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

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 (√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@Pranavs-MacBook-Air Mini Project % 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
```

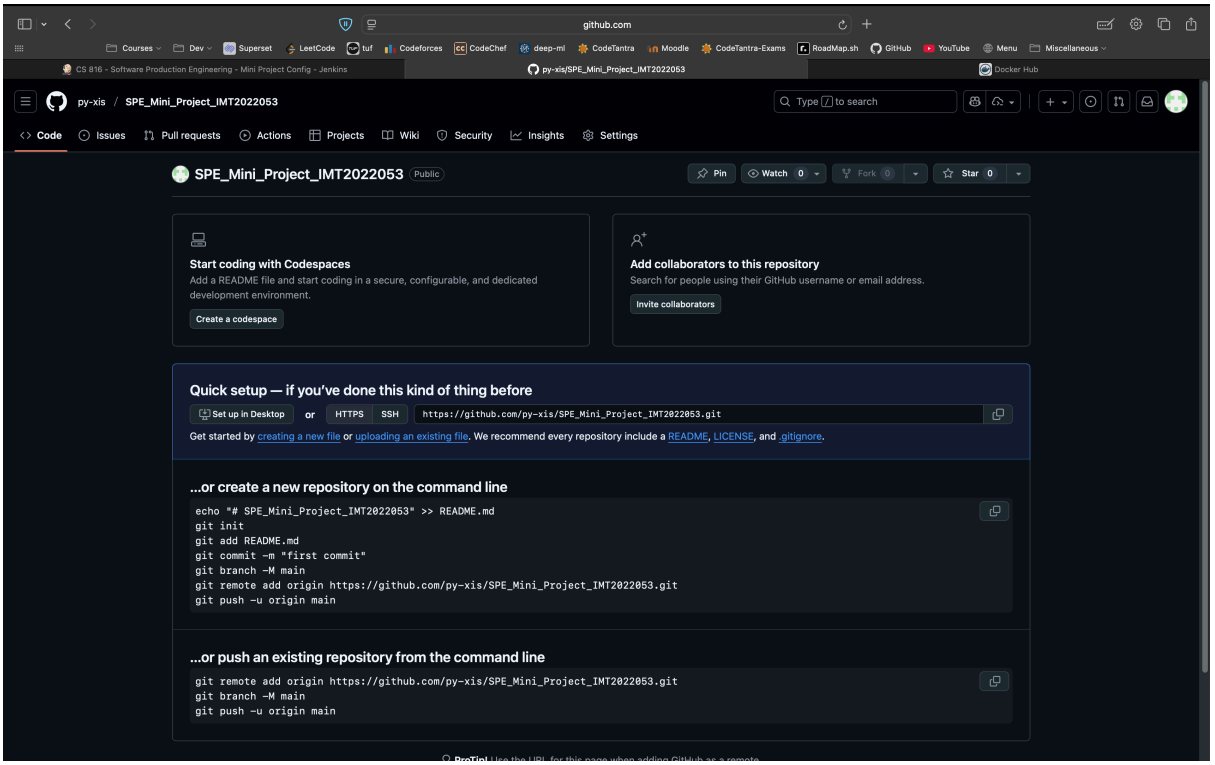
Initial Directory Structure

```
.
├── 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@Pranavs-MacBook-Air Mini Project % git init
Initialized empty Git repository in /Users/pranav_kulkarni/Desktop/SPE/Projects/Mini Project/.git/
```

Initialising Git Repo

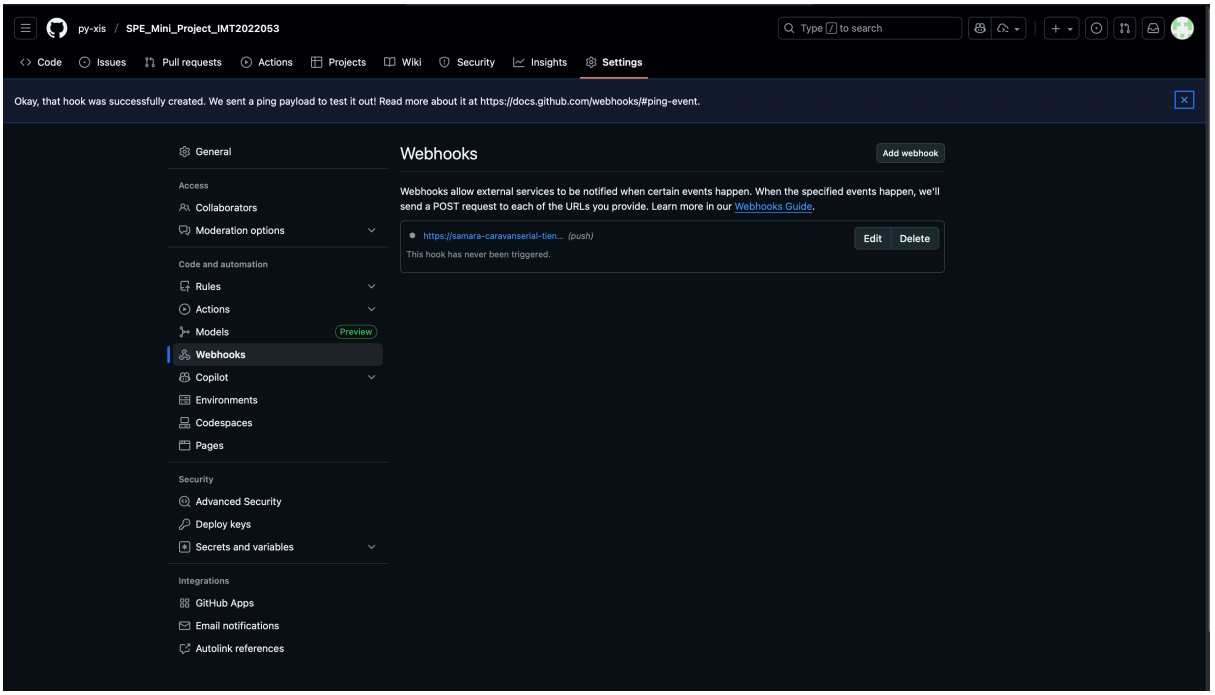


Creating the GitHub Repository

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.

