International Institute of Information Technology, Bangalore

Software Production Engineering Mini Project Report

Scientific Calculator

Submitted by

Parv Gatecha (MT2024108)

In the guidance of Prof. B. Thangaraju

Contents:

Abstract:

A software program called the Scientific Calculator project was created to carry out a number of mathematical and scientific operations, including factororial, square root, logarithms, and powers of supplied numbers. DevOps techniques are incorporated into this project to optimize the development, testing, and deployment procedures. The project guarantees scalability and dependability by utilizing automated testing, containerization, and continuous integration and deployment (CI/CD) pipelines. The project showcases contemporary software engineering techniques that improve automation, maintainability, and teamwork through the use of DevOps tools including GitHub Actions, Docker, Jenkins, and Ansible.

An application for a small calculator that can do scientific and mathematical calculations is called the Scientific Calculator project. The project incorporates DevOps tools for automation, testing, and deployment and is created utilizing contemporary software development methodologies.

DevOps:

What is DevOps? DevOps is a software development methodology that integrates operations and development teams to collaborate on an application's lifetime. Accelerated application and service delivery is the aim of DevOps.

The rationale behind DevOps is that it helps companies deliver software more quickly and effectively by automating and optimizing procedures. Additionally, DevOps facilitates better teamwork.

DevOps advantages include:

- Quicker delivery: DevOps can assist companies in releasing software upgrades more regularly.
- Higher quality: DevOps can assist in lowering errors and raising the caliber of the final product.
- Improved cooperation: DevOps can assist in dismantling the divisions between the operations and development teams.
- Improved security: DevOps can assist in incorporating security procedures into the process of development.
- Better scalability: By automating processes and streamlining workflows, DevOps can assist in managing scalable solutions.
- Improved customer experience: DevOps can assist in producing software that is of a high caliber and satisfies client requirements.
- Cost reduction: DevOps can assist in lowering the expenses related to sluggish and ineffective software delivery procedures.

Tools and Technologies Used:

Programming Language: Python

Version Control: Git & GitHub

Build Automation: FastApi

CI/CD Pipeline: GitHub Actions / Jenkins

Containerization: Docker

Testing Framework: Pytest

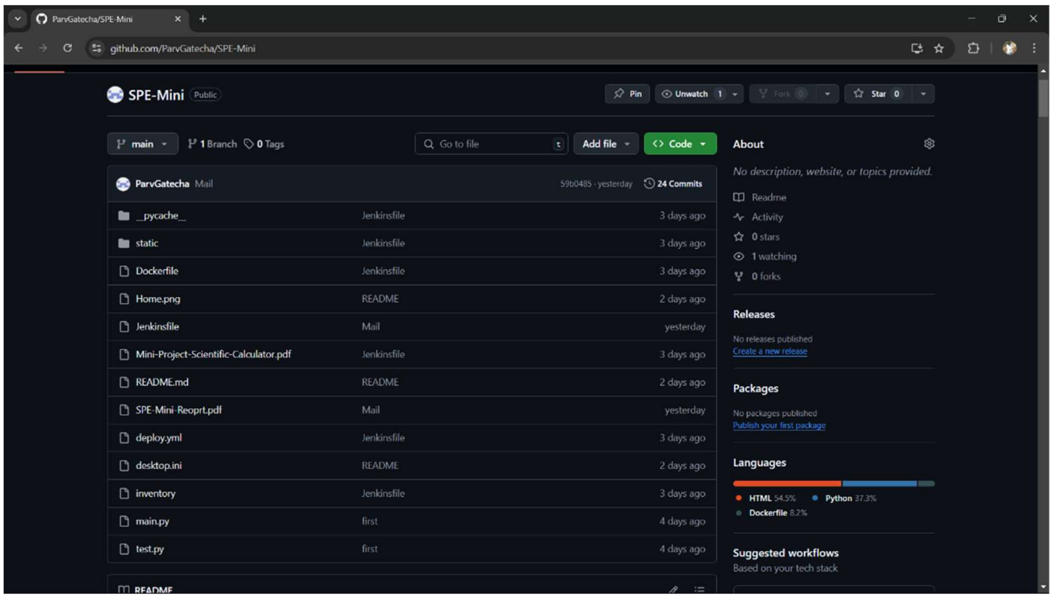
Source Code Management

SCM tracks changes made to a source code repository. SCM monitors a code base's modifications and assists in resolving disputes when combining updates from several authors. Version Control is another name for SCM.

The local machine (repository) is where the complete project is initially built. After then, this project gets uploaded to GitHub's remote repository.

To push the local repository to the remote repository, a set of instructions is run.

- git init
- git status
- git add
- git commit -m "Commit Message"
- git remote add origin https://github.com/ParvGatecha/SPE-Mini.git
- git push -u origin master



SPE-Mini Public

main · 1 Branch · 0 Tags

ParvGatecha Mail · 59b0485 · yesterday · 24 Commits

pycache	Jenkinsfile	3 days ago
static	Jenkinsfile	3 days ago
Dockerfile	Jenkinsfile	3 days ago
Home.png	README	2 days ago
Jenkinsfile	Mail	yesterday
Mini-Project-Scientific-Calculator.pdf	Jenkinsfile	3 days ago
README.md	README	2 days ago
SPE-Mini-Reoprt.pdf	Mail	yesterday
deploy.yml	Jenkinsfile	3 days ago
desktop.ini	README	2 days ago
inventory	Jenkinsfile	3 days ago
main.py	first	4 days ago
test.py	first	4 days ago

About

No description, website, or topics provided.

Readme · Activity · 0 stars · 1 watching · 0 forks

Releases: No releases published. Create a new release

Packages: No packages published. Publish your first package

Languages: HTML 54.5% · Python 37.3% · Dockerfile 8.2%

Suggested workflows: Based on your tech stack

Source Code Management

Docker:

Software can be delivered as a package using Docker, a kernel-level virtualization tool. The image needed to run the jar file produced by the building to Maven is constructed using Docker. Other computers can pull the image and create containers to use the file after it has been posted to the Docker Hub.

