.
- **Maven** - Maven is a powerful build automation tool primarily used for Java projects. It simplifies the build process, manages dependencies, and provides a standard project structure. It handles compilation, packaging, testing, and deployment.
- **JUnit** - JUnit is a widely used unit testing framework for Java applications. It allows developers to write and run automated tests to verify the correctness of their code at the individual unit level.
- **Jenkins** - Jenkins is an open-source automation server that facilitates continuous integration and continuous delivery (CI/CD) pipelines. It automates build, test, and deployment processes, enabling frequent and reliable software releases.
- **Docker** - Docker is a platform for building, shipping, and running applications in containers. Containers allow developers to package an application and its dependencies into a standardized unit for software delivery, ensuring that the application runs consistently across different environments.
- **Docker Hub** - Docker Hub is a cloud-based registry service provided by Docker for storing and sharing Docker images. It acts as a central repository for developers to publish and retrieve pre-built images.
- **Ansible** - Ansible is an automation tool for configuration management, application deployment, and task automation. It uses a simple, agentless architecture to manage and configure systems, making it easy to automate repetitive tasks and deploy applications across multiple servers.

3 Project Setup

3.1 Creating the GitHub Repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: Prabhav49 / Repository name *: Scientific-Calculator

Scientific-Calculator is available.

Great repository names are short and memorable. Need inspiration? How about super-octo-spork ?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Figure 1: GitHub Repository Setup

```
    git init
    git remote add origin
  https://github.com/Prabhav49/ScientificCalculator.git
    git add .
    git commit -m "Initial commit"
    git push -u origin master
```

3.2 Installing Dependencies

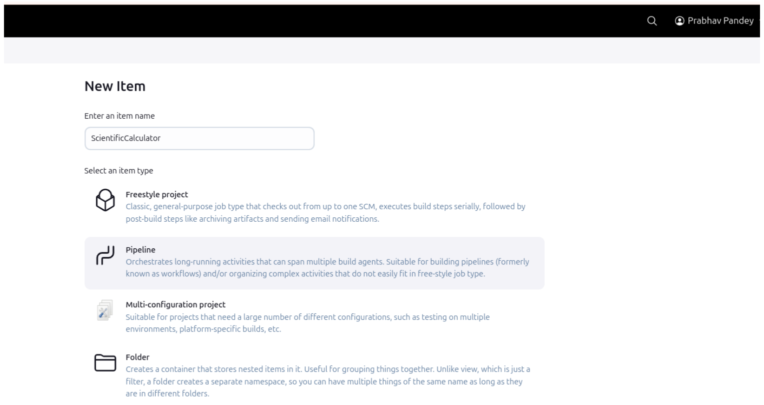
`sudo apt install git maven jenkins docker.io ansible -y`

3.3 Jenkins Setup

- Installed Jenkins and started the service.

```
    - sudo apt update
    - sudo apt install openjdk-17-jdk -y
    - wget -O-
  https://pkg.jenkins.io/debian-stable/jenkins.io.key |
  sudo tee /usr/share/keyrings/jenkins-keyring.asc >
  /dev/null
    - echo "deb
  [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
  https://pkg.jenkins.io/debian-stable binary/" | sudo tee
  /etc/apt/sources.list.d/jenkins.list > /dev/null
    - sudo apt update
    - sudo apt install jenkins -y
    - sudo systemctl enable jenkins
    - sudo systemctl start jenkins
    - sudo systemctl status jenkins
```

- Created a pipeline job with a Jenkins Dashboard.



New Item

Enter an item name

ScientificCalculator

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

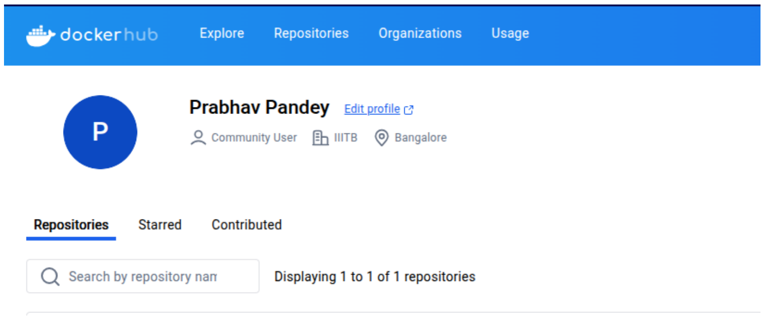
Figure 2: Created Pipeline Job through Jenkins

3.4 Install Plugin in Jenkins

- Email Notification
- Pipeline Stage View Plugin.
- Docker plugin.
- Docker pipeline.
- Docker-build-step
- Ansible plugin
- Ansible Tower plugin
- SSH Agent plugin
- SSH pipeline Steps
- SSH plugin

4 Docker Hub Account

Go to **Docker Hub site** and create an account.



