SPE Miniproject Report

Done by: MT2024108 Parv Gatecha

Quick links

- Github Repo
- Docker image on Dockerhub

Problem statement

Create a scientific calculator program with the following user menu driven operations:

- Square root function- √x
- Factorial function- x!
- Natural logarithm (base e)- ln(x)
- Powerfunction- xb

Understanding DevOps

DevOps is an **approach to software development and IT operations** that emphasizes **collaboration, automation, and efficiency** to streamline the software delivery lifecycle. It brings together **development (Dev)** and **operations (Ops)** teams to enable **continuous integration, continuous deployment (CI/CD), rapid releases, and improved system reliability**.

Core Principles of DevOps:

1. **Team Collaboration** – Encouraging seamless interaction between development and operations teams.
2. **Process Automation** – Implementing automation for testing, deployment, and infrastructure management.
3. **CI/CD Pipeline** – Ensuring frequent code integration and smooth automated deployments.
4. **Monitoring & Feedback** – Continuously tracking system performance and incorporating user feedback.
5. **Security Integration (DevSecOps)** – Embedding security measures throughout the development pipeline.

Why DevOps?

Traditional software development models, such as the **Waterfall approach**, often struggle with **delayed deployments, communication gaps, and inefficient workflows**. DevOps overcomes these challenges by:

1. **Accelerating Delivery** – Automated pipelines streamline software releases.
2. **Enhancing Quality** – Continuous testing minimizes defects before production.
3. **Ensuring Scalability & Reliability** – Cloud-native DevOps practices optimize performance.
4. **Fostering Collaboration** – Developers and IT teams work in sync for seamless operations.
5. **Reducing Costs** – Automation cuts down manual work, downtime, and operational expenses.

In modern **cloud computing, microservices, and AI-driven systems**, DevOps plays a vital role in maintaining agility and efficiency.

DevOps = Speed + Quality + Stability!

DevOps Tools

- **Git**
    - A distributed version control system (VCS) used for tracking changes, managing repositories, and collaborating through platforms like GitHub.
- **Docker**
    - A containerization tool that encapsulates applications into portable, lightweight containers.
- **Ansible**
    - A configuration management and automation tool for streamlining deployment and infrastructure setup.
- **Jenkins**
    - A CI/CD automation platform that facilitates building, testing, and deploying applications efficiently.
- **Python**
    - The core programming language used for developing the scientific calculator application.
- **FastAPI**
    - A high-performance web framework designed for building APIs, enabling seamless communication for the calculator.
- **Shell Scripting**
    - Utilized for automating deployment tasks and system configurations.

Project structure

Root Files

- **Dockerfile** - Defines instructions for building a Docker image.
- **Jenkinsfile** - Specifies CI/CD pipeline steps for Jenkins automation.
- **README.md** - Documentation file explaining the project setup and usage.

Source Code & Logic

- **main.py** - Main script containing the scientific calculator logic.
- **test.py** - Test script for verifying functionality (probably using `pytest`).

Deployment & Automation

- **deploy.yml** - Ansible playbook for automated deployment.
- **inventory** - Ansible inventory file listing target servers.

Dependencies

- **requirements.txt** - Lists Python dependencies required for the project.

Frontend Files

- **static/** - Contains static web files for the frontend.
    - **index.html** - The HTML file for the web-based calculator UI.

Compiled Files (Ignored in Version Control)

- **pycache/** - Stores compiled Python bytecode files for faster execution.
    - `calculator.cpython-312.pyc`
    - `calculator.cpython-313.pyc`
    - `test.cpython-313-pytest-8.3.4.pyc`

Project Overview

This project follows a well-structured approach to **development, deployment, and automation**:

Development & Testing

- The core logic is in `main.py`, and `test.py` ensures correctness.

Automation & Deployment

- **Jenkins** automates CI/CD pipelines via `Jenkinsfile`.

- **Ansible** manages deployment with `deploy.yml` and `inventory`.

Containerization

- The `Dockerfile` ensures a portable and reproducible environment.

User Interface

- The **web-based frontend** is in `static/index.html`.

This structured approach ensures **scalability, efficiency, and automation** in software development.

