SPE Mini Project

IMT2019010 Ankit Agrawal

Project Report

Guided By: Prof. B. Thangaraju

Problem Statement

The goal of the project is to create a Scientific Calculator that supports the following operations:

1. Square Root Function - ($\sqrt{x}$)
2. Factorial Function - ($x!$)
3. Natural Logarithm (base e) - ($\ln x$)
4. Power Function - (a^b)

However, one needs to follow the complete DevOps Principles while building the program. This would also include using a CI/CD pipeline to control the life-cycle of program building.

What is DevOps?

DevOps is a set of cultural concepts, practices and tools that help organizations create better application and services, while reducing the time required to do so.

The main idea behind DevOps is to break down the barrier between:

1. **Software Developers:** who want to add new features to applications and services
2. **Operators:** who want to provide stability to client, and ensure high availability of services to clients.

DevOps does so, by:

1. Ensuring that both the teams work together throughout the life-cycle of the application, from building of application to its deployment, which reduces friction between these two teams and help them deliver quality services, faster.
2. Employing best-practices to automate the majority of tasks that were previously done manually, such as building and deployment.
3. Following the CALMS principle:
    a. Culture: Teams support each other, avoid finger pointing and conflicts
    b. Automation: Using tools that enable faster building of services, and automatic deployment

c. **L**ean: Focus on value for end-user, and proceed in small steps while building applications and services
d. **M**easure: Measure everything, and show improvements
e. **S**haring: Collaboration between Developers and Operations teams

Why DevOps?

There are a lot of benefits of following DevOps principles while building applications and services, some of which are:

1. **Faster time-to-market:** which beats competition and increases profits.
2. **Better quality applications and services:** developers not only build code, but also build test cases to check the correctness of their applications, ensuring they are of high quality.
3. **Easier management of application life-cycle:** Since both Developers and Operators co-operate in building applications, the management of the application life-cycle becomes easier.

Tools Used in DevOps

A plethora of tools are available that help in building the CI/CD pipeline. Some of these tools are given in the table below:

Purpose	Tools
Building CI/CD Pipeline	Jenkins Travis CodeShip
Version Control System	Git Mercurial Subversion
Build Tools	Maven Ant Gradle
Testing Applications	JUnit Selenium CppUnit

Deployment of Applications	Docker Amazon AWS
Operating on Services/ Infrastructure as Code	Ansible Chef Kubernetes
Monitoring Services	ELK Stack

Tools used For Project

I chose the following tools while doing the Mini-Project:

1. **Git:** A fantastic Version Control System, that allows us to track changes in code, and manage its different versions. More information about it can be found here.
2. **GitHub:** GitHub acts as a Code Repository, and we can interact with it through Git. I used it for storing my code base. Its official website is this. It also provides facilities such as WebHooks, that allow us to trigger builds remotely as soon as any changes to code base are made.
3. **Jenkins:** An excellent tool for managing the CI/CD pipeline. It is a free and open-source and more information about it can be found here. I used it for creating the CI/CD pipeline for the project.
4. **Maven:** A build automation tool for applications created using the Java programming language. More information about it can be found here. I used Maven to build code automatically for the project.
    - I also used **JUnit** with Maven to test the code. Its official website can be found here.
5. **Docker:** Used for building containers, Docker is used to ensure that applications and services work as intended when deployed on the actual machine. Its official website can be found here. I used Docker for creating images, and making containers which run the calculator.
6. **Ansible:** It is used for automating a lot of tasks that are repetitive in nature, and tend to be tedious when done manually. More information about it can be found here. I used Ansible for downloading Docker images and then running them.
7. **Ngrok:** It is used for exposing local IP address of a system to the Internet, so that applications running on say localhost, can be accessed

via Internet also. More information about it can be found here. I used it for exposing Jenkins, so that my Webhooks can reach to it via GitHub, whenever any new changes are pushed to the repository.

