

# CS 816 - Software Production Engineering: Mini Project Report

**Project Title:** Scientific Calculator with End-to-End CI/CD Pipeline

**Name:** Aayush Bhargav

**Roll Number:** IMT2022089

**GitHub Repository Link:** [https://github.com/Aayush-Bhargav/SPE\\_MiniProject](https://github.com/Aayush-Bhargav/SPE_MiniProject)

**DockerHub Repository Link:** <https://hub.docker.com/u/aayushbhargav57>

## 1. Introduction: What and Why of DevOps?

### What is DevOps?

DevOps (Development and Operations) is a modern approach that integrates software development and IT operations to enable faster, more reliable software delivery. It focuses on automating the entire software life cycle — from code integration and testing to deployment and monitoring — while fostering close collaboration between developers, testers, and operations teams.

Rather than functioning as isolated departments, DevOps encourages shared ownership and continuous feedback throughout the development pipeline. This collaboration ensures that code changes move seamlessly from development to production while maintaining stability and quality. DevOps is both a **culture** and a **set of practices** that emphasize automation, continuous integration and deployment (CI/CD), monitoring, and rapid feedback loops. The ultimate goal is to deliver high-quality software quickly, consistently, and aligned with user needs.

### Why Use DevOps?

Organizations adopt DevOps because it addresses several long-standing challenges in traditional software delivery models. The primary benefits include:

- **Speed and Agility:**  
DevOps enables faster and more frequent releases through automation and continuous integration. This agility allows teams to respond quickly to customer feedback and changing market demands.

- **Reliability:**  
Automated testing, continuous monitoring, and early error detection ensure that software is stable before reaching users. This reduces the likelihood of failures in production environments.
  - **Scalability:**  
With tools like Infrastructure as Code (IaC) and containerization, DevOps allows teams to scale systems efficiently. Environments can be replicated and configured automatically, minimizing human error.
  - **Collaboration and Culture:**  
DevOps bridges the gap between development, testing, and operations teams. It builds a culture of shared responsibility, where all members are collectively accountable for performance, stability, and user satisfaction.
  - **Continuous Feedback:**  
Through automated monitoring and logging, teams receive real-time insights about system performance and user behavior. This feedback drives constant improvement in both code and processes.
- 

## Why DevOps is Preferred Over Traditional Models

Traditional models like **Waterfall** and even **Agile** have limitations that DevOps overcomes. In the **Waterfall model**, the software development process is linear and sequential. Each phase — from requirement gathering to deployment — happens one after the other, leading to long release cycles and delayed feedback. Problems often surface only at the final stages, making them expensive and time-consuming to fix.

The **Agile model** improved on this by introducing iterative development and faster delivery through sprints. However, Agile primarily focuses on collaboration and flexibility within the development team. It still often leaves a gap between development and operations — meaning code may be ready faster, but deployment and maintenance still involve manual steps and delays.

**DevOps goes beyond Agile** by extending the principles of iteration, automation, and collaboration to the entire software delivery pipeline. It eliminates the friction between development and operations by using automated deployment, continuous monitoring, and shared workflows. As a result, updates can be released multiple times a day rather than a few times a month. Moreover, since testing and deployment are automated, quality assurance becomes part of the ongoing process rather than a final checkpoint.

In essence, DevOps combines the development speed of Agile with the operational stability that traditional models aimed for. It transforms software delivery into a continuous, integrated process — reducing time to market while improving reliability and scalability.

## 2. Problem Statement and Tools Used

### 2.1 Problem Statement

The goal of this project was to implement a command-line Scientific Calculator program in Java, featuring the following menu-driven operations:

1. Square root function ( $\text{sqrt}(x)$ )
2. Factorial function ( $!x$ )
3. Natural logarithm ( $\ln(x)$ )
4. Power function ( $x^b$ )

This application must be deployed via a complete **8-step CI/CD pipeline**.

### 2.2 Tools Used

Category	Tool	Description
Language	Java 17	Programming language used for the core application logic.
SCM	GitHub	Source Code Management and collaboration.
Build Tool	Apache Maven	Used to compile, package (JAR), and run unit tests.
Testing	JUnit 5	Framework used for writing and executing unit tests.
CI/CD Orchestration	Jenkins	Automation server to orchestrate the entire pipeline via the <a href="#">Jenkinsfile</a> .

<b>Containerization</b>	Docker	Used to package the Java application and its environment into a portable image.
<b>Registry</b>	Docker Hub	Public cloud repository for storing and distributing the Docker image.
<b>Deployment</b>	Ansible	Configuration Management tool used to deploy the final container onto the local managed host.

### 3. Initial Development, Testing Setup, and Maven Configuration

**Brief:** The project began by establishing the core Java application logic and integrating the JUnit 5 testing framework, all managed and standardized by Apache Maven. This stage was essential for creating the tested, executable artifact required for the subsequent CI/CD pipeline stages.

**Application Development and Structure:** Development was initiated within the **IntelliJ IDEA** environment. The use of IntelliJ was deliberate, as its powerful, integrated features for **running and debugging JUnit test cases** provided immediate, superior feedback essential for test-driven development.

The project structure relied on a crucial partnership between the IDE and the build tool:

- **Apache Maven's Role:** Maven is a **Project Management and Comprehension Tool**. Its primary function is to define the project's entire lifecycle (dependencies, compiling, testing, and packaging) in a standardized, repeatable way via the `pom.xml` file.
- **IntelliJ's Partnership:** While Maven handles the build logic, IntelliJ acts as the user interface, reading the `pom.xml` to automatically configure the project, download dependencies, and provide shortcuts to execute Maven phases (like `mvn test`) without needing command-line interaction.

The source code was organized into the standard Maven directory structure using the package `com.spe.calculator`:

- **Main.java:** Located in `src/main/java`, implements the **menu-driven Command-Line Interface (CLI)**. It serves as the primary **entry point** for the application, handling user choices (1-5) and managing the application flow. Screenshots have not been included to compact the report.
- **ScientificCalculator.java:** Located in `src/main/java`, contains the **core business logic** for the required mathematical functions (`sqrt(x)`, `!x`, `ln(x)`, `x^b`). This file was built with robust exception handling, including a custom

`FactorialOverflowException` to manage large input limits, ensuring stability. Screenshots have not been included to compact the report.

- **ScientificCalculatorTest.java**: Located in `src/test/java`, this file houses the comprehensive **JUnit 5 test suite**. Test cases were designed to cover boundary conditions, zero, negative inputs, and large input overflow cases, proving the functional correctness of the code.

**Maven (`pom.xml`) Configuration: The Build Gatekeeper** The `pom.xml` serves as the central control file, configuring how the code is compiled, tested, and packaged. The following plugins and dependencies were crucial for linking the development phase to the DevOps pipeline:

Element	Plugin/Dependency	CI/CD Purpose
Testing Framework	<code>junit-jupiter-api</code> & <code>junit-jupiter-engine</code>	These dependencies satisfy the <b>Testing</b> requirement. They provide the necessary APIs for writing tests and the execution engine for running them, which is vital for the CI pipeline's immediate validation step.
Compiler Standard	<code>maven-compiler-plugin:3.11.0</code>	Configured the <code>&lt;source&gt;</code> and <code>&lt;target&gt;</code> to <b>Java 17</b> . This ensured the compiled code was standardized, guaranteeing consistent behavior whether it was run locally, by Jenkins, or inside the Docker container.
Test Runner	<code>maven-surefire-plugin:3.2.3</code>	<b>Critical for the Continuous Integration (CI) gate.</b> This plugin automatically discovers and executes all JUnit tests during the Maven <code>test</code> lifecycle phase. If tests fail, the Maven build (and thus the Jenkins pipeline) fails, preventing unstable code from moving forward.
Executable Packaging	<code>maven-jar-plugin:3.3.0</code>	<b>Crucial for the Build and Containerization.</b> The nested <code>&lt;mainClass&gt;com.spe.calculator.Main&lt;/mainClass&gt;</code> configuration makes the final JAR file <b>self-executable</b> ( <code>java -jar ...</code> ). This eliminates the need to specify the main class when the Docker container runs the application.

## 1. The Maven/Directory Separation

The primary separation is handled by the build tool (Maven) and the directory structure:

- `src/main/java`: This folder is for production code (the code that gets deployed and run by the end-user).
- `src/test/java`: This folder is strictly for test code (JUnit files) and is *not* included in the final executable JAR.

This directory separation is standardized by Maven and is critical for the Continuous Integration (CI) pipeline, as it allows Jenkins to run tests (`mvn test`) independently before packaging the final application code (`mvn clean package`).