Code snippets

This Python code defines a simple **FastAPI-based scientific calculator** that provides various mathematical operations as API endpoints. The `@app.get` decorators define routes for different calculations:

- **`/static/index.html`** → Serves the `index.html` file for the frontend.
- **`/sqrt/{x}`** → Returns the square root of `x`.
- **`/factorial/{x}`** → Returns the factorial of `x`.
- **`/ln/{x}`** → Returns the natural logarithm of `x`.
- **`/power/{x}/{b}`** → Computes `x` raised to the power of `b`.

Each function extracts parameters from the URL, performs the operation using Python's `math` module, and returns the result as JSON.

This Python code contains unit tests for a FastAPI-based scientific calculator using a test client. Each function sends a GET request to an API endpoint and verifies the response:

- **`test_square_root()`**: Checks if `/sqrt/9` returns `3.0`.
- **`test_factorial()`**: Ensures `/factorial/5` returns `120`.
- **`test_natural_log()`**: Tests if `/ln/1` returns `0.0`.
- **`test_power()`**: Verifies `/power/2/3` returns `8.0`.

Each test asserts that:

1. The API response has a **status code of `200`**.
2. The computed result matches the expected value.

🛠️ Setup and Installation for local machine

1 Clone Repository

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

2 Running the FastAPI Server

Start the application using Uvicorn:

```
python main.py
```

The API will be available at http://localhost:8000.

3 Running Tests

To verify the API functionality, run the test suite using **pytest**:

```
pytest test.py
```

Docker build and run

```
docker build -t parvg/scientific-calculator .
```

This will use the Dockerfile in our directory to create a docker image

It can then be run with

```
❯ docker run -d --name sci-cal -p 8000:8000 parvg/scientific-calculator
```

Expose the port 8000 of the container to port 8000 of the local machine. The API will be available at http://localhost:8000.

Docker Image to DockerHub

Once the Image is built, we can push it to DockerHub to make it publically accessible.

1 Docker Login

Login into docker.

```
docker login
```

2 Tag the Image

```
docker tag parvg/scientific-calculator parvg/scientific-calculator:latest
```

3 Push Image to DockerHub

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

Deploy Using Ansible

Prerequisites

- Ansible installed on your local machine
- Docker installed on the target machine
- SSH access to the target machine
- Required Ansible collections installed (`community.docker`)

Steps to Deploy the Container

1. Set Up Ansible Environment

Ensure Ansible is using the correct Python interpreter:

```
vars:
  ansible_python_interpreter: /usr/bin/python
```

2. Ensure Docker is Installed

Check if Docker is installed, and install it if necessary:

```
- name: Install Docker
      apt:
        name: docker.io
        state: present
```

3. Start Docker (if Not Running)

Ensure Docker is running before proceeding:

```
- name: Start Docker (if not running)
  shell: open -a Docker
  ignore_errors: yes

- name: Wait for Docker to be available
  command: docker info
  register: docker_status
  until: docker_status.rc == 0
  retries: 10
  delay: 5
```

4. Pull the Latest Docker Image

Download the latest version of the scientific calculator Docker image:

```
- name: Pull the latest Docker image
  community.docker.docker_image:
    name: "parvg/scientific-calculator"
    source: pull
```

5. Remove Existing Container (if Running)

Ensure no previous instance of the container is running:

```
- name: Remove existing container (if running)
  community.docker.docker_container:
    name: sci-cal
    state: absent
```

6. Deploy the Scientific Calculator Container

Start a new container with the latest image:

```
- name: Run the container
  community.docker.docker_container:
    name: calculator-container
    image: "parvg/scientific-calculator:latest"
    state: started
    restart_policy: always
    ports:
      - "8000:8000" # Corrected port mapping for FastAPI
```

Automated Build, Test and Deployment using Jenkins

1. Checkout Code

Clones the latest code from the GitHub repository.

```
stage('Checkout') {
            steps {
                git branch: 'main', url: "https://github.com/ParvGatecha/SPE-
Mini.git"
            }
        }
```

2. Run Unit Tests

Executes unit tests using `pytest`.

```
stage('Test') {
            steps {
                sh 'pip install pytest httpx fastapi uvicorn'
                sh 'pytest test.py'
            }
        }
```

4. Build Docker Image

Builds a Docker image for the project.

```
stage('Build Docker Image') {
            steps {
                sh 'docker build -t parvg/scientific-calculator .'
            }
        }
```