Prabhav Pandey

Community User IIITB Bangalore

Displaying 1 to 1 of 1 repositories

Figure 3: Docker Hub Profile

5 Install Docker, Ansible

- Docker.

```
sudo apt install -y docker.io
```

- Ansible

```
sudo apt install -y ansible
```

6 Adding credentials in jenkins

- Go to Dashboard and click on "Manage Jenkins"

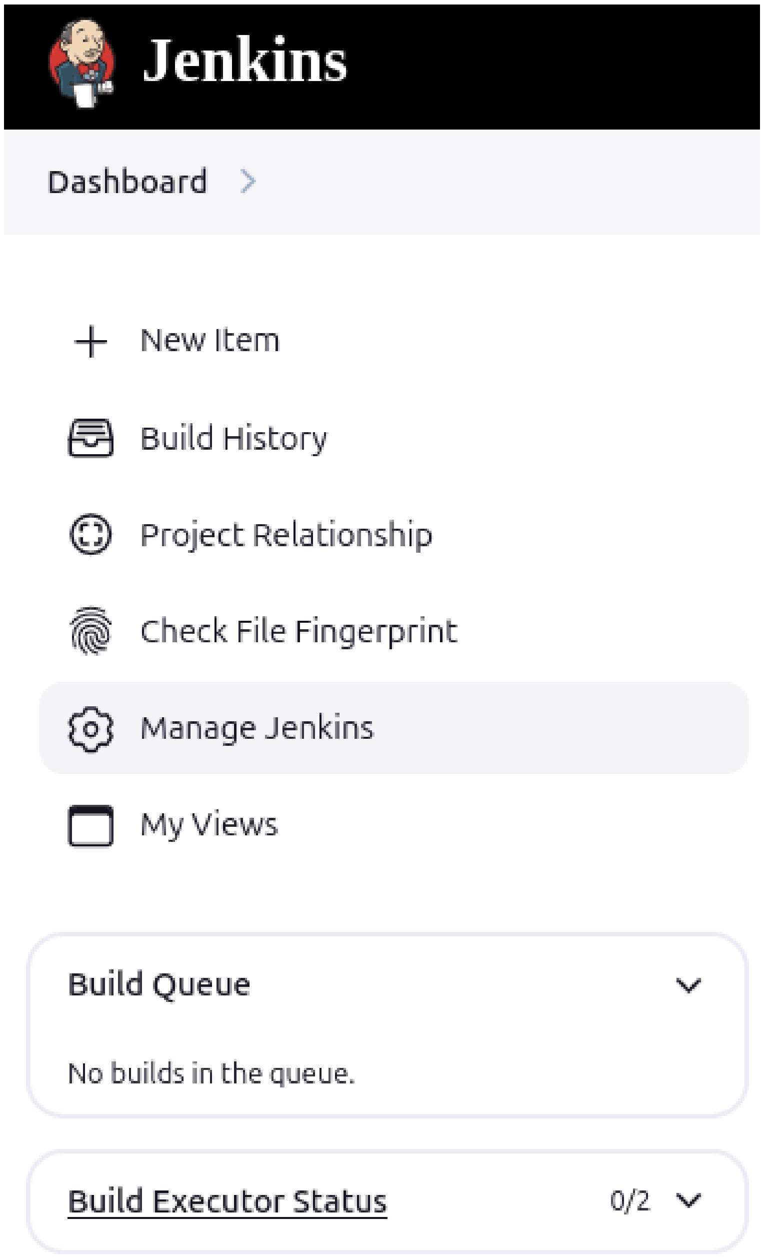
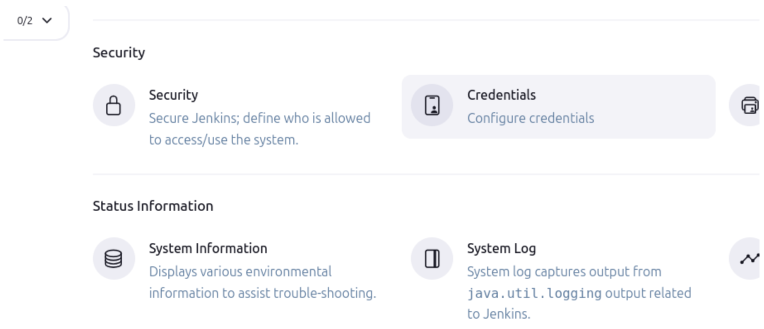


