SPE
Mini Project Report
Scientific Calculator with DevOps Pipeline

Name: Prateek Rath
Roll Number: IMT2022017

Overview

This report explains the implementation of a Scientific Calculator command line application and the creation of a DevOps pipeline for its continuous development, integration, and deployment. The project uses Github for source control management, JUnit for testing the Java code, Gradle as the build tool, Jenkins for continuous integration, Docker for containerization, DockerHub for artifact management, and Ansible for configuration management and deployment on the 'localhost' machine. The calculator program supports square root ($\sqrt{x}$), factorial ($!x$), natural logarithm ($\ln(x)$), and power (x^b). As much as possible, steps, configurations, commands, and results are documented with explanations and supporting screenshots. **The links to the GitHub repository and DockerHub registry are shown below.**

Project Links

Public GitHub Repository Link

`https://github.com/Prateek-Rath/spe_miniproj_IMT2022017.git`

Docker Hub Repository Link

`https://hub.docker.com/r/pratster20/miniproj_image1`

Introduction

This project focuses on building a simple Scientific Calculator application and implementing a comprehensive DevOps toolchain to automate its software production lifecycle. The integration of application development and automated infrastructure management is central to modern software engineering practices.

What is DevOps?

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the systems development life cycle and provide continuous delivery with high software quality. It is a cultural, automation, and platform-based approach that aims to streamline the process from code creation to deployment in production.

Why use DevOps?

The adoption of DevOps practices offers several significant benefits.

- **Faster Delivery Cycle**: Automation in the pipeline (CI/CD) drastically reduces the time from commit to deployment.
- **Increased Reliability**: Automated testing and configuration management ensure consistency and reduce human error.

- **Improved Collaboration**: By breaking down silos between the Dev and Ops teams, communication is enhanced, leading to faster issue resolution.
- **Scalability**: Tools such as Docker and Ansible allow for easy, repeatable, and scalable deployments across various environments.

Implementation Details

The application was developed using Java as a command-line application. The main code is present in App.java while the calculator functions are in the Calculator.java file. The rest is generated by running 'gradle init'. The test file is called CalculatorTest.java.

