CS 816 - Software Production Engineering
Mini Project Report

Scientific Calculator with DevOps

Varnit Mittal
`Varnit.Mittal@iiitb.ac.in`
Roll Number: **IMT2022025**
Github
Docker Front-end
Docker Back-end

October 11, 2025

Contents

1 Introduction to DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. DevOps is about breaking down silos between development and operations teams, encouraging a culture of collaboration and shared responsibility.

1.1 Why DevOps?

The primary goal of DevOps is to improve collaboration, automate processes, and increase the speed and reliability of software delivery. By automating the build, test, and deployment pipeline, organizations can:

- **Increase Deployment Frequency:** Release software more often.
- **Lower Failure Rate:** Reduce the number of failed deployments.
- **Faster Time to Market:** Deliver features to users more quickly.
- **Improve Reliability:** Ensure the stability and performance of the application.

2 Project Overview

This project implements a web-based scientific calculator and a complete DevOps pipeline to automate its lifecycle from code commit to deployment.

2.1 Application Architecture

The application follows a two-tier architecture:

- **Frontend:** A user interface built with **React**, allowing users to perform calculations. It runs on port **5173**.
- **Backend:** A REST API built with **Spring Boot** that handles the business logic for the mathematical operations (Square Root, Factorial, Natural Logarithm, Power). It runs on port **8081**.

2.2 DevOps Toolchain

The following tools were used to construct the CI/CD pipeline:

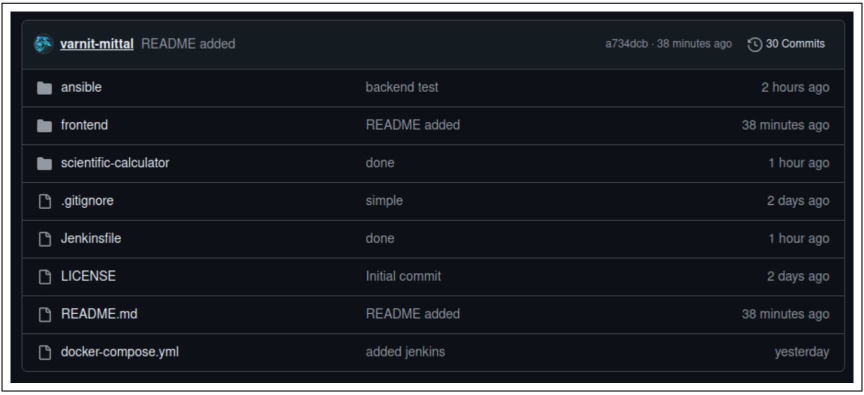
- **Source Control:** Git & GitHub
- **Continuous Integration:** Jenkins
- **Containerization:** Docker & Docker Compose
- **Artifact Registry:** Docker Hub
- **Configuration Management & Deployment:** Ansible

3 DevOps Pipeline Implementation

3.1 Step 1: Source Control Management (GitHub)

Source Control Management (SCM) is crucial for tracking and managing changes to source code. **Git** is the distributed version control system used, and **GitHub** hosts the central repository.

The repository was structured to separate concerns, with folders for the backend (‘scientific-calculator‘), frontend, and Ansible configuration. Key files like ‘Jenkinsfile’ and ‘docker-compose.yml’ are at the root for easy access by the CI/CD system.



varnit-mittal README added a734dcb · 38 minutes ago 30 Commits

ansible	backend test	2 hours ago
frontend	README added	38 minutes ago
scientific-calculator	done	1 hour ago
.gitignore	simple	2 days ago
Jenkinsfile	done	1 hour ago
LICENSE	Initial commit	2 days ago
README.md	README added	38 minutes ago
docker-compose.yml	added jenkins	yesterday

Figure 1: GitHub Repository Structure.

3.2 Step 2: Containerization (Docker)

Containerization packages an application and its dependencies into an isolated unit called a container. **Docker Compose** is a tool for defining and running multi-container applications.