8. **Apache Log4j:** It is a popular application that enables logging facility into programs. More information about it can be found here. I used it for enabling logging in my programs.
9. **ELK Stack:** It is used for monitoring the applications and services that are running. I used it for monitoring my program's health by giving it logs generated from Apache Log4j while using my program. More information about it can be found here.

Links

My GitHub repository can be accessed using the below link:

SPE Mini Project

This repo contains code for the SPE Mini Project!

My DockerHub repository can be accessed using the below link:

Basic Repository

Repository containing images with the Calculator program

Steps

Installing Git

Git comes pre-installed with most of the Linux distributions. It can be installed on Ubuntu using the below commands:

```
$ sudo apt update
$ sudo apt install -y git-all
```

The Git version can be checked using:

```
$ git version
```

One can now configure their username and e-mail that will be used while pushing their commits to GitHub using:

```
$ git config --global user.name "<username>"
$ git config --global user.email "<email>"
```

Now, we can initialize Git in any directory by navigating to the directory and using:

```
$ git init
```

After updating code for our project, we can add it to the staging area, using the command:

```
$ git add .
```

We can then, commit our changes in the staging area by using the command:

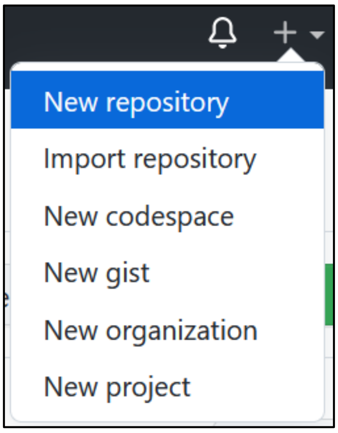
```
$ git commit -m “<Commit Message>”
```

After committing, we can push changes to our GitHub repository, which is explained in the next section.

Creating Account on GitHub

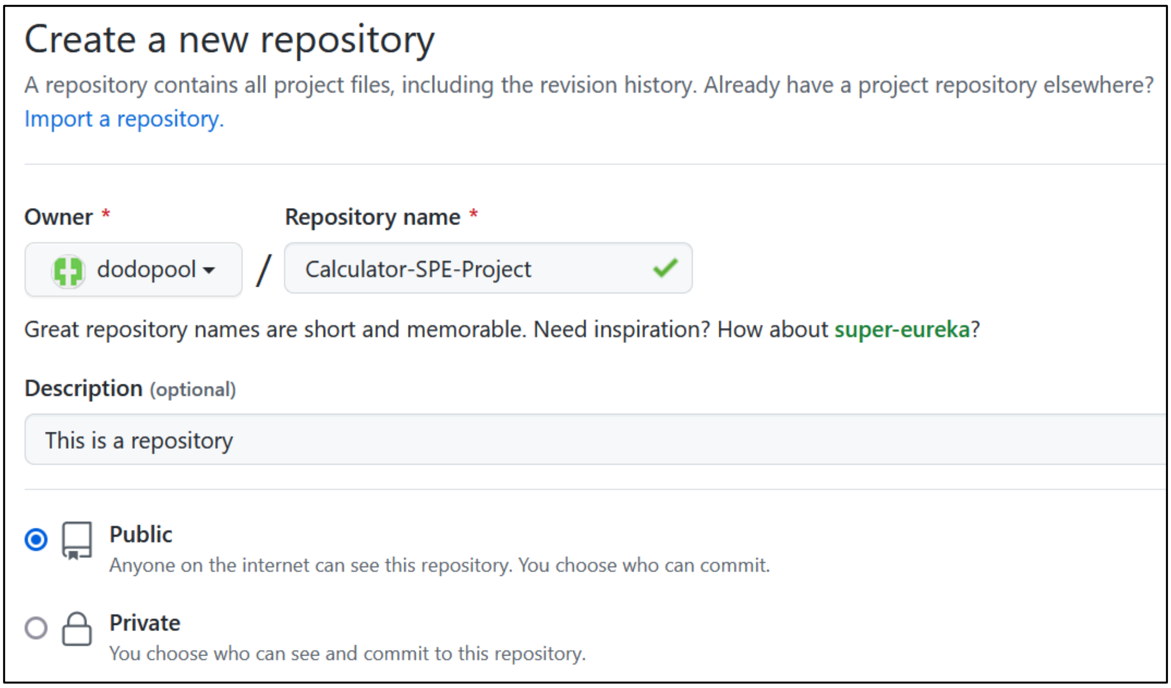
We will use GitHub for keeping our code base. In order setup GitHub repository where we would be storing our code, we firstly create an account at GitHub. Next, we create a repository in GitHub by following the below steps:

1. Click on the + button on the top right hand of the screen, and click on new repository:



[Step 1] Creating a New repository in GitHub

2. We will be taken to a page where we need to enter repository details. Once done, we click on the Create Repository button to create the repository:



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * dodopool / Repository name * Calculator-SPE-Project ✓

Great repository names are short and memorable. Need inspiration? How about super-eureka?

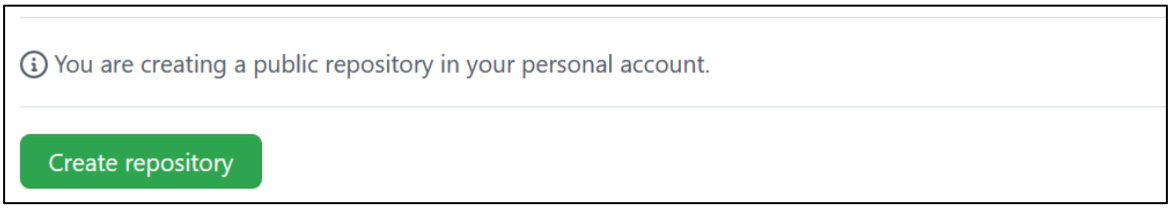
Description (optional)

This is a repository

● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

[Step 2] Enter Repository Details



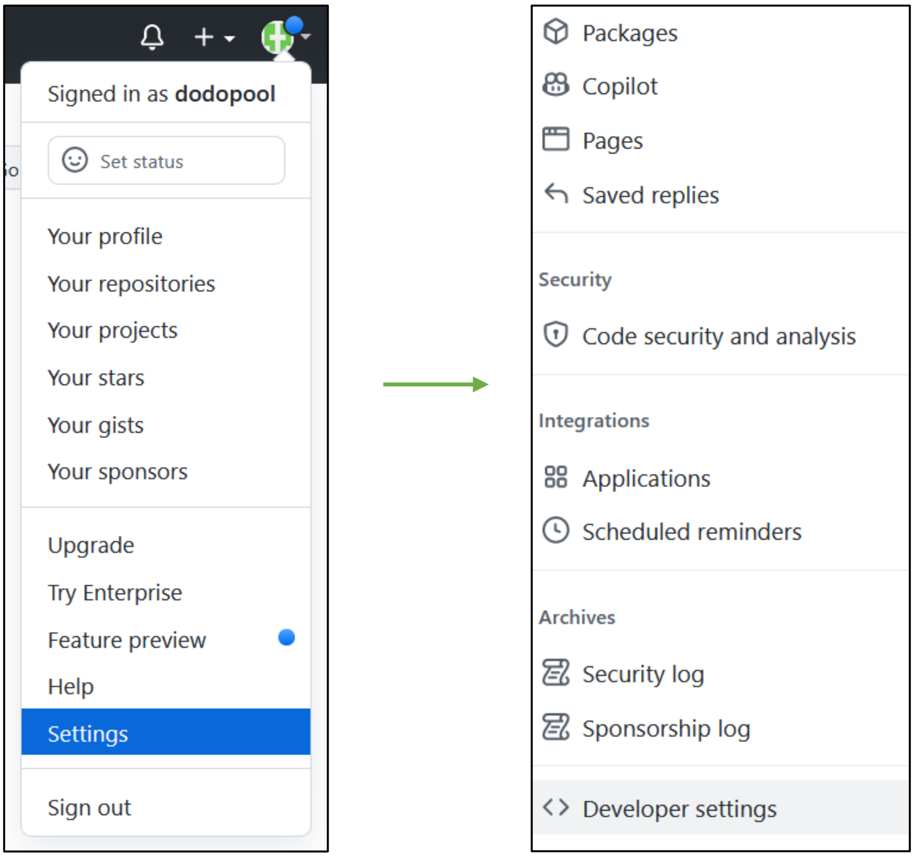
ⓘ You are creating a public repository in your personal account.