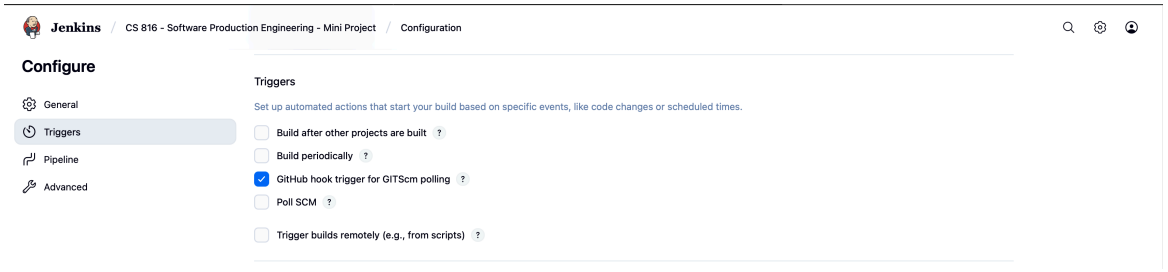
From that point, a single git push updated the running calculator without any manual step — a complete CI/CD setup.

---

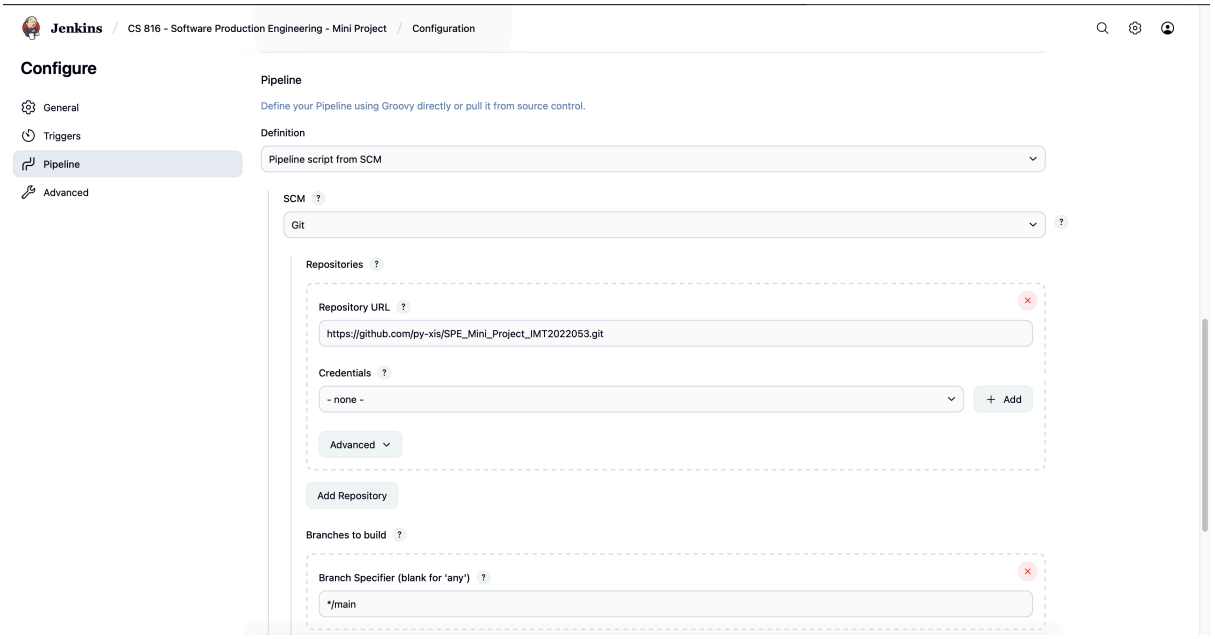
3. Setting Up Jenkins Job (Continuous Integration)

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

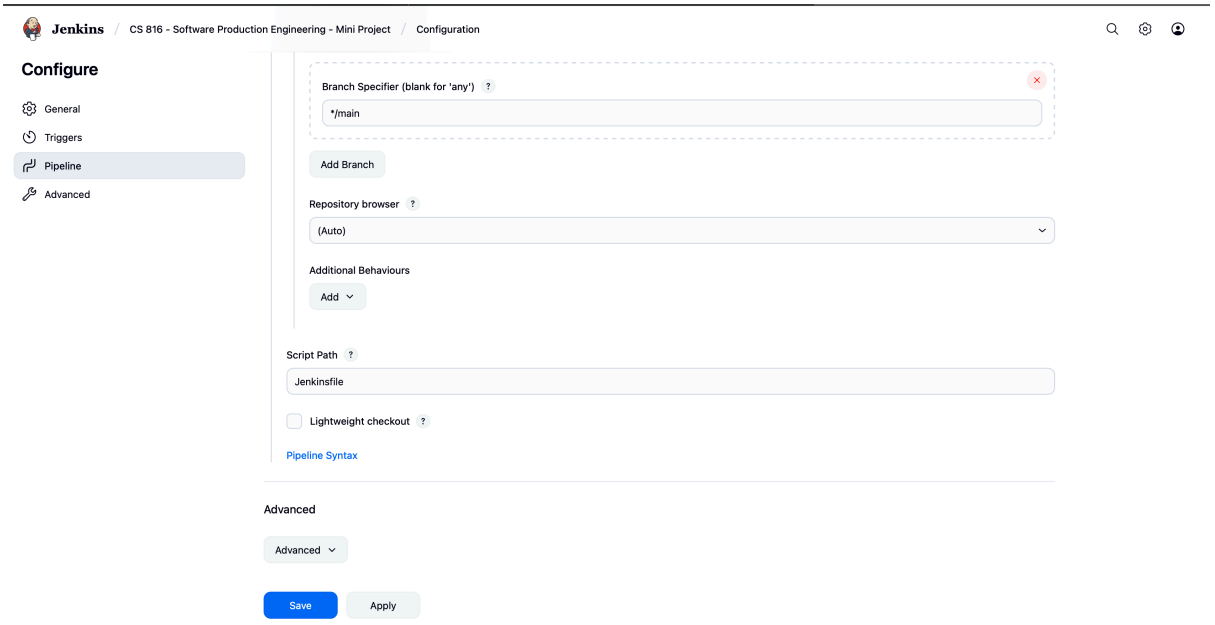
After installation, I created a **Pipeline job** titled *SPE_Mini_Project_IMT2022053*.