Figure 4: Manage Jenkins

- Go to ”Security and click on ”Credentials”



0/2

Security

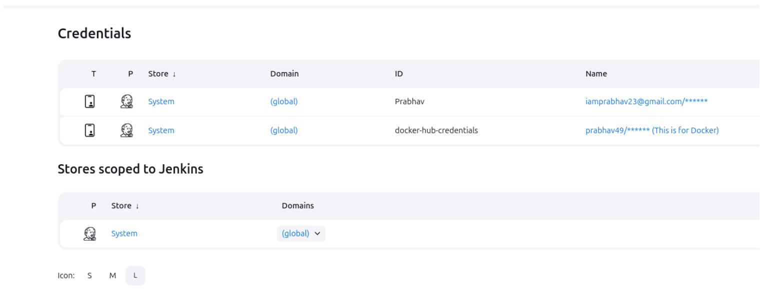
Security Secure Jenkins; define who is allowed to access/use the system.	**Credentials** Configure credentials

Status Information

System Information Displays various environmental information to assist trouble-shooting.	**System Log** System log captures output from `java.util.logging` output related to Jenkins.

Figure 5: Click on Credentials

- Go to Stores scoped to Jenkins and click on global



Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	Prabhav	iamprabhav23@gmail.com/******
		System	(global)	docker-hub-credentials	prabhav49/****** (This is for Docker)

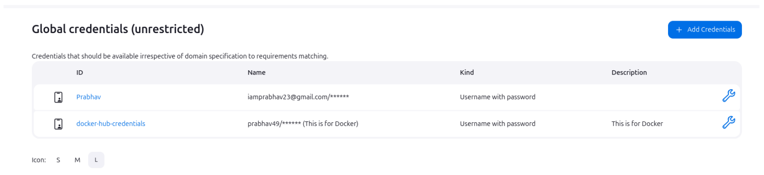
Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Icon: S M L

Figure 6: Click on global

- Click on ”Add Credentials”



Global credentials (unrestricted)

+ Add Credentials

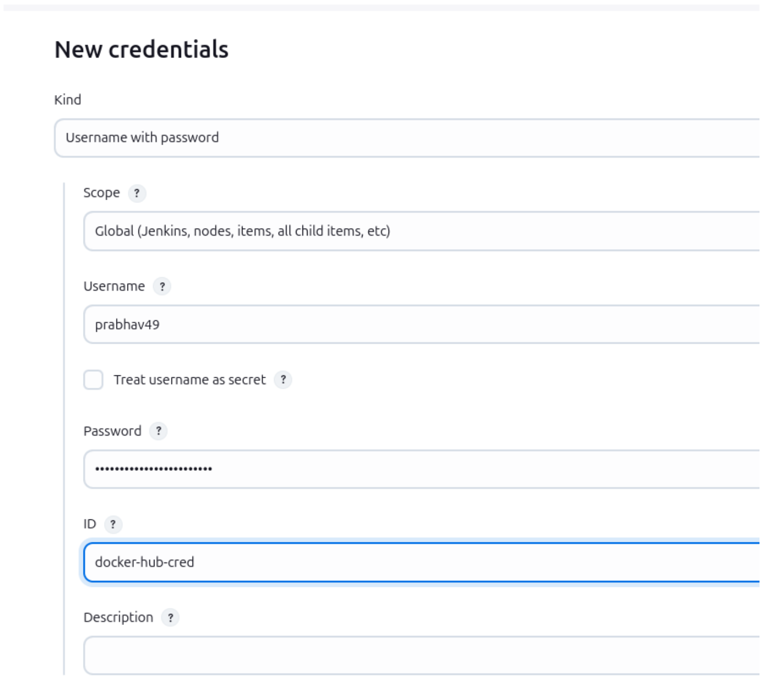
Credentials that should be available irrespective of domain specification to requirements matching.