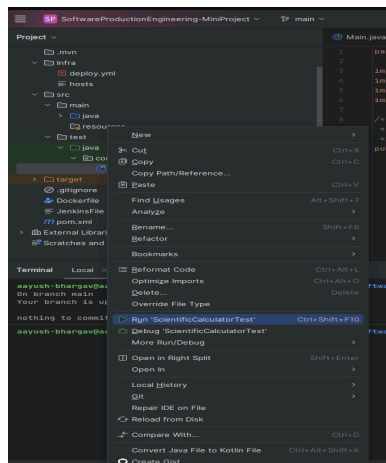
---

## 2. The Package Naming Convention (`com.spe.calculator`)

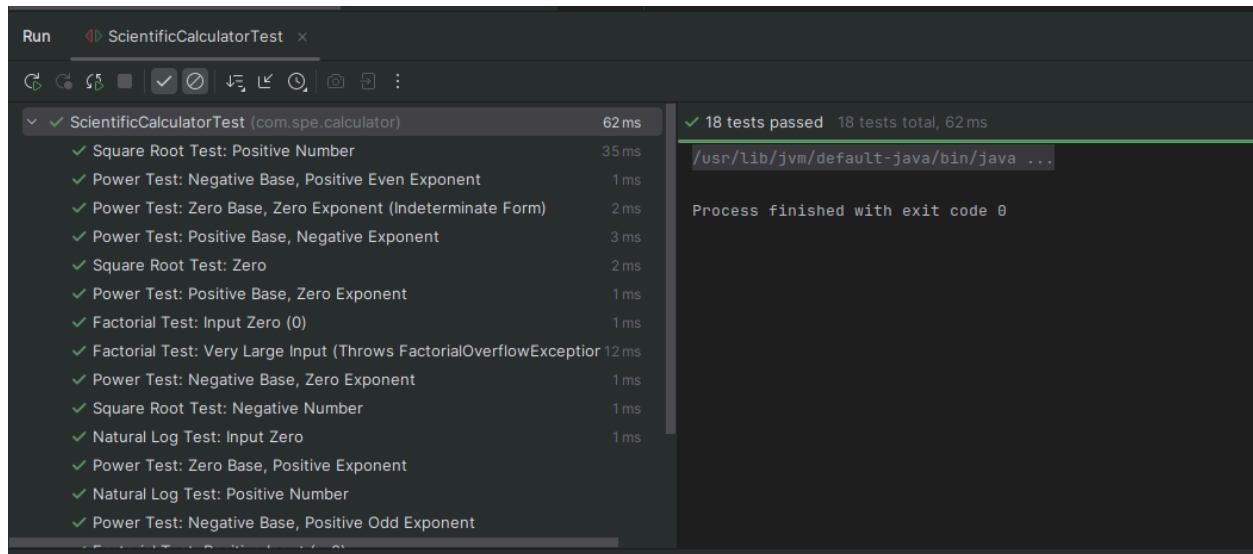
The Java package name handles the logical grouping and identity:

- Logical Grouping: By declaring `package com.spe.calculator;` in both `Main.java` and `ScientificCalculatorTest.java`, you are telling the Java compiler that these files are logically part of the same unit.
- No Imports Required: Since both classes belong to the same package, you do not need to write an `import` statement in `ScientificCalculatorTest.java` to use the `ScientificCalculator` class. This simplifies development and reduces boilerplate code.

## Running The Tests:

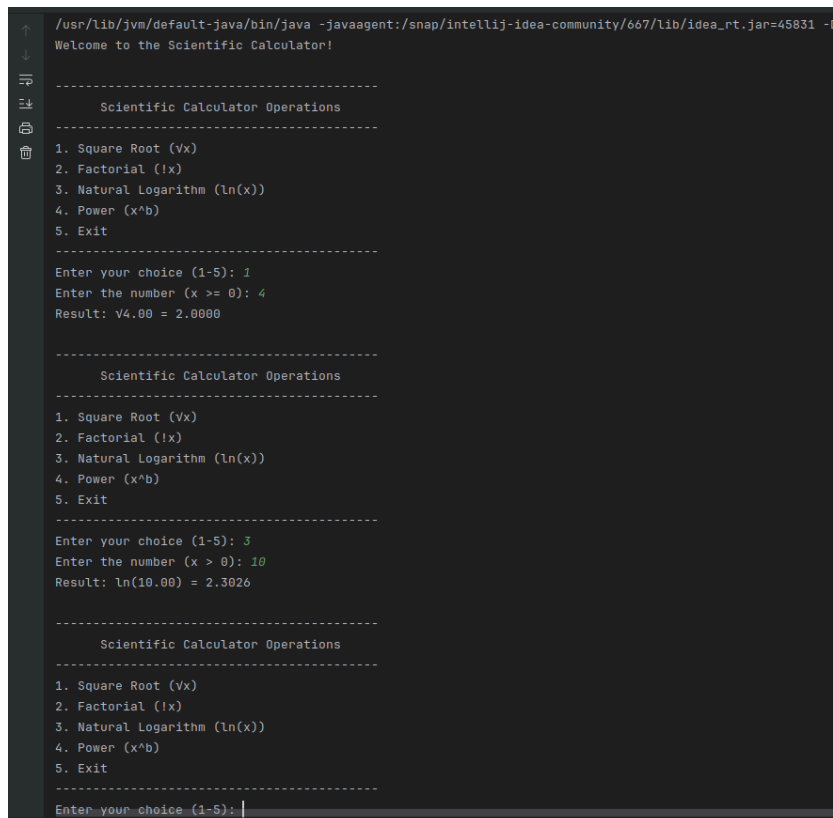


## Output:



All 18 tests passed successfully.

## Running the Program:



## 4. Source Control Management (SCM) and Initial Push

**Brief:** Source Control Management (SCM) is the foundational step in any DevOps pipeline. We utilized GitHub to host the project repository, allowing for collaborative development, version tracking, and, most importantly, providing the centralized source from which the Continuous Integration (CI) server (Jenkins) pulls the code.

### Setup and Configuration:

1. **Repository Creation:** The process began by creating a new public repository on GitHub named `SPE_MiniProject`.

Link to the GitHub repository:

[https://github.com/Aayush-Bhargav/SPE\\_MiniProject](https://github.com/Aayush-Bhargav/SPE_MiniProject)

2. **Local Repository Initialization:** In the IntelliJ project directory, Git was initialized, and the remote GitHub repository was linked.

Action	Command Used	Purpose
Initialize local Git	<code>git init</code>	Converts the local project directory into a Git repository.
Link Remote URL	<code>git remote add origin https://github.com/Aayush-Bhargav/SPE_MiniProject.git</code>	Links the local <code>origin</code> branch to the remote GitHub repository URL.
Stage Files	<code>git add .</code>	Adds all existing project files ( <code>.java</code> , <code>pom.xml</code> , etc.) to the staging area.
Commit Changes	<code>git commit -m "First commit"</code>	Records the staged changes permanently in the local repository history.

Authentication via Personal Access Token (PAT): When executing the final push



command, standard username and password **authentication failed, as GitHub** deprecated password usage for Git operations over HTTPS in 2021 for security reasons.

Action	Command Used	Authentication Required
Push to GitHub	<code>git push -u origin main</code>	Prompts for Username and Password/PAT.

Explanation of Personal Access Token (PAT): A Personal Access Token (PAT) is a secure, temporary string that serves as a modern, revocable, and audited alternative to a full account password for accessing the GitHub API or performing Git operations.

- Why it is Used: PATs enhance security by allowing specific permissions (scopes, e.g., 'repo' access only) and set expiration dates, limiting potential damage if the token is compromised, unlike a permanent account password.
- PAT Generation: The token was generated through **Settings** → **Developer settings** → **Personal access tokens -> Tokens (classic)** on the GitHub website, with the necessary `repo` scope checked to allow code pushes.

The PAT was entered when prompted for the password, successfully completing the initial push and ensuring the project source code was ready for the next phase.

Screenshots for the creation of PAT:

- GitHub Apps
- OAuth Apps
- Personal access tokens ^
  - Fine-grained tokens
  - Tokens (classic)

## New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Note

spe-mini\_project-token

What's this token for?

### Expiration

📅 30 days (Oct 29, 2025) ▾

The token will expire on the selected date

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows
<input type="checkbox"/> <b>write:packages</b>	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry

re selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

- GitHub Apps
- OAuth Apps
- Personal access tokens ^
  - Fine-grained tokens
  - Tokens (classic)

## Personal access tokens (classic)

Generate new token ▾

Tokens you have generated that can be used to access the [GitHub API](#).

🔔 Make sure to copy your personal access token now. You won't be able to see it again!		
✓ ghp_wIgDMT5MSHdGRytdou5LSS01l1S3Mo3TNqD7		Delete
jenkins-webhook-token — admin:repo_hook		
Expires on Tue, Oct 7 2025.		Never used Delete

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Screenshots of actions in the local repository leading to the first commit:

```

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   .idea/.gitignore
    new file:   .idea/encodings.xml
    new file:   .idea/misc.xml
    new file:   .idea/vcs.xml
    new file:   pom.xml
    new file:   src/main/java/com/spe/calculator/Main.java
    new file:   src/main/java/com/spe/calculator/ScientificCalculator.java
    new file:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java
    new file:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   pom.xml
    modified:   src/main/java/com/spe/calculator/Main.java
    deleted:    src/test/java/com/spe/calculator/ScientificCalculatorTest.java
    modified:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java

```

```

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git add .
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   .idea/.gitignore
    new file:   .idea/encodings.xml
    new file:   .idea/misc.xml
    new file:   .idea/vcs.xml
    new file:   pom.xml
    new file:   src/main/java/com/spe/calculator/Main.java
    new file:   src/main/java/com/spe/calculator/ScientificCalculator.java
    new file:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git commit -m "First Commit"
[master (root-commit) 2a992b5] First Commit
 9 files changed, 499 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/encodings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 pom.xml
 create mode 100644 src/main/java/com/spe/calculator/Main.java
 create mode 100644 src/main/java/com/spe/calculator/ScientificCalculator.java
 create mode 100644 src/test/java/com/spe/calculator/ScientificCalculatorTest.java

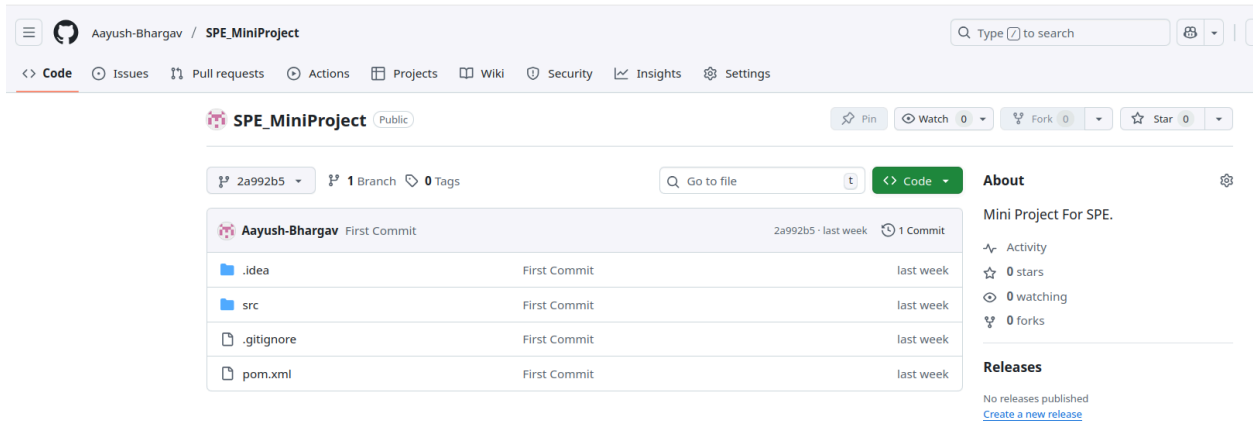
```

```

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git push -u origin main
Username for 'https://github.com': Aayush-Bhargav
Password for 'https://Aayush-Bhargav@github.com':
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (23/23), 6.10 KiB | 2.03 MiB/s, done.
Total 23 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Aayush-Bhargav/SPE_MiniProject.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$

```



**State of the Repository after the First Commit.**

## 5. Continuous Integration Setup (CI Trigger)

### What is Jenkins?

Jenkins is an open-source automation server used to build, test, and deploy software automatically. It is a key DevOps tool that supports Continuous Integration (CI) and Continuous Delivery (CD), helping developers detect and fix issues early by automating the software build and deployment pipeline. You need to have Jenkins installed on your system and running locally for this project.