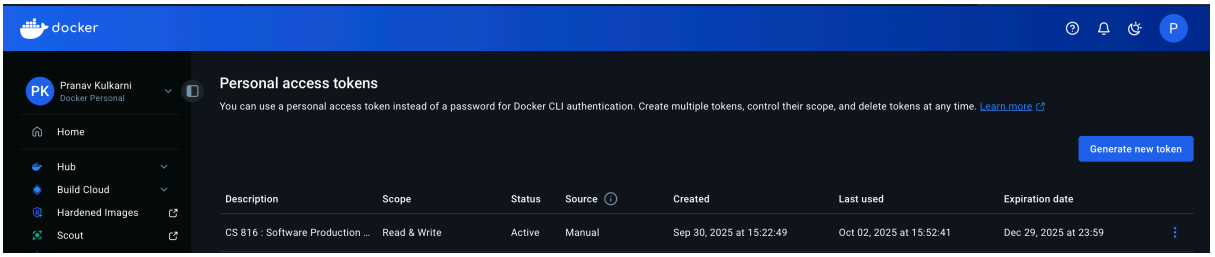
Configuring the job to trigger whenever a commit is done to the repository



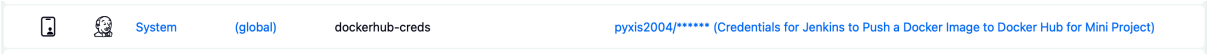
Adding the GitHub Repository URL, Branch of the Repository and Source of the Jenkinsfile



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



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.

