SPE Mini Project Report

IMT2022076 Mohit Naik

GitHub Repository

DockerHub Repository

Oct 10, 2025

Contents

1 Introduction **3**
- 1.1 What is DevOps . . . 3
- 1.2 How to implement DevOps . . . 3

2 Requirements **3**
- 2.1 Functional requirements . . . 3
- 2.2 Non-functional requirements . . . 4

3 Tools Used & Setup **4**
- 3.1 Java & Maven . . . 4
- 3.2 Git & GitHub . . . 5
- 3.3 Docker . . . 5
- 3.4 Jenkins . . . 6
- 3.5 Ansible . . . 6
- 3.6 Ngrok . . . 7

4 Credentials Required **7**
- 4.1 GitHub . . . 7
- 4.2 Docker Hub . . . 7
- 4.3 Gmail . . . 8
- 4.4 Ngrok . . . 8

5 Setting up Jenkins **9**

6 Project Workflow & Setup **12**
- 6.1 Creating a project in VSCode . . . 12
- 6.2 Building & testing with maven . . . 13

7 Setting Up the Project with Git and GitHub **14**
- 7.1 Initializing a git repository . . . 14
- 7.2 Creating a .gitignore file . . . 15
- 7.3 Staging and commiting changes . . . 15
- 7.4 Connecting to GitHub . . . 15

8 Containerizing the Project using Docker **15**
8.1 Creating a Dockerfile 15
8.2 Explanation of Dockerfile 16
8.3 Creating a .dockerignore file 16
8.4 Pushing image to Docker Hub 16

9 Ansible Setup **17**
9.1 Creating the inventory file 17
9.2 Creating the playbook file 17
9.3 Generating SSH keys 18

10 Jenkins Pipeline Setup **18**
10.1 Creating a new pipeline 18
10.2 Configuring the pipeline 19
10.3 Jenkins pipeline explanation 19

11 GitHub Webhook Setup **21**

1 Introduction

1.1 What is DevOps

DevOps is a modern approach to software development that builds on earlier methodologies like Agile and the traditional Waterfall model. Over time, these methods have helped teams work more efficiently, but DevOps takes things even further. While the Waterfall model followed a strict, step-by-step process, Agile introduced shorter development cycles, allowing teams to deliver features more quickly. However, even Agile can leave certain phases somewhat isolated, with developers, operations, and testers still working in separate silos.

DevOps breaks down those barriers. It treats software development as a continuous, interconnected process, where everyone - developers, operations, testers - collaborates closely. It's not just about writing code; it's about creating a continuous feedback loop. Features are developed, deployed, and refined in real time, with input from all parts of the team. The biggest advantage of DevOps is this seamless integration between development and operations. It enables faster development, quicker deployment of new features, and constant feedback from users, creating a more responsive and adaptive software process.

1.2 How to implement DevOps

To implement DevOps effectively, you need a well-coordinated set of tools that facilitate collaboration across all stakeholders. DevOps relies on automation, continuous testing, and rapid feedback to ensure that new features are quickly validated without breaking existing functionality.

Some important tools and practices include:

- An IDE with Git and build tool integration (VSCode).
- A tool for build automation and dependency management (Maven).
- Local and remote version control (Git/GitHub).
- Continuous integration and deployment, and pipeline automation (Jenkins).
- A tool for containerization and consistent deployments (Docker).
- A registry for storing and sharing container images (DockerHub).
- Tools for environment management and deployment automation (Ansible).

2 Requirements

2.1 Functional requirements

The project aims to develop a scientific calculator program that allows the user to perform the following operations:

- **Square root (`sqrt(x)`)**: Calculates the square root of a given number x, i.e., $\sqrt{x}$.
- **Factorial (`fact(x)`)**: Computes the factorial of a non-negative integer x, i.e., $x! = x \times (x-1) \times \cdots \times 1$.

- **Natural logarithm (`nlog(x)`)**: Finds the natural logarithm (base e) of a given number x, i.e., $\log_e x$.
- **Power (`pow(x,b)`)**: Raises a number x to the power b, i.e., x^b.

Each operation should be selectable through a user-friendly menu interface. Error checking should handle edge cases such as negative inputs for square roots and logarithms, or non-integer values for factorial.

2.2 Non-functional requirements

- **Tech Stack**: Choose an appropriate stack (Spring Boot).
- **DevOps Suitability**: Ensure DevOps is a suitable approach for project goals and requirements.
- **User Assistance**: Provide clear error/help messages.
- **Code Quality**: Maintain clean, well-documented, and understandable code.
- **Version Control**: Use Git and GitHub for source code management.
- **Build Automation**: Automate builds with Maven or equivalent tools.
- **CI/CD**: Use Jenkins (or equivalent) for integration, testing, and deployment automation.
- **Containerization**: Package the application into Docker containers.
- **Environment Management**: Use Ansible (or equivalent) for consistent deployments.
- **Documentation**: Provide complete project documentation for future reference.

3 Tools Used & Setup

You need a Linux-based system for all the steps below. You can also use a VM, WSL, or a Linux container.

3.1 Java & Maven

Java is a high-level, object-oriented programming language widely used for building platform-independent applications. It ensures compatibility with tools like Maven for automation and build management. The Java Runtime Environment (JRE) provides the execution environment.

Maven is a build automation tool that simplifies project management by automating tasks such as compiling, testing, and packaging. It ensures consistent builds across environments by handling dependencies via central repositories.

```
$ sudo apt update
$ sudo apt install openjdk-17-jdk maven

$ java --version
openjdk 17.0.16 2025-07-15 LTS
```

```
OpenJDK Runtime Environment Temurin-17.0.16+12 (build 17.0.16+12-LTS)
OpenJDK 64-Bit Server VM Temurin-17.0.16+12 (build 17.0.16+12-LTS,
mixed mode, sharing)
```

```
$ mvn --version
Apache Maven 3.9.9
Maven home: /usr/share/maven
Java version: 17.0.6, vendor: Eclipse Adoptium, runtime:
/usr/lib/jvm/temurin-17-jdk-amd64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.6.87.2-microsoft-standard-wsl2",
arch: "amd64", family: "unix"
```

3.2 Git & GitHub

Git is a distributed version control system for tracking changes, rolling back, and managing parallel development, while GitHub is a cloud-based platform for hosting Git repositories, enabling remote collaboration, pull requests, and integrations with Jenkins.

```
$ sudo apt update
$ sudo apt install git
```

```
$ git --version
git version 2.47.3
```