Application Demonstration

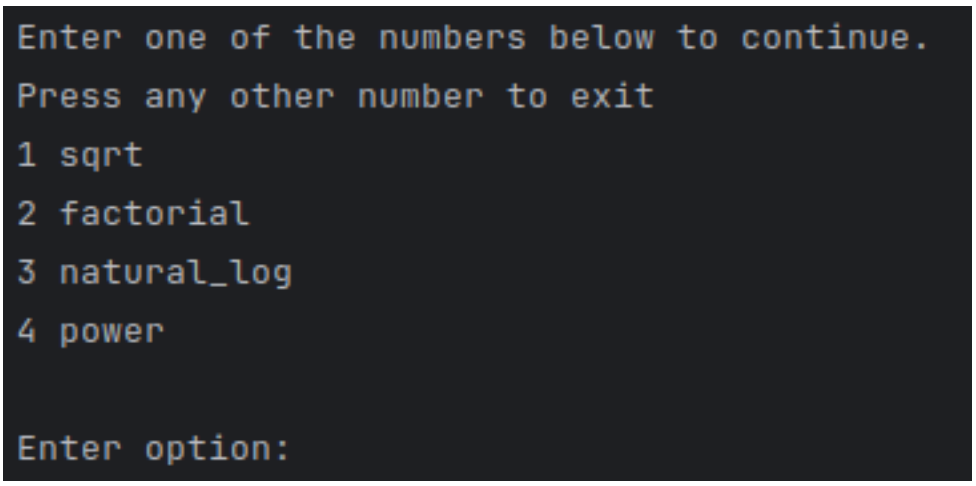


Figure 1: Screenshot of Scientific Calculator Application Main Menu

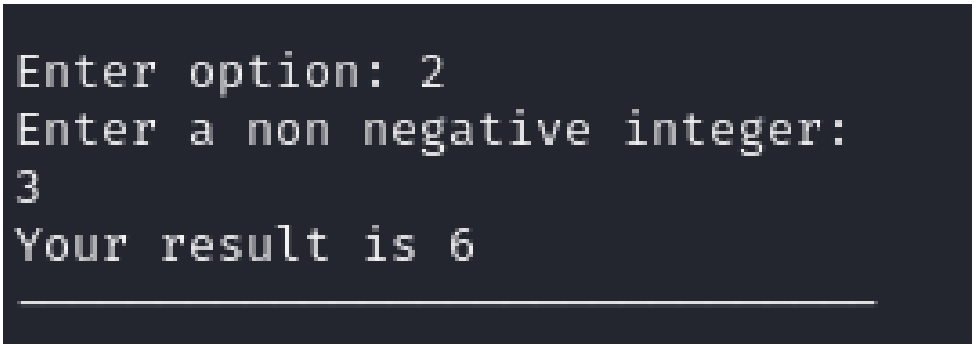


Figure 2: Screenshot of Scientific Calculator - Factorial Operation

Source Control Management (SCM)

Tool Used: GitHub

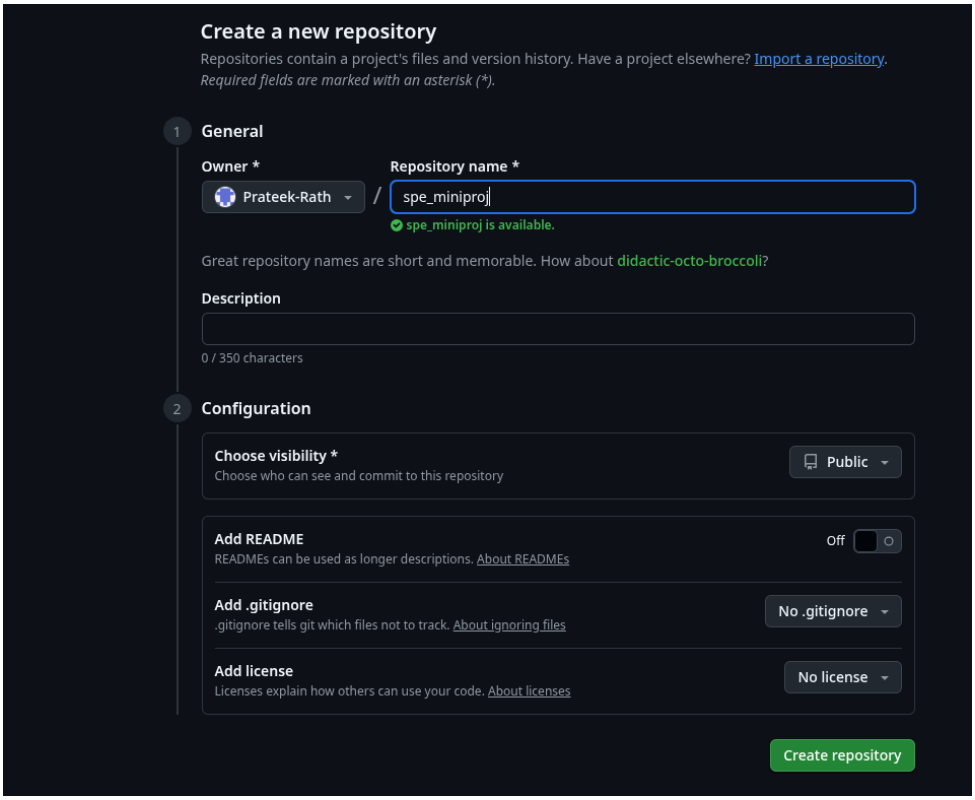
GitHub is a web-based platform for version control using Git. It allows multiple developers to collaborate on a project, track changes, and manage different versions of the source

code.

Setup and Configuration

The project repository was created on GitHub.

Figure 3: Screenshot of Gtihub Repo Creation



The local repository was initialized and linked using the following commands after the source code(majority) was written:

Listing 1: Initial Git Commands

```
git init
git add .
git commit -m "intial commit"
git remote add origin "https://github.com/Prateek-Rath/miniproj_IMT2022017.git"
git push
```

Some code was also added to build.gradle. Build.gradle is a config file that tells gradle(a build automation tool) how exactly it needs to compile the java project. More detail is provided in the build section.