The following command can be used to build the Docker image:

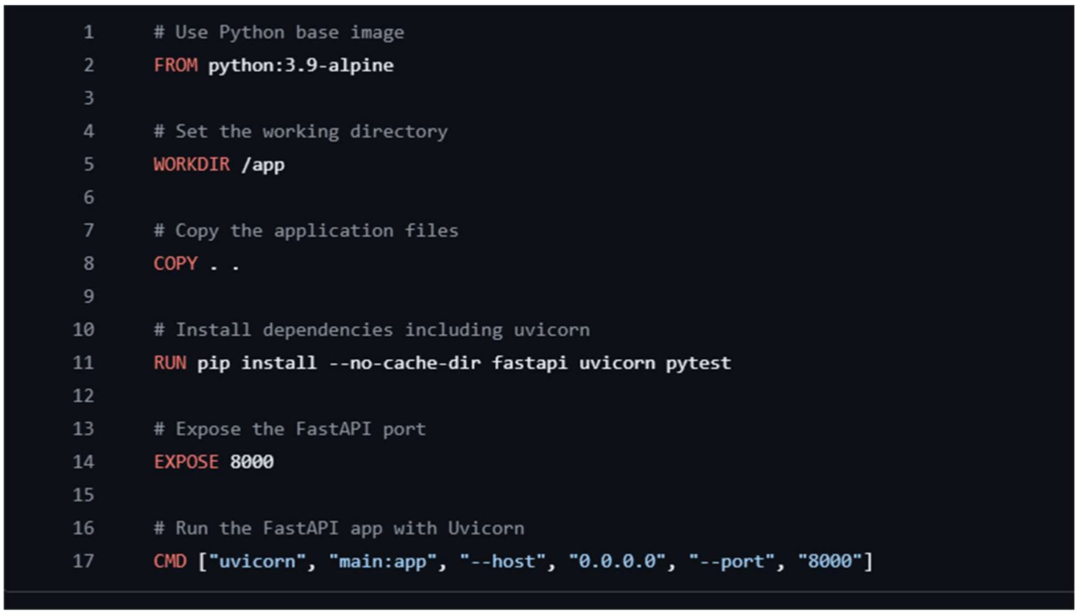
docker build –t <USERNAME>/<IMAGE NAME> : <TAG>

To push the docker image use command:

docker push <USERNAME>/<REPOSITORY_NAME>:tagname

The docker image is pushed from jenkins after the build stage and pulled on from ansible job. The docker image can be run using command

docker run –i –t <image_name>. After pushing the created docker image to Docker Hub, we can check on our Docker hub profile.

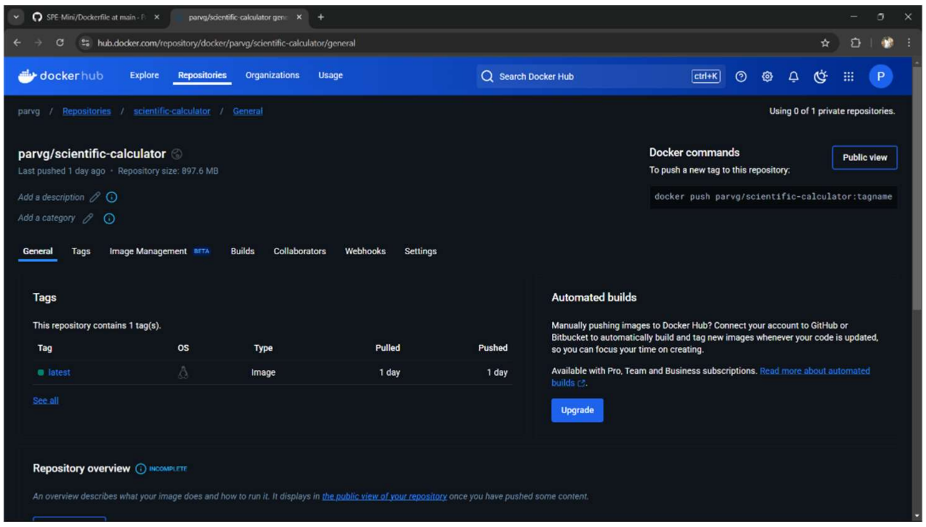


```dockerfile
# Use Python base image
FROM python:3.9-alpine

# Set the working directory
WORKDIR /app

# Copy the application files
COPY . .

# Install dependencies including uvicorn
RUN pip install --no-cache-dir fastapi uvicorn pytest

# Expose the FastAPI port
EXPOSE 8000

# Run the FastAPI app with Uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

DockerFile

To build the docker image and push it to Docker Hub remote repository, we will specify steps in Jenkins pipeline script.

The Docker image is built and pushed on the Docker Hub repository of the user. The local image on the machine is deleted after the image is pushed on the Docker Hub.



hub.docker.com/repository/docker/parvg/scientific-calculator/general

parvg / Repositories / scientific-calculator / General

Using 0 of 1 private repositories.

parvg/scientific-calculator

Last pushed 1 day ago · Repository size: 897.6 MB

Add a description

Add a category

Docker commands

Public view

To push a new tag to this repository:

```
docker push parvg/scientific-calculator:tagname
```

General | Tags | Image Management BETA | Builds | Collaborators | Webhooks | Settings

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	1 day	1 day

See all

Automated builds

Manually pushing images to Docker Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Upgrade

Repository overview INCOMPLETE

An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.

Docker Hub Repository

Jenkins:

Regardless of the platform you are working on, Jenkins is a robust tool that enables continuous integration and continuous project delivery.

Any type of build or continuous integration can be handled by this free source program. Jenkins can be integrated with a variety of deployment and testing tools.

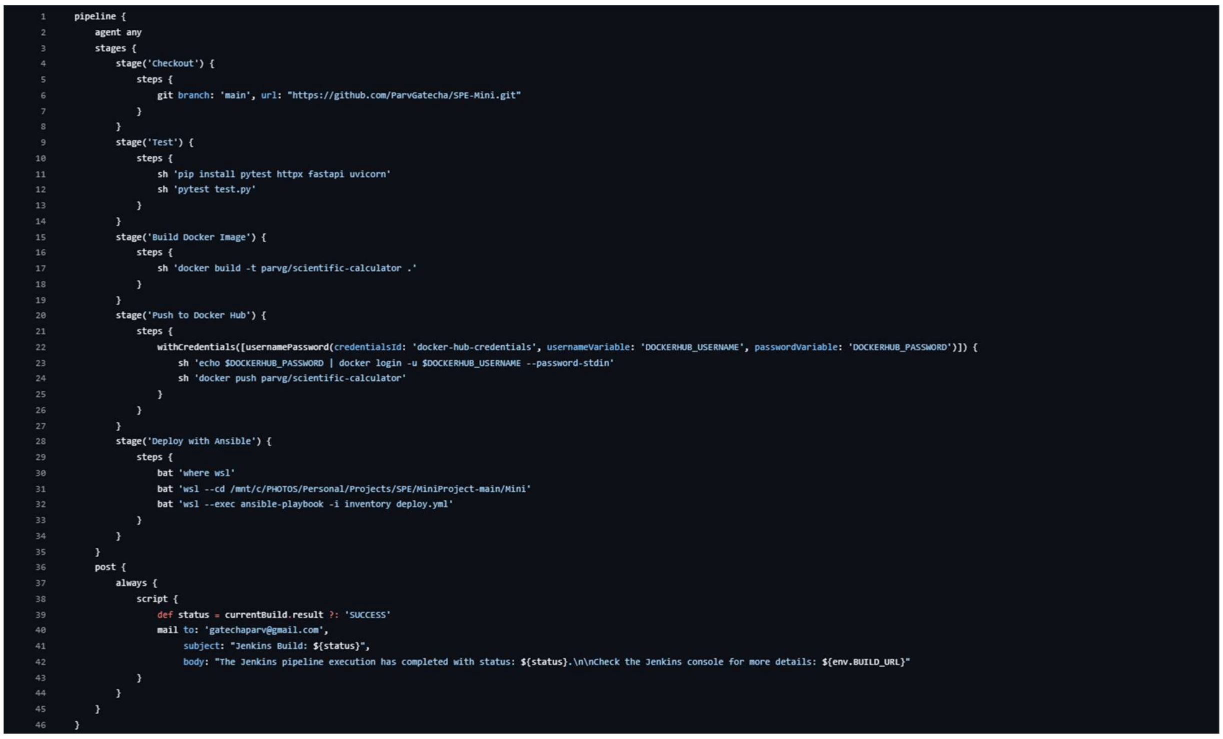
The pipeline is set up such that it is activated and all of its phases are carried out whenever a new commit is made to the GitHub repository.

An automated expression that illustrates your software acquisition procedure for version control is called a Jenkins pipeline. As a result, each modification made to your software must pass through several intricate procedures before it can be made public. Additionally, it entails building software in a dependable and repeatable way and advancing it through several testing and deployment phases.
The Jenkins pipeline is used to integrate all of the jobs. Pulling the updates from GitHub, building the project with Maven, creating the Docker image, pushing the image to Docker Hub, and running the Ansible playbook are all steps in the pipeline.

Create CI Pipeline

An automated expression that illustrates your software acquisition procedure for version control is called a Jenkins pipeline. As a result, each modification made to your software must pass through several intricate procedures before it can be made public. Additionally, it entails building software in a dependable and repeatable way and advancing it through several testing and deployment phases.

There are various stages in Jenkins Pipeline including Checkout, Build Docker Image, Push Docker Image and Run ansible playbook then there is post action to send email for acknowledge the status of pipeline.