[Step 2] Clicking on Create repository creates the repository for us

3. Our repository now gets created on GitHub. We can push our changes in our local system to the GitHub repository using the commands mentioned on the page, namely:

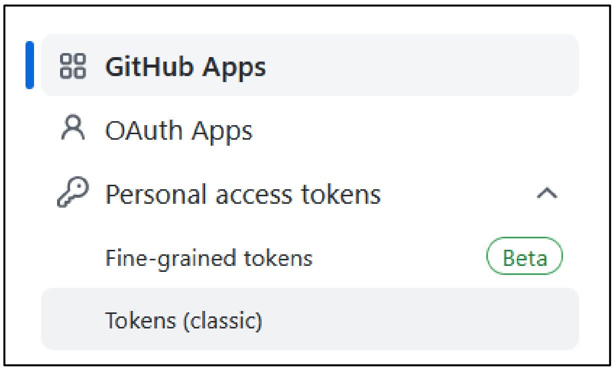
```
$ git remote add origin https://github.com/dodopool/Calculator-SPE-Project.git
$ git branch -M main
$ git push -u origin main
```

4. While pushing changes to GitHub, we will be asked to provide our GitHub username and password. The username would be our GitHub username; however, the password will be a secret authentication token, that can be generated by:
    a. Going to our Profile settings, and scrolling all the way down until we find Developer Settings on the left:



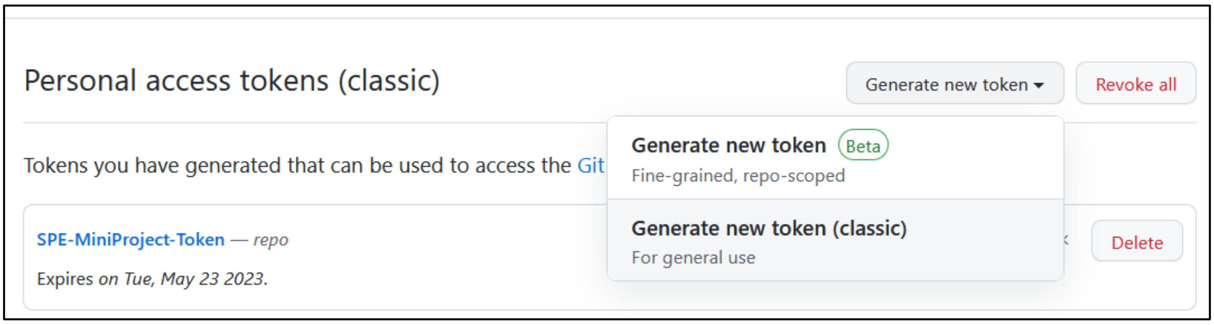
[Step 4.a] Opening the Developer settings in GitHub

    b. In the new page that opens us, we choose Tokens (classic), as follows:



[Step 4.b] Choosing the Tokens (classic) for our authentication

c. On the right-hand side, we click on Generate New Token, and then click on Generate New Token (classic):



Personal access tokens (classic)