Testing

Tool Used: JUnit

JUnit, or more specifically, JUnit Jupiter, is a Java unit testing framework used to ensure that individual components (functions) of the calculator work as intended. Automated

tests are critical for catching regressions early in the development cycle.

Test Setup and Commands

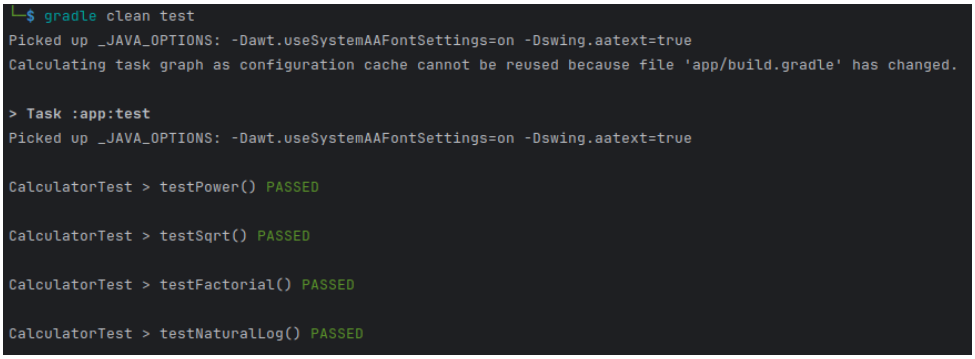
Unit tests were written for all four scientific functions.

The tests are executed via the command:

Listing 2: Test Execution Command

```
gradle test
```

Figure 4: Screenshot of Test Results



Build

Tool Used: Gradle

Gradle is a build automation tool used to compile the source code, run tests, and package the application into a deployable artifact (e.g., a JAR file for Java).

Build Configuration and Commands

In any project created via the gradle init command, gradle generates some default files along with a standard directory structure. It also creates a special file called build.gradle which tells gradle how it needs to compile the project.

The main command used for the build process is:

Listing 3: Build Command

```
gradle clean build
```

Then we can run the project using:

Listing 4: Build Command

```
gradle run
```

Continuous Integration (CI)

Tool Used: Jenkins

Jenkins is an open-source automation server that orchestrates the entire CI/CD pipeline. It pulls code from SCM, runs tests, triggers the build, and passes the artifact to the next stages.

Setup and Configuration

In Jenkins, we create a new pipeline project and configure it to use a webhook to trigger a build any time code is pushed into the github repository. The Jenkins file contains the pipeline script which is detailed later.

We add the DockerHub credentials to Jenkins. We need these to be able to push images onto DockerHub.

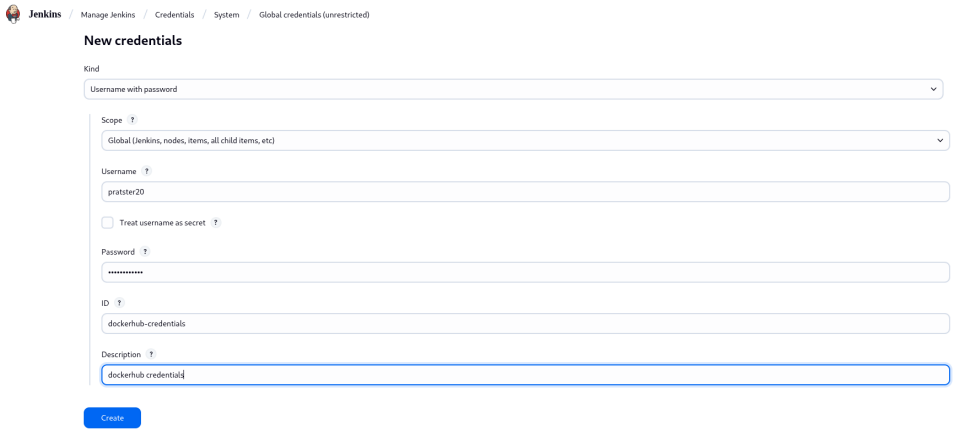


Figure 5: Screenshot of Jenkins Credential Management(Add dockerhub credentials)

We then create a Jenkins pipeline project that uses a Jenkinsfile located in the github repository.

