

# CS 816 : Software Production Engineering - Mini Project Report

## Scientific Calculator with DevOps

**Name:** Pranav Kulkarni

**Roll Number:** IMT2022053

GitHub Repo URL : [https://github.com/py-xis/SPE\\_Mini\\_Project\\_IMT2022053](https://github.com/py-xis/SPE_Mini_Project_IMT2022053)

DockerHub Repo URL :

<https://hub.docker.com/repository/docker/pyxis2004/scicalc/general>

---

## 0. What and Why DevOps ?

### What is DevOps:

DevOps is a combination of **Development (Dev)** and **Operations (Ops)** — a set of cultural philosophies, practices, and tools that aim to **bridge the gap between software development and IT operations**.

It emphasises **automation, collaboration, and continuous feedback** to deliver software faster, more reliably, and with higher quality.

### Why DevOps:

Traditionally, developers wrote code and handed it over to operations teams for deployment — often causing delays, configuration mismatches, and communication gaps.

DevOps solves these challenges by:

- **Automating** repetitive tasks like builds, testing, and deployment.
- **Integrating** all stages of the software lifecycle into a single continuous flow (CI/CD).
- **Improving reliability** through consistent, containerised environments.

- **Reducing time to market** by ensuring every code change can be built, tested, and deployed automatically.
- **Encouraging collaboration** between developers, testers, and system administrators.

In short, **DevOps is not just a set of tools but a mindset** — it transforms how teams build, test, release, and maintain software by turning manual, isolated workflows into automated, collaborative pipelines.

## 1. Introduction

For my Software Production Engineering mini-project, I implemented a **Scientific Calculator** in **Go (Golang)** and automated its entire build, test, and deployment lifecycle using a **DevOps pipeline**.

The project uses the following tools:

- **GitHub** – for source code management.
- **Jenkins** – for continuous integration and pipeline automation.
- **Docker** – for containerisation.
- **DockerHub** – for storing and versioning images.
- **Ansible** – for automated deployment.

The calculator performs four mathematical operations:

1. Square root ( $\sqrt{x}$ )
2. Factorial ( $!x$ )
3. Natural logarithm ( $\ln x$ )
4. Power ( $x^b$ )

The aim was to integrate all DevOps concepts—automation, reproducibility, and continuous delivery—into one seamless workflow.

---

## 2. Beginning the Project

I began from a **local working directory** on my local machine where I had implemented a basic Go scaffold for a menu-driven calculator.

After verifying that the code worked locally, I decided to take it through a complete CI/CD workflow.

I initialised a Git repository locally and connected it to a new public repository on GitHub:

```
pranav_kulkarni@Pranav's-MacBook-Air ~ % tree
.
├── ansible
│   └── deploy.yml
│   └── inventory.ini
├── cmd
│   └── server
│       └── main.go
├── Dockerfile
└── go.mod
├── internal
│   ├── fact
│   ├── ln
│   └── pow
│       └── sqrt
│           └── sqrt_test.go
│           └── sqrt.go
└── Jenkinsfile
SPE - Mini Project Instructions.pdf

9 directories, 9 files
```