```groovy
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: "https://github.com/ParvGatecha/SPE-Mini.git"
            }
        }
        stage('Test') {
            steps {
                sh 'pip install pytest httpx fastapi uvicorn'
                sh 'pytest test.py'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t parvg/scientific-calculator .'
            }
        }
        stage('Push to Docker Hub') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'docker-hub-credentials', usernameVariable: 'DOCKERHUB_USERNAME', passwordVariable: 'DOCKERHUB_PASSWORD')]) {
                    sh 'echo $DOCKERHUB_PASSWORD | docker login -u $DOCKERHUB_USERNAME --password-stdin'
                    sh 'docker push parvg/scientific-calculator'
                }
            }
        }
        stage('Deploy with Ansible') {
            steps {
                bat 'where wsl'
                bat 'wsl --cd /mnt/c/PHOTOS/Personal/Projects/SPE/MiniProject-main/Mini'
                bat 'wsl --exec ansible-playbook -i inventory deploy.yml'
            }
        }
    }
    post {
        always {
            script {
                def status = currentBuild.result ?: 'SUCCESS'
                mail to: 'gatechaparv@gmail.com',
                     subject: "Jenkins Build: ${status}",
                     body: "The Jenkins pipeline execution has completed with status: ${status}.\n\nCheck the Jenkins console for more details: ${env.BUILD_URL}"
            }
        }
    }
}
```

Stages of Pipeline

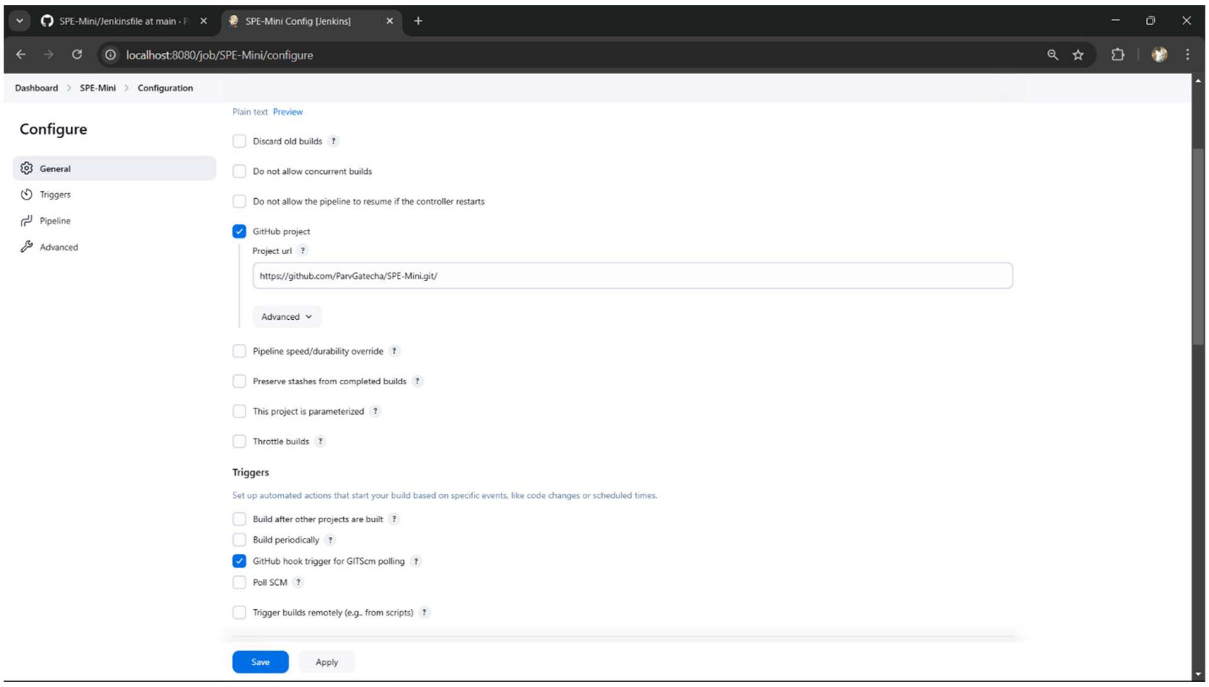
Configuration of Project in Jenkins

To run Jenkins in local machine there is a command to run it

sudo systemctl start jenkins

To check status of Jenkins:

sudo systemctl status jenkins


SPE-Mini/Jenkinsfile at main
SPE-Mini Config [Jenkins]
localhost:8080/job/SPE-Mini/configure
Dashboard > SPE-Mini > Configuration
Configure
General
Triggers
Pipeline
Advanced
Plain text Preview

☐ Discard old builds

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the controller restarts

☑ GitHub project

Project url

https://github.com/ParvGatecha/SPE-Mini.git/

Advanced

☐ Pipeline speed/durability override

☐ Preserve stashes from completed builds

☐ This project is parameterized

☐ Throttle builds

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

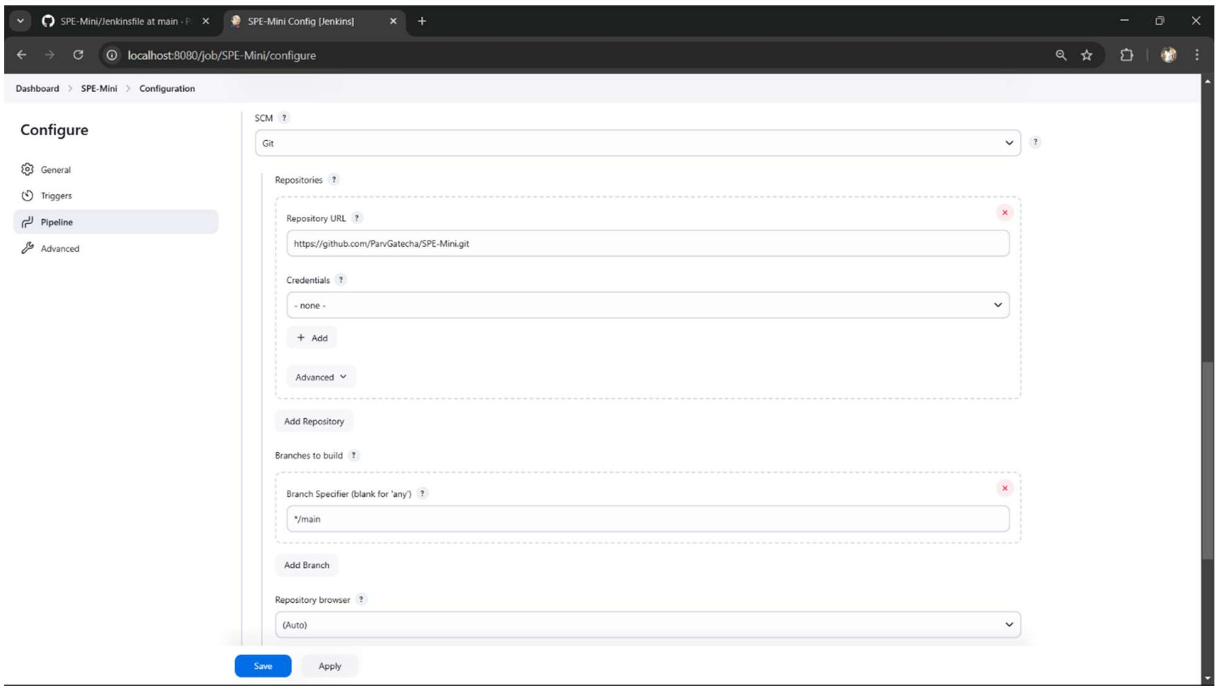
☐ Build after other projects are built

☐ Build periodically

☑ GitHub hook trigger for GITScm polling