**Brief:** Continuous Integration requires an immediate trigger when source code changes. Since the Jenkins server was running locally, this step involved exposing the local server to the public internet using ngrok and configuring a GitHub Webhook to send events to that public address.

**Tool:** ngrok (Public Tunneling Service)

ngrok is a crucial tool used to create a secure, public tunnel to a locally running service (like Jenkins on `localhost:8080`). This public URL allows external services, such as GitHub's webhook mechanism, to communicate with the local CI server.

**ngrok Setup and Execution:**

**Installation and Authentication:** ngrok was downloaded, extracted, and authenticated using the personal auth token copied from the ngrok dashboard.

# 1. Extract ngrok to system path

```
$ sudo tar xvf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
```

# 2. Add Authtoken for secure connection

```
$ ngrok authtoken <token>
```

**Exposing Jenkins:** The local Jenkins instance, running on port 8080, was exposed to the public internet.

# Execute ngrok and copy the generated public HTTPS URL

```
$ ngrok http 8080
```

```
ngrok
🐼 Decouple policy and sensitive data with vaults: https://ngrok.com/r/secrets

Session Status      online
Account             Aayush Bhargav (Plan: Free)
Update              update available (version 3.30.0, Ctrl-U to update)
Version             3.27.0
Region              India (in)
Latency              31ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://cbf9aab8f605.ngrok-free.app -> https://localhost:8080

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

The resulting HTTPS URL (e.g., <https://cbf9aab8f605.ngrok-free.app>) served as the Payload URL for the GitHub webhook.

## Jenkins Server Global Configuration

To ensure Jenkins correctly identifies its own public address and securely communicates with GitHub, global system settings were configured:

**1. Jenkins Location Update:** The public ngrok URL was set as the official Jenkins address.

- Location: *Manage Jenkins* → *System* → *Jenkins Location*

- Configuration: The ngrok address (<https://cbf9aab8f605.ngrok-free.app>) was entered into the Jenkins URL field. This allows Jenkins to generate correct links for webhooks and internal resources.

Jenkins Location

Jenkins URL ?

<https://52d64359d9cb.ngrok-free.app>

System Admin e-mail address ?

Jenkins-Master <aayushbhargav0507@gmail.com>

## 2. GitHub Server Credential Setup: The second PAT created for the webhook was registered in Jenkins.

- Location: *Manage Jenkins* → *System* → *GitHub Servers*
- Configuration: A new credential was added (Type: Secret Text) containing the PAT created with the [admin:repo\\_hook](#) scope. This is used by Jenkins plugins to make verified requests back to GitHub (e.g., updating commit statuses).

### Second PAT screenshot:

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

Edit personal access token (classic)

Make sure to copy your token now as you will not be able to see it again.

ghp\_demQ1myya67oWHVYFkQ0hbrWVIMXTa37UmpM

Note

jenkins-webhook-token

What's this token for?

Expiration

This token expires on Wed, Oct 29 2025. To set a new expiration date, you must [regenerate the token](#).

Select scopes

☒ admin:repo\_hook

Full control of repository hooks

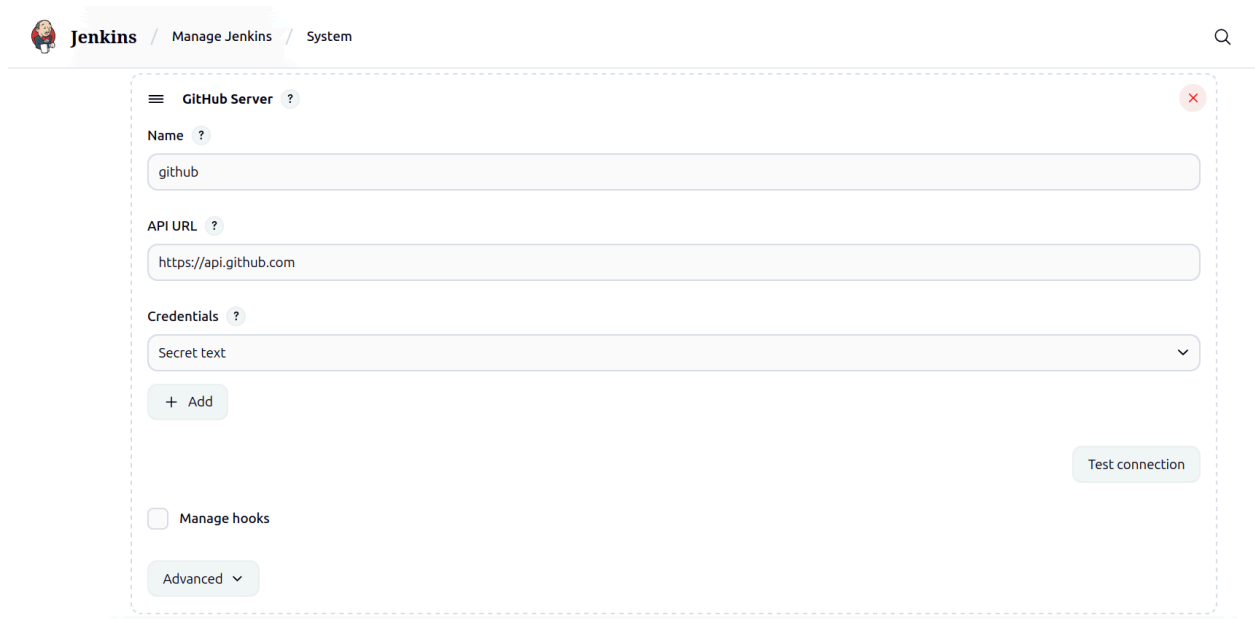
☒ write:repo\_hook

Write repository hooks

☒ read:repo\_hook

Read repository hooks

## Adding the secret text into the credentials in Jenkins Settings:



The screenshot shows the Jenkins configuration page for a 'GitHub Server'. The page has a header with the Jenkins logo and navigation links: 'Jenkins', 'Manage Jenkins', and 'System'. A search icon is in the top right. The main content area is titled 'GitHub Server' with a help icon. It contains several input fields: 'Name' with the value 'github', 'API URL' with the value 'https://api.github.com', and 'Credentials' with a dropdown menu showing 'Secret text'. Below the 'Credentials' field is a '+ Add' button. To the right of the 'Add' button is a 'Test connection' button. At the bottom, there is a checkbox for 'Manage hooks' and an 'Advanced' dropdown menu.

Jenkins / Manage Jenkins / System

GitHub Server ?

Name ?  
github

API URL ?  
https://api.github.com

Credentials ?  
Secret text

+ Add

Test connection

☐ Manage hooks

Advanced ▾

**Click on Add and use the Second PAT created as the Secret.**

Manage Jenkins / System

### Jenkins Credentials Provider: Jenkins

#### Add Credentials

Domain

Global credentials (unrestricted) ▼

Kind

Secret text ▼

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Secret

.....

ID ?

Description ?

## GitHub Webhook Configuration

To complete the trigger mechanism, a webhook was configured in the GitHub repository settings.

- 1. Generating the Webhook PAT:** A second Personal Access Token (PAT), named `jenkins-webhook-token`, was created specifically for the webhook.
  - Scope: This PAT was granted the `admin:repo_hook` scope. This restricted permission is necessary because it allows GitHub to manage the webhook itself and verify the connection, without granting full repository write access.
- 2. Webhook Settings:** The webhook was configured with the following parameters:

Parameter	Value	Purpose
-----------	-------	---------



Payload URL	<code>https://52d6435d9cb.ngrok-free.app/github-webhook/</code>	The public address (from ngrok) plus the standard Jenkins endpoint for GitHub triggers.
Content type	<code>application/x-www-form-urlencoded</code>	Specifies the format of data GitHub sends to Jenkins.
Secret	The generated <code>jenkins-webhook-token</code> PAT.	Secures the connection; Jenkins uses this token to verify the incoming request is genuinely from GitHub.
SSL Verification	Enabled	Ensures data integrity and security between GitHub and the ngrok tunnel.

### 3.

**Delivery Test:** A test delivery was executed, and the system returned a successful HTTP status code (200 OK), confirming the Continuous Integration trigger was successfully linked.

**Screenshot of ngrok running:**

```
ngrok
🛡️ Block threats before they reach your services with new WAF actions → https://ngrok.com/r/waf

Session Status      online
Account             Aayush Bhargav (Plan: Free)
Version             3.27.0
Region              India (in)
Latency              28ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://876a1ca2658f.ngrok-free.app -> http://localhost:8080

Connections          ttl      opn      rt1      rt5      p50      p90
                    33       0        0.00     0.01     0.68     33.16