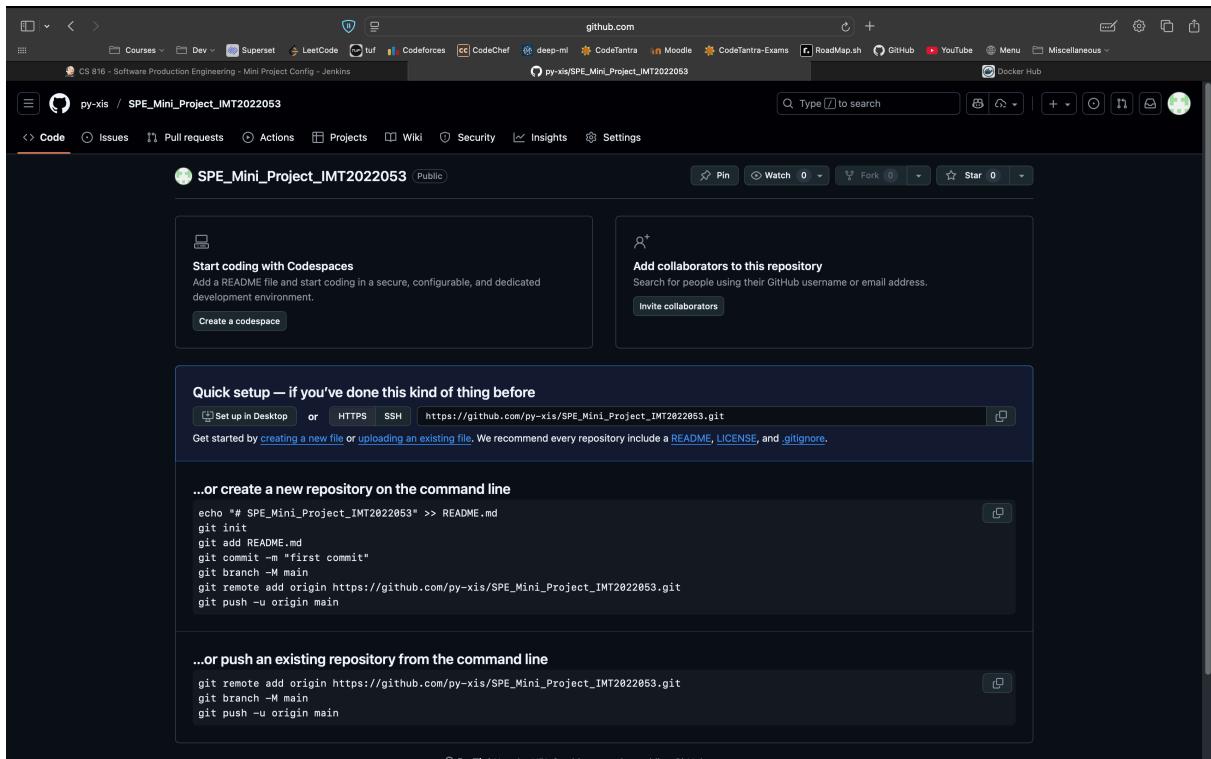
Initial Directory Structure

```
.
├── ansible
│   └── deploy.yml
│   └── inventory.ini
├── cmd
│   └── server
│       └── main.go
├── Dockerfile
└── go.mod
├── internal
│   ├── fact
│   │   ├── fact_test.go
│   │   └── fact.go
│   ├── ln
│   │   ├── ln_test.go
│   │   └── ln.go
│   └── pow
│       ├── pow_test.go
│       └── pow.go
└── sqrt
    └── sqrt_test.go
    └── sqrt.go
└── Jenkinsfile
└── scicalc
SPE - Mini Project Instructions.pdf
```

Final Directory Structure

```
• pranav_kulkarni@Pranav's-MacBook-Air ~ % git init
Initialized empty Git repository in /Users/pranav_kulkarni/Desktop/SPE/Projects/Mini Project/.git/
```