☐ Poll SCM

☐ Trigger builds remotely (e.g., from scripts)


SPE-Mini/Jenkinsfile at main
SPE-Mini Config [Jenkins]
localhost:8080/job/SPE-Mini/configure
Dashboard > SPE-Mini > Configuration
Configure
General
Triggers
Pipeline
Advanced

SCM

Git

Repositories

Repository URL

https://github.com/ParvGatecha/SPE-Mini.git

Credentials

- none -

+ Add
Advanced
Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/main

Repository browser

(Auto)

Configuration of Project

Here, we created github project in jenkins and give URL of our github repository and specify the jenkins file path in configuration.

Set System Credentials for Docker Hub and Local Machine for Ansible playbook, here we run project in local machine and pull the image from docker image so we have to specify that system configuration in jenkins system credentials and add two credentials one for Docker Hub Repository and other one for Local Machine.

To set DockerHub credentials we have to specify the username of dockerhub from where we have to pull the image.

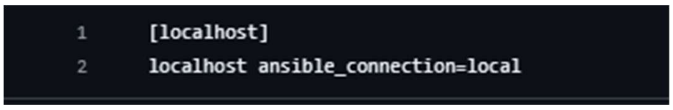
Ansible:

One tool for continuous deployment is Ansible. where the deliverables are distributed on what are known as managed nodes, which have a single control node.

Only the control node has to have Ansible installed. Ansible uses SSH to establish a connection with the controlled nodes. Ansible provides a collection of commands in the form of a yml file that must be run on the managed node. We refer to it as a playbook. We offer details on the managed nodes in addition to the playbook. It is known as inventory.

To execute this playbook we run the following command:

ansible-playbook <playbook> -i <inventory>



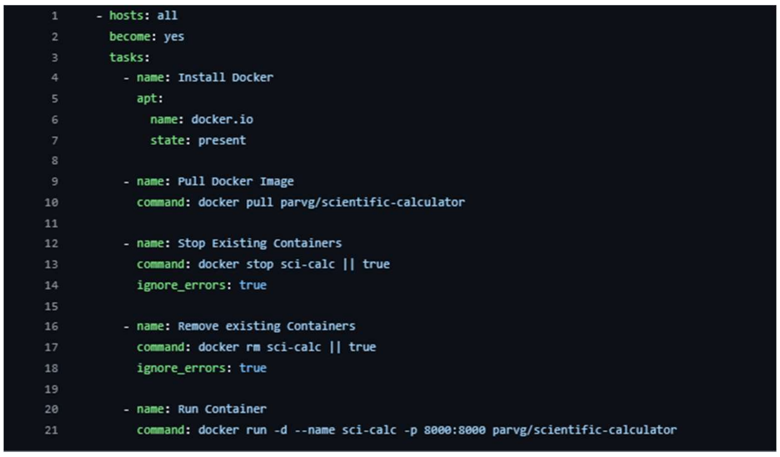
```
[localhost]
localhost ansible_connection=local
```

Inventory File

The hosts and host groups that a playbook's commands, modules, and tasks run on are specified in the Ansible inventory file. Depending on your Ansible environment and plugins, the file may be in one of several formates.

The configuration, deployment, and orchestration functions of Ansible are documented and carried out by the Ansible playbooks.

To launch the application's docker image, we must install Python pip on the host in addition to docker. The container is launched on the designated hosts when the docker image is downloaded from the docker hub. The Ansible playbook handles this process.

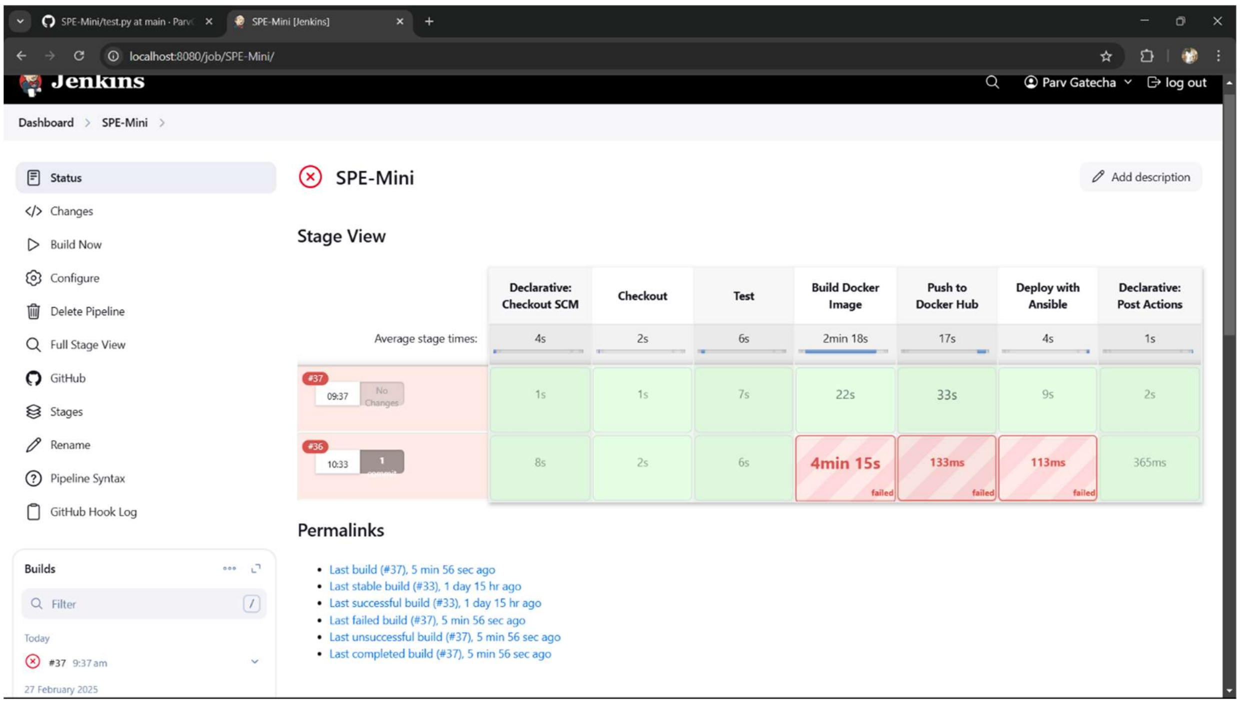