	ID	Name	Kind	Description
	Prabhav	iamprabhav23@gmail.com/******	Username with password	
	docker-hub-credentials	prabhav49/****** (This is for Docker)	Username with password	This is for Docker

Icon: S M L

Figure 7: Add Credentials

- Now you can see your added Credentials



New credentials

Kind
Username with password

Scope
Global (Jenkins, nodes, items, all child items, etc)

Username
prabhav49

Treat username as secret

Password
••••••••••••••••••••••••

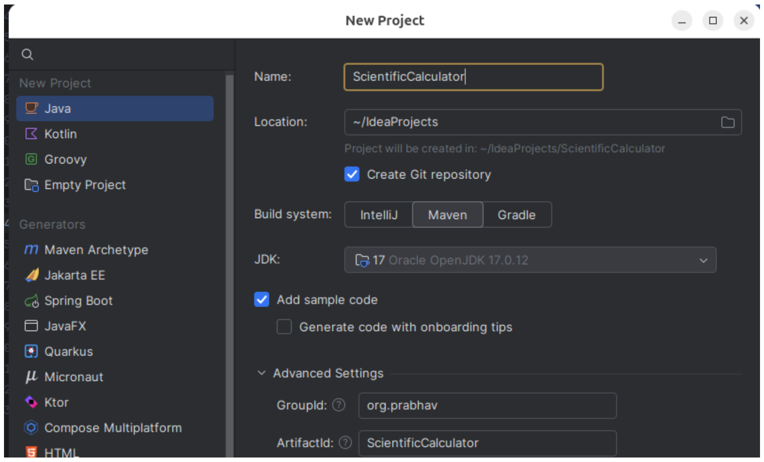
ID
docker-hub-cred

Description

Figure 8: Check your added credentials

7 Create Calculator console Application using Java

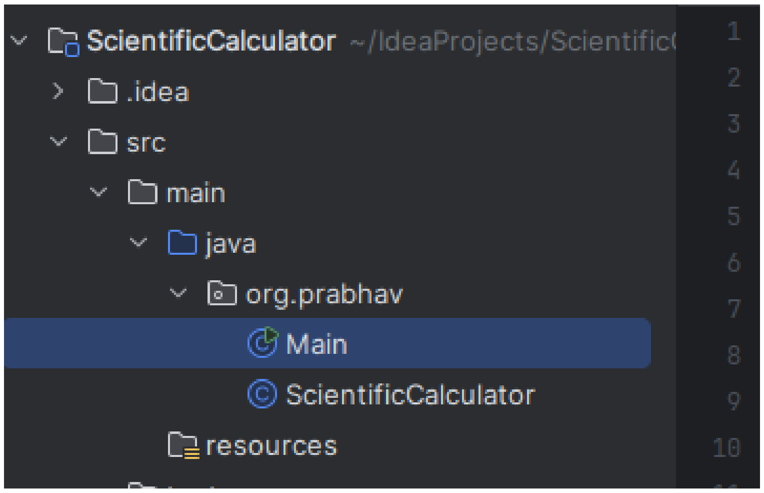
- First Create New Project
- Choose Maven for build system



Name: ScientificCalculator
Location: ~/IdeaProjects
Project will be created in: ~/IdeaProjects/ScientificCalculator
Create Git repository
Build system: Maven
JDK: 17 Oracle OpenJDK 17.0.12
Add sample code
Generate code with onboarding tips
GroupId: org.prabhav
ArtifactId: ScientificCalculator