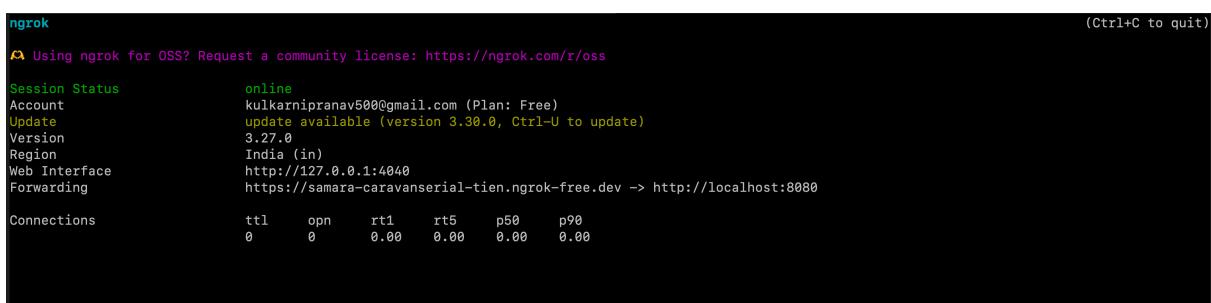
Initialising Git Repo



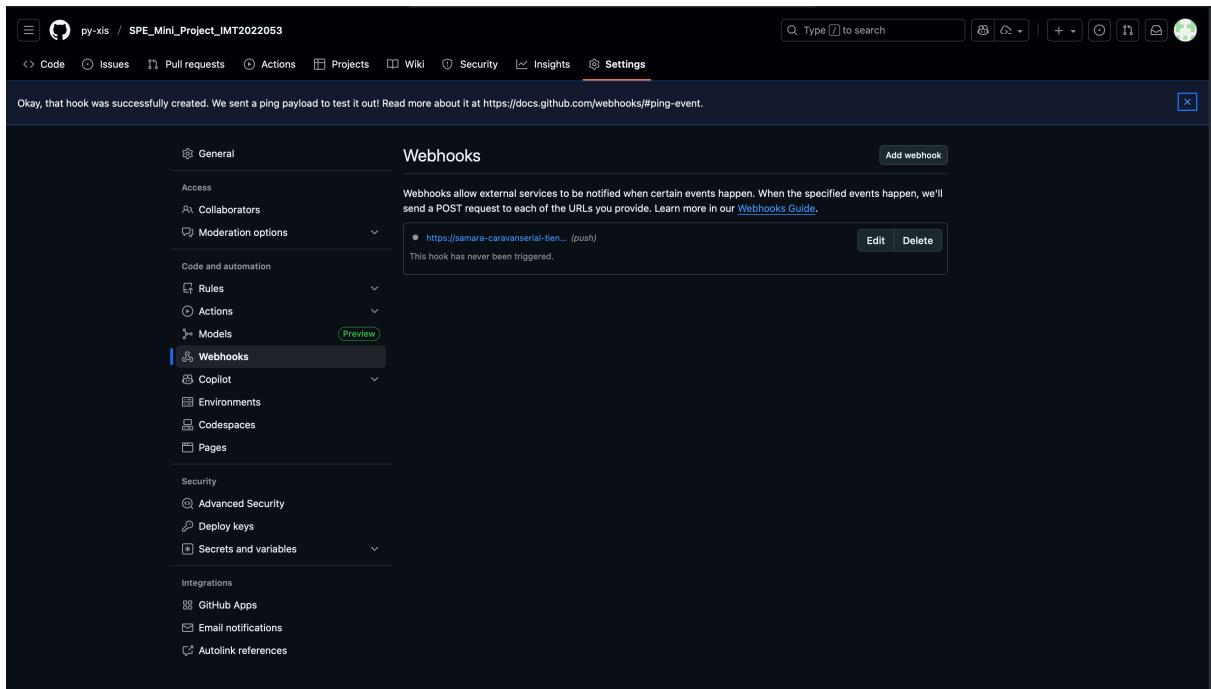
### Creating the GitHub Repository

To make the process completely automatic, I configured a **GitHub webhook**.

Since Jenkins was running locally, I exposed it using **ngrok**.



### Setting up ngrok



Configuring the WebHook for the Repository

I added the ngrok-generated static HTTPS URL under GitHub → Repository Settings → WebHooks.

Now, every time I committed and pushed changes to GitHub:

1. The webhook triggered Jenkins automatically.
2. Jenkins pulled the latest commit and ran the entire pipeline.
3. The new image was built, tested, pushed, and deployed automatically.

From that point, a single git push updated the running calculator without any manual step — a complete CI/CD setup.

### 3. Setting Up Jenkins Job (Continuous Integration)

I installed **Jenkins LTS** and started it locally at port 8080.

