

DevOps Project Report: Scientific Calculator

Harsh Dhruv
Roll Number: IMT2022008

October 10, 2025

1 Introduction: What and Why of DevOps

What is DevOps?

DevOps is a set of practices combining software development (Dev) and IT operations (Ops), aiming to shorten the development lifecycle and provide continuous delivery with high software quality.

Why DevOps?

- Automates build, test, and deployment pipelines
- Improves collaboration between development and operations
- Enables faster feedback loops and more reliable software releases
- Reduces manual errors and increases deployment frequency

2 Tools Used

Stage	Tool	Purpose
Source Control Management	GitHub	Stores and manages project source code
Build Automation	Maven	Compiles and packages Java application
Continuous Integration / Delivery	Jenkins	Automates build, test, and deployment
Containerization	Docker	Packages the application with dependencies
Configuration Management	Ansible	Automates deployment of containers
Container Registry	DockerHub	Stores and shares Docker images

3 Step-by-Step Explanation

3.1 Calculator.java File Explanation

The `Calculator.java` file implements a simple scientific calculator supporting four mathematical operations: square root, factorial, natural logarithm, and power.

```
public class Calculator {  
    // Square root  
    public double sqrt(double x) {  
        if (x < 0) throw new IllegalArgumentException("sqrt: negative input");  
        return Math.sqrt(x);  
    }  
    // Factorial  
    public BigInteger factorial(int n) {  
        if (n < 0) throw new IllegalArgumentException("factorial: negative input");  
        BigInteger res = BigInteger.ONE;  
        for (int i = 2; i <= n; i++) res = res.multiply(BigInteger.valueOf(i));  
        return res;  
    }  
    // Natural logarithm  
    public double ln(double x) {  
        if (x <= 0) throw new IllegalArgumentException("ln: input must be > 0");  
        return Math.log(x);  
    }  
    // Power x^b  
    public double pow(double x, double b) {  
        return Math.pow(x, b);  
    }  
}
```

Figure 1: Calculator

- **sqrt(double x):** Calculates the square root of a number using `Math.sqrt()`. Throws an exception for negative inputs.
- **factorial(int n):** Computes the factorial of a non-negative integer using a loop and `BigInteger` to handle large results.
- **ln(double x):** Returns the natural logarithm ($\ln x$) using `Math.log()`. Only accepts positive inputs.
- **pow(double x, double b):** Calculates x^b using the built-in `Math.pow()` method for exponentiation.

Each function includes basic input validation and uses Java's standard math library to ensure accuracy.

3.2 JUnit Test File: CalculatorTest.java

This file contains automated unit tests for the `Calculator` class using JUnit 5. Each test case verifies the correctness and exception handling of the calculator functions.

```

import static org.junit.jupiter.api.Assertions.*;

class CalculatorTest {

    Calculator calc = new Calculator();

    @Test
    void testSqrtPositive() {
        assertEquals(4.0, calc.sqrt(16.0), 1e-9);
    }

    @Test
    void testSqrtNegative() {
        assertThrows(IllegalArgumentException.class, () -> calc.sqrt(-1.0));
    }

    @Test
    void testFactorialSmall() {
        assertEquals(BigInteger.valueOf(120), calc.factorial(5));
    }

    @Test
    void testLn() {
        assertEquals(Math.log(2.0), calc.ln(2.0), 1e-9);
    }

    @Test
    void testPow() {
        assertEquals(8.0, calc.pow(2.0, 3.0), 1e-9);
    }
}

```

Figure 2: Junit tests

Explanation:

- `testSqrtPositive()`: Tests correct computation of square root. - `testSqrtNegative()`: Ensures exception is thrown for negative inputs. - `testFactorialSmall()`: Verifies factorial calculation for small numbers. - `testLn()`: Checks correctness of natural logarithm calculation. - `testPow()`: Validates power function output.

3.3 Dockerfile Explanation

```
# 1st Stage: Build using Maven with JDK 21

FROM maven:3.9.9-eclipse-temurin-21 AS build (last pushed 4 months ago)
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests

# 2nd Stage: Run with JDK 21

FROM eclipse-temurin:21-jdk-alpine (last pushed 16 hours ago)
WORKDIR /app
COPY --from=build /app/target/miniproject-1.0-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Figure 3: Dockerfile

The Dockerfile defines how the scientific calculator application is built and run inside a container. It uses a multi-stage build to make the image smaller and more efficient.

- Stage 1 (Build Stage): This stage uses the image `maven:3.9.9-eclipse-temurin-21` to compile the Java project. The `pom.xml` file is first copied to download dependencies using the command `mvn dependency:go-offline`. Then the source code is copied and the project is packaged using `mvn clean package -DskipTests`.
- Stage 2 (Runtime Stage): This stage uses the lightweight image `eclipse-temurin:21-jdk-alpine` to run the built application. The jar file created in the build stage is copied into the container and executed using the command `java -jar app.jar`.

This setup separates the build and runtime environments, reduces image size, and ensures faster and more secure deployment.

3.4 Jenkinsfile Explanation

The Jenkinsfile defines a declarative pipeline that automates building, testing, containerizing, and deploying the scientific calculator application. It also integrates email notifications for build results.

3.4.1 Agent and Environment Variables