3.3 Docker

Docker is a platform for deploying applications inside lightweight, portable containers. Containers encapsulate the code, runtime, libraries, and dependencies, ensuring consistency across environments. This project uses Docker to containerize the scientific calculator.

```
$ sudo apt remove docker docker-engine docker.io containerd runc
$ sudo apt update
$ sudo apt install ca-certificates curl gnupg
$ sudo install -m 0755 -d /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
$ echo \ "deb [arch=$(dpkg --print-architecture)
  signed-by=/etc/apt/keyrings/docker.gpg]
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
$ sudo apt update
$ sudo apt install docker-ce docker-ce-cli containerd.io
  docker-buildx-plugin docker-compose-plugin
```

```
$ sudo usermod -aG docker $USER
```

```
$ newgrp docker
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

3.4 Jenkins

Jenkins is an open-source automation server used for continuous integration and deployment. It automates builds, testing, and deployments, supporting plugin-based integration with GitHub, Docker, and more.

```
$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
$ echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

$ sudo apt update
$ sudo apt install jenkins
$ sudo systemctl start jenkins
$ sudo systemctl status jenkins
jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded
     Active: active (running)
 Invocation: 3fe5b8957df0485cb3228bdcb4f08bf4
   Main PID: 219 (java)
      Tasks: 57 (limit: 9082)
     Memory: 596.2M (peak: 708.3M)
        CPU: 25.596s
     CGroup: /system.slice/jenkins.service
     ...
```

3.5 Ansible

Ansible is an open-source automation tool for managing and configuring environments. It uses YAML playbooks to define deployment tasks. In this project, Ansible deploys Docker containers to consistent environments, reducing errors and simplifying scaling.

```
$ sudo apt update
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install -y ansible
$ ansible --version
ansible [core 2.19.0b6]
  config file = None
  configured module search path = ['/home/mohit/.ansible/plugins/modules',
  '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/mohit/.ansible/collections:
  /usr/share/ansible/collections
  executable location = /usr/bin/ansible
```

```
python version = 3.13.5 (main, Jun 25 2025, 18:55:22)
[GCC 14.2.0] (/usr/bin/python3)
jinja version = 3.1.6
pyyaml version = 6.0.2 (with libyaml v0.2.5)
```

3.6 Ngrok

Ngrok is a tool that exposes local servers to the public internet through secure tunnels. It creates temporary public URLs for local applications, enabling testing, demos, and remote access without complex network configuration. In this project, ngrok is used to obtain a static public URL for the local Jenkins server to allow GitHub webhooks to work.