```yaml
- hosts: all
  become: yes
  tasks:
    - name: Install Docker
      apt:
        name: docker.io
        state: present

    - name: Pull Docker Image
      command: docker pull parvg/scientific-calculator

    - name: Stop Existing Containers
      command: docker stop sci-calc || true
      ignore_errors: true

    - name: Remove existing Containers
      command: docker rm sci-calc || true
      ignore_errors: true

    - name: Run Container
      command: docker run -d --name sci-calc -p 8000:8000 parvg/scientific-calculator
```

Playbook file

Run Project as Pipeline in Jenkins

To run the project as pipeline in jenkins we have to click build now to see each stages of pipeline and show the output of each stages.



localhost:8080/job/SPE-Mini/

Dashboard > SPE-Mini >

SPE-Mini

Stage View

	Declarative: Checkout SCM	Checkout	Test	Build Docker Image	Push to Docker Hub	Deploy with Ansible	Declarative: Post Actions
Average stage times:	4s	2s	6s	2min 18s	17s	4s	1s
#37 09:37 No Changes	1s	1s	7s	22s	33s	9s	2s
#36 10:33 1	8s	2s	6s	4min 15s failed	133ms failed	113ms failed	365ms

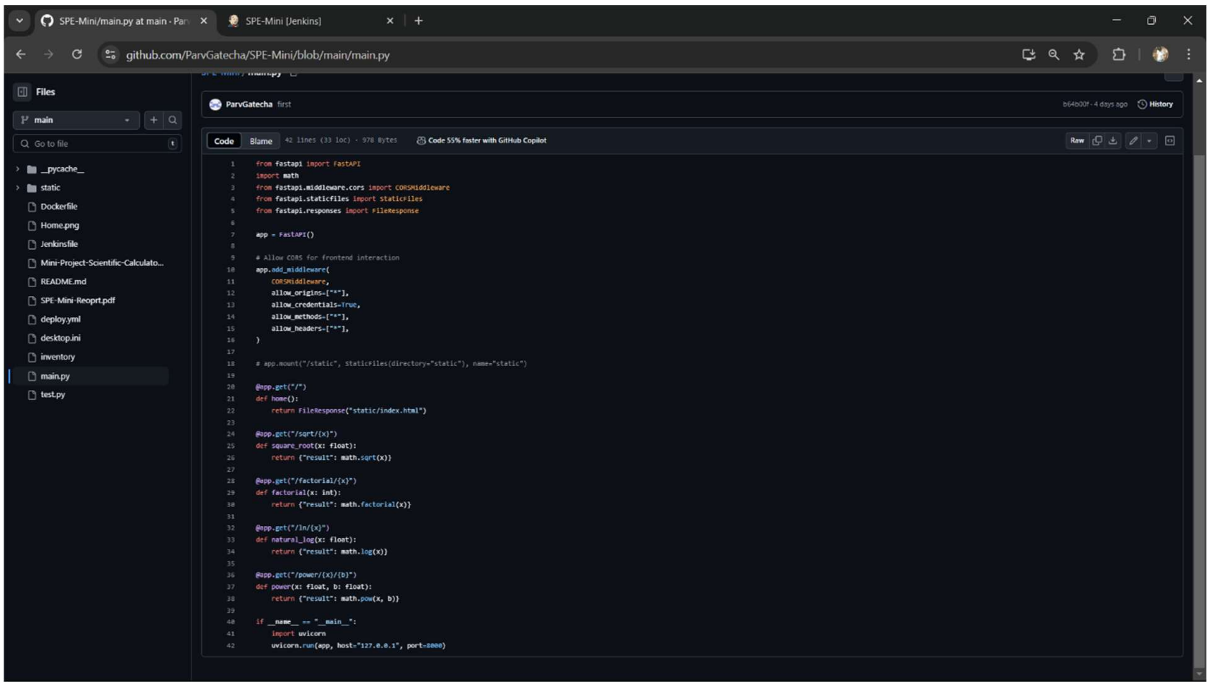
Permalinks

- Last build (#37), 5 min 56 sec ago
- Last stable build (#33), 1 day 15 hr ago
- Last successful build (#33), 1 day 15 hr ago
- Last failed build (#37), 5 min 56 sec ago
- Last unsuccessful build (#37), 5 min 56 sec ago
- Last completed build (#37), 5 min 56 sec ago

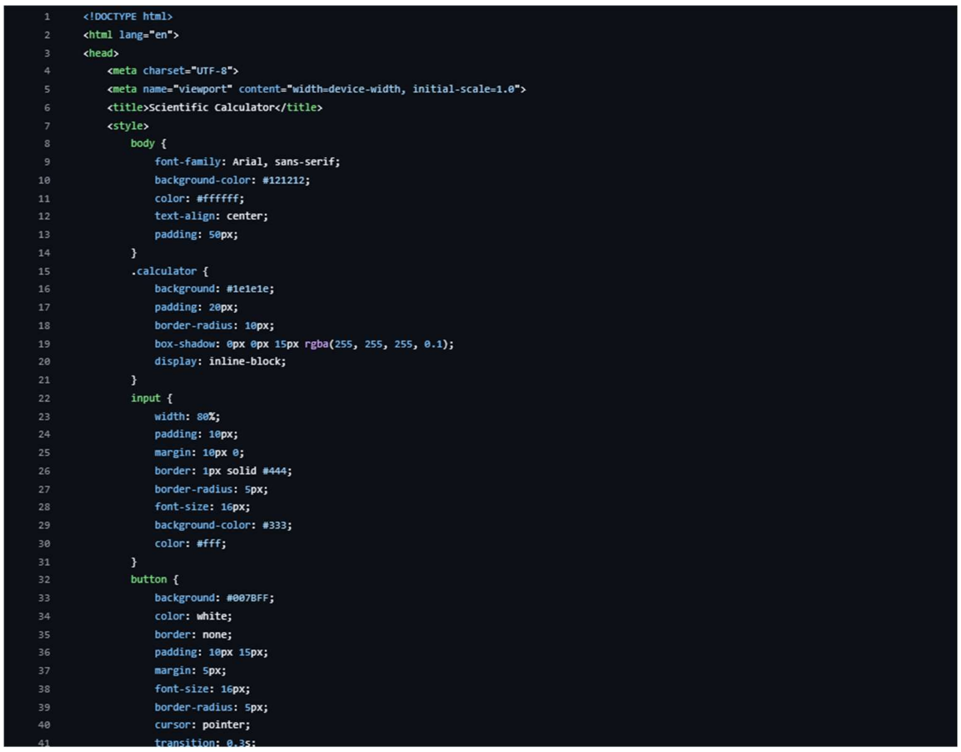
View of Pipeline Stages

Scientific Calculator Source Code:

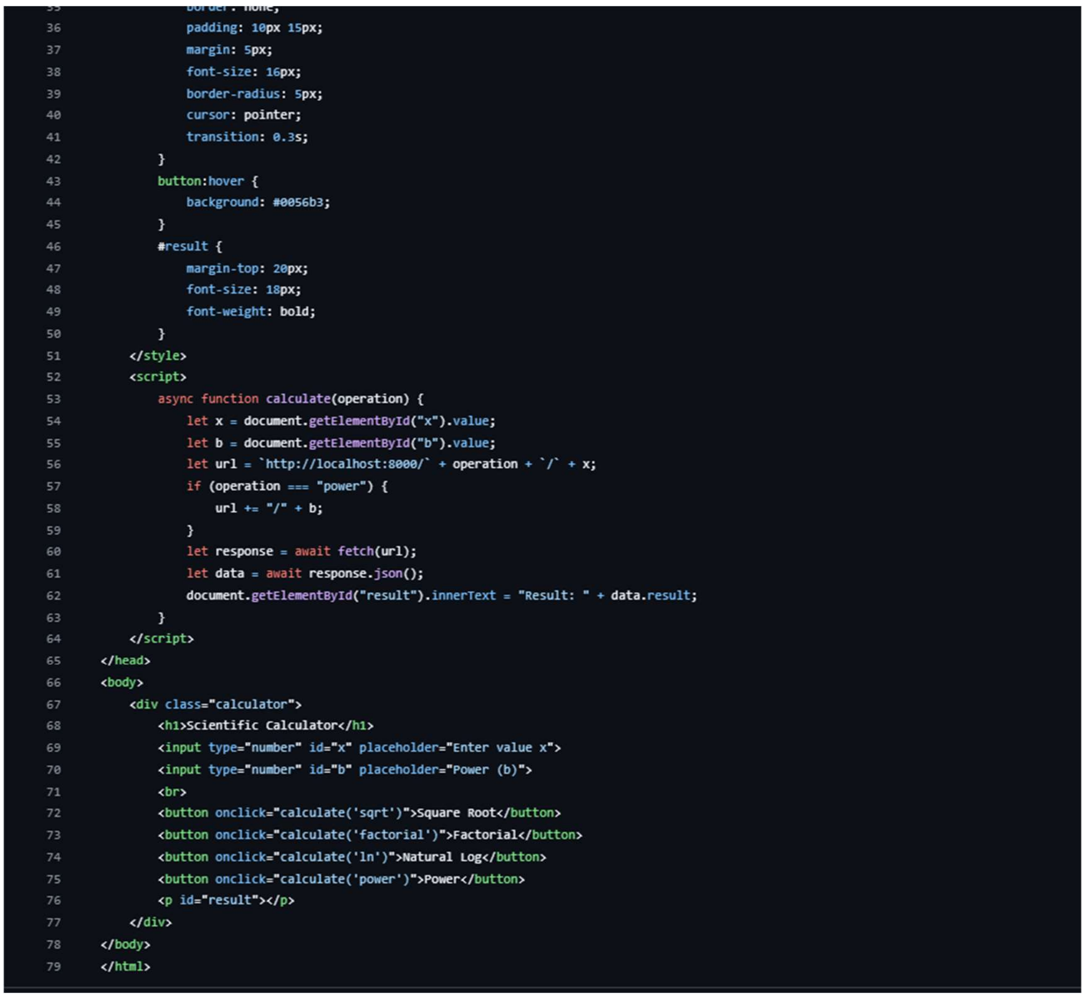
Below attached the source code of scientific calculator in python