The ‘docker-compose.yml’ file defines two services: ‘backend’ and ‘frontend’. It defines how to build their respective images, maps the ports, and connects them via a shared network.

```yaml
version: "3.9"
services:
  backend:
    build: ./scientific-calculator
    image: varn03/sci-calc-backend:latest
    container_name: sci-calc-backend
    ports:
      - "8081:8081"
    networks:
      - spe-net
  frontend:
    build: ./frontend
    image: varn03/sci-calc-frontend:latest
    container_name: sci-calc-frontend
    ports:
      - "5173:80"
    depends_on:
      - backend
    networks:
      - spe-net
networks:
  spe-net:
    driver: bridge
```

Listing 1: docker-compose.yml

3.3 Step 3: Continuous Integration (Jenkins)

Jenkins is an open-source automation server used to automate the non-human part of the software development process. It orchestrates the entire pipeline from code checkout to deployment.

A declarative pipeline was created using a ‘Jenkinsfile’. This “pipeline-as-code” approach allows us to version control our CI/CD process alongside our application code.

3.3.1 Jenkins Pipeline Stages

1. **Checkout:** Clones the source code from the GitHub repository.
2. **Build Docker Image:** Uses ‘docker-compose build’ to create the frontend and backend images as defined in the ‘docker-compose.yml’ file.
3. **Push to Docker Hub:** Logs into Docker Hub using stored credentials and pushes the newly built images to the registry. This makes the images available for deployment.
4. **Deploy with Ansible:** Executes the Ansible playbook to pull the latest images from Docker Hub and deploy the application on the target machine.
5. **Post Actions:** Sends an email notification indicating the success or failure of the pipeline.

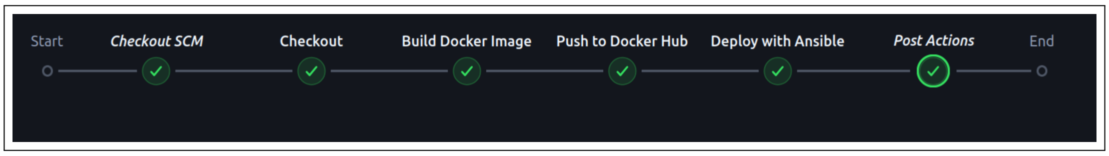


Figure 2: Jenkins view of a Successful Pipeline Run.

```
pipeline {
    agent any

    environment {
        DOCKERHUBREPO = ’your-dockerhub-username/spe-sci-calc’
        ANSIBLEPLAYBOOK = "${WORKSPACE}/ansible/deploy.yml"
        ANSIBLEINVENTORY = "${WORKSPACE}/ansible/inventory.ini"
        DOCKERCREDENTIALSID = ’dockerhub-creds’
        GITREPOURL =
   ’https://github.com/varnit-mittal/SPE-sci-calc.git’
        MAILRECIPIENTS = ’varnit03mittal@gmail.com’
    }

    stages {
        stage(’Checkout’) { /* ... */ }
        stage(’Build Docker Image’) { /* ... */ }
        stage(’Push to Docker Hub’) { /* ... */ }
        stage(’Deploy with Ansible’) { /* ... */ }
    }

    post {
        success { /* ... */ }
        failure { /* ... */ }
    }
}
```

Listing 2: Jenkinsfile

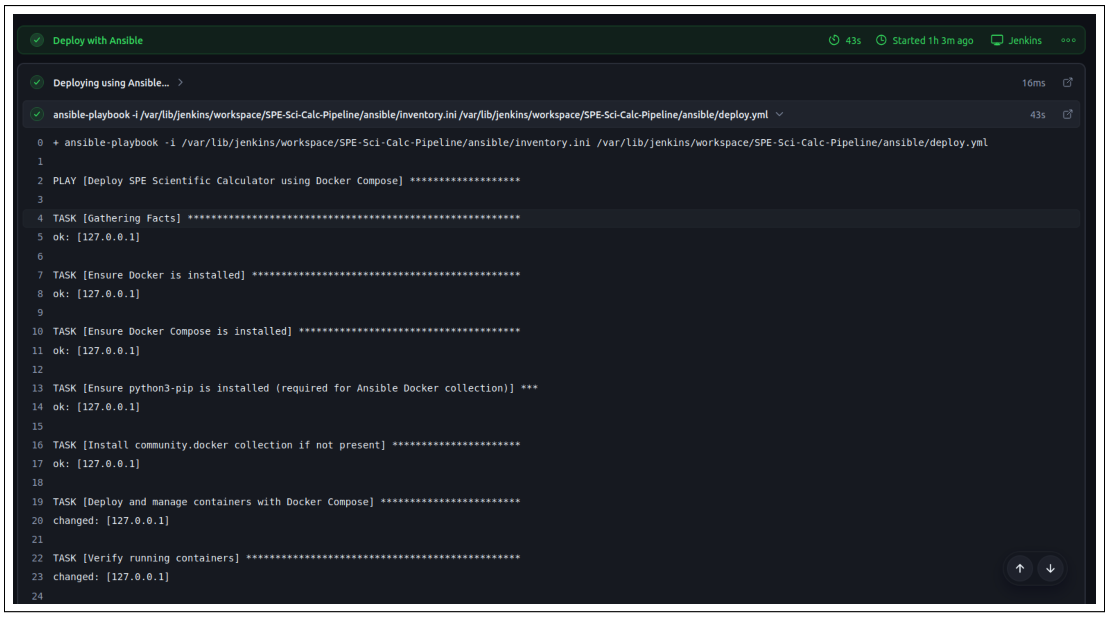
3.4 Step 4: Deployment (Ansible)