```
$ curl -sSL https://ngrok-agent.s3.amazonaws.com/ngrok.asc \
  | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null \
  && echo "deb https://ngrok-agent.s3.amazonaws.com bookworm main" \
  | sudo tee /etc/apt/sources.list.d/ngrok.list \
  && sudo apt update \
  && sudo apt install ngrok
```

4 Credentials Required

Before proceeding with the setup, implementation and deployment of the project, the following credentials are required:

4.1 GitHub

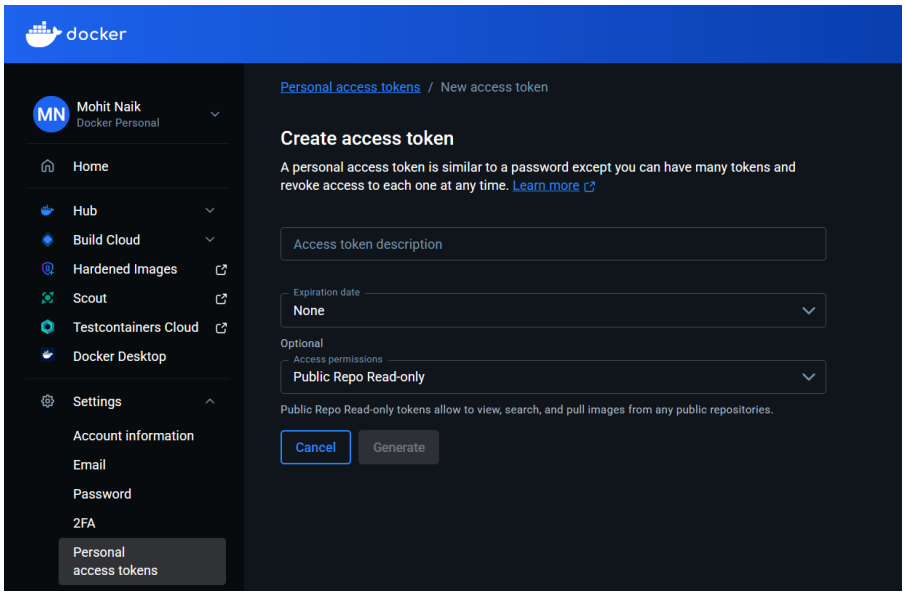
- **Username**: Required for cloning private repositories.
- **Personal Access Token**: Generated on GitHub, used instead of password for authentication via CLI or API.

GitHub credentials are only required when the repository is private. In case of a public repository, this is not needed.

4.2 Docker Hub

- **Username**: Required for pushing/pulling docker images.
- **Password / Access Token**: Used for authentication to interact with Docker Hub.

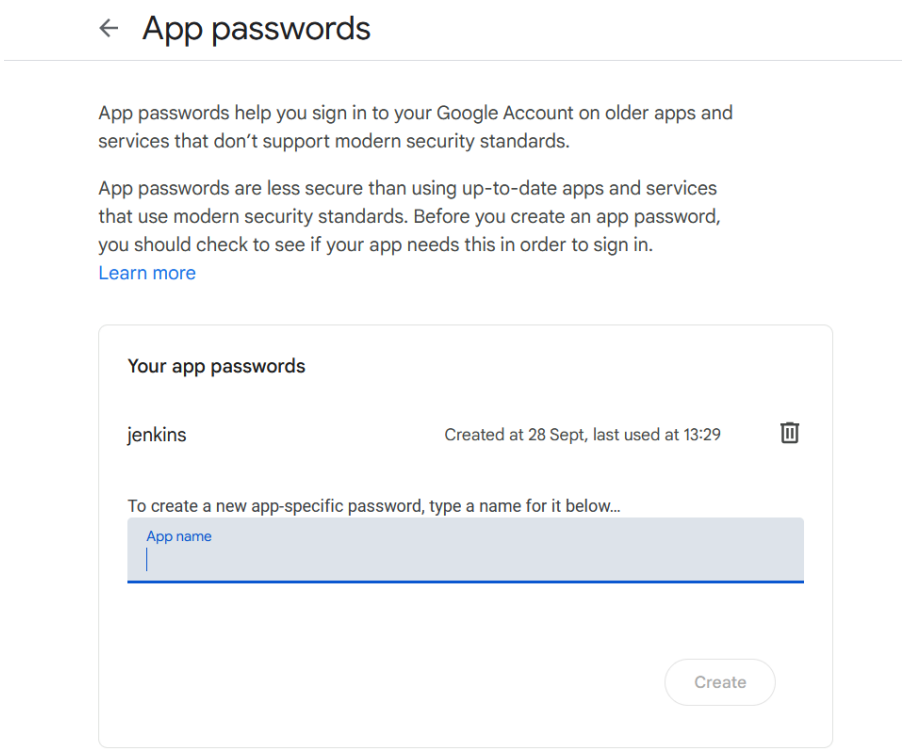
To set this up, go to `app.docker.com/accounts/your-username/settings/personal-access-tokens`, and click on the `Generate new token` button. Then fill in the details and save the created token.



4.3 Gmail

- **Email Address**: Required for Jenkins email notifications.
- **App Password**: App-specific password for secure SMTP authentication.

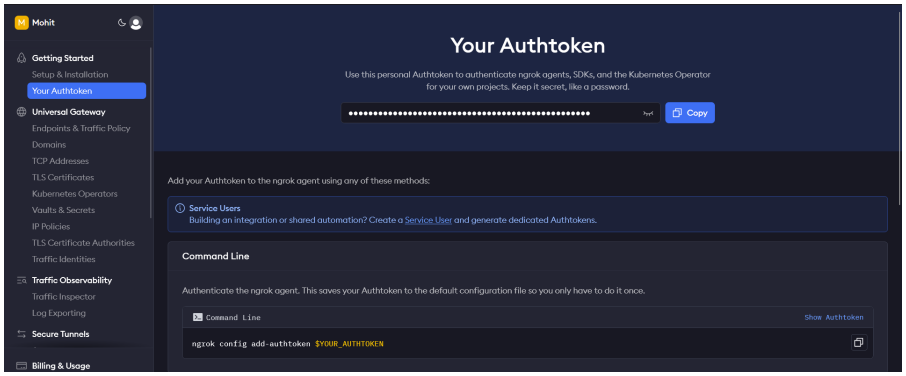
To set this up, go to `myaccount.google.com`, search for `app passwords`, follow the steps and save the created app password.

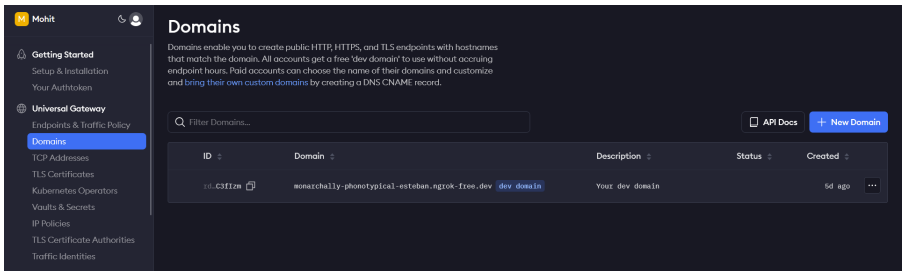


4.4 Ngrok

To expose your local app via a public URL using Ngrok, you need an auth token. Log in / Sign up to Ngrok and go to `dashboard.ngrok.com/get-started/your-authtoken`, and follow the

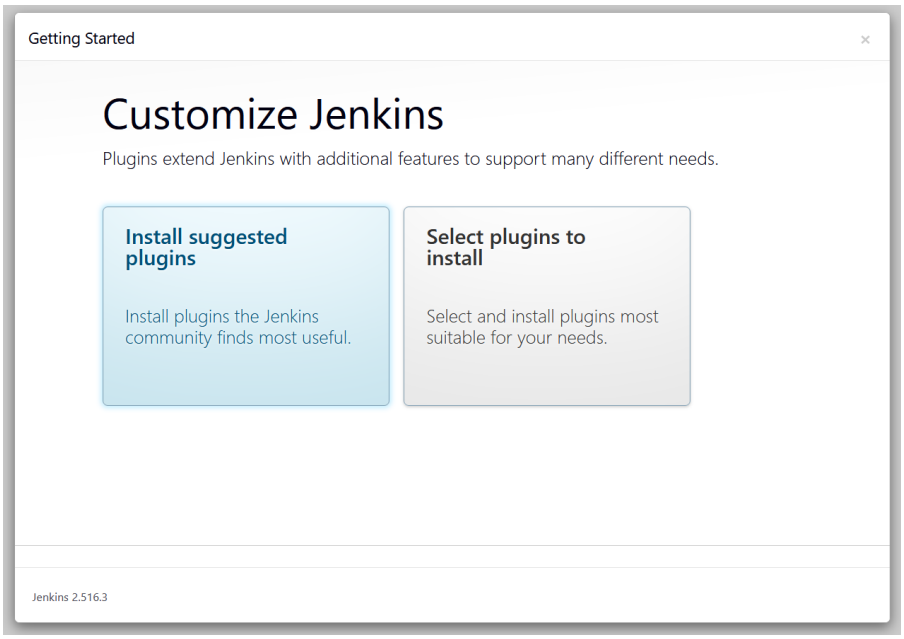
steps given there. You can also find your static public URL at `dashboard.ngrok.com/domains`.

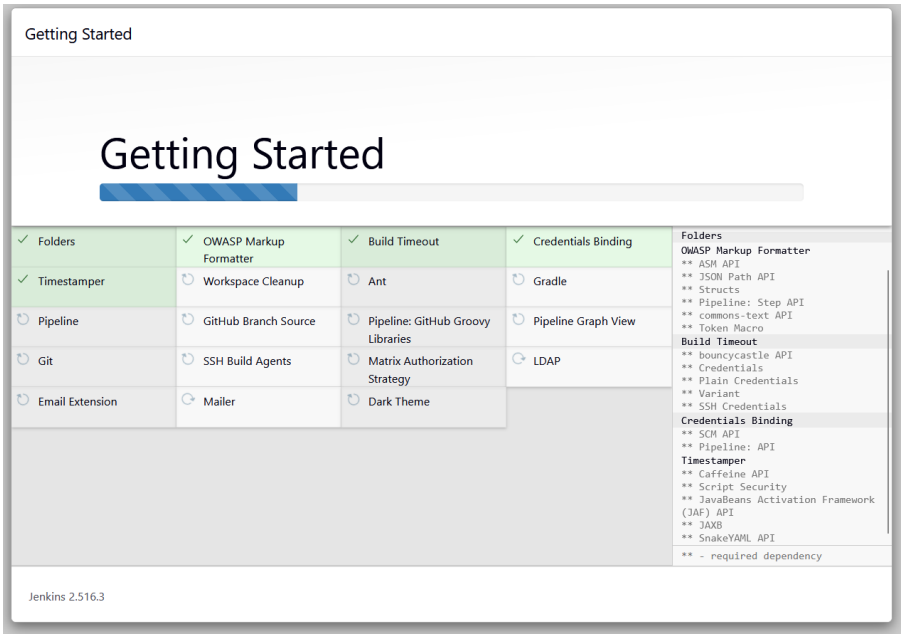




5 Setting up Jenkins

You can use Jenkins locally at `localhost:8080`. The first time, you will need to give your administrator password. You can then install plugins and set up new users.

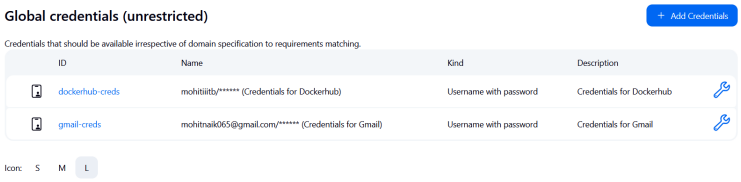

Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.
Jenkins 2.516.3

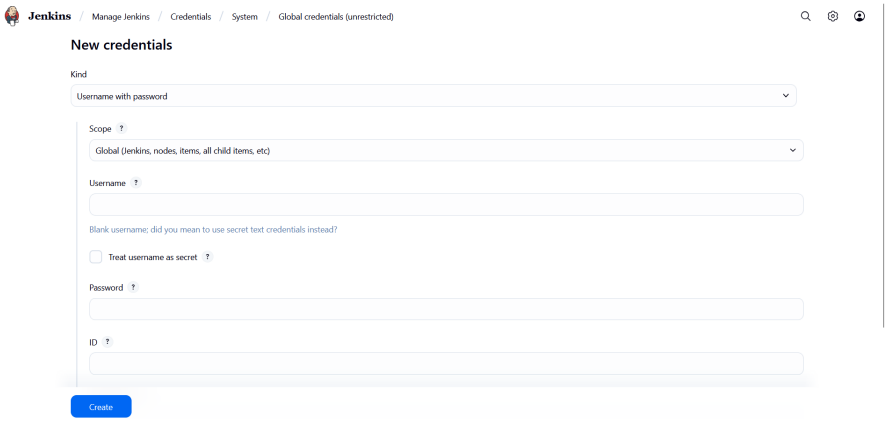

Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline Graph View
Git
SSH Build Agents
Matrix Authorization Strategy
LDAP
Email Extension
Mailer
Dark Theme
Folders
OWASP Markup Formatter
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** commons-text API
** Token Macro
Build Timeout
** bouncycastle API
** Credentials
** Plain Credentials
** Variant
** SSH Credentials
Credentials Binding
** SCM API
** Pipeline: API
Timestamper
** Caffeine API
** Script Security
** JavaBeans Activation Framework (JAF) API
** JAXB
** SnakeYAML API
** - required dependency
Jenkins 2.516.3

Getting Started
Create First Admin User
Username
mohit
Password
Confirm password
Full name
Mohit Naik
E-mail address
Jenkins 2.516.3
Skip and continue as admin
Save and Continue

To securely manage credentials in Jenkins without hardcoding them in the Jenkinsfile:

- Navigate to `Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted)`.
- Click on `Add Credentials`.

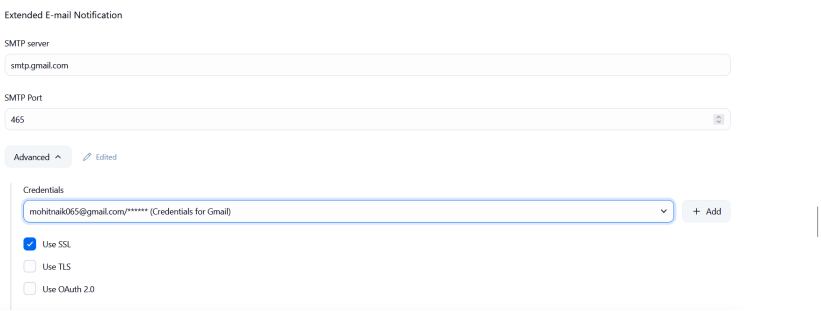


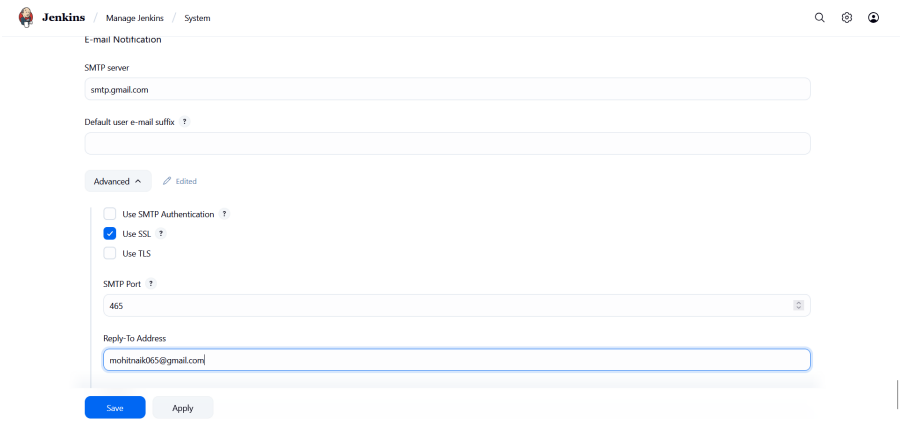


Add the following credentials:

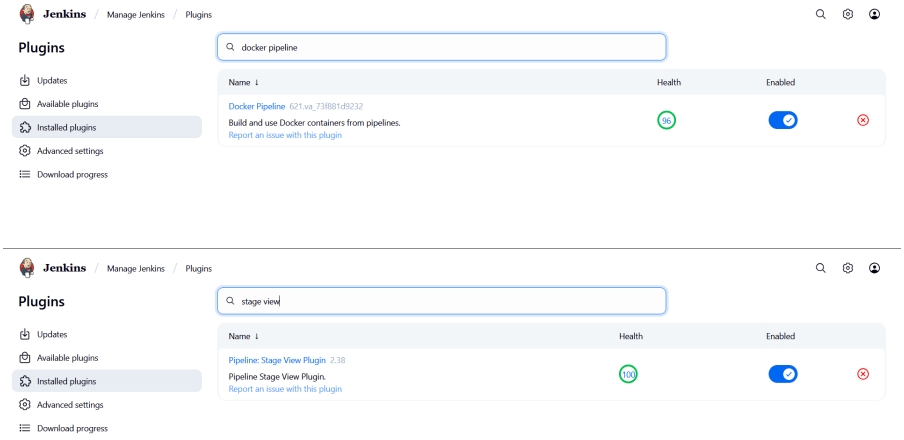
- **Docker Hub Credentials**: ID: `dockerhub-creds`
  Password: The personal access token obtained from Docker Hub
- **Google SMTP Credentials**: ID: `gmail-creds`
  Password: The app password you set up previously.

You also need to set up email notifications so that you get an email providing the status for each build. To do that, go to `System > Extended Email Notifications` and `System > Email Notification` and fill as follows:





You also need to install the Stage View plugin to get a brief overview of your previous builds, and the Docker Pipeline plugin to support secure login to Docker Hub via your Jenkinsfile using `docker.withRegistry()`.



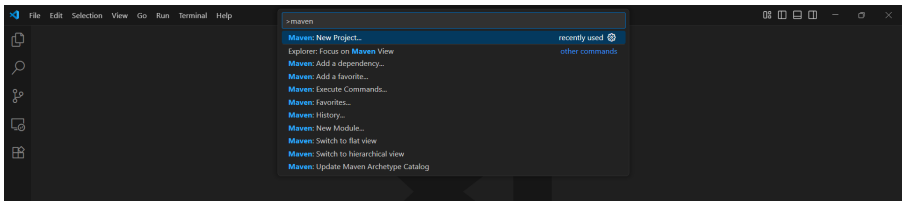
6 Project Workflow & Setup

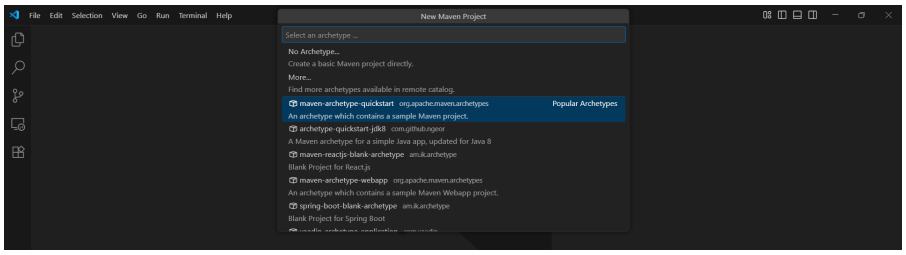
6.1 Creating a project in VSCode

1. Open VSCode and press `Ctrl + Shift + P`.
2. Search for `Maven: New Project`.
3. Select `maven-archetype-quickstart`.
4. Enter the package name and artifact ID.

This generates a blank Maven project with a structure similar to one below. You can then add source files and tests as needed. Any dependencies, versions, configurations etc. can be written in `pom.xml`.