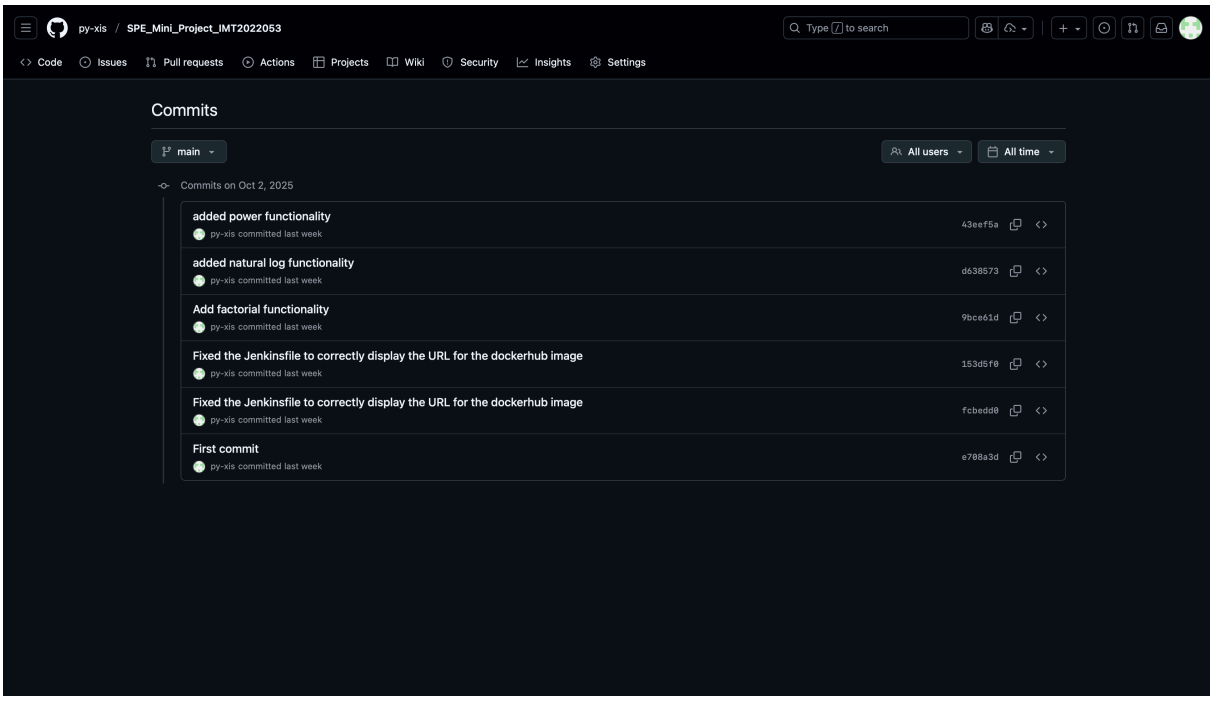
At this point, I had a fully functional Jenkins pipeline integrated with GitHub and Docker and Ansible

---

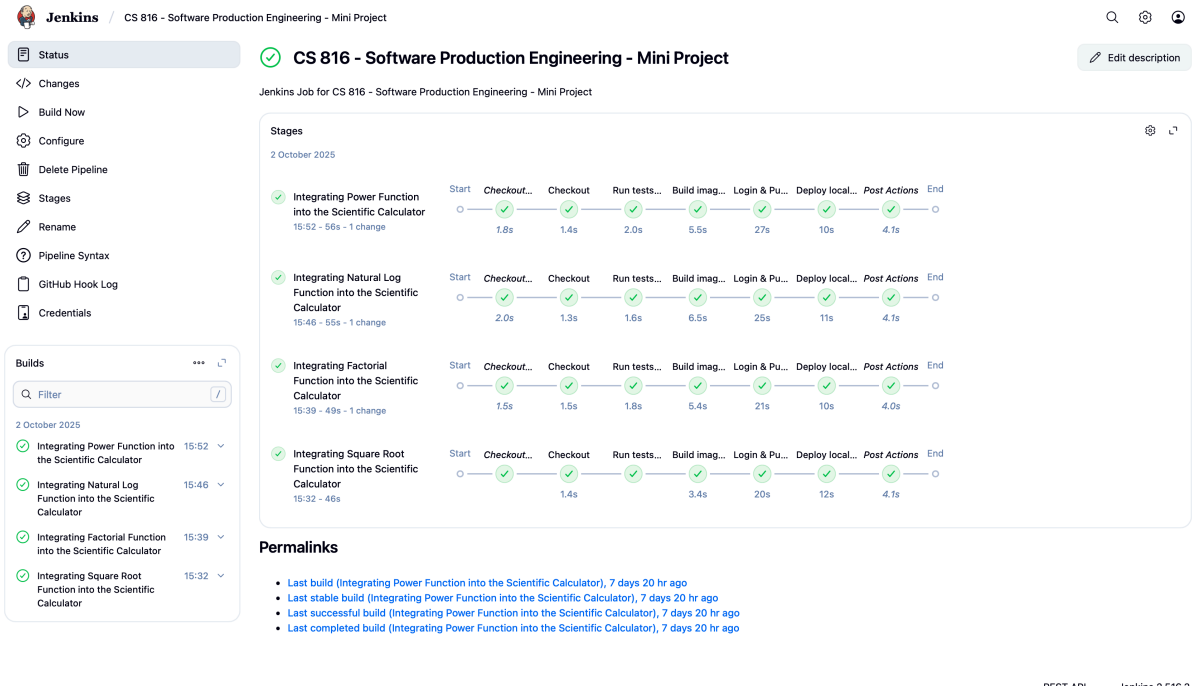
5. Incremental Development and CI/CD Workflow

I did not push all my code at once.