Figure 9: Maven Build System

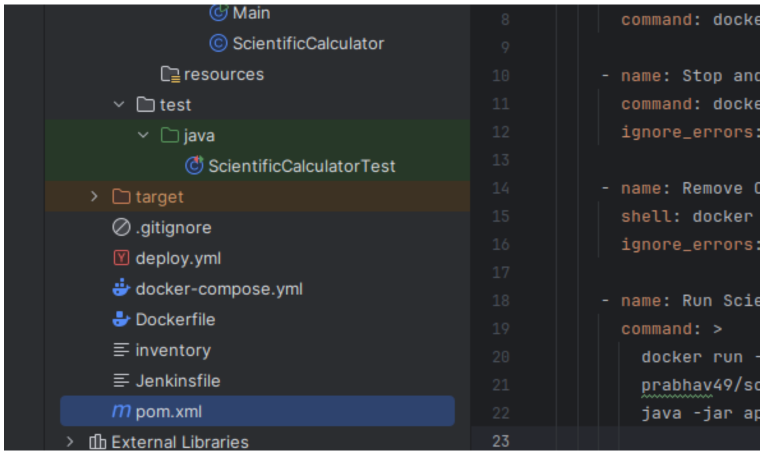
- Then go to: src → main → java → org → example → Main.java



```
ScientificCalculator ~/IdeaProjects/Scientific
  .idea
  src
    main
      java
        org.prabhav
          Main
          ScientificCalculator
      resources
```

Figure 10: Main Java class

- Then write your calculator logic code
- Then go to pom.xml file



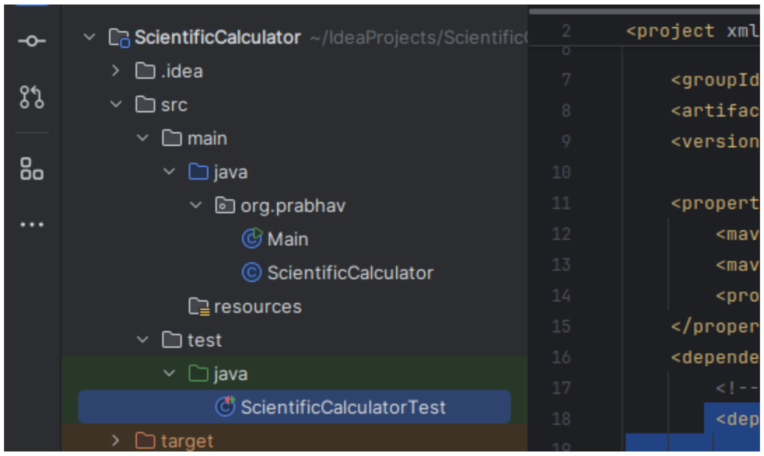
```
    command: docke

  - name: Stop anc
    command: docke
    ignore_errors:

  - name: Remove (
    shell: docker
    ignore_errors:

  - name: Run Scie
    command: >
      docker run -
      prabhav49/sc
      java -jar ap
```

Figure 11: pom.xml file

- Paste JUnit dependency under dependencies

```
         <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
```

- Then go to test folder then write testing logic for your main code



```
<project xml

    <groupId
    <artifac
    <version

    <propert
        <mav
        <mav
        <pro
    </proper
    <depende
        <!--
        <dep
```

Figure 12: Test file

- Then run below command to build your code, this will create .jar file under target folder.

```
        mvn clean package
```

- Then run below command to test your code

```
        mvn test
```

8 Continuous Integration and Deployment (CI/CD) Pipeline

8.1 Jenkins Pipeline Configuration: Jenkinsfile

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'prabhav49/scientific-calculator:latest'
        ANSIBLE_PLAYBOOK = 'deploy.yml'
    }
    stages {
        stage('Clone Repository') {
            steps {
                git
   'https://github.com/Prabhav49/ScientificCalculator.git'
            }
        }

        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }

        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                sh 'docker build -t $DOCKER_IMAGE .'
            }
        }

        stage('Push Docker Image') {
            steps {

   withCredentials([usernamePassword(credentialsId:
   'docker-hub-credentials', usernameVariable: 'DOCKER_USER',
   passwordVariable: 'DOCKER_PASS')]) {
                    sh 'echo "$DOCKER_PASS" | docker login -u
   "$DOCKER_USER" --password-stdin'
                    sh 'docker push $DOCKER_IMAGE'
                }
            }
        }

//         stage('Deploy with Ansible') {
//                     steps {
//                         script {
//                             def deployStatus = sh(script:
   'ansible-playbook --connection=local --become deploy.yml',
   returnStatus: true)
//                             if (deployStatus != 0) {
//                                 error "Deployment failed!"
//                             }
//                         }
//                     }
//                 }

        stage('Run Ansible Playbook') {
                    steps {
                        script {
                            ansiblePlaybook(
                                playbook: 'deploy.yml',
                                inventory: 'inventory'
                            )
                        }
                    }
                }
```