```
pom.xml
src/
    main/java/com/example/App.java
    test/java/com/example/AppTest.java
target/
```



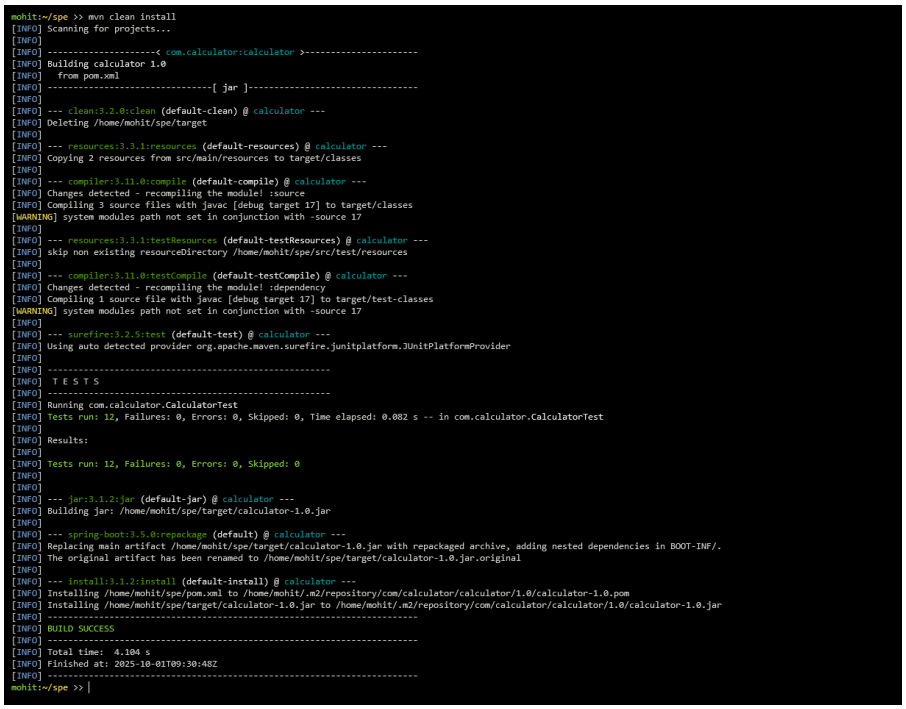


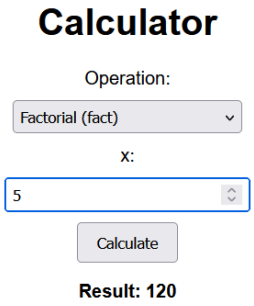
6.2 Building & testing with maven

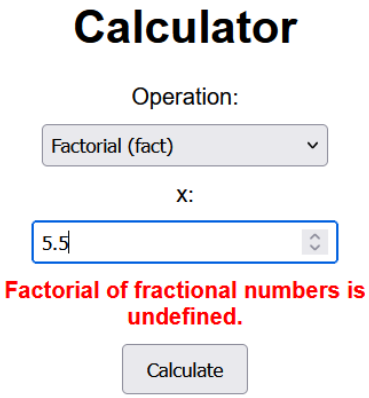
After you are done implementing the code and tests, run the following commands:

```
// Remove the target directory (previous build) to ensure a fresh build.
$ mvn clean

// Execute all tests in \texttt{src/test/java}.
$ mvn test

// Package the app into an executable jar file.
$ mvn package (mvn install will also work)

// Run the app via the jar file
$ java -jar target/calculator-1.0.jar
```

You can then access the app at `localhost:8081`.







7 Setting Up the Project with Git and GitHub

To incorporate version control in the project and allow for collaboration, we can add the project to a Git/GitHub repository.

7.1 Initializing a git repository

Navigate to your project root folder and run

```
$ git init
```

This initializes a git repository at that folder.

7.2 Creating a .gitignore file

A `.gitignore` prevents unnecessary files from being tracked. Usually you dont want the compiled classes, executables and local settings to be tracked, so the `.gitignore` can look as follows:

```
target/
.vscode/
```

7.3 Staging and commiting changes

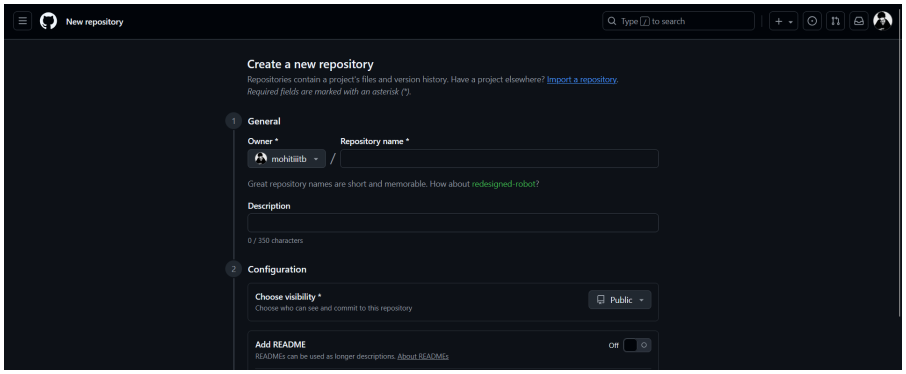
Your changes now become a part of the version history, and you can revert to this state at any time.

```
$ git add .
$ git commit -m "Initial commit"
```

7.4 Connecting to GitHub

This is to save your code to a remote repository so other people can see and use it. Log in to GitHub and create a public repository `calculator-spe`. Then in the project root, run

```
$ git remote add origin https://github.com/your-username/calculator-spe.git
$ git push -u origin main
```



8 Containerizing the Project using Docker

8.1 Creating a Dockerfile

A `Dockerfile` is a set of instructions for setting up the environment, dependencies and files in a Docker image. In our case, we only need a minimal Java runtime and the executable `.jar` file, so our Dockerfile can look as follows:

```
FROM eclipse-temurin:17-jre-alpine
WORKDIR /app
COPY target/calculator-1.0.jar app.jar
EXPOSE 8081
ENTRYPOINT ["java","-jar","app.jar","--server.port=8081"]
```

8.2 Explanation of Dockerfile

- `FROM eclipse-temurin:17-jre-alpine`: Uses the lightweight Alpine-based image of Eclipse Temurin Java 17 runtime as the base.
- `WORKDIR /app`: Sets `/app` as the working directory inside the container. All subsequent commands run relative to this directory.
- `COPY target/calculator-1.0.jar app.jar`: Copies your built Java JAR from the host (target/calculator-1.0.jar) into the container as `app.jar`.
- `EXPOSE 8081`: Declares that the container listens on port 8081 (informational, used for mapping ports).
- `ENTRYPOINT ["java","-jar","app.jar","--server.port=8081"]`: Specifies the command to run when the container starts: it runs the JAR with Java and sets the server port to 8081.

8.3 Creating a .dockerignore file

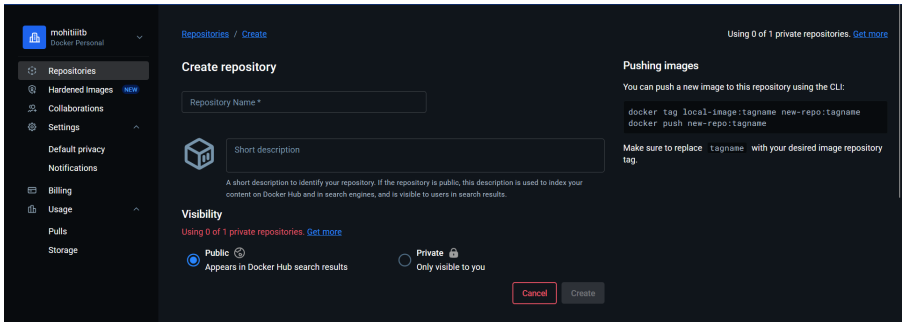
Docker only requires the `.jar` file, so we can exclude all other files to make the image lighter and faster. The `.dockerignore` file will look as follows:

```
**
.*
!target/calculator-1.0.jar
```

8.4 Pushing image to Docker Hub

Push the image to the Docker Hub registry so others can view, pull and run the image.

```
// Build Docker image
$ docker build -t your-dockerhub-username/calculator-spe:latest .