5. Push Docker Image to Docker Hub

Tags and pushes the Docker image to the repository.

```
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'
                }
            }
        }
```

7. Clean Up Docker Images

Removes old Docker images to free space.

```
stage('Clean Up Docker Images') {
    steps {
        sh "docker rmi ${DOCKER_TAG} || true"  // Remove old images
        sh "docker rmi ${DOCKER_IMAGE_NAME} || true"
    }
}
```

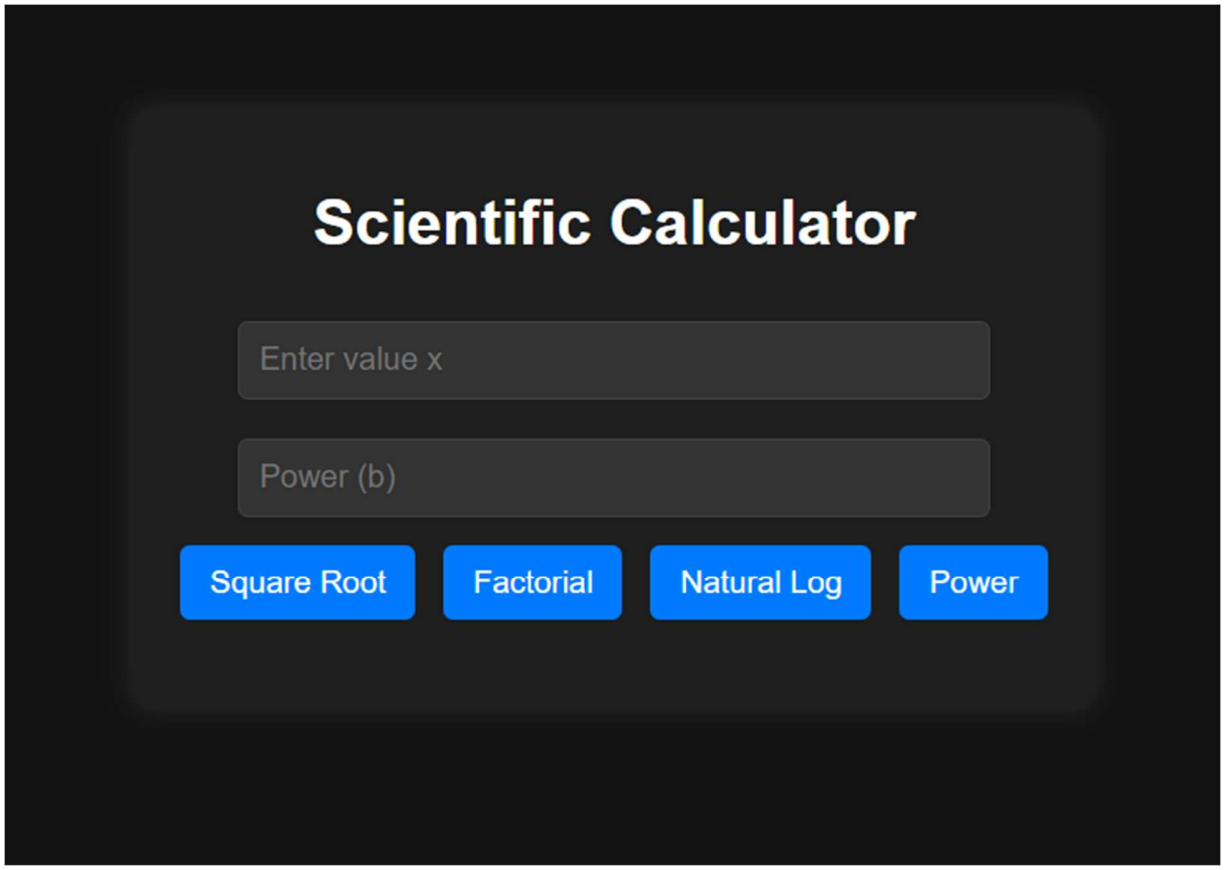
8. Deploy Using Ansible

Deploys the containerized application using Ansible.

```
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'
            }
        }
```

Deployment Complete 🎉

The scientific calculator container is now up and running! We can access it at http://localhost:8000.


Scientific Calculator
Enter value x
Power (b)
Square Root
Factorial
Natural Log
Power