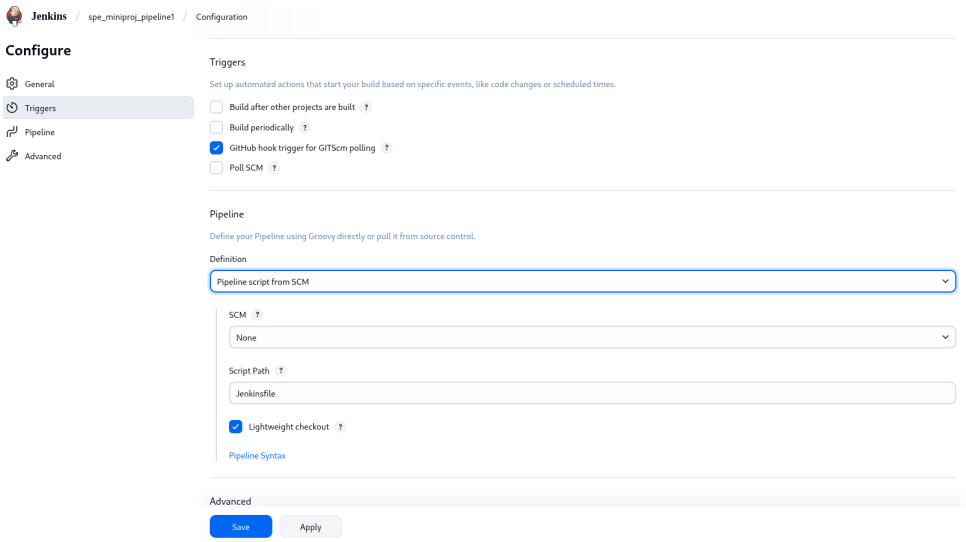


Figure 6: Creating a jenkins pipeline project

Jenkins Pipeline Script (`Jenkinsfile`)

The pipeline is defined as code in a `Jenkinsfile`.

We need a public IP address (or a public URL like the one provided by ngrok) for your local Jenkins instance, and we need to add this URL to a GitHub webhook because GitHub cannot directly communicate with a server running only on localhost. So we use ngrok to obtain an ip address using the command 'ngrok http 8080', since jenkins is running on port 8080 on localhost.

Listing 5: ngrok command

```
ngrok http 8080
```

We add the public ip that ngrok gives us to our GitHub repository's webhook. Screenshots are shown below.

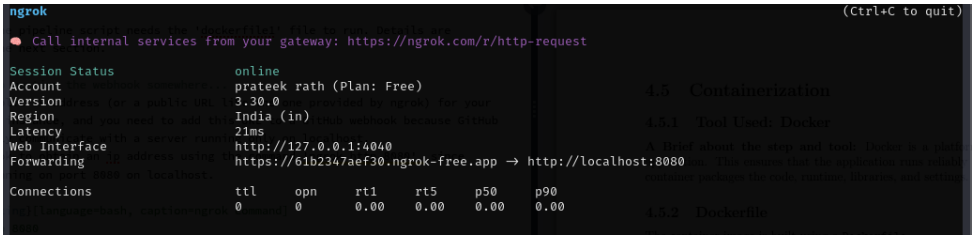


Figure 7: ngrok gives a public ip for jenkins server

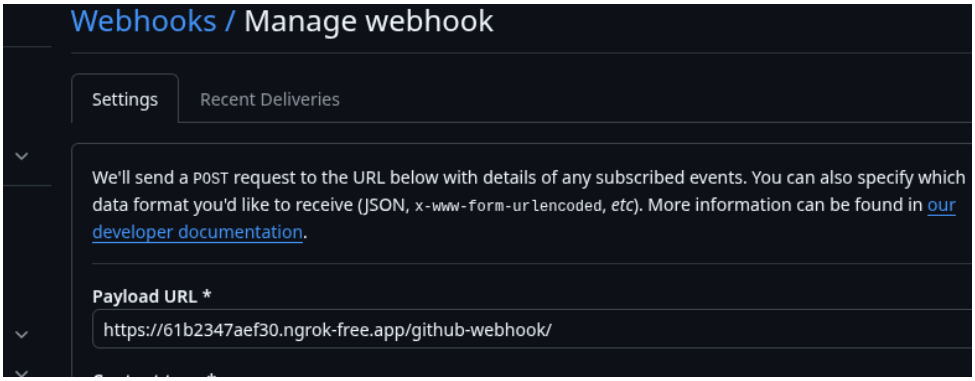


Figure 8: GitHub webhook

Whenever a push is made to the GitHub repository, it tells Jenkins via the GitHub Webhook to start executing the pipeline. The pipeline script consists of the following stages:

1. **Checkout**: Jenkins pulls the source code from the GitHub repository.
2. **Test**: Jenkins runs the tests on the new source code. It uses the Gradle wrapper to do this.
3. **Build jar**: Next, Jenkins cleans the class files and recompiles the java source code to produce a single jar executable.
4. **Build Docker Image**: We then pack the jar file into a docker image, instructions to build the image are provided in the Dockerfile. Refer to that section for details.

5. **Push Docker image**: Next, we login, push the image onto DockerHub and then logout. This image can later be pulled by another host and used to run the container and the executable inside it.

6. **Deploy with Ansible**: We use the inventory file along with the playbook to deploy to localhost. Details are provided in the deployment section.

7. **Post Actions**: These actions are performed upon the success/failure of the Jenkins pipeline. Jenkins sends an email with the appropriate message using the 'emailext' extension. We also make Jenkins clean the workspace. This deletes everything in the workspace directory($WORKSPACE) and ensures a clean slate for the next build.

   Prevents leftover files (e.g., build artifacts, caches, logs) from affecting future builds

Jenkins Pipeline View

At runtime or after the run, this view helps us see which stage of the build we are currently in and thereby understand the progress of the project. A screenshot is shown below.

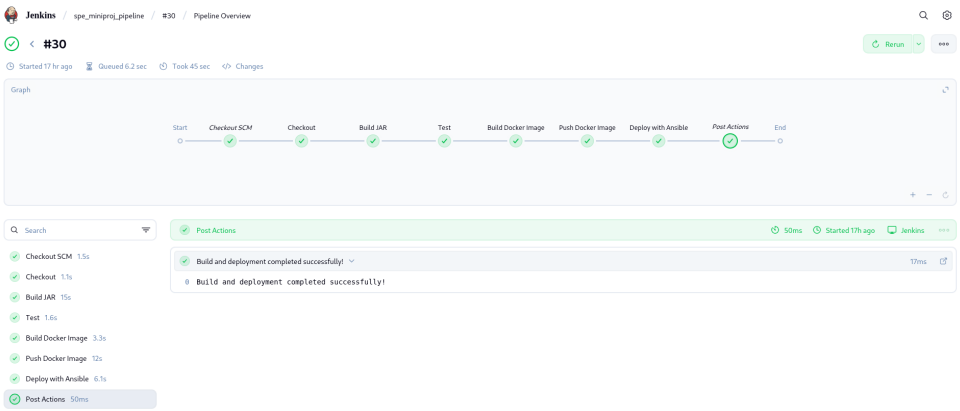


Figure 9: Jenkins Pipeline View

Containerization

Tool Used: Docker

Docker is a platform used to containerize the application. This ensures that the application runs reliably in any environment, as the container packages the code, runtime, libraries, and settings.

Dockerfile

The container image is built using a `Dockerfile`. Here, the Dockerfile contains the openjdk:21-jdk base image and the necessary commands to run and files to copy to build the container from the base image. It also has a single entry point command(java -jar app.jar) that is run when the container starts.

Listing 6: Docker Build Command

```
docker build -f dockerfile1 -t pratster20/miniproj_image1 .
```