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



Git Commit History



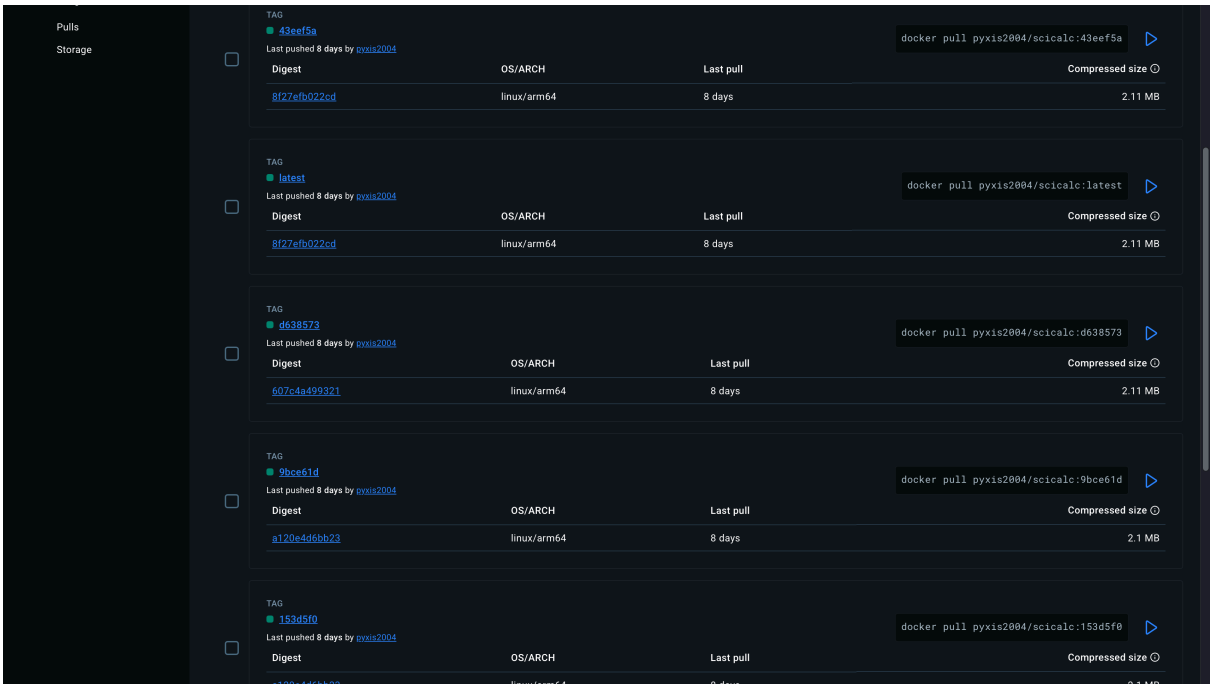
Build Triggered in Jenkins. Please note that I manually renamed the names of the build to make it more convenient to present.

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

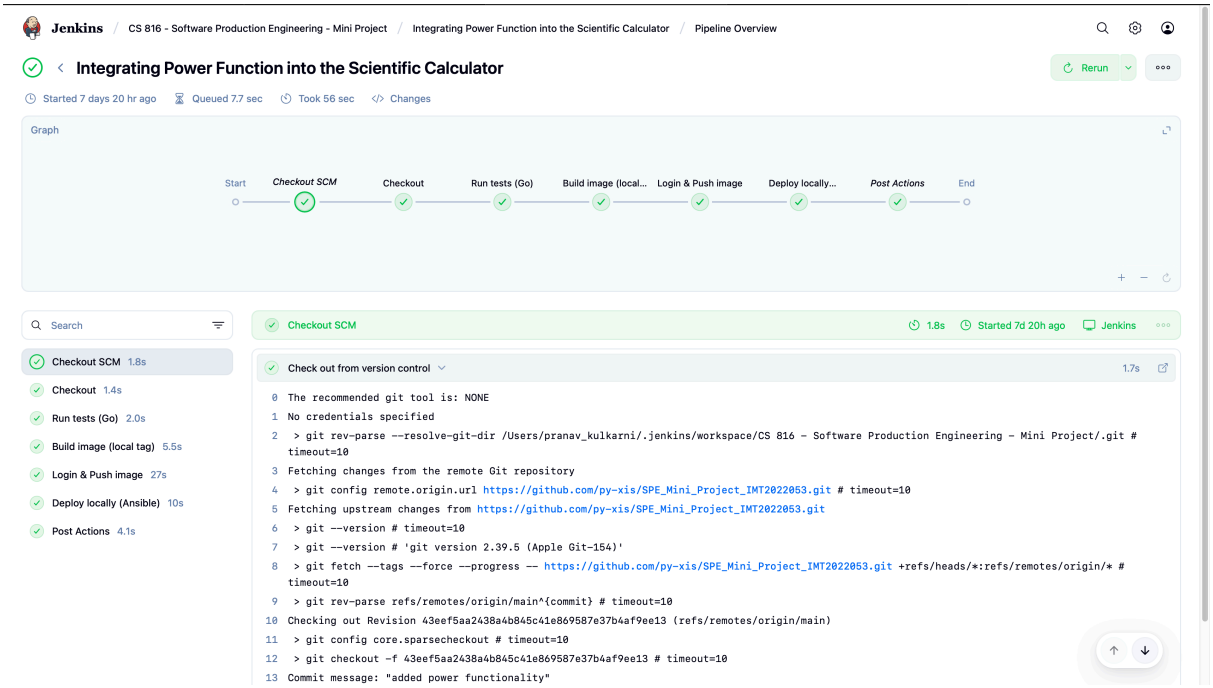


This incremental development style validated the entire **Continuous Integration → Continuous Deployment** concept in practice.