```
    }
    post {
        success {
            emailext(
                to: 'iam49smith@gmail.com',
                subject: "SUCCESS: ${env.JOB_NAME}
   #${env.BUILD_NUMBER}",
                body: """<p>The build and deployment were
   <b>successful!</b></p>
                         <p>Check the build details: <a
   href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>"""
            )
        }
        failure {
            emailext(
                to: 'iam49smith@gmail.com',
                subject: "FAILURE: ${env.JOB_NAME}
   #${env.BUILD_NUMBER}",
                body: """<p>The build or deployment
   <b>failed!</b></p>
                         <p>Check the build details: <a
   href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>"""
            )
        }
        always {
            cleanWs()
        }
    }
}
```

8.2 Containerization Setup: Dockerfile

```
FROM openjdk:17-jdk-slim

# Set the working directory inside the container
WORKDIR /app

COPY
   target/ScientificCalculator-1.0-SNAPSHOT-jar-with-dependencies.jar
   app.jar

ENTRYPOINT ["sh", "-c", "sleep infinity"]
CMD ["java", "-jar", "app.jar"]
```

8.3 Service Orchestration: docker-compose.yml

```
version: '3.8'

services:
  calculator:
    image: scientific-calculator
    container_name: scientific_calculator
    stdin_open: true  # Keep STDIN open for interactive input
   (useful if needed later)
    tty: true
    entrypoint: ["java", "-jar", "app.jar"]
```

8.4 Configuration Management: Ansible Playbook

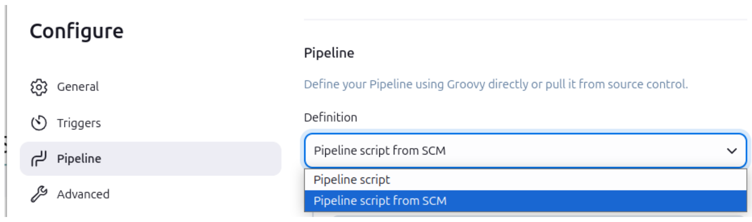
```
- name: Deploy Scientific Calculator
  hosts: localhost
  remote_user: prabhav
  become: false
  tasks:

    - name: Pull Latest Docker Image
      command: docker pull
   prabhav49/scientific-calculator:latest

    - name: Stop and Remove Existing Container
      command: docker rm -f scientific_calculator
      ignore_errors: yes

    - name: Remove Old Docker Images (Optional)
      shell: docker images -q prabhav49/scientific-calculator |
   xargs -r docker rmi -f
      ignore_errors: yes

    - name: Run Scientific Calculator Container
      command: >
        docker run -d --name scientific_calculator
        prabhav49/scientific-calculator:latest
        java -jar app.jar 1 4 2
```

8.5 Inventory Configuration: inventory

```
ansible_host=localhost
ansible_user=<name of your server>
ansible_ssh_pass=<password of your system>
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

9 Creating Pipeline Job and configuration

- Go to Jenkins and go to the project that you created while setuping the project.
- Go to "configure"
- Go to "Pipeline" section then "Definition" then select "Pipeline script from SCM"



Pipeline

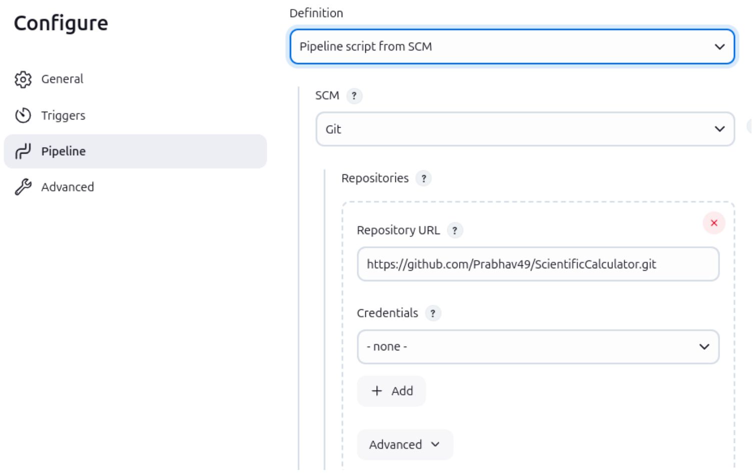
Define your Pipeline using Groovy directly or pull it from source control.