```
pipeline {
    agent any

    environment {
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-credentials-id')
        DOCKERHUB_REPO = 'harsh4710/scientific-calculator'
        NOTIFY_EMAIL = 'Harshdhruv889@gmail.com'
    }
}
```

Figure 4: Jenkins file

The pipeline runs on any available Jenkins agent. It defines environment variables such as DockerHub credentials, the Docker repository name, and the recipient email address for notifications.

3.4.2 Stage 1: Checkout

```
stages {
    stage('Checkout') {
        steps {
            echo "Checking out repository..."
            git branch: 'main', url: 'https://github.com/hrdhruv/Calculator'
        }
    }
}
```

Figure 5: Checkout stage

In this stage, Jenkins pulls the latest source code from the main branch of the GitHub repository using the git command. This ensures that the most recent code version is used for the build process.

3.4.3 Stage 2: Build and Test

```
stage('Build & Test') {
    steps {
        echo "Building and testing application..."
        sh 'mvn clean package'
    }
}
```

Figure 6: Build and test stage

The project is compiled and tested using Maven. The command 'mvn clean package' cleans any previous builds and packages the application into a jar file. If any tests fail, the pipeline stops and marks the build as failed.

3.4.4 Stage 3: Build Docker Image

```
stage('Build Docker Image') {
    steps {
        echo "Building Docker image..."
        script {
            sh "docker build -t ${DOCKERHUB_REPO}:${BUILD_NUMBER} ."
        }
    }
}
```

Figure 7: Docker build stage

After successful compilation, a Docker image of the application is built using the Dockerfile present in the repository. The image is tagged with the current Jenkins build number for version tracking.

3.4.5 Stage 4: Push to DockerHub

```
stage('Push to DockerHub') {
    steps {
        echo "Pushing image to DockerHub..."
        script {
            sh "echo ${DOCKERHUB_CREDENTIALS_PSW} | docker login -u ${DOCKERHUB_CREDENTIALS_USR} --password-stdin"
            sh "docker push ${DOCKERHUB_REPO}:${BUILD_NUMBER}"
            sh "docker tag ${DOCKERHUB_REPO}:${BUILD_NUMBER} ${DOCKERHUB_REPO}:latest"
            sh "docker push ${DOCKERHUB_REPO}:latest"
        }
    }
}
```

Figure 8: Docker hub stage

In this stage, Jenkins logs into DockerHub using stored credentials and pushes the newly built image to the specified DockerHub repository. It also tags the image as 'latest' to mark it as the current stable version.

3.4.6 Stage 5: Deploy with Ansible

```
stage('Deploy with Ansible') {  
    steps {  
        echo "Deploying container using Ansible..."  
        script {  
            sh "ansible-playbook -i inventory.ini playbook.yml"  
        }  
    }  
}
```

Figure 9: Ansible stage

This stage automates the deployment of the container using Ansible. Jenkins executes the Ansible playbook (playbook.yml) with the specified inventory file (inventory.ini) to deploy the Docker container on the target system.

3.4.7 Post Section: Notifications

```
post {  
    success {  
        echo "Build ${BUILD_NUMBER} completed successfully!"  
        mail to: "${NOTIFY_EMAIL}",  
            subject: "SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER}",  
            body: ""Build SUCCESSFUL  
  
Project: ${env.JOB_NAME}  
Build Number: ${env.BUILD_NUMBER}  
Status: SUCCESS  
  
Check details at: ${env.BUILD_URL}  
""  
    }  
    failure {  
        echo "Build ${BUILD_NUMBER} failed!"  
        mail to: "${NOTIFY_EMAIL}",  
            subject: "FAILURE: ${env.JOB_NAME} #${env.BUILD_NUMBER}",  
            body: ""Build FAILED  
  
Project: ${env.JOB_NAME}  
Build Number: ${env.BUILD_NUMBER}  
Status: FAILED  
  
Check console output: ${env.BUILD_URL}console  
""  
    }  
    unstable {  
        echo "Build ${BUILD_NUMBER} is unstable!"  
        mail to: "${NOTIFY_EMAIL}",  
            subject: "UNSTABLE: ${env.JOB_NAME} #${env.BUILD_NUMBER}",  
            body: ""Build UNSTABLE  
  