HTTP Requests
-----
18:59:20.782 IST POST /github-webhook/ 200 OK
18:54:16.044 IST POST /widget/ExecutorsWidget/ajax 200 OK
18:54:16.016 IST POST /widget/BuildQueueWidget/ajax 200 OK
18:54:10.791 IST GET /static/107d7d82/jsbundles/styles.css
18:54:10.631 IST GET / 200 OK
18:54:10.959 IST GET /i18n/resourceBundle 200 OK
18:54:09.587 IST POST /widget/ExecutorsWidget/ajax 200 OK
18:54:09.587 IST POST /widget/BuildQueueWidget/ajax 200 OK
18:49:39.084 IST POST /widget/ExecutorsWidget/ajax 200 OK
18:49:39.083 IST POST /widget/BuildQueueWidget/ajax 200 OK
```

**Screenshot of GitHub Webhook Configuration:**

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Models

Webhooks

Copilot

Environments

Codespaces

Pages

Security

Advanced Security

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL \*

https://cbf9aab8f605.ngrok-free.app

Content type \*

application/json

Secret

ghp\_demQ1myya67oWHVYFKQ0hbrVViMXTa37UmpM

SSL verification

By default, we verify SSL certificates when delivering payloads.

Enable SSL verification

Disable (not recommended)

Which events would you like to trigger this webhook?

Just the push event.

Send me everything.

Let me select individual events.

Active

We will deliver event details when this hook is triggered.

Add webhook

/ SPE\_MiniProject

Type to search

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

General

Access

Collaborators

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Models

Webhooks

Copilot

Environments

Codespaces

Pages

Security

Advanced Security

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Webhooks / Manage webhook

Settings

Recent Deliveries

6c895f3a-9d36-11f0-922b-77acd2674059

ping

redelivery

2025-09-29 18:59:21

Request

Response

200

Redeliver

Completed in 1.0 second.

Headers

Request URL: https://876a1ca2658f.ngrok-free.app/github-webhook/

Request method: POST

Accept: \*/\*

Content-Type: application/x-www-form-urlencoded

User-Agent: GitHub-Hookshot/V2-eeff06b

X-GitHub-Delivery: 6c895f3a-9d36-11f0-922b-77acd2674059

X-GitHub-Encoded-Secret: Z2hwX2RlbVExbXl5YTY3bDlVlG6a1EwaGJyV1ZpTVhuYTMs3VW1wTQ==

X-GitHub-Event: ping

X-GitHub-Hook-ID: 572398374

X-GitHub-Hook-Installation-Target-ID: 1066392424

X-GitHub-Hook-Installation-Target-Type: repository

X-Hub-Signature: sha1=907004cc5f5a785058a0de16a62e45ddcd6c4e98

X-Hub-Signature-256: sha256=4ba9dec7e73685dedfb07dee2520460b87476eb7897ecb008e707099591f17a7

Payload