Definition: Pipeline script from SCM

Pipeline script
Pipeline script from SCM

Figure 13: Select Pipeline script from SCM

- Select GIT in SCM and enter the repository URL in "Repository URL"



Definition: Pipeline script from SCM

SCM: Git

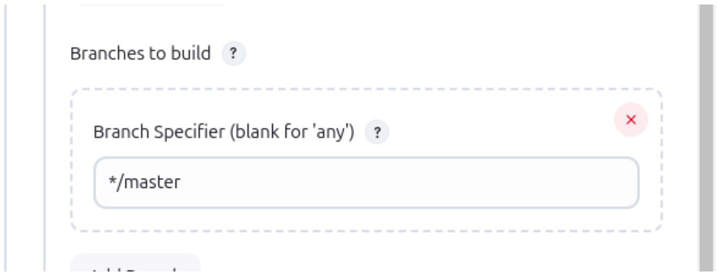
Repositories

Repository URL: https://github.com/Prabhav49/ScientificCalculator.git

Credentials: - none -

Figure 14: Add Repository URL in SCM

- Change branch according to your repository, default branch



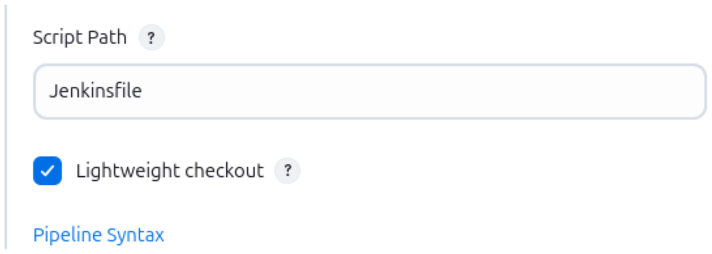
Branches to build

Branch Specifier (blank for 'any')

*/master

Figure 15: Change Branch of your repository

- In script path enter ”Jenkinsfile” (file name of your pipeline script created earlier).



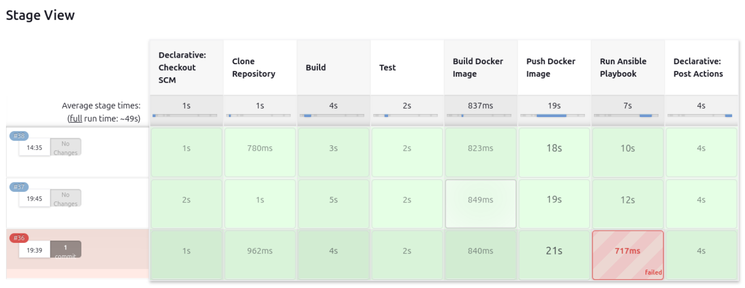
Script Path

Jenkinsfile

☑ Lightweight checkout

Figure 16: Add script file in path

- Click on save
- Go back and click on ”Build Now then you can see your pipeline stage view”



Stage View

	Declarative: Checkout SCM	Clone Repository	Build	Test	Build Docker Image	Push Docker Image	Run Ansible Playbook	Declarative: Post Actions
Average stage times: (full run time: ~49s)	1s	1s	4s	2s	837ms	19s	7s	4s
#38 14:35 No Changes	1s	780ms	3s	2s	823ms	18s	10s	4s
#37 19:45 No Changes	2s	1s	5s	2s	849ms	19s	12s	4s
#36 19:39 1 commit	1s	962ms	4s	2s	840ms	21s	717ms failed	4s

Figure 17: Stage view of pipeline

- If any error occurs, then you can see that error on console output.

10 Execution and Results

- After successful deployment, the application runs in a Docker container, also it is successfully pushed into your dockerhub profile.