// Login to Docker Hub
$ docker login --username your-dockerhub-username

// Push image
$ docker push your-dockerhub-username/calculator-spe:latest
```



9 Ansible Setup

Ansible allows you to pull Docker images and run them in a specified deterministic environment and configuration, for consistency and convenience.

9.1 Creating the inventory file

The inventory file is what tells Ansible which machines (hosts) it should manage and how to connect to them. In this case, we only need to manage the local machine, so the `inventory.ini` file will look as follows:

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

9.2 Creating the playbook file

The playbook defines tasks and deployment steps to be performed in the machines listed in the inventory. In our case, we want to remove existing outdated calculator images and containers, pull the latest image and run it. So, the `playbook.yml` will look as follows:

```
---
- hosts: calculator_hosts
  tasks:

    - name: Ensure old container is removed
      community.docker.docker_container:
        name: calculator_app
        state: absent
      ignore_errors: true

    - name: Remove existing Docker image
      community.docker.docker_image:
        name: mohitiiitb/calculator-spe:latest
        state: absent
      ignore_errors: true

    - name: Pull the latest Docker image
      community.docker.docker_image:
        name: mohitiiitb/calculator-spe:latest
        source: pull

    - name: Run Calculator container in detached mode
      community.docker.docker_container:
        name: calculator_app
        image: mohitiiitb/calculator-spe:latest
        state: started
        tty: yes
        detach: yes
        recreate: true
        published_ports:
          - "8081:8081"
```

```
    - name: Print access URL
      debug:
        msg: "Application running at http://localhost:8081"
```

To run the playbook, enter

```
$ ansible-playbook -i inventory.ini playbook.yml
```

If everything works correctly, the playbook should pull and run the image, and you will be able to access the application on `localhost:8081`. Note that you need Docker to be able to run without `sudo`, otherwise this step will not work.

9.3 Generating SSH keys

You need SSH keys to establish connections between the Jenkins server and the Ansible server, and Ansible and the target servers. But in this case, everything is taking place on the same machine (`localhost`), so you dont need to set up any SSH keys.

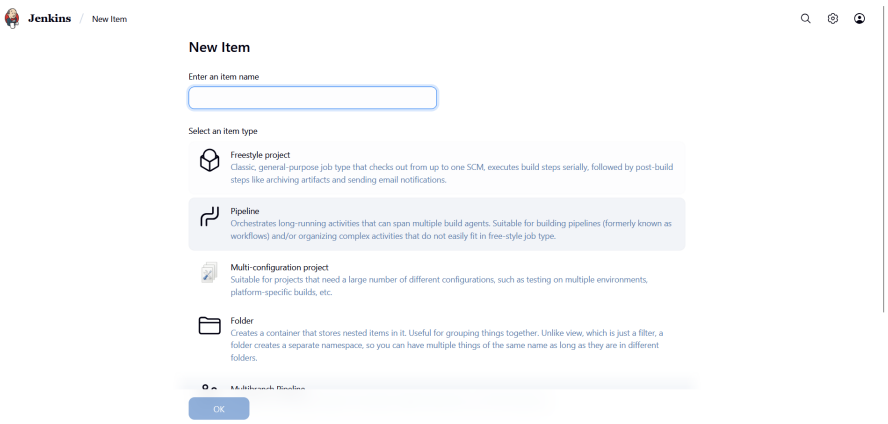
```
// Generate key pair
$ ssh-keygen -t rsa