```
{
  "zen": "Approachable is better than simple.",
  "hook_id": 572398374,
  "hook": {
    "type": "Repository",
    "id": 572398374,
  }
}
```

## 6. Jenkinsfile Creation

A **Jenkinsfile** is a text file that defines a **Jenkins pipeline** — a sequence of automated steps that enable continuous integration and continuous deployment (**CI/CD**). Our file is written using the **declarative** syntax based on Groovy and stored within the project's version control system. This allows the entire build and deployment process to be versioned alongside the source code, ensuring transparency, reproducibility, and collaboration.

In a **CI/CD pipeline**, the Jenkinsfile acts as the **blueprint** that automates the software development lifecycle — from fetching the latest code to testing, building, containerizing, and deploying it. It ensures that code changes are continuously integrated, verified, and delivered in a consistent and reliable manner without manual intervention.

A typical Jenkinsfile defines several key aspects:

- **Agent:** Specifies where the pipeline will run (e.g., any available node or a Docker container).
- **Environment Variables:** Store credentials and configuration values securely.
- **Stages:** Represent distinct steps such as *checkout*, *build*, *test*, *package*, *deploy*, etc.
- **Post Actions:** Define cleanup or notification steps that run after the pipeline completes.

By automating repetitive tasks, a Jenkinsfile enhances **developer productivity**, reduces **human errors**, and ensures **faster feedback** on code quality. It forms the backbone of modern DevOps workflows by integrating tools such as Maven for build management, Docker for containerization, and Ansible for deployment — as demonstrated in this project.

Our JenkinsFile can be seen on GitHub in the root of the project.

### Detailed Stage Breakdown:

Stage Name	Command/ Instruction	Step/ Tool	Explanation
Checkout	<code>git branch: 'main', url: '...'</code>	SCM (Git)	Pulls the latest source code from the specified GitHub URL and ensures the pipeline is building from the <b>main</b> branch, guaranteeing synchronization.
Test	<code>sh 'mvn test'</code>	Testing (Maven Surefire)	Executes the test phase defined in <code>pom.xml</code> , running all JUnit test cases within <code>ScientificCalculatorTest.java</code> . The pipeline fails immediately if any test asserts are violated.
Build JAR	<code>sh 'mvn clean package'</code>	Build (Maven)	Compiles the Java code and packages it, along with dependencies, into the final executable JAR file ( <code>scientific-calculator-project-1.0-SNAPSHOT.jar</code> ). This artifact is necessary for the next stage.
Build Docker Image	<code>sh 'docker build -t ... .'</code>	Containerize (Docker)	Executes the <code>Dockerfile</code> using the Jenkins workspace as the build context. This creates the final application image.
Push Docker Image	<code>`sh 'echo \$PSW</code>	docker login ..."	Push the image to Docker Hub

<b>Deploy with Ansible</b>	<code>sh 'ansible-playbook -i ...'</code>	Deployment (Ansible)	Executes the configuration management tool, which manages the lifecycle of the container on the target host ( <code>localhost</code> ).
----------------------------	---	----------------------	---

The `post` block is the final section of the pipeline, defining actions that must run after all stages are complete, regardless of the overall pipeline result. Its primary role is to provide Continuous Feedback (email notification) and ensure Cleanup.

## Continuous Feedback and Cleanup

The `post` section utilizes conditional blocks to execute specific commands based on the outcome of the preceding stages:

Condition	Action Executed	Purpose
<b>success</b>	The <code>mail</code> step sends a success notification, including the unique <code>\$BUILD_NUMBER</code> for easy reference. This informs the developer that the new code has been tested, built, and deployed.	<b>Continuous Feedback</b> (Success Notification)
<b>failure</b>	The <code>mail</code> step sends an immediate failure notification, directing the developer to check the Jenkins console for the specific stage failure.	<b>Continuous Feedback</b> (Immediate Alerting)
<b>always</b>	Contains steps that run regardless of whether the pipeline passed or failed.	Essential Cleanup

<b>cleanWs()</b>	A built-in Jenkins step that <b>deletes the contents of the Jenkins workspace</b> ( <code>/var/lib/jenkins/workspace /...</code> ).	<b>Resource Management.</b> This prevents leftover files and build artifacts from interfering with subsequent builds, ensuring the next run starts from a clean state.
------------------	---	--

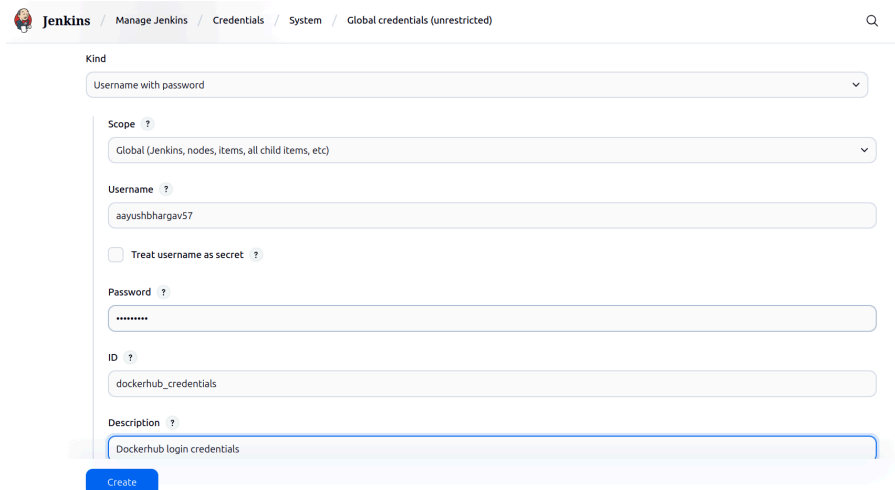
## Docker Hub Credential Management and Security

### What is DockerHub?

Docker Hub is a cloud-based registry service where Docker users can **store, share, and manage container images**. It acts like a “GitHub for Docker images,” allowing developers to pull pre-built images or push their own for use in different environments.

To execute the **Push Docker Image** stage, Jenkins requires secure access to Docker Hub. This was achieved by configuring a global credential.

1. **Access:** *Manage Jenkins → Credentials → System → Global credentials (unrestricted) → Add Credentials.*
2. **Configuration:** A credential of kind "**Username with password**" was created with my DockerHub username and password.
3. **Identifier:** The **ID** was set to `dockerhub_credentials`. This identifier must match the name used in the `environment` block of the `Jenkinsfile`.
4. **Purpose:** This mechanism ensures the Docker Hub login details (username/password or PAT) are never exposed directly in the `Jenkinsfile` or the build logs, fulfilling a fundamental security requirement of CI/CD.



The screenshot shows the Jenkins 'Global credentials (unrestricted)' configuration page. The 'Kind' is set to 'Username with password'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'aayushbhargav57'. There is an unchecked checkbox for 'Treat username as secret'. The 'Password' field is masked with dots. The 'ID' is 'dockerhub\_credentials'. The 'Description' is 'Dockerhub login credentials'. A blue 'Create' button is at the bottom.

Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted)

Kind  
Username with password

Scope  
Global (Jenkins, nodes, items, all child items, etc)

Username  
aayushbhargav57

☐ Treat username as secret

Password  
\*\*\*\*\*

ID  
dockerhub\_credentials

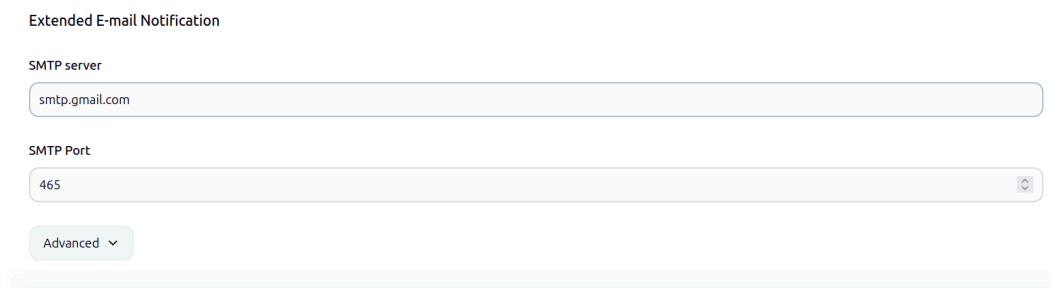
Description  
Dockerhub login credentials

Create

## Jenkins Global Configuration (Setup)

Before the pipeline can send emails, the Jenkins server itself must be configured with an SMTP server (like Gmail, Outlook, or a company server).

1. Go to **Manage Jenkins** → **System**.
2. Scroll down to the **"Extended E-mail Notification"** section (or the standard **"E-mail Notification"** section).
3. **Configure the SMTP Server Details:**
  - **SMTP server:** (set it to `smtp.gmail.com` since we are using Gmail)
  - **Check "Use SMTP Authentication"** and enter your full email address and the corresponding **App Password**.
  - **Use SSL/TLS** and specify the correct port (465).
4. **Test Configuration:** Use the "Test Configuration" button at the bottom to ensure Jenkins can successfully send a test email.



The screenshot shows the 'Extended E-mail Notification' configuration section in Jenkins. The 'SMTP server' is set to 'smtp.gmail.com'. The 'SMTP Port' is set to '465'. There is an 'Advanced' dropdown menu at the bottom.

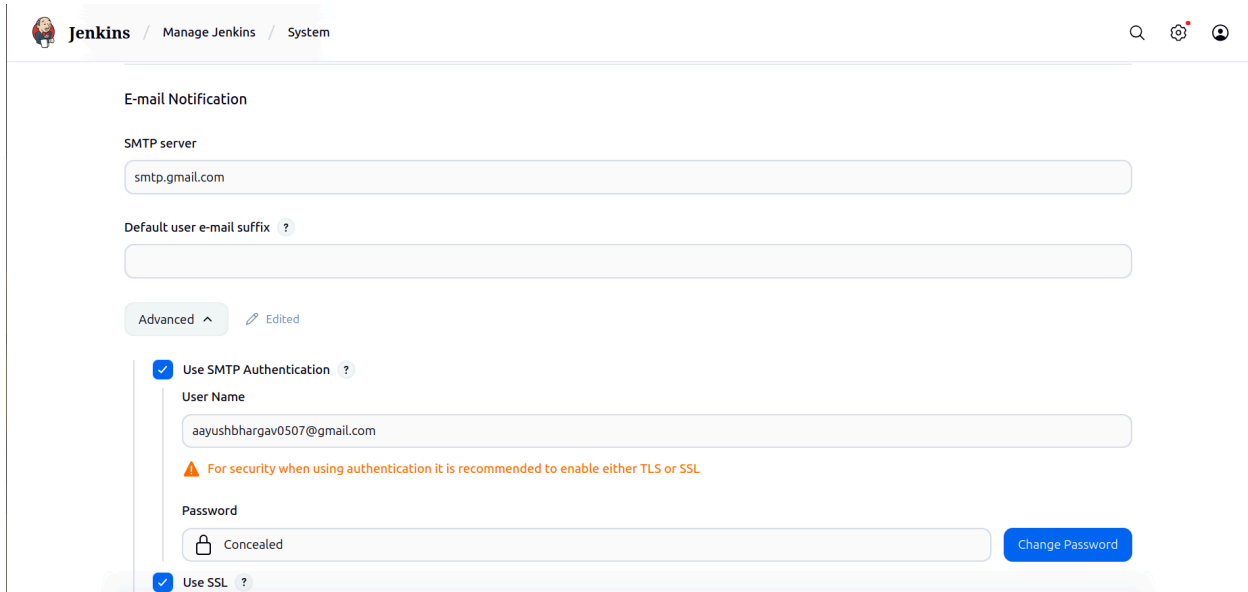
Extended E-mail Notification

SMTP server  
smtp.gmail.com

SMTP Port  
465

Advanced





The screenshot shows the Jenkins 'E-mail Notification' configuration page. The breadcrumb navigation at the top is 'Jenkins / Manage Jenkins / System'. The page title is 'E-mail Notification'. There are three input fields: 'SMTP server' with the value 'smtp.gmail.com', 'Default user e-mail suffix' which is empty, and an 'Advanced' section. The 'Advanced' section is expanded, showing 'Use SMTP Authentication' checked. Below it, 'User Name' is 'aayushbhargav0507@gmail.com' and 'Password' is 'Concealed'. A warning message states: 'For security when using authentication it is recommended to enable either TLS or SSL'. There is a 'Change Password' button next to the password field. At the bottom of the advanced section, 'Use SSL' is checked.

Jenkins / Manage Jenkins / System

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix ?

Advanced ^ Edited

☒ Use SMTP Authentication ?

User Name

aayushbhargav0507@gmail.com

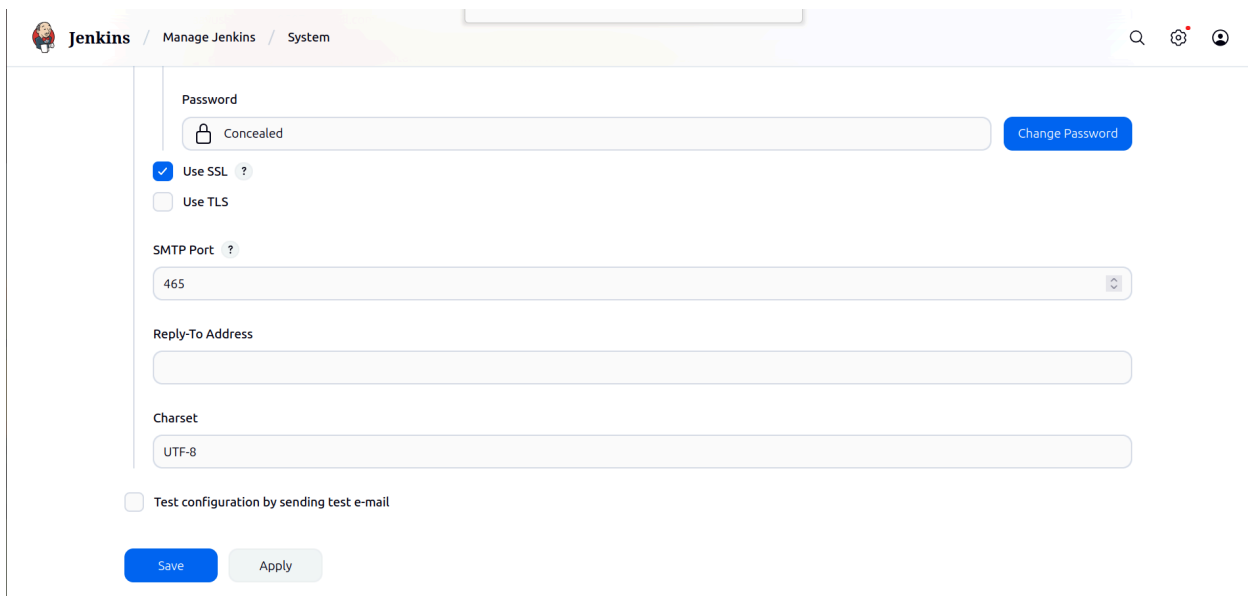
⚠ For security when using authentication it is recommended to enable either TLS or SSL

Password

Concealed

Change Password

☒ Use SSL ?



This screenshot shows the continuation of the Jenkins 'E-mail Notification' configuration page. It includes the 'Password' field (concealed) with a 'Change Password' button. Below this, 'Use SSL' is checked and 'Use TLS' is unchecked. The 'SMTP Port' is set to '465'. There are empty input fields for 'Reply-To Address' and 'Charset' (set to 'UTF-8'). At the bottom, there is an unchecked checkbox for 'Test configuration by sending test e-mail' and two buttons: 'Save' and 'Apply'.

Jenkins / Manage Jenkins / System

Password

Concealed

Change Password

☒ Use SSL ?

☐ Use TLS

SMTP Port ?

465

Reply-To Address

Charset

UTF-8

☐ Test configuration by sending test e-mail

Save Apply

## 7. Dockerfile Definition (Containerization)

### What is Docker?

Docker is a platform that allows applications to run in **containers** — lightweight, isolated environments that package code along with all its dependencies. This ensures the application runs consistently across different systems, making deployment faster, more reliable, and easier to scale.

The **Dockerfile** is the blueprint for creating the portable, self-contained environment that wraps the Java application, fulfilling the **Containerization (Step 5)** requirement.