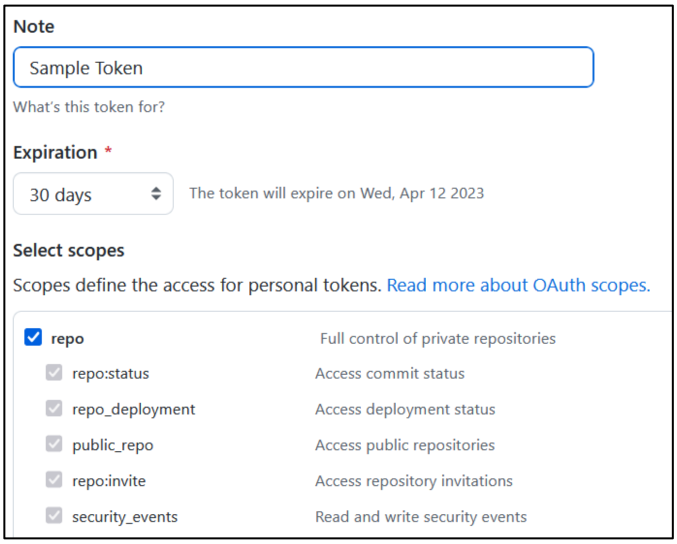
Tokens you have generated that can be used to access the Git

SPE-MiniProject-Token — *repo*

Expires *on Tue, May 23 2023.*

[Step 4.c] Generating a new token (classic) on GitHub

d. We will now be asked to authenticate ourselves on GitHub. After that, we will be taken to a screen where we can enter our token name, choose an expiration date. We also choose repo as the scope of our token:



Note

Sample Token

What's this token for?

Expiration *

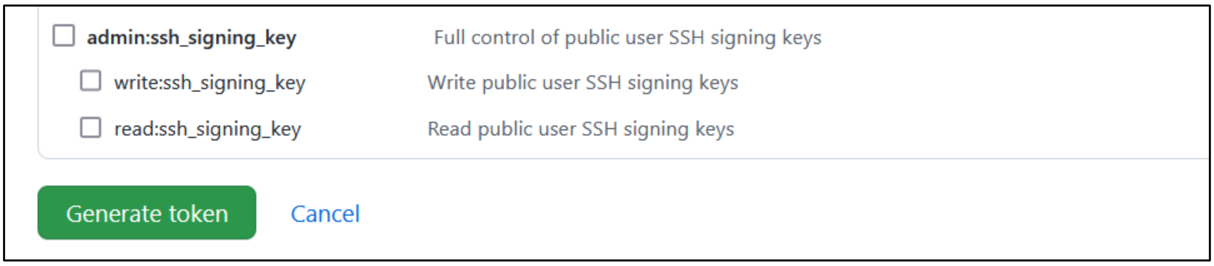
30 days ⬍ The token will expire on Wed, Apr 12 2023

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ repo	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☑ security_events	Read and write security events