// Copy key to server
$ ssh-copy-id your_username@127.0.0.1

$ ansible myhosts -i inventory.ini -m ping
// Should return a success message if configured correctly
```

10 Jenkins Pipeline Setup

We can now set up a Jenkins pipeline so that all the above steps (testing, building, containerizing, pushing, pulling, running) can take place with a single button click.

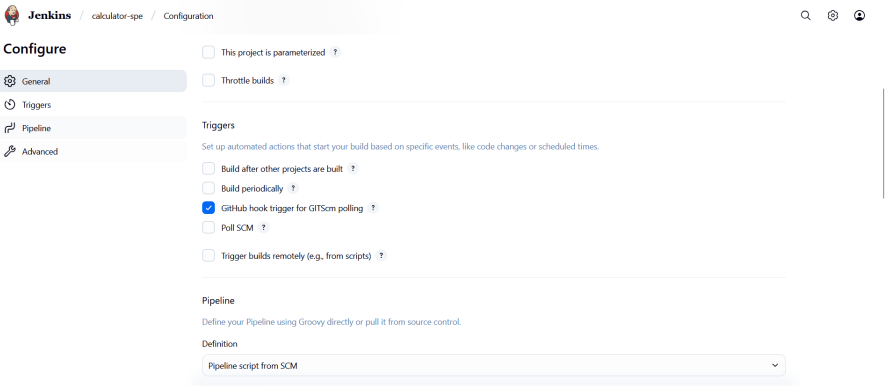
10.1 Creating a new pipeline

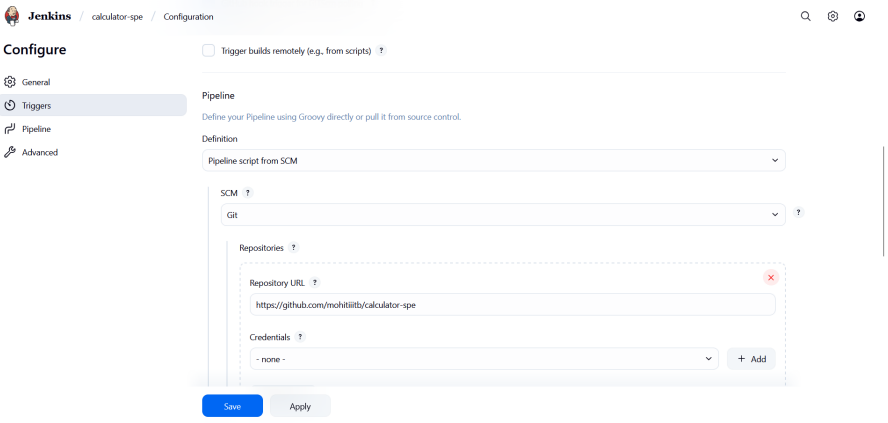
1. Open Jenkins Dashboard.
2. Click on `New Item > Enter project name`.
3. Select `Pipeline` and click OK.



10.2 Configuring the pipeline

- **Build Trigger**: Select `GitHub hook trigger for GITSCM polling`.
- **Pipeline from SCM**: Choose Git, add repository URL, select GitHub credentials.
- **Script Path**: Default `Jenkinsfile`.





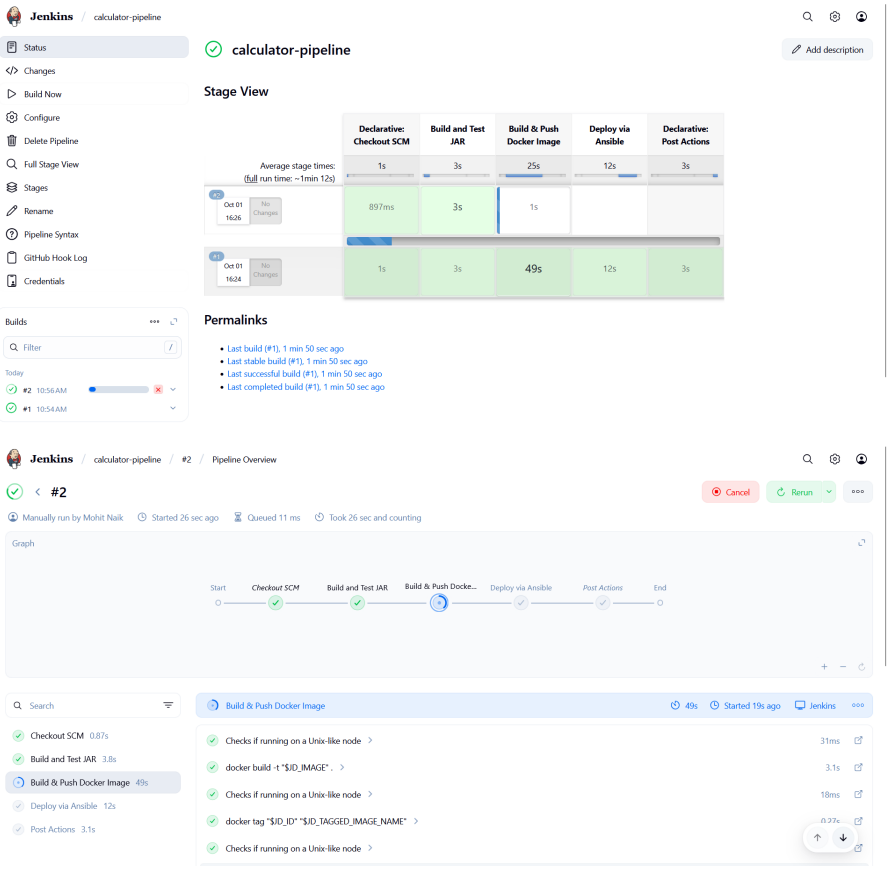
10.3 Jenkins pipeline explanation

The Jenkinsfile should have the following stages:

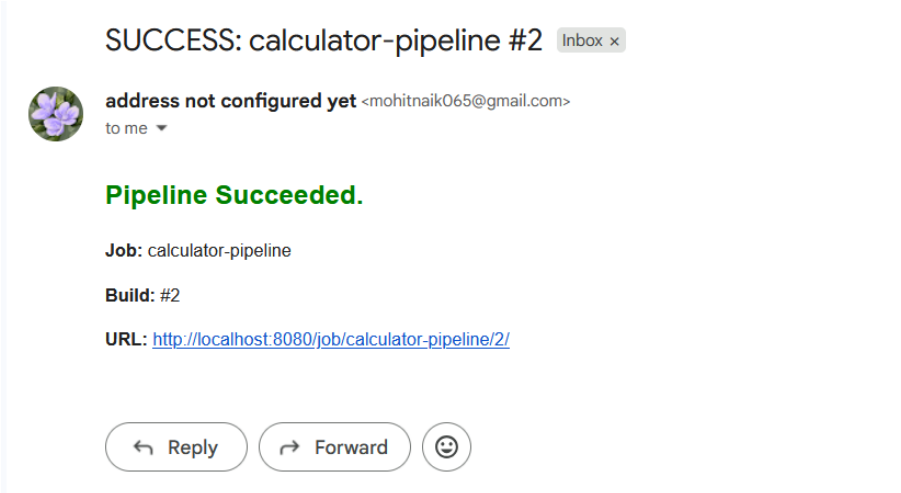
- **Checkout SCM**: Fetch the source code from GitHub that triggered the build.
- **Build and Test**: Executes unit tests using Maven and builds the `.jar` executable.

- **Build Docker Image**: Builds and tags the application Docker image containing the executable.
- **Login to DockerHub**: Authenticates using stored credentials.
- **Push Image**: Pushes the built image to Docker Hub.
- **Deploy via Ansible**: Runs an Ansible playbook to deploy the application.
- **Post Actions**: Notify success/failure to the configured email address. If failed, also clean the workspace.

A successful build will look as follows:



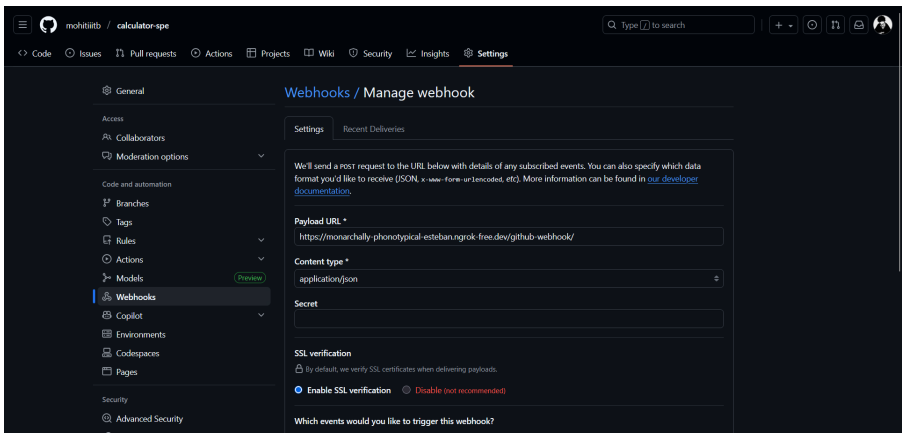
And you will also get a success notification on your email:



11 GitHub Webhook Setup

To automatically start the Jenkins build pipeline when you push your commits to GitHub, we need to set up a Github webhook. To do that, you need a public static Jenkins URL, which you can get via Ngrok.

1. In your terminal, run `ngrok http --domain=your-ngrok-domain 8080` (If Jenkins is on `localhost:8080`.
2. Open your repository on GitHub and go to `Settings > Webhooks`.
3. Click on `Add Webhook`.
4. In `Payload URL`, enter your ngrok domain.
5. Set `Content type` to `application/json`.
6. Select `Just the push event`.
7. Click `Add Webhook`.



You can then push a commit to GitHub and verify that Jenkins automatically starts the pipeline.