Project: ${env.JOB_NAME}  
Build Number: ${env.BUILD_NUMBER}  
Status: UNSTABLE  
  
Review console output: ${env.BUILD_URL}console  
""  
    }  
}
```

Figure 10: Email notification stage

After the pipeline completes, Jenkins automatically sends email notifications: 1. On success, it sends a message with build details and a link to the build page. 2. On failure,

it sends a failure message with a link to the console output for debugging. 3. On unstable builds, it notifies that the build passed with issues.

This Jenkinsfile fully automates the CI/CD workflow, ensuring smooth integration, deployment, and monitoring of the project.

3.5 Ansible Inventory File: `inventory.ini`

The `inventory.ini` file defines the target hosts where Ansible will execute tasks. In this project, we are deploying the Docker container locally. The content of the file is:

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

Explanation:

`local` defines a group of hosts named `local`.

- `localhost` specifies that the tasks will run on the same machine where Ansible is executed.
- `ansible_connection=local` tells Ansible to use a local connection instead of SSH.

This setup is useful for testing and deployment on the local machine without requiring a remote server.

3.6 Ansible Playbook: `playbook.yml`

The `playbook.yml` file automates the deployment of the Scientific Calculator Docker container. Its content is:

```
- name: Deploy Calculator App
  hosts: local

  tasks:
    - name: Remove old containers
      docker_container:
        name: scientific-calculator
        state: absent
        force_kill: yes

    - name: Deploy Calculator App
      docker_container:
        name: scientific-calculator
        image: harsh4710/scientific-calculator
        state: started
        interactive: yes
        detach: yes
```

Explanation:

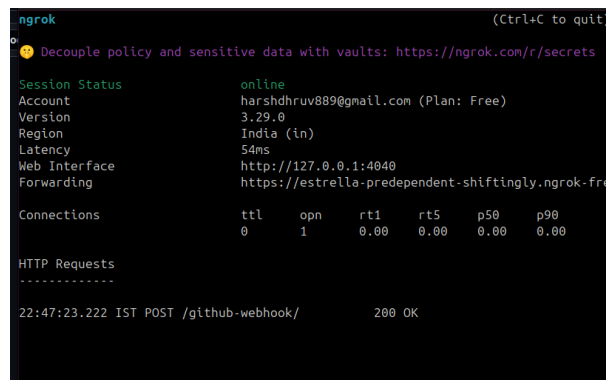
- `name: Deploy Calculator App` defines the playbook name. - `hosts: local` specifies that the playbook will run on the local group defined in the inventory. - `Remove old containers`

task removes any previously running container named scientific-calculator using state: absent and forcekill: yes. - Deploy Calculator App task starts a new container using the Docker image harsh4710/scientific-calculator. - interactive: yes and detach: yes ensure the container runs in the background but remains interactive.

This playbook ensures that old containers are removed before deploying the latest version, making deployment clean and repeatable.

3.7 Ngrok Tunnel for GitHub Webhook

Ngrok is used to expose the local Jenkins server to the internet so that GitHub webhooks can trigger builds. The session output is as follows:



```
ngrok (Ctrl+C to quit)
🔑 Decouple policy and sensitive data with vaults: https://ngrok.com/r/secrets

Session Status      online
Account             harshdhruv889@gmail.com (Plan: Free)
Version             3.29.0
Region              India (in)
Latency              54ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://estrella-predependent-shiftingly.ngrok-free

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      1      0.00   0.00   0.00   0.00

HTTP Requests
-----
22:47:23.222 IST POST /github-webhook/ 200 OK
```

Figure 11: ngrok

Explanation:

- The **Session Status** shows that the tunnel is active and online.
- **Web Interface** provides a local dashboard to monitor traffic.
- **Forwarding** gives the public URL that GitHub uses to trigger Jenkins builds.
- The **HTTP Requests** section logs incoming webhook events and their response status.
- Ngrok allows testing webhooks without deploying the Jenkins server publicly.

4 Final Application Output

The Scientific Calculator application supports addition, subtraction, multiplication, and division.