```
ulator.java  pom.xml (scientific-calculator-project)  .gitignore  ScientificCalculatorTest.java
1  # Use a lightweight official Java Runtime Environment (JRE) as the base image
2  FROM openjdk:26-slim
3
4  # Set the working directory inside the container
5  WORKDIR /app
6
7  # Copy the executable JAR file created by Maven's 'package' goal
8  # from the host's 'target/' directory into the container's '/app' directory.
9  # NOTE: The JAR name must match your pom.xml's artifactId and version.
10 COPY target/scientific-calculator-project-1.0-SNAPSHOT.jar app.jar
11
12 # Command to run the JAR file when the container starts
13 # The Main class (com.spe.calculator.Main) runs the command line menu.
14 ENTRYPOINT ["java", "-jar", "app.jar"]
15
```

Dockerfile Instruction	Code	Explanation
FROM	FROM openjdk:26-slim	Uses a minimal, lightweight <b>Java Runtime Environment (JRE)</b> based on OpenJDK 26.
WORKDIR	WORKDIR /app	Sets the application's root directory inside the container.
COPY	COPY target/scientific-calculator-project-1.0-SNAPSHOT.jar app.jar	This is the crucial step: it transfers the compiled Maven JAR (the build artifact) from the Jenkins workspace into the /app directory inside the container.

<b>ENTRYPOINT</b>	ENTRYPOINT \["java", "-jar", "app.jar"]	Defines the command that executes by default when the container starts.
-------------------	---	---

## 8. Configuration Management and Deployment - Ansible

The final phase of the pipeline uses **Ansible** to manage the target host and deploy the Docker containerized application. Since the deployment target is the local Jenkins host, Ansible's role is to ensure the latest tested image is pulled and run successfully.

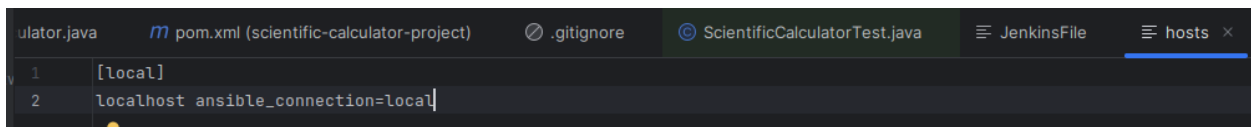
### A. What is Ansible and Its Role?

**Ansible** is an open-source automation engine used for software provisioning, configuration management, and application deployment. Unlike other tools, it is **agentless**, meaning it manages remote hosts over SSH (or local connections) without requiring any special software installed on the target machine.

In this project, Ansible's role is to ensure the final deployment step—**running the Docker container**—is executed reliably, idempotently, and repeatably on the local host machine.

### B. Ansible Inventory File (**Infra/hosts**)

The **hosts** file defines the inventory—the list of target machines Ansible will manage.



```

1 [local]
2 localhost ansible_connection=local

```

**[local]**: Defines a group named "local."

**localhost**: Specifies the target machine is the local host where Jenkins and Ansible are running.

**ansible\_connection=local:** Instructs Ansible to execute commands directly on the local machine rather than connecting via SSH, making the deployment instantaneous for this single-host setup.

## C. Ansible Playbook (Infra/deploy.yml)

The `deploy.yml` is the core configuration file written in YAML. It defines the sequence of tasks required to deploy the container.

```
1  ---
2  - name: Deploy Scientific Calculator Docker Container
3    hosts: local
4    # This requires the 'jenkins' user to have NOPASSWD access in /etc/sudoers
5    # to run Docker commands as root (unless jenkins user is in the 'docker' group)
6    become: yes
7
8    vars:
9      # IMPORTANT: The image name MUST match the one defined in your Jenkinsfile
10     app_image: "aayushbhargav57/scientific-calculator:latest"
11     container_name: "scientific_calculator_app"
12
13    tasks:
14      # FIX: Switching back to pip, using the 'ansible.builtin.pip' module with
15      # the 'extra_args' flag to bypass the externally-managed-environment error
16      # on modern Linux distributions. This is the cleanest fix for CI systems.
17      - name: Ensure Python Docker module is installed (via pip bypass)
18        ansible.builtin.pip:
19          name: docker
20          state: present
21          executable: pip3
22          extra_args: "--break-system-packages" # Flag to override PEP 668 restriction
23          # This task requires elevated privileges (become: yes)
24          # No 'when' condition needed as this fix is generally system-agnostic for CI
25
26      - name: Stop and remove existing container (if running)
27        # Use the fully qualified collection name for the docker_container module
28        community.docker.docker_container:
29          name: "{{ container_name }}"
30          state: absent
31      - name: ignore_errors: yes
32
33      - name: Pull the latest Docker image from Docker Hub
34        community.docker.docker_image:
35          name: "{{ app_image }}"
36          source: pull
37
38      - name: Run the new container
39        community.docker.docker_container:
40          name: "{{ container_name }}"
41          image: "{{ app_image }}"
```

```

- name: Run the new container
  community.docker.docker_container:
    name: "{{ container_name }}"
    image: "{{ app_image }}"
    state: started
    # Detach the container so the Jenkins job can finish
    interactive: true
    detach: true
    # The app is command-line only; it will run the entrypoint (Main class)
    # and exit after execution unless in an interactive environment.
    # For continuous running CLI, you'd use a loop in the Entrypoint/CMD,
    # but for this assignment, a started container is sufficient.

```

## Detailed Task Explanation:

Task Name	Module Used	Purpose and Why
Play Setup	<code>hosts: local, become: yes</code>	The playbook targets the <code>local</code> group and runs tasks with elevated privileges ( <code>become: yes</code> ), which is essential for managing the Docker service.
Ensure Python Docker module is installed	<code>ansible.builtin.pip</code>	This addresses a common CI/CD environmental issue. Ansible requires the <b>Python docker library</b> to manage containers. We used <code>extra_args: "--break-system-packages"</code> to bypass system Python security restrictions (PEP 668), ensuring the deployment dependency is installed reliably.

<b>Stop and remove existing container</b>	<code>community.docker.docker_container</code>	This enforces <b>idempotency</b> . It guarantees that the new deployment starts clean by stopping and removing the previous version of the container. <code>ignore_errors: yes</code> handles the case where the container does not yet exist.
<b>Pull the latest Docker image</b>	<code>community.docker.docker_image</code>	Instructs the target Docker Daemon to pull the image ( <code>aayushbhargav57/scientific-calculator:latest</code> ) from Docker Hub, ensuring the host always runs the version that was just built and pushed by Jenkins.
<b>Run the new container</b>	<code>community.docker.docker_container</code>	<b>Final Deployment Step.</b> Starts the new container using the freshly pulled image. The setting <code>detach: true</code> ensures that the Jenkins pipeline does not hang, allowing the job to finish successfully immediately after the container starts running the Java application's <code>ENTRYPOINT</code> .

## 9. Jenkins Job Configuration (Orchestration)

The last step in establishing Continuous Integration (CI) was the creation and configuration of the Jenkins Pipeline job itself. This job acts as the central orchestrator, executing the `Jenkinsfile` and managing the entire flow from code commit to deployment.

### A. Understanding the Pipeline Job

The job was created as a **Pipeline** type, which is superior to traditional Freestyle projects because it executes **Pipeline-as-Code** (the **JenkinsFile** stored in GitHub). This ensures the build process is always version-controlled and synchronized with the code it builds.

## B. Step-by-Step Configuration

The job was named **SPE-ScientificCalculator-MiniProject**. The key settings required to link GitHub, the webhook, and the **JenkinsFile** are detailed below:

### 1. General Configuration:

- **GitHub Project:** Checked, with the URL provided as [https://github.com/Aayush-Bhargav/SPE\\_MiniProject.git](https://github.com/Aayush-Bhargav/SPE_MiniProject.git). This links the job visually to the repository.

### 2. Continuous Integration Trigger:

- **GitHub hook trigger for GITScm polling:** Selected. This setting is mandatory. It tells Jenkins to listen for the incoming push events sent by the webhook (via ngrok) and immediately start the pipeline when a commit is received.

### 3. Pipeline Definition: This section directs Jenkins to find and execute the build instructions stored in your SCM.

- **Definition: Pipeline script from SCM** was chosen.
  - *Purpose:* This forces Jenkins to read the code's configuration from the repository, ensuring the CI process is governed by the version of the **JenkinsFile** checked into GitHub.
- **SCM:** Git was selected.
- **Repositories:**
  - **Repository URL:**  
[https://github.com/Aayush-Bhargav/SPE\\_MiniProject.git](https://github.com/Aayush-Bhargav/SPE_MiniProject.git) (The public URL).
  - **Credentials:** Left as (**none**) since the repository is public.
- **Branches to build:** Set to **\*/main**. This restricts automated builds to the primary development branch.
- **Script Path:** Set to **JenkinsFile**. This is the exact name of the file in the repository root containing the pipeline logic.

**Summary:** This configuration successfully automated the entire workflow. A commit to the **main** branch now instantly triggers Jenkins, which fetches the **JenkinsFile** and begins executing the eight stages of the CI/CD pipeline.



Jenkins / All / New Item

## New Item

Enter an item name

SPE-ScientificCalculator-MiniProjec

Select an item type



### Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



### Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



### Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



### Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

OK



Jenkins / SPE-ScientificCalculator-MiniProject / Configuration



## Configure

General

Triggers

Pipeline

Advanced

## General

Enabled

Description

This is the CI/CD pipeline job for the SPE Mini Project.

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/Aayush-Bhargav/SPE\_MiniProject/

Advanced ▾

Save

Apply





Jenkins

/ SPE-ScientificCalculator-MiniProject

/ Configuration



## Configure

General

Triggers

Pipeline

Advanced

### 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) ?

### Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

#### Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Save

Apply



Jenkins

/ SPE-ScientificCalculator-MiniProject

/ Configuration



## Configure

General

Triggers

Pipeline

Advanced

Repositories ?

Repository URL ?

https://github.com/Aayush-Bhargav/SPE\_MiniProject

Credentials ?

- none -

+ Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Save

Apply

The screenshot shows the Jenkins Configuration page for a project named 'SPE-ScientificCalculator-MiniProject'. The left sidebar contains a 'Configure' section with sub-items: General, Triggers, Pipeline, and Advanced (which is selected). The main content area is titled 'Configuration' and includes a 'Repository browser' dropdown set to '(Auto)', an 'Additional Behaviours' section with an 'Add' button, a 'Script Path' field containing 'JenkinsFile', and a checked 'Lightweight checkout' option. A 'Pipeline Syntax' link is also present. Below the configuration fields is an 'Advanced' section with a dropdown menu set to 'Advanced'. At the bottom are 'Save' and 'Apply' buttons.

## 10. Running The Pipeline


To run the pipeline, we can go to the created Jenkins pipeline job and click on the **‘Build Now’** option.

The screenshot shows the Jenkins job page for 'SPE-ScientificCalculator-MiniProject'. The left sidebar lists various actions: Status (selected), Changes, Build Now, Configure, Delete Pipeline, GitHub, Stages, Rename, Pipeline Syntax, and GitHub Hook Log. The main content area features a green checkmark icon, the job name, and an 'Edit description' button. Below this is a description: 'This is the CI/CD pipeline job for the SPE Mini Project.' and a 'Permalinks' section with a list of build links: 'Last build (#16), 3 hr 24 min ago', 'Last stable build (#16), 3 hr 24 min ago', 'Last successful build (#16), 3 hr 24 min ago', 'Last failed build (#12), 7 days 10 hr ago', 'Last unsuccessful build (#12), 7 days 10 hr ago', and 'Last completed build (#16), 3 hr 24 min ago'.

## Before the build, the docker containers on my local host:

```
Oct 7 22:59
aayush-bhargav@aayush-bhargav-Inspiron-15-3520: ~
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
3af7fdd478bc   ubuntu        "/bin/bash"             38 hours ago   Exited (0)    38 hours ago   ansible-container
1f3835151a8f   ansible-client "/bin/sh -c 'service..." 39 hours ago   Exited (137)  39 hours ago   node1
eaacd137ed6f   ubuntu        "/bin/bash"             39 hours ago   Exited (0)    37 hours ago   ansible_container
2e646d88a9c1   hello-world   "/hello"                39 hours ago   Exited (0)    39 hours ago   clever_clarke
c1d3d09e8dc3   hello-world   "/hello"                2 months ago   Exited (0)    2 months ago   keen_hamilton
6c362fa3937b   hello-world   "/hello"                2 months ago   Exited (0)    2 months ago   loving_stonebraker
689d0dcfcb22   hello-world   "/hello"                2 months ago   Exited (0)    2 months ago   admiring_khayyam
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~$
```

## Pipeline job successfully completed:

 Jenkins

SPE-ScientificCalculator-MiniProject

#20

🔍 ⚙️ 👤

Status

Changes

Console Output

Edit Build Information

Delete build '#20'

Polling Log

Timings

Git Build Data

Pipeline Overview

Restart from Stage

Replay

Pipeline Steps

Workspaces

Previous Build

✅ #20 (Oct 8, 2025, 11:46:14 AM)

Add description

Keep this build forever

🕒 Started by GitHub push by Aayush-Bhargav

Started 26 min ago  
Took 49 sec

🕒 This run spent:

- 7.7 sec waiting;
- 49 sec build duration;
- 57 sec total from scheduled to completion.

📦 git

Revision: 145c0ef4e7eda95b9ef644bd918a9de2264b091b  
Repository: [https://github.com/Aayush-Bhargav/SPE\\_MiniProject](https://github.com/Aayush-Bhargav/SPE_MiniProject)

- refs/remotes/origin/main

⚠️ The following steps that have been detected may have insecure interpolation of sensitive variables ([click here for an explanation](#)):

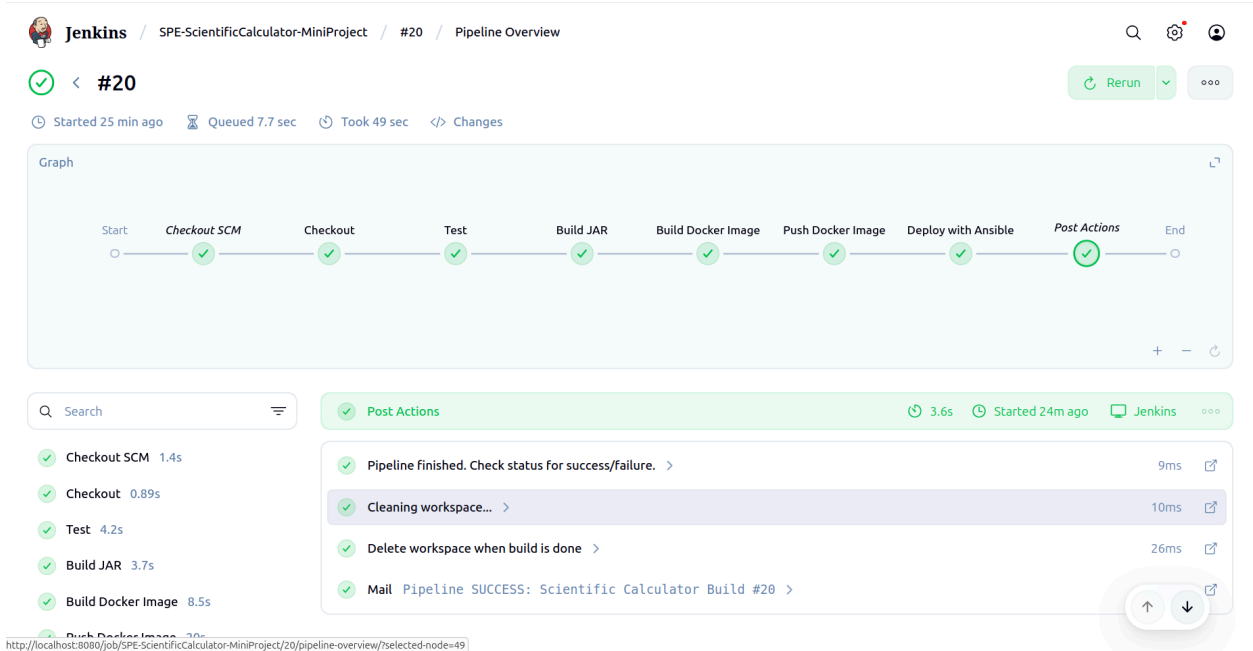
- sh: [DOCKERHUB\_CREDENTIALS\_PSW]

</> Changes

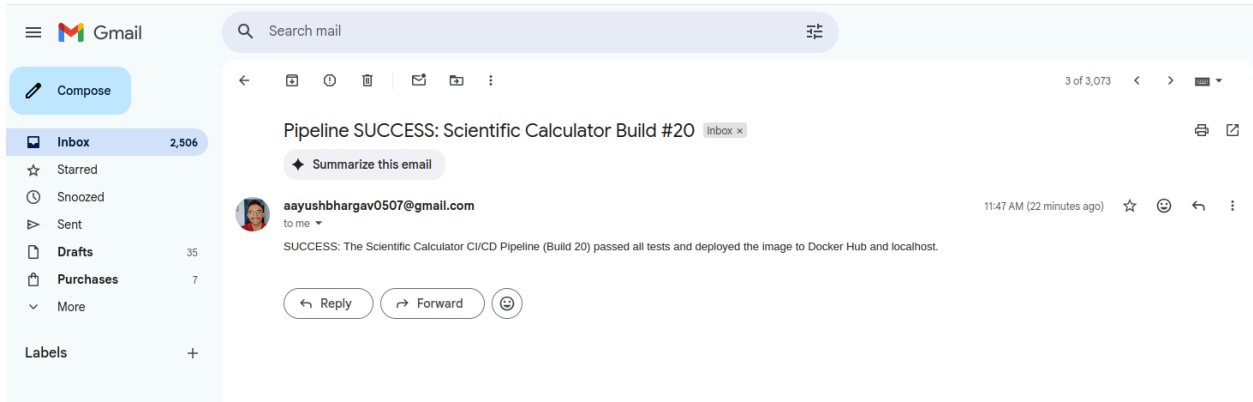
1. Added Email notification Feature (commit: 145c0ef) ([details](#) / [githubweb](#))

REST API

Jenkins 2.516.1



Email Confirmation:



After the build, the docker containers on my local host:

```
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS          NAMES
7a69d71e8a02   aayushbhargav57/scientific-calculator:latest   "java -jar app.jar"     10 seconds ago   Up 10 seconds          scientific_calculator_app
3af7fdd478bc   ubuntu                                "/bin/bash"             2 days ago      Exited (0) 2 days ago   ansible-container
1f3835151a8f   ansible-client                          "/bin/sh -c 'service..." 2 days ago      Exited (137) 2 days ago   node1
eaacd137ed6f   ubuntu                                "/bin/bash"             2 days ago      Exited (0) 47 hours ago   ansible_container
2a64d6d08a9e1   hello-world                            "/hello"                 2 days ago      Exited (0) 2 days ago   clever_clarke
cid3409a8dc3   hello-world                            "/hello"                 2 months ago     Exited (0) 2 months ago   keen_hamilton
6c362fa3937b   hello-world                            "/hello"                 2 months ago     Exited (0) 2 months ago   loving_stonebraker
689d0dcfcb22   hello-world                            "/hello"                 2 months ago     Exited (0) 2 months ago   admiring_khayyam
```

The fact that you see the container status as **Up** means the Ansible playbook is working perfectly and the container is running in the background in the local host, waiting.

The next step you need to do: **use `docker exec` to attach a new interactive process to that running container.**

## The Command to Connect

Since the application requires an interactive terminal (**Scanner**), you need to run a new process **inside** the container that provides that terminal connection.

Here is the command to run on your host machine:

```
docker exec -it scientific_calculator_app java -jar app.jar
```

## Explanation of the Process

1. **`docker exec`**: Executes a new command inside the already running container.
2. **`-it`**: This combination is what unlocks the **interactive terminal**. It connects your current keyboard/terminal to the container's input/output streams.
3. **`scientific_calculator_app`**: The target container ID or name.
4. **`java -jar app.jar`**: The command you want to run. This manually launches the Java application again inside the established interactive session.

When you run this command, you will see your **calculator menu appear**, and you can then enter a choice (1-5) and interact with the application fully!