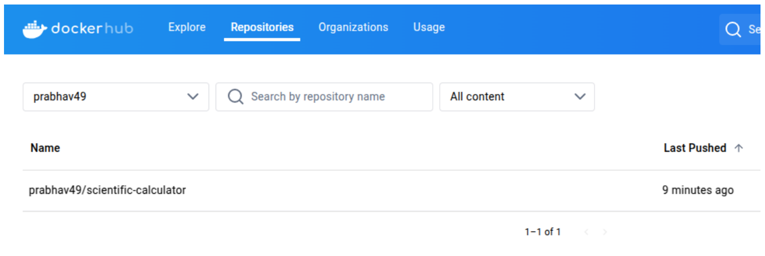


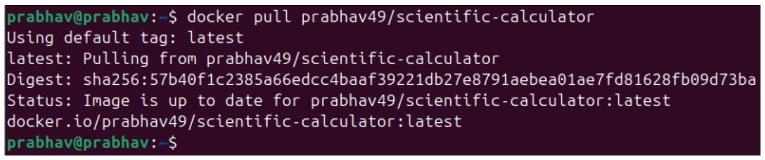
Figure 18: Image pushed to your dockerhub account

- To verify this on your terminal type below command:

```
sudo docker images
```

- Pull the Image using the command: (prabhav49 is the username of the dockerhub profile and scientific-calculator is name of my image)

```
sudo docker pull prabhav49/scientific-calculator
```



```
prabhav@prabhav:~$ docker pull prabhav49/scientific-calculator
Using default tag: latest
latest: Pulling from prabhav49/scientific-calculator
Digest: sha256:57b40f1c2385a66edcc4baaf39221db27e8791aebea01ae7fd81628fb09d73ba
Status: Image is up to date for prabhav49/scientific-calculator:latest
docker.io/prabhav49/scientific-calculator:latest
prabhav@prabhav:~$
```

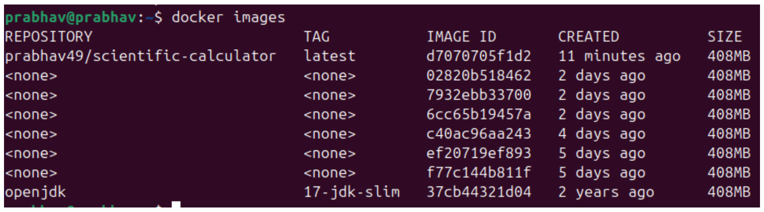
Figure 19: Pulling Docker image

- To see image is successfully pulled type below command

```
sudo docker images
```

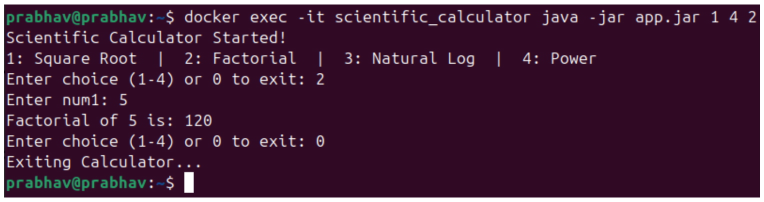
- To run the image type below command

```
docker exec -it scientific_calculator java -jar
app.jar 1 4 2
```



```
prabhav@prabhav:~$ docker images
REPOSITORY                          TAG          IMAGE ID       CREATED          SIZE
prabhav49/scientific-calculator     latest       d7070705f1d2   11 minutes ago   408MB
<none>                              <none>       02820b518462   2 days ago       408MB
<none>                              <none>       7932ebb33700   2 days ago       408MB
<none>                              <none>       6cc65b19457a   2 days ago       408MB
<none>                              <none>       c40ac96aa243   4 days ago       408MB
<none>                              <none>       ef20719ef893   5 days ago       408MB
<none>                              <none>       f77c144b811f   5 days ago       408MB
openjdk                             17-jdk-slim  37cb44321d04   2 years ago      408MB
```

Figure 20: Check images is pulled



```
prabhav@prabhav:~$ docker exec -it scientific_calculator java -jar app.jar 1 4 2
Scientific Calculator Started!
1: Square Root  |  2: Factorial  |  3: Natural Log  |  4: Power
Enter choice (1-4) or 0 to exit: 2
Enter num1: 5
Factorial of 5 is: 120
Enter choice (1-4) or 0 to exit: 0
Exiting Calculator...
prabhav@prabhav:~$
```

Figure 21: Running the Image

- To check whether the image is running and to check logs:

```
docker ps -a
docker logs calculator
```

11 Conclusion

The Scientific Calculator project successfully demonstrates DevOps principles, including automation, containerization, and continuous deployment.

12 References

References

[1] GitHub Repository, `https://github.com/Prabhav49/ScientificCalculator`

[2] Docker Hub Repository, `https://hub.docker.com/r/prabhav49/scientific-calculator`