After installation, I created a **Pipeline job** titled *SPE\_Mini\_Project\_IMT2022053*.

The screenshot shows the Jenkins job configuration page for 'CS 816 - Software Production Engineering - Mini Project'. In the 'Configure' section, the 'Triggers' tab is selected. Under 'Triggers', there is a list of options: 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling' (which is checked), 'Poll SCM', and 'Trigger builds remotely (e.g., from scripts)'. A note above the list says: 'Set up automated actions that start your build based on specific events, like code changes or scheduled times.'

Configuring the job to trigger whenever a commit is done to the repository

The screenshot shows the Jenkins job configuration page for 'CS 816 - Software Production Engineering - Mini Project'. In the 'Configure' section, the 'Pipeline' tab is selected. Under 'Pipeline', it says 'Define your Pipeline using Groovy directly or pull it from source control.' Below this, a dropdown menu shows 'Pipeline script from SCM'. A 'SCM' section follows, with 'Git' selected. It contains fields for 'Repository URL' (set to 'https://github.com/py-xis/SPE\_Mini\_Project\_IMT2022053.git'), 'Credentials' (set to '- none -'), and 'Advanced' settings. Below the SCM section is 'Branches to build', with 'Branch Specifier (blank for "any")' set to '\*/\*main'.

Adding the GitHub Repository URL, Branch of the Repository and Source of the Jenkinsfile

The screenshot shows the Jenkins job configuration page for 'CS 816 - Software Production Engineering - Mini Project'. In the 'Configure' section, the 'Pipeline' tab is selected. Under 'Pipeline Syntax', the 'Script Path' is set to 'Jenkinsfile'. Below this, there is a checkbox for 'Lightweight checkout'. At the bottom of the pipeline configuration, there is an 'Advanced' section with a dropdown menu set to 'Advanced'.

Mentioning the Name of the Pipeline Script i.e. Jenkinsfile

Description	Scope	Status	Source	Created	Last used	Expiration date
CS 816 - Software Production ...	Read & Write	Active	Manual	Sep 30, 2025 at 15:22:49	Oct 02, 2025 at 15:52:41	Dec 29, 2025 at 23:59

Personal Access Token for dockerhub

Personal Access Token Configured in Jenkins

I configured the project to do the following :

1. Trigger whenever a commit was pushed to Github. (GitHub hook trigger for GitSCM polling).
2. Use the pipeline script from SCM named Jenkinsfile present at the root of the repository in the main branch to orchestrate the build.
3. I created a personal access token with read/write permissions and configured them under the dockerhub-creds in Settings → Manage Jenkins → Credentials for pushing and pulling from dockerhub repository. This allowed Jenkins to automatically build and push Docker images securely without exposing any passwords.

## 4. Building the Pipeline

My Jenkinsfile defined multiple automated stages:

1. **Checkout** – Clones the repository from GitHub.
2. **Run Tests** – Executes Go test cases.
3. **Build Image** – Builds the Docker image.
4. **Push to DockerHub** – Pushes the image to my DockerHub repository.
5. **Deploy (Ansible)** – Deploys the container locally using the Ansible playbook.

At this point, I had a fully functional Jenkins pipeline integrated with GitHub and Docker and Ansible

## 5. Incremental Development and CI/CD Workflow

I did not push all my code at once.

Instead, I followed an **incremental CI/CD approach**, where every function was committed separately and automatically triggered a full pipeline run.

The screenshot shows a GitHub repository interface. At the top, there are navigation links: Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A search bar is located at the top right. Below the header, a section titled "Commits" is shown. A dropdown menu indicates the branch is "main". Filter options "All users" and "All time" are also present. A list of commits is displayed, all made by a user named "py-xis" last week. The commits are:

- added power functionality
- added natural log functionality
- Add factorial functionality
- Fixed the Jenkinsfile to correctly display the URL for the dockerhub image
- Fixed the Jenkinsfile to correctly display the URL for the dockerhub image
- First commit

Each commit has a commit hash (e.g., 43eef5a, d638573, 9bce61d, 153d5f0, fcbedd0, e708a3d) and a copy icon.