```python
from fastapi import FastAPI
import math
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse

app = FastAPI()

# Allow CORS for frontend interaction
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/")
def home():
    return FileResponse("static/index.html")

@app.get("/sqrt/{x}")
def square_root(x: float):
    return {"result": math.sqrt(x)}

@app.get("/factorial/{x}")
def factorial(x: int):
    return {"result": math.factorial(x)}

@app.get("/ln/{x}")
def natural_log(x: float):
    return {"result": math.log(x)}

@app.get("/power/{x}/{b}")
def power(x: float, b: float):
    return {"result": math.pow(x, b)}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scientific Calculator</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #121212;
            color: #ffffff;
            text-align: center;
            padding: 50px;
        }
        .calculator {
            background: #1e1e1e;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(255, 255, 255, 0.1);
            display: inline-block;
        }
        input {
            width: 80%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #444;
            border-radius: 5px;
            font-size: 16px;
            background-color: #333;
            color: #fff;
        }
        button {
            background: #007BFF;
            color: white;
            border: none;
            padding: 10px 15px;
            margin: 5px;
            font-size: 16px;
            border-radius: 5px;
            cursor: pointer;
            transition: 0.3s;
```



```
            border: none;
            padding: 10px 15px;
            margin: 5px;
            font-size: 16px;
            border-radius: 5px;
            cursor: pointer;
            transition: 0.3s;
        }
        button:hover {
            background: #0056b3;
        }
        #result {
            margin-top: 20px;
            font-size: 18px;
            font-weight: bold;
        }
    </style>
    <script>
        async function calculate(operation) {
            let x = document.getElementById("x").value;
            let b = document.getElementById("b").value;
            let url = `http://localhost:8000/` + operation + `/` + x;
            if (operation === "power") {
                url += "/" + b;
            }
            let response = await fetch(url);
            let data = await response.json();
            document.getElementById("result").innerText = "Result: " + data.result;
        }
    </script>
</head>
<body>
    <div class="calculator">
        <h1>Scientific Calculator</h1>
        <input type="number" id="x" placeholder="Enter value x">
        <input type="number" id="b" placeholder="Power (b)">
        <br>
        <button onclick="calculate('sqrt')">Square Root</button>
        <button onclick="calculate('factorial')">Factorial</button>
        <button onclick="calculate('ln')">Natural Log</button>
        <button onclick="calculate('power')">Power</button>
        <p id="result"></p>
    </div>
</body>
</html>
```

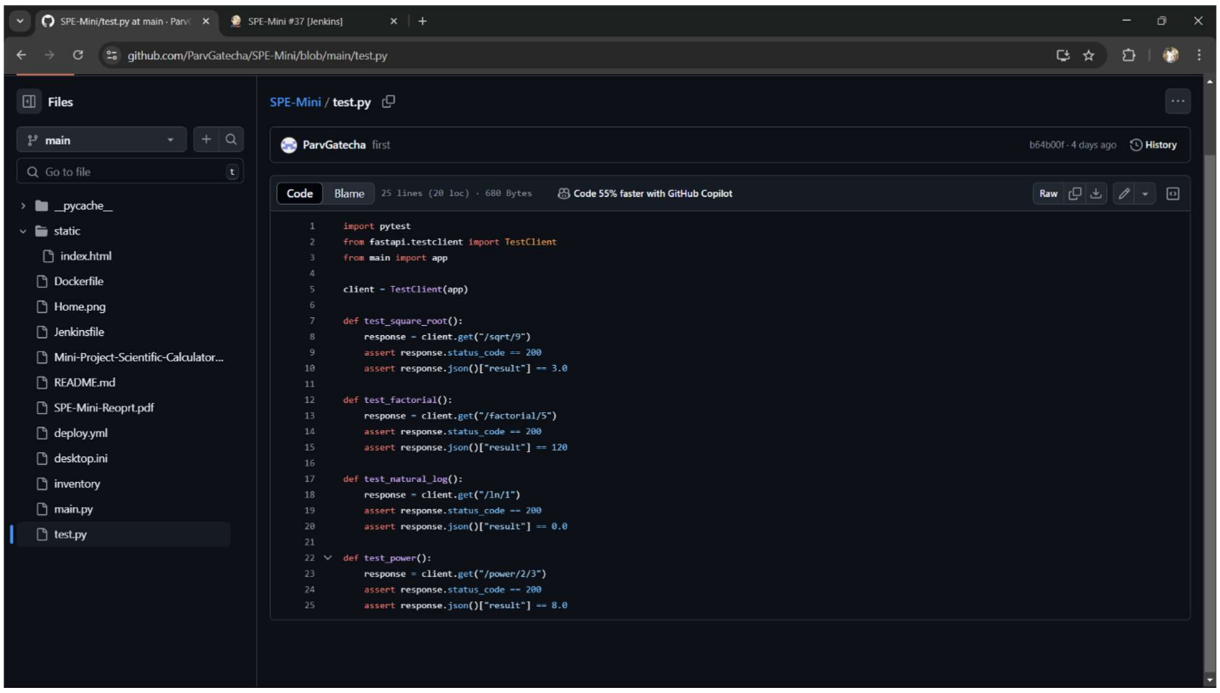
Adding Test Cases:

What is Unit Testing?

As the name implies, unit testing is the process of testing discrete code pieces. The smallest piece of code that can be retrieved from the system is denoted as a unit in this context. This little chunk could be a class, a method, or a line of code. Because smaller portions of code tend to run faster, the smaller the code, the better. Additionally, this offers a deeper understanding of the code's functionality.

Pytest:

pytest is a popular testing framework for Python. It is used to write simple as well as complex tests with ease. It supports features like fixtures, parameterized testing, assertions with better error reporting, and plugins for extended functionality. It’s widely used for unit, functional, and integration testing. To run tests, you just need to write functions starting with test_ and execute them using the pytest command.

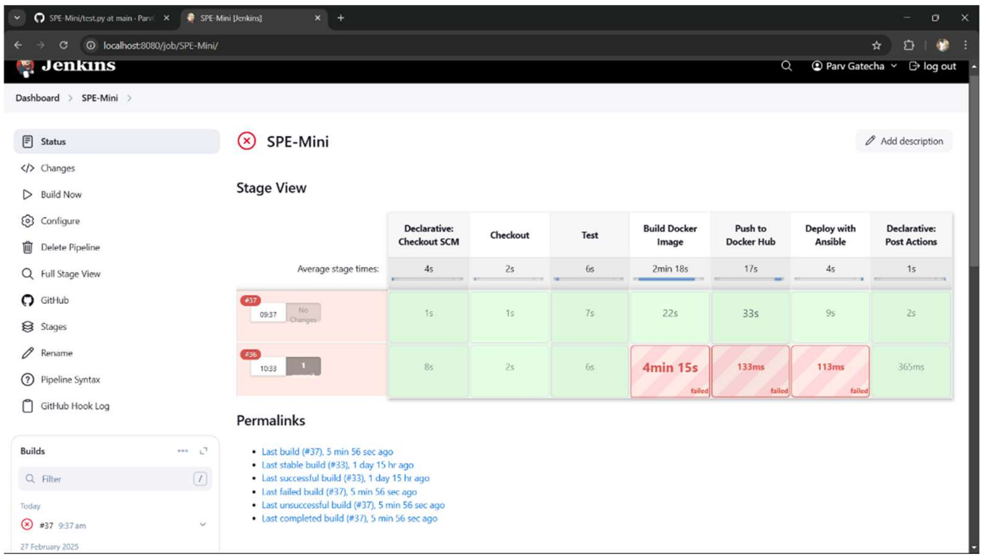