```
aayush-bhargav@aayush-bhargav-Inspiron-15-3520: $ docker exec -it scientific_calculator_app java -jar app.jar
Welcome to the Scientific Calculator!

-----
Scientific Calculator Operations
-----
1. Square Root ( $\sqrt{x}$ )
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power ( $x^b$ )
5. Exit
-----
Enter your choice (1-5): 2
Enter the non-negative integer (x <= 20): 15
Result: !15 = 1307674368000

-----
Scientific Calculator Operations
-----
1. Square Root ( $\sqrt{x}$ )
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power ( $x^b$ )
5. Exit
-----
Enter your choice (1-5): 5
Thank you for using the Scientific Calculator. Goodbye!
```

Instead of the build now option in jenkins, we can also make a change to the code ourselves, commit it and see the job being executed.

## Writing a new test case:

```
// --- Square Root Function Tests ---

@Test
@DisplayName("Square Root Test: Positive Number")
void squareRoot_positiveInput_returnsCorrectValue() {
    assertEquals( expected: 5.0, calculator.squareRoot( 25.0), DELTA, message: "Square root of 25 should be 5.0.");
}

// new testcase added

@Test
@DisplayName("Square Root Test: Positive Number")
void squareRoot_anotherPositiveInput_returnsCorrectValue() {
    assertEquals( expected: 6.0, calculator.squareRoot( 36.0), DELTA, message: "Square root of 36 should be 6.0.");
}
```

Let us commit the changes and push the changes to github.

```
Terminal Local x + v
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java

no changes added to commit (use "git add" and/or "git commit -a")

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git add .
aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   src/test/java/com/spe/calculator/ScientificCalculatorTest.java

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git commit -m "Added new Testcase"
[main 9d4451f] Added new Testcase
1 file changed, 7 insertions(+)

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

aayush-bhargav@aayush-bhargav-Inspiron-15-3520:~/IdeaProjects/SoftwareProductionEngineering-MiniProject$ git push -u origin main
Username for 'https://github.com': aayush-bhargav
Password for 'https://aayush-bhargav@github.com':
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 12 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (9/9), 697 bytes | 348.88 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/aayush-bhargav/SPE_MiniProject.git
   5276548..9d4451f  main -> main
branch 'main' set up to track 'origin/main'.
```

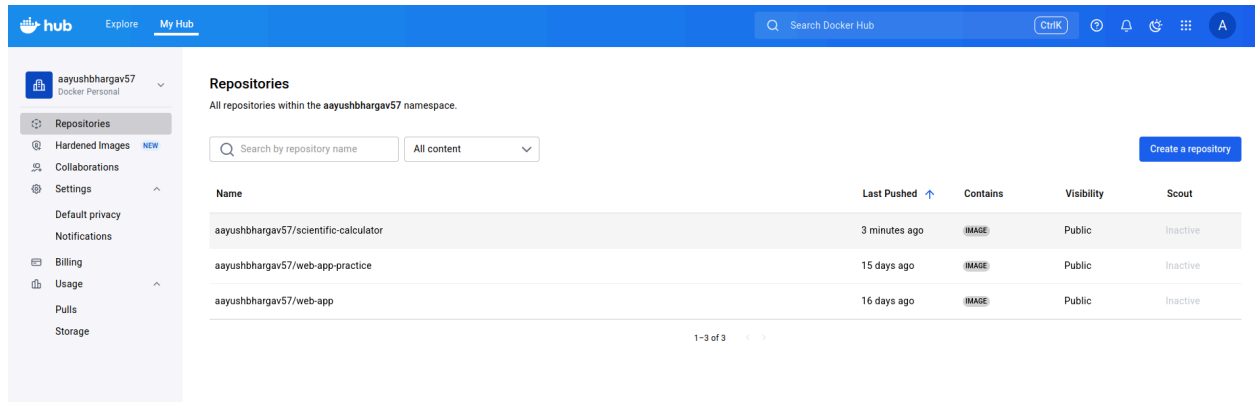
You can clearly see, that the jenkins job has been triggered:

The screenshot shows the Jenkins web interface for a project named "SPE-ScientificCalculator-MiniProject". On the left sidebar, there are links for "Build Now", "Configure", "Delete Pipeline", "GitHub", "Stages", "Rename", "Pipeline Syntax", and "GitHub Hook Log". The main area displays "Permalinks" with a list of build links: "Last build (#18), 2.7 sec ago", "Last stable build (#17), 11 min ago", "Last successful build (#17), 11 min ago", "Last failed build (#12), 7 days 10 hr ago", "Last unsuccessful build (#12), 7 days 10 hr ago", and "Last completed build (#17), 11 min ago". Below this, a "Builds" section shows a list of builds for "Today": build #18 at 11:14 PM (status: failed), build #17 at 11:03 PM (status: successful), and build #16 at 7:34 PM (status: successful). Each build entry has a status icon, a number, a time, and a dropdown arrow.

**Build completed Successfully as well!**

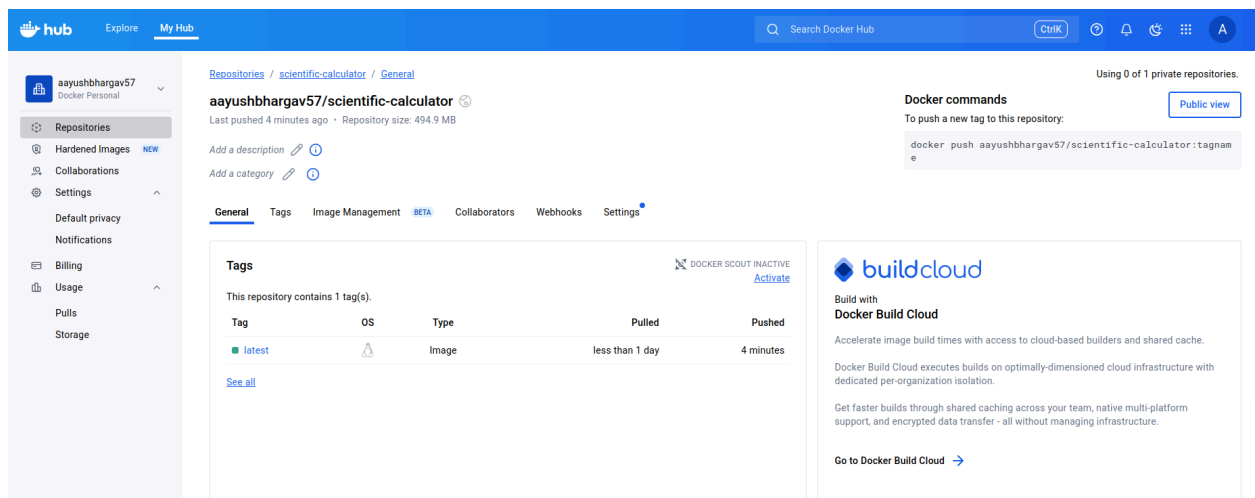
This screenshot shows the Jenkins interface with the "Status" tab selected in the left sidebar. The main area displays a green checkmark icon followed by the text "SPE-ScientificCalculator-MiniProject". Below this, it says "This is the CI/CD pipeline job for the SPE Mini Project." The "Permalinks" section lists build links: "Last build (#18), 1 min 12 sec ago", "Last stable build (#18), 1 min 12 sec ago", "Last successful build (#18), 1 min 12 sec ago", "Last failed build (#12), 7 days 10 hr ago", "Last unsuccessful build (#12), 7 days 10 hr ago", and "Last completed build (#18), 1 min 12 sec ago". The "Builds" section at the bottom shows only one build for "Today": build #18 at 11:14 PM (status: successful). The "Edit des" button is visible in the top right corner.

The docker container will be on the local host and we can run it again as before.  
**You can also verify that the image has been created and pushed to my dockerhub :**



The screenshot shows the Docker Hub interface for the user 'aayushbhargav57'. The left sidebar contains navigation links: Repositories, Hardened Images, Collaborations, Settings, Default privacy, Notifications, Billing, Usage, Pulls, and Storage. The main content area is titled 'Repositories' and shows a list of repositories within the user's namespace. The list includes 'aayushbhargav57/scientific-calculator', 'aayushbhargav57/web-app-practice', and 'aayushbhargav57/web-app'. Each repository entry shows its name, last pushed time, content type (IMAGE), visibility (Public), and Scout status (Inactive).

Name	Last Pushed	Contains	Visibility	Scout
aayushbhargav57/scientific-calculator	3 minutes ago	IMAGE	Public	Inactive
aayushbhargav57/web-app-practice	15 days ago	IMAGE	Public	Inactive
aayushbhargav57/web-app	16 days ago	IMAGE	Public	Inactive



The screenshot shows the Docker Hub repository page for 'aayushbhargav57/scientific-calculator'. The left sidebar is the same as the previous screenshot. The main content area shows the repository details, including the name, last pushed time, and repository size. The 'Tags' section shows a single tag 'latest' with OS 'linux', Type 'Image', Pulled 'less than 1 day', and Pushed '4 minutes'. The 'Docker commands' section shows the command 'docker push aayushbhargav57/scientific-calculator:tagname'. The 'Build with Docker Build Cloud' section is also visible.

**Tags**

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	less than 1 day	4 minutes

**Docker commands**

To push a new tag to this repository:

```
docker push aayushbhargav57/scientific-calculator:tagname
```

**Build with Docker Build Cloud**

Accelerate image build times with access to cloud-based builders and shared cache.

Docker Build Cloud executes builds on optimally-dimensioned cloud infrastructure with dedicated per-organization isolation.

Get faster builds through shared caching across your team, native multi-platform support, and encrypted data transfer - all without managing infrastructure.

[Go to Docker Build Cloud](#)

Link to public Github Repository:

[https://github.com/Aayush-Bhargav/SPE\\_MiniProject](https://github.com/Aayush-Bhargav/SPE_MiniProject)

Link to Dockerhub:

<https://hub.docker.com/u/aayushbhargav57>