Git Commit History

The screenshot shows a Jenkins job dashboard. On the left, a sidebar lists Jenkins management options: Status, Changes, Build Now, Configure, Delete Pipeline, Stages, Rename, Pipeline Syntax, GitHub Hook Log, and Credentials. The main area is titled "CS 816 - Software Production Engineering - Mini Project". It displays the Jenkins Job for CS 816 - Software Production Engineering - Mini Project. The "Stages" section shows four parallel pipelines, each with a green checkmark indicating success. The stages are: Start, Checkout..., Checkout, Run tests..., Build imag..., Login & Pu..., Deploy local..., Post Actions, and End. The times for each stage are listed below. The pipelines are:

- Integrating Power Function into the Scientific Calculator (15:52 - 56s - 1 change)
- Integrating Natural Log Function into the Scientific Calculator (15:46 - 55s - 1 change)
- Integrating Factorial Function into the Scientific Calculator (15:39 - 49s - 1 change)
- Integrating Square Root Function into the Scientific Calculator (15:32 - 46s)

The times for the stages are approximately: Start (0s), Checkout... (1.8s, 2.0s, 1.5s, 1.4s), Checkout (1.4s, 1.6s, 1.5s, 1.4s), Run tests... (2.0s, 6.5s, 5.4s, 3.4s), Build imag... (5.5s, 27s, 21s, 20s), Login & Pu... (27s, 25s, 21s, 20s), Deploy local... (10s, 11s, 10s, 12s), Post Actions (4.7s, 4.1s, 4.0s, 4.1s). The "Builds" section on the left shows a list of builds from October 2, 2025, with the same names as the pipelines above. The "Permalinks" section at the bottom provides links to the last build, stable build, successful build, and completed build for each pipeline.

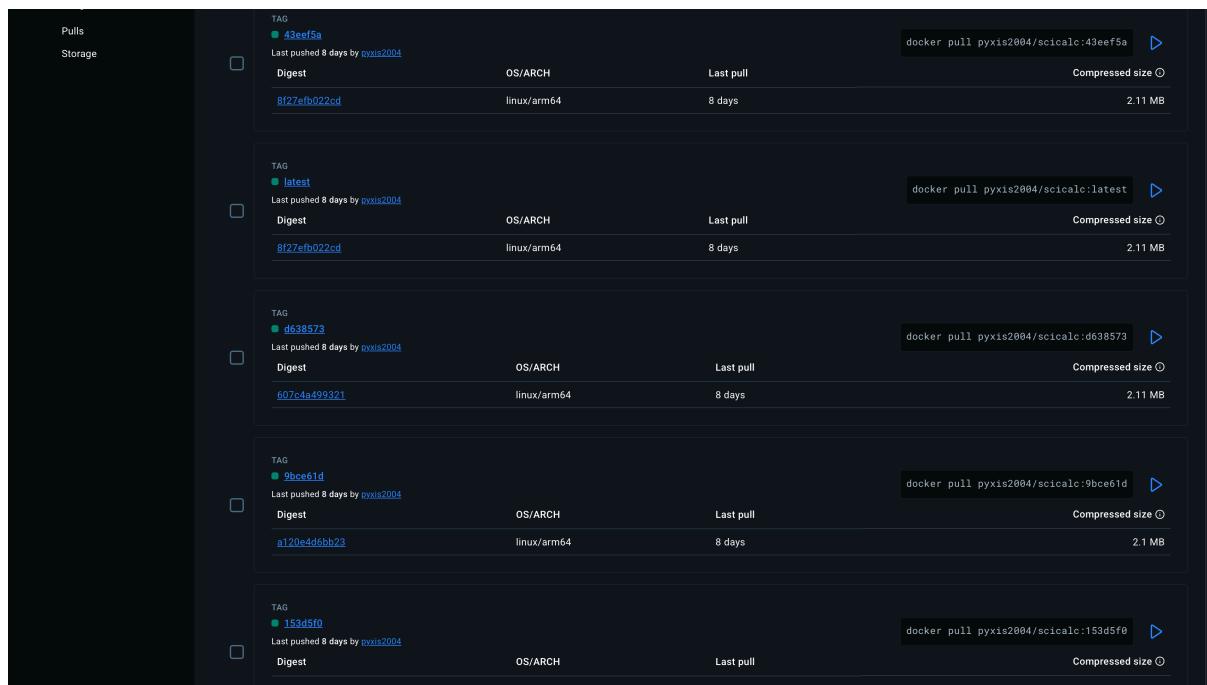
Build Triggered in Jenkins. Please note that I manually renamed the names of the build to make it more convenient to present.