6. Running the Pipeline

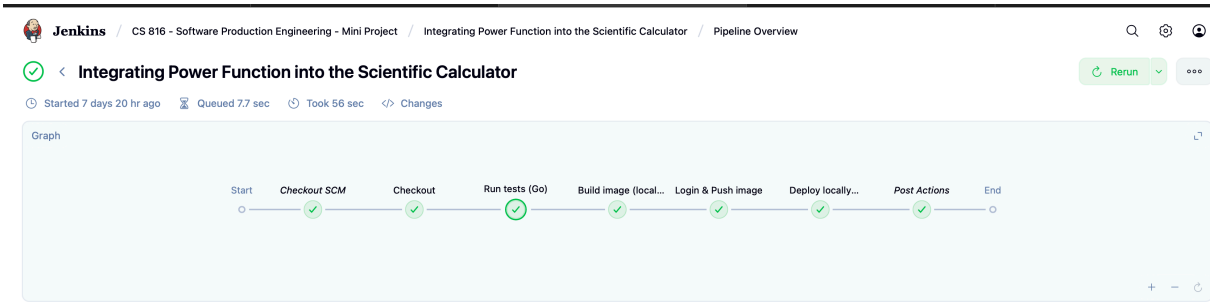
Every time I pushed a commit, the pipeline started automatically:

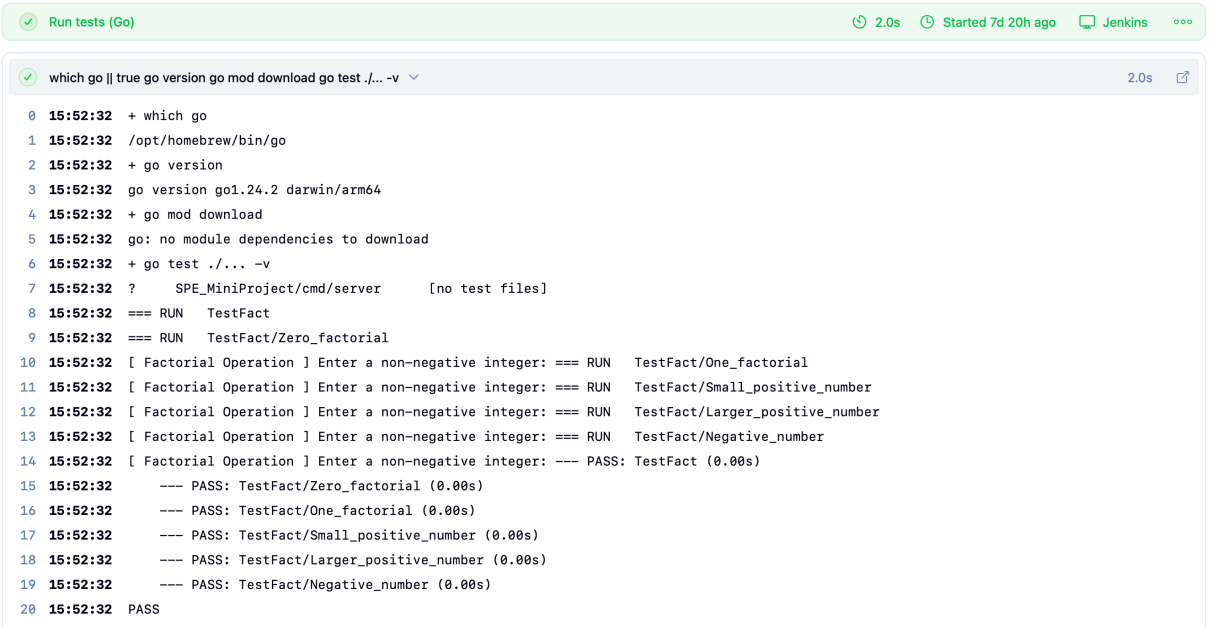
Stage 1 – Checkout



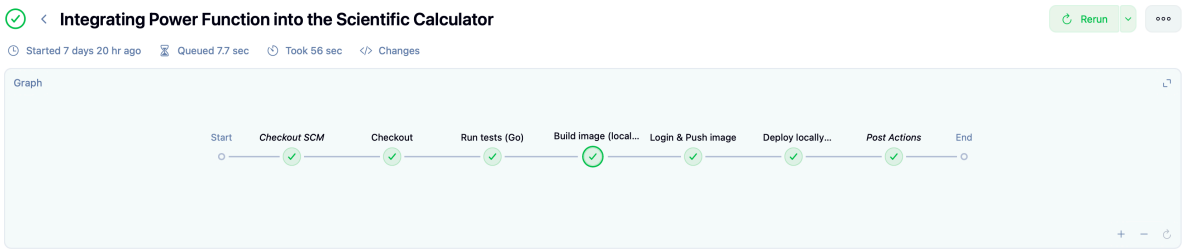
Jenkins cloned the GitHub repository for the specific commit.