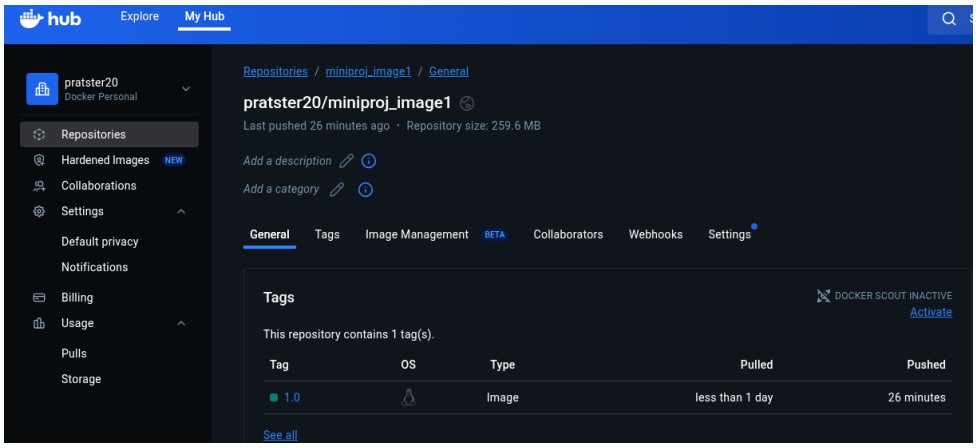
Push to Docker Hub

Tool Used: Docker Hub

Docker Hub is a cloud-based registry service that enables the sharing and management of Docker container images. The built image is pushed here to serve as the final artifact for deployment. Note that dockerhub credentials are stored as global credentials in Jenkins, allowing us to login, push an image, and then logout from DockerHub.

Setup and Commands

Figure 10: Screenshot of Docker Hub Repository



The push command is automatically executed by the Jenkins pipeline.

Listing 7: Docker Push Command

```
docker push pratster20/miniproj_image1:1.0
```

Deployment and Configuration Management

Tool Used: Ansible

Ansible is an IT automation engine that uses SSH to deploy applications and manage configurations on managed hosts. It will be used to pull the Docker image from Docker Hub and run it on the target machine (here it is localhost).

Ansible Configuration and Commands

Our inventory file is called hosts here and is located in '/app/infra'. It contains relevant information about the hosts on which we want to deploy. Since here we are only interested

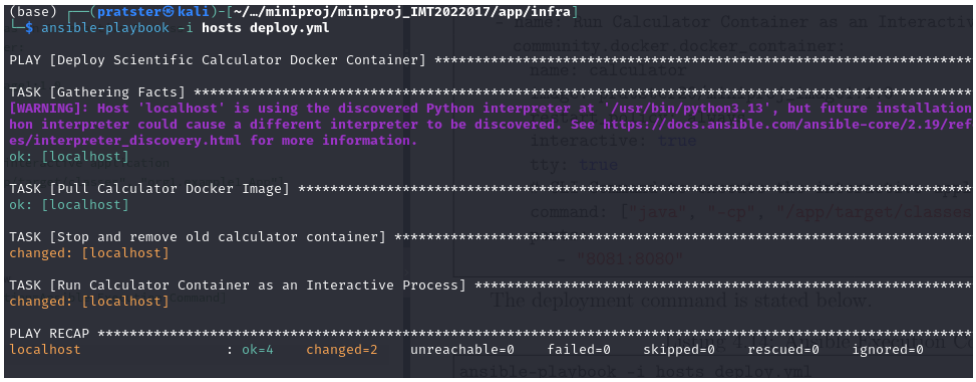
in deploying to localhost, we just provide that information. We don't need ssh for this, only a local connection.

The deployment instructions are present in the deploy.yml file. This file is also called a playbook file. It uses declarative yaml syntax to specify what the localhost machine should do. Here, our script asks it to pull the docker image, stop and remove any existing or old images and then starts the container(in the background) as an interactive process. We can then later attach to the same container and interact with the calculator program. The deployment command is stated below. We don't have to explicitly use it as the jenkins pipeline takes care of it.

Listing 8: Ansible Execution Command

```
ansible-playbook -i hosts deploy.yml
```

Figure 11: Screenshot of Successful Ansible Playbook Run



Once the above steps are complete, we can just attack to the already started docker container and run the application using the command below. Since our entry point command already runs the jar file, running docker start suffices and gives us the input window to start with.

Figure 12: Screenshot of Deployed Application running via Docker

Conclusion

This project successfully implemented a scientific calculator application and established a robust end-to-end DevOps pipeline, and helped build a decent understanding of key DevOps tools and methodologies. The automated pipeline, from code commit to containerized deployment, ensures rapid and reliable software delivery.