My GitHub commit history clearly shows this step-by-step evolution:

- **First commit:** Jenkinsfile, Dockerfile, Ansible playbook and the square root function.
- **Fixed Jenkinsfile:** Adjusted DockerHub URL to correctly reference image path.
- **Add factorial functionality:** Introduced the factorial operation and pushed the commit. Jenkins immediately detected it via webhook, ran tests, built the new image, pushed it to DockerHub, and deployed the updated container.
- **Add natural log functionality:** Another commit, another automated pipeline run. The logs confirmed successful test execution and deployment.
- **Add power functionality:** Final mathematical function added, triggering one last pipeline cycle. The pipeline again executed all stages automatically and deployed the latest version successfully.

Each git commit lead to a commit ID. The first few characters of this commit ID was used to tag the docker image.

<https://hub.docker.com/r/pyxis2004/scicalc>



This incremental development style validated the entire **Continuous Integration → Continuous Deployment** concept in practice.

## 6. Running the Pipeline

Every time I pushed a commit, the pipeline started automatically:

## **Stage 1 – Checkout**

Jenkins / CS 816 - Software Production Engineering - Mini Project / Integrating Power Function into the Scientific Calculator / Pipeline Overview

Integrating Power Function into the Scientific Calculator

Started 7 days 20 hr ago | Queued 7.7 sec | Took 56 sec | Changes

Graph

Start → Checkout SCM → Checkout → Run tests (Go) → Build image (Local...) → Login & Push image → Deploy locally... → Post Actions → End

Checkout SCM

Checkout from version control

Checkout SCM 1.8s | Started 7d 20h ago | Jenkins

Checkout 1.4s

Run tests (Go) 2.0s

Build image (Local tag) 5.5s

Login & Push image 27s

Deploy locally (Ansible) 10s

Post Actions 4.1s

0 The recommended git tool is: NONE  
1 No credentials specified  
2 > git rev-parse --resolve-git-dir /Users/pranav\_kulkarni/.jenkins/workspace/CS 816 - Software Production Engineering - Mini Project/.git #  
timeout=10  
3 Fetching changes from the remote Git repository  
4 > git config remote.origin.url https://github.com/py-xis/SPE\_Mini\_Project\_IMT2022053.git # timeout=10  
5 Fetching upstream changes from https://github.com/py-xis/SPE\_Mini\_Project\_IMT2022053.git  
6 > git --version # timeout=10  
7 > git --version # 'git' version 2.39.5 (Apple Git-154)'  
8 > git fetch --tags --force --progress -- https://github.com/py-xis/SPE\_Mini\_Project\_IMT2022053.git +refs/heads/\*:refs/remotes/origin/\* #  
timeout=10  
9 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10  
10 Checking out Revision 43eef5aa2438a4b845c1e869587e37b4af9ee13 (refs/remotes/origin/main)  
11 > git config core.sparsecheckout # timeout=10  
12 > git checkout -f 43eef5aa2438a4b845c1e869587e37b4af9ee13 # timeout=10  
13 Commit message: "added power functionality"

Jenkins cloned the GitHub repository for the specific commit.