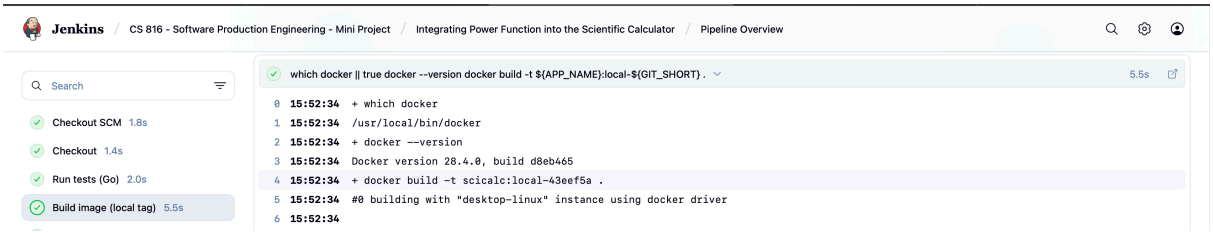
Stage 2 – Run Tests





Stage 3 – Build Docker Image

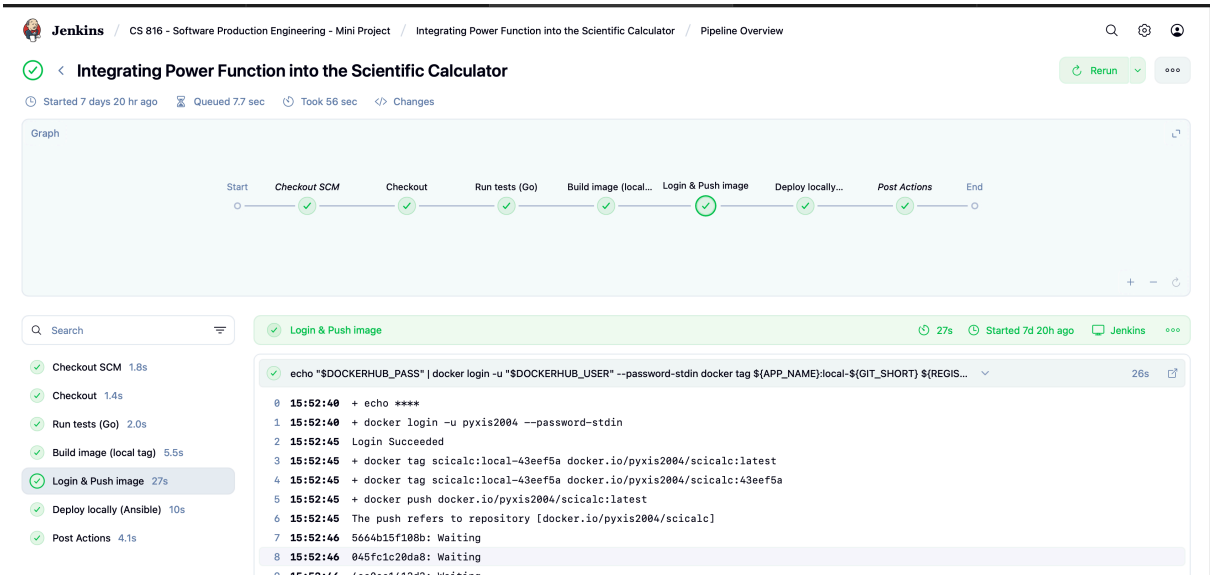




The Dockerfile used a two-stage build process resulting in a small, secure container image.

Stage 4 – Push to DockerHub

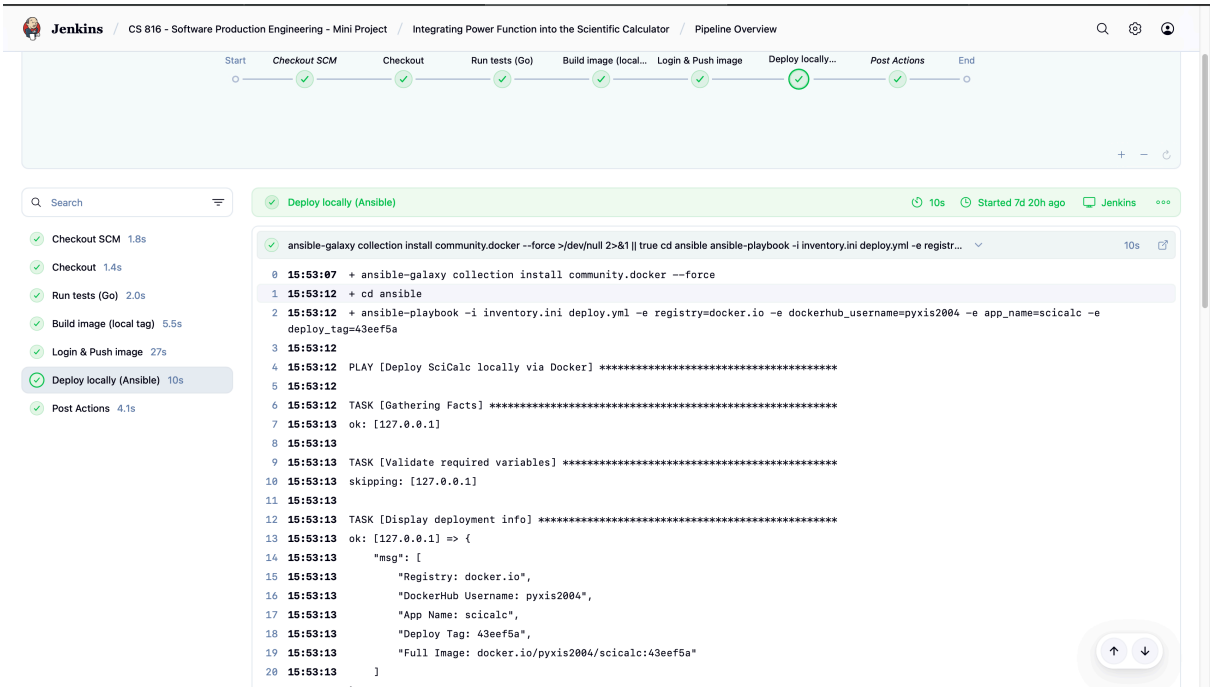
After a successful build, Jenkins logged into DockerHub using credentials and pushed both the commit-tagged image and the latest tag:



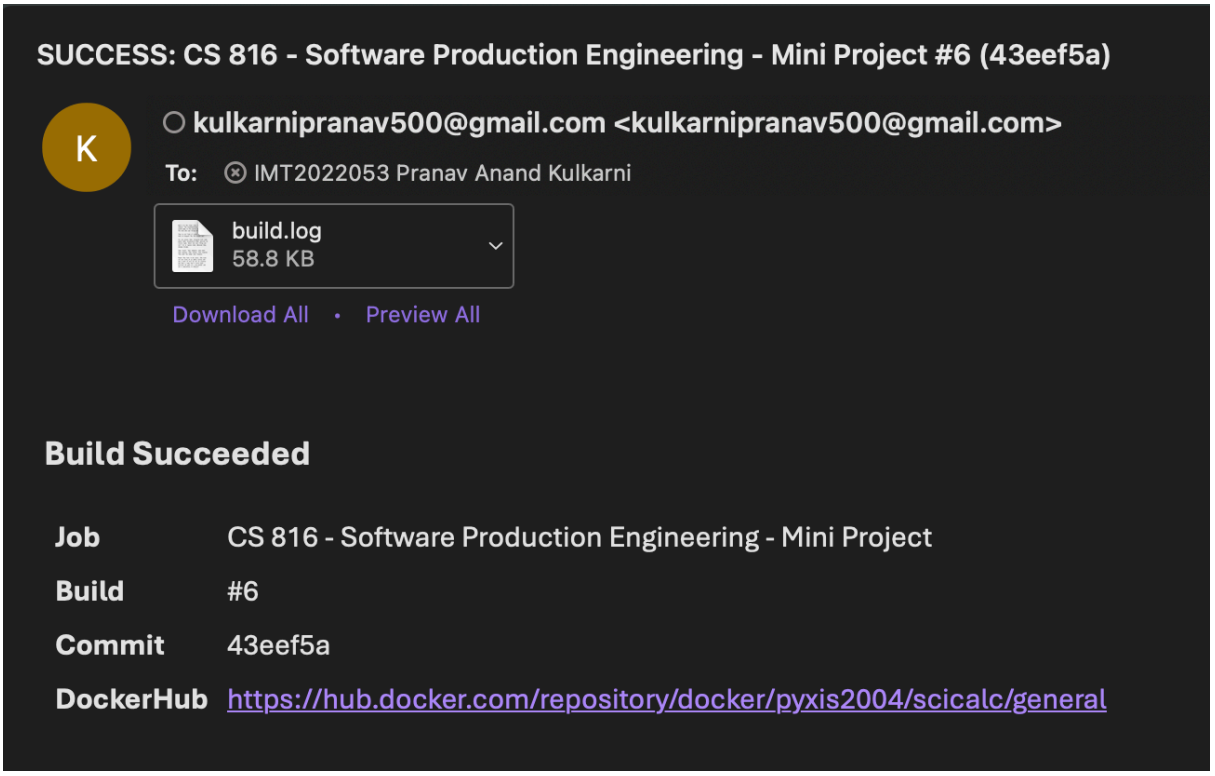
The console confirmed the successful upload, and DockerHub displayed the new image tags.

Stage 5 – Deploy via Ansible

Finally, Jenkins triggered my Ansible playbook:



Stage 6 : Post Actions:



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@Pranavs-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
7b74a986598f   pyxis2004/scicalc:fcbedd0   "/usr/local/bin/scic…"   14 seconds ago   Up 13 seconds   0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp   scical
c
```

I then ran the application inside the container:

```
pranav_kulkarni@Pranavs-MacBook-Air ~ % docker run -it --rm pyxis2004/scicalc:43eef5a --help

  ________      .__________        .__
 /   _____/ ____ |__\_   ___ \_____  |  |   ____
 \_____  \_/ ___\|  /    \  \/\__  \ |  | _/ ___\
 /        \  \___|  \     \____/ __ \|  |_\  \___
/_______  /\___  >__|\______  (____  /____/\___  >
        \/     \/           \/     \/          \/

Choose your operation[1, 2, 3, 4]
[1] Square Root of a Number (√x)
[2] Factorial of a Number (x!)
[3] Natural Logarithm of a Number(ln(x))
[4] Power Function (x ^ b)
Enter q to exit
1
[ Sqrt Operation ] Enter a number: 9
Result: 3.000000
Choose your operation[1, 2, 3, 4]
[1] Square Root of a Number (√x)
[2] Factorial of a Number (x!)
[3] Natural Logarithm of a Number(ln(x))
[4] Power Function (x ^ b)
Enter q to exit
q
Exiting...
```

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