```python
import pytest
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def test_square_root():
    response = client.get("/sqrt/9")
    assert response.status_code == 200
    assert response.json()["result"] == 3.0

def test_factorial():
    response = client.get("/factorial/5")
    assert response.status_code == 200
    assert response.json()["result"] == 120

def test_natural_log():
    response = client.get("/ln/1")
    assert response.status_code == 200
    assert response.json()["result"] == 0.0

def test_power():
    response = client.get("/power/2/3")
    assert response.status_code == 200
    assert response.json()["result"] == 8.0
```

Add Test Cases

Run Project:

To run project first run pipeline in Jenkins



SPE-Mini

Stage View

	Declarative: Checkout SCM	Checkout	Test	Build Docker Image	Push to Docker Hub	Deploy with Ansible	Declarative: Post Actions
Average stage times:	4s	2s	6s	2min 18s	17s	4s	1s
#37 09:37 No Changes	1s	1s	7s	22s	33s	9s	2s
#36 10:33 1	8s	2s	6s	4min 15s failed	133ms failed	113ms failed	365ms

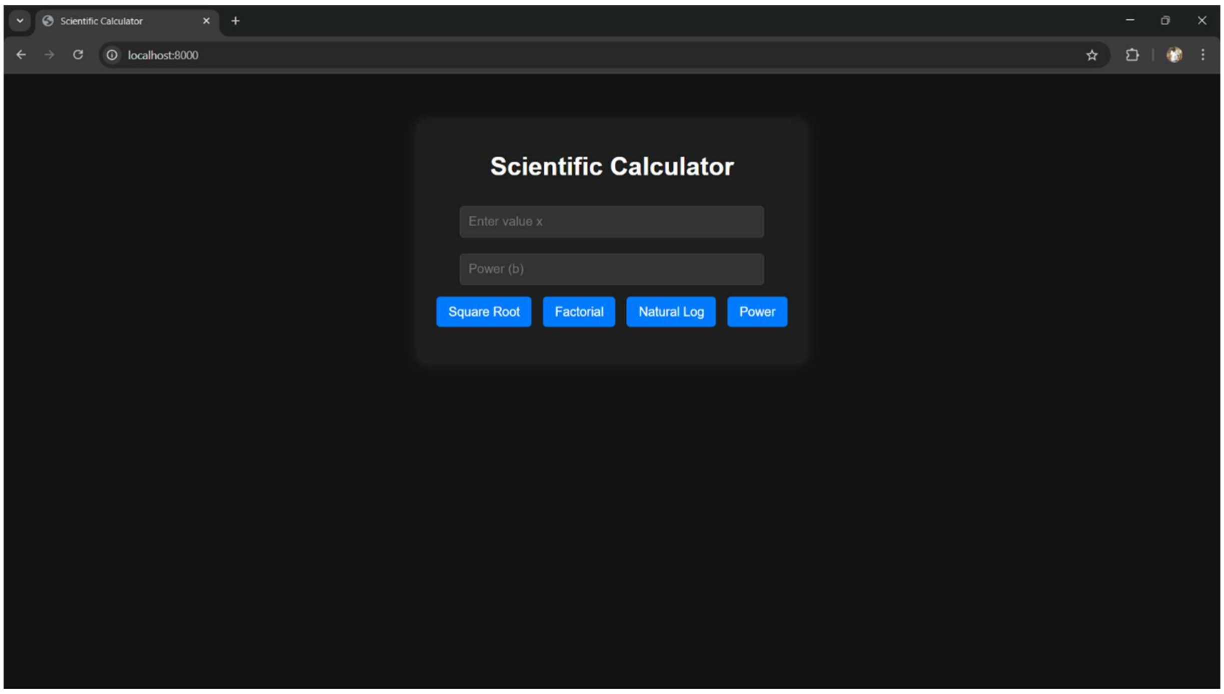
Permalinks

- Last build (#37), 5 min 56 sec ago
- Last stable build (#33), 1 day 15 hr ago
- Last successful build (#33), 1 day 15 hr ago
- Last failed build (#37), 5 min 56 sec ago
- Last unsuccessful build (#37), 5 min 56 sec ago
- Last completed build (#37), 5 min 56 sec ago

Pipeline View

Project Runs successfully and now our docker image is pulled from docker hub and run in local machine to check whether it runs or not we can access localhost:8000 on our local machine

So, right now our container is run and to execute this we use below command to enter the container and check whether our code is working properly or not.



GitHub Repo : https://github.com/ParvGatecha/SPE-Mini.git

DockerHub Repo: https://hub.docker.com/r/parvg/scientific-calculator