## **Stage 2 – Run Tests**

Jenkins / CS 816 - Software Production Engineering - Mini Project / Integrating Power Function into the Scientific Calculator / Pipeline Overview

🔍 ⚙️ 🌐

✓ < Integrating Power Function into the Scientific Calculator

🕒 Started 7 days 20 hr ago | 🛠 Queued 7.7 sec | ⏱ Took 56 sec | 🛡 Changes

⟳ Rerun ⋮

Graph

```
graph LR; Start((Start)) --> SCM((Checkout SCM)); SCM --> Checkout((Checkout)); Checkout --> RunTests((Run tests (Go))); RunTests --> BuildImage((Build image (local...))); BuildImage --> LoginPush((Login & Push image)); LoginPush --> DeployLocally((Deploy locally...)); DeployLocally --> PostActions((Post Actions)); PostActions --> End((End))
```

Run tests (Go)

which go || true go version go mod download go test ./... -v

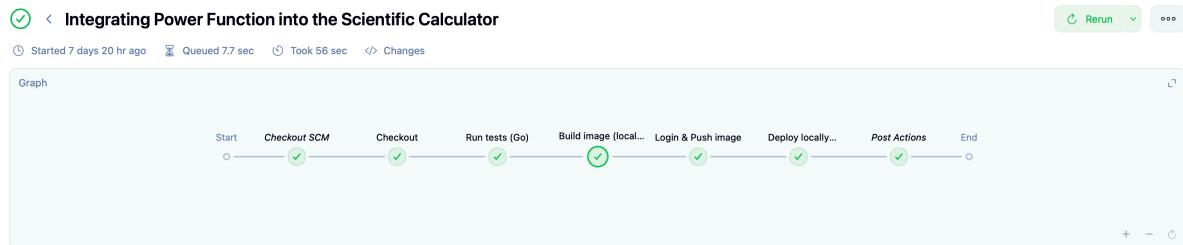
```

0 15:52:32 + which go
1 15:52:32 /opt/homebrew/bin/go
2 15:52:32 + go version
3 15:52:32 go version go1.24.2 darwin/arm64
4 15:52:32 + go mod download
5 15:52:32 go: no module dependencies to download
6 15:52:32 + go test ./... -v
7 15:52:32 ?     SPE_MiniProject/cmd/server      [no test files]
8 15:52:32 === RUN  TestFact
9 15:52:32 === RUN  TestFact/Zero_factorial
10 15:52:32 [ Factorial Operation ] Enter a non-negative integer: === RUN  TestFact/One_factorial
11 15:52:32 [ Factorial Operation ] Enter a non-negative integer: === RUN  TestFact/Small_positive_number
12 15:52:32 [ Factorial Operation ] Enter a non-negative integer: === RUN  TestFact/Larger_positive_number
13 15:52:32 [ Factorial Operation ] Enter a non-negative integer: === RUN  TestFact/Negative_number
14 15:52:32 [ Factorial Operation ] Enter a non-negative integer: --- PASS: TestFact (0.00s)
15 15:52:32     --- PASS: TestFact/Zero_factorial (0.00s)
16 15:52:32     --- PASS: TestFact/One_factorial (0.00s)
17 15:52:32     --- PASS: TestFact/Small_positive_number (0.00s)
18 15:52:32     --- PASS: TestFact/Larger_positive_number (0.00s)
19 15:52:32     --- PASS: TestFact/Negative_number (0.00s)
20 15:52:32 PASS

```

2.0s Started 7d 20h ago Jenkins ...

## Stage 3 – Build Docker Image



Jenkins / CS 816 - Software Production Engineering - Mini Project / Integrating Power Function into the Scientific Calculator / Pipeline Overview

Search

which docker || true docker --version docker build -t \${APP\_NAME}:local-\${GIT\_SHORT} .

```

0 15:52:34 + which docker
1 15:52:34 /usr/local/bin/docker
2 15:52:34 + docker --version
3 15:52:34 Docker version 28.4.0, build d8eb465
4 15:52:34 + docker build -t scicalc:local-43eef5a .
5 15:52:34 #0 building with "desktop-linux" instance using docker driver
6 15:52:34

```

5.5s

The Dockerfile used a two-stage build process resulting in a small, secure container image.