Ansible is an open-source tool for configuration management, application deployment, and task automation. It uses a human-readable language, YAML, to define automation jobs in "playbooks".

The 'deploy.yml' playbook handles the deployment to the local machine.

3.4.1 Ansible Playbook Tasks

1. **Pre-tasks:** Ensures that Docker, Docker Compose, and other dependencies are installed on the target host.
2. **Deploy Containers:** Uses the 'community.docker.docker_compose' module to pull the latest images and run the application. The 'pull: true' parameter ensures we always use the newest images from Docker Hub.
3. **Verify Deployment:** Runs 'docker ps' to verify that the containers are up and running, and displays the output in the Jenkins logs.



```
ansible-playbook -i /var/lib/jenkins/workspace/SPE-Sci-Calc-Pipeline/ansible/inventory.ini /var/lib/jenkins/workspace/SPE-Sci-Calc-Pipeline/ansible/deploy.yml
```

```
+ ansible-playbook -i /var/lib/jenkins/workspace/SPE-Sci-Calc-Pipeline/ansible/inventory.ini /var/lib/jenkins/workspace/SPE-Sci-Calc-Pipeline/ansible/deploy.yml

PLAY [Deploy SPE Scientific Calculator using Docker Compose] *******************

TASK [Gathering Facts] *********************************************************
ok: [127.0.0.1]

TASK [Ensure Docker is installed] **********************************************
ok: [127.0.0.1]

TASK [Ensure Docker Compose is installed] **************************************
ok: [127.0.0.1]

TASK [Ensure python3-pip is installed (required for Ansible Docker collection)] ***
ok: [127.0.0.1]

TASK [Install community.docker collection if not present] **********************
ok: [127.0.0.1]

TASK [Deploy and manage containers with Docker Compose] ************************
changed: [127.0.0.1]

TASK [Verify running containers] ***********************************************
changed: [127.0.0.1]

```

Figure 3: Ansible Playbook Execution Log in Jenkins Console.

```yaml
---
- name: Deploy SPE Scientific Calculator using Docker Compose
  hosts: local
  become: true
  vars:
    project_dir: "{{ lookup('env','WORKSPACE') }}"

  pre_tasks:
    - name: Ensure Docker is installed
      apt: { name: docker.io, state: present }
    # ... more pre-tasks

  tasks:
    - name: Deploy and manage containers with Docker Compose
      community.docker.docker_compose:
        project_src: "{{ project_dir }}"
        state: present
        pull: true
        restarted: true
      become: true

    - name: Verify running containers
      shell: docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"
      register: docker_ps_output
    # ... more tasks
```

Listing 3: ansible/deploy.yml

4 Conclusion

This project successfully demonstrated the implementation of a full CI/CD pipeline for a multi-tier web application. By integrating Git, Jenkins, Docker, and Ansible, the entire process of building, testing, and deploying the Scientific Calculator was automated. This DevOps approach significantly improves efficiency, reduces the risk of manual errors, and accelerates the delivery of new features.

The final result is a robust, repeatable, and scalable system where any change pushed to the GitHub repository automatically triggers the pipeline, deploying the updated application without manual intervention.

5 Project Links

- **GitHub Repository:** https://github.com/varnit-mittal/SPE-sci-calc
- **Docker Hub Repository:** https://hub.docker.com/u/varn03