```

harsh-d@harsh-d-L0Q-15ARP9:~/Desktop/mini_project$ docker build -t scientific-calculator .
[+] Building 2.3s (17/17) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 430B
=> [internal] load metadata for docker.io/library/eclipse-temurin:21-jdk-alpine
=> [internal] load metadata for docker.io/library/maven:3.9.9-eclipse-temurin-21
=> [auth] library/eclipse-temurin:pull token for registry-1.docker.io
=> [auth] library/maven:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [stage-1 1/3] FROM docker.io/library/eclipse-temurin:21-jdk-alpine@sha256:89517925fa675c6c4b778bee7c44d38a7763212741b0d6fca5a5103caab21a97
=> [build 1/6] FROM docker.io/library/maven:3.9.9-eclipse-temurin-21@sha256:3a4ab3276a087b7276f79cae96b1af04f53731bec53fb2e651aca79e4b10211e
=> [internal] load build context
=> => transferring context: 276B
=> CACHED [build 2/6] WORKDIR /app
=> CACHED [build 3/6] COPY pom.xml .
=> CACHED [build 4/6] RUN mvn dependency:go-offline
=> CACHED [build 5/6] COPY src ./src
=> CACHED [build 6/6] RUN mvn clean package -DskipTests
=> CACHED [stage-1 2/3] WORKDIR /app
=> [stage-1 3/3] COPY --from=build /app/target/miniproject-1.0-SNAPSHOT.jar app.jar
=> => exporting layers
=> => writing image sha256:3d0849dcb405a8d724c252086d443570f165bedfa5cedb7f60284b75e1c78435
=> => naming to docker.io/library/scientific-calculator
harsh-d@harsh-d-L0Q-15ARP9:~/Desktop/mini_projects$

```

Figure 12: docker build

```

harsh-d@harsh-d-L0Q-15ARP9:~/Desktop/mini_project$ docker pull harsh4710/scientific-calculator:latest
latest: Pulling from harsh4710/scientific-calculator
Digest: sha256:f671dclbda7754facf300fc0c80c171ae04e39cea765d54e2457f407865cb5ad
Status: Image is up to date for harsh4710/scientific-calculator:latest
docker.io/harsh4710/scientific-calculator:latest
harsh-d@harsh-d-L0Q-15ARP9:~/Desktop/mini_project$ docker run -it --rm harsh4710/scientific-calculator:latest

Scientific Calculator - Menu
1) Square root
2) Factorial
3) Natural log(ln x)
4) Power(x^b)
5) Exit
Choose option: 1
Enter x: 3
√3.0 = 1.7320508075688772

Scientific Calculator - Menu
1) Square root
2) Factorial
3) Natural log(ln x)
4) Power(x^b)
5) Exit
Choose option: 4
Enter x: 4
Enter b: 3
4.0^3.0 = 64.0

Scientific Calculator - Menu
1) Square root
2) Factorial
3) Natural log(ln x)
4) Power(x^b)
5) Exit
Choose option: 5
Exiting...
harsh-d@harsh-d-L0Q-15ARP9:~/Desktop/mini_project$


```

Figure 13: docker run

	Declarative: Checkout SCM	Checkout	Build & Test	Build Docker Image	Push to DockerHub	Deploy with Ansible	Declarative: Post Actions
Average stage times: (full run time: ~54s)	1s	1s	1s	3s	26s	2s	1s
#17 Oct 09 22:47 1 commit	1s	1s	2s	17s	33s	3s	4s
#16 Oct 08 19:25 1 commit	1s	847ms	2s	878ms	18s	3s	3s
#15 Oct 08 19:24 1 commit	1s	923ms	2s	879ms	18s	3s	3s
#14 Oct 05 18:49 2 commits	2s	1s	2s	866ms	1min 4s	6s	78ms
#13 Oct 05 18:47 No Changes							
#12 Oct 05 18:41 1 commit	2s	870ms	2s	1s	49s	912ms failed	67ms
#11 Oct 05 18:37 No Changes	957ms	988ms failed	67ms failed	64ms failed	58ms failed	63ms failed	83ms
#10 Oct 05 18:26 4 commits	2s	6s failed	127ms failed	66ms failed	71ms failed	75ms failed	96ms

Figure 14: pipeline view

SUCCESS: Calculator_mini #17 Inbox x

 **address not configured yet** <harshdhruv889@gmail.com>
to me ▾

Build **SUCCESSFUL**

Project: Calculator_mini
Build Number: 17
Status: SUCCESS

Check details at: https://estrella-predependent-shiftingly.ngrok-free.dev/job/Calculator_mini/17/

↩ Reply
➦ Forward
😊

Figure 15: email notification

```

harsh-d@harsh-d-L00-15ARP9:~/Desktop/mini_projects$ ansible-playbook -i inventory.ini playbook.yml
PLAY [Deploy Calculator App] *****
TASK [Gathering Facts] *****
ok: [localhost]

TASK [Remove old containers] *****
changed: [localhost]

TASK [Deploy Calculator App] *****
changed: [localhost]

PLAY RECAP *****
localhost : ok=3 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

harsh-d@harsh-d-L00-15ARP9:~/Desktop/mini_projects$

```

Figure 16: ansible output

5 Public Links

- GitHub Repository: <https://github.com/hrdhruv/Calculator>
- DockerHub Repository: <https://hub.docker.com/r/harsh4710/scientific-calculator>

6 Conclusion

The DevOps pipeline successfully automated the build, test, Docker image creation, deployment, and ensured consistent application execution.