## Stage 4 – Push to DockerHub

After a successful build, Jenkins logged into DockerHub using credentials and pushed both the commit-tagged image and the latest tag:

```

    echo "$DOCKERHUB_PASS" | docker login -u "$DOCKERHUB_USER" --password-stdin docker tag ${APP_NAME}:local-${GIT_SHORT} ${REGIS...
    0 15:52:40 + echo *****
    1 15:52:40 + docker login -u pyxis2004 --password-stdin
    2 15:52:45 Login Succeeded
    3 15:52:45 + docker tag scicalc:local-43eef5a docker.io/pyxis2004/scicalc:latest
    4 15:52:45 + docker tag scicalc:local-43eef5a docker.io/pyxis2004/scicalc:43eef5a
    5 15:52:45 + docker push docker.io/pyxis2004/scicalc:latest
    6 15:52:45 The push refers to repository [docker.io/pyxis2004/scicalc]
    7 15:52:46 5664b15f108b: Waiting
    8 15:52:46 045fc1c20da8: Waiting
    ...

```

The console confirmed the successful upload, and DockerHub displayed the new image tags.

## Stage 5 – Deploy via Ansible

Finally, Jenkins triggered my Ansible playbook:

```

    ansible-galaxy collection install community.docker --force >/dev/null 2>&1 || true cd ansible ansible-playbook -i inventory.ini deploy.yml -e registr...
    0 15:53:07 + ansible-galaxy collection install community.docker --force
    1 15:53:12 + cd ansible
    2 15:53:12 + ansible-playbook -i inventory.ini deploy.yml -e registry=docker.io -e dockerhub_username=pyxis2004 -e app_name=scicalc -e
    deploy_tag=43eef5a
    3 15:53:12
    4 15:53:12 PLAY [Deploy SciCalc locally via Docker] ****
    5 15:53:12 TASK [Gathering Facts] ****
    6 15:53:13 ok: [127.0.0.1]
    8 15:53:13
    9 15:53:13 TASK [Validate required variables] ****
    10 15:53:13 skipping: [127.0.0.1]
    11 15:53:13
    12 15:53:13 TASK [Display deployment info] ****
    13 15:53:13 ok: [127.0.0.1] => {
    14 15:53:13     "msg": [
    15 15:53:13         "Registry: docker.io",
    16 15:53:13         "DockerHub Username: pyxis2004",
    17 15:53:13         "App Name: scicalc",
    18 15:53:13         "Deploy Tag: 43eef5a",
    19 15:53:13         "Full Image: docker.io/pyxis2004/scicalc:43eef5a"
    20 15:53:13     ]

```

## Stage 6 : Post Actions:

**SUCCESS: CS 816 - Software Production Engineering - Mini Project #6 (43eef5a)**

○ kulkarnipranav500@gmail.com <kulkarnipranav500@gmail.com>

To: ⓧ IMT2022053 Pranav Anand Kulkarni

 build.log  
58.8 KB

Download All • Preview All

**Build Succeeded**

**Job** CS 816 - Software Production Engineering - Mini Project

**Build** #6

**Commit** 43eef5a

**DockerHub** <https://hub.docker.com/repository/docker/pyxis2004/scicalc/general>

Post actions I had configured the Jenkins pipeline to send a email irrespective of success/failure with build logs attached, Job Name, Build No. and Commit ID/tag along with the URL to DockerHub containing the repository.

## **Verifying the Deployment**

After deployment, I verified the container status:

```
pranav_kulkarni@Pranav's-MacBook-Air ~ % docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
7b74a986598f pyxis2004/scicalc:fcbbed0 "/usr/local/bin/scic..." 14 seconds ago Up 13 seconds 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp scicalc
```

I then ran the application inside the container:

All operations executed perfectly, proving that my deployment pipeline was working end-to-end.