[Step 4.d] Entering Token Details



□ **admin:ssh_signing_key** Full control of public user SSH signing keys

□ write:ssh_signing_key Write public user SSH signing keys

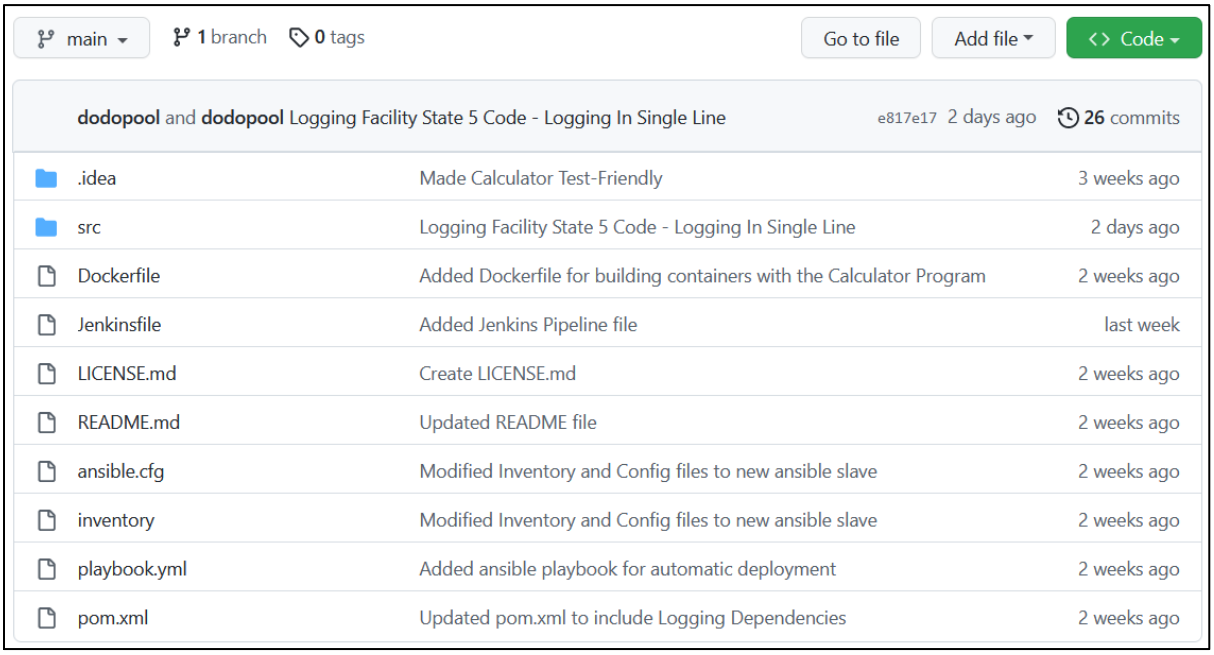
□ read:ssh_signing_key Read public user SSH signing keys

[Step 4.d] We click on Generate token to create a new token

e. Now, we will be shown the token. We need to keep it safely somewhere, as it won't be shown to us again. When Git asks us for password while pushing our repository, we can supply this token for authentication.

That's it! We are now ready to start working on our project and pushing changes to GitHub.

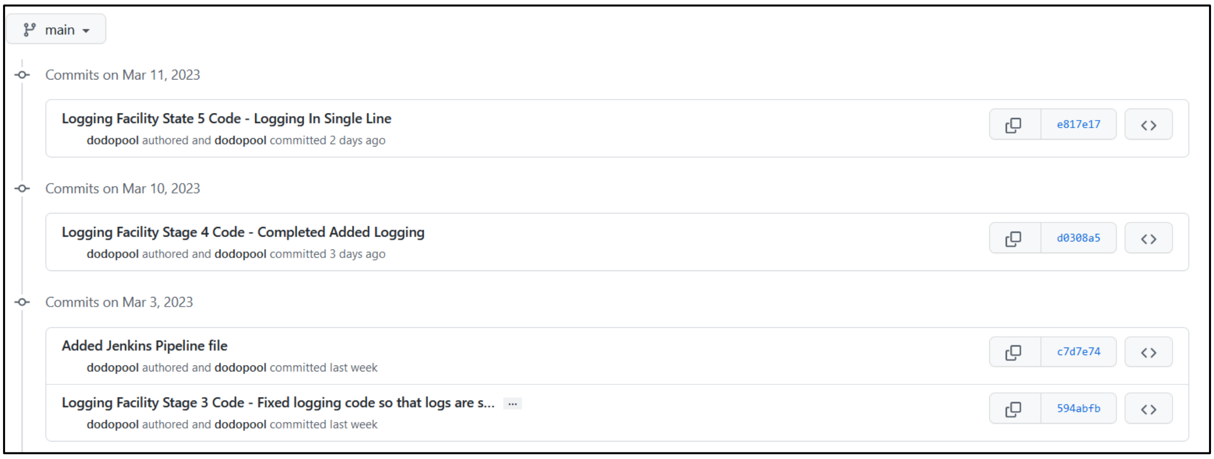
My GitHub repository looks as follows:



dodopool and **dodopool** Logging Facility State 5 Code - Logging In Single Line e817e17 2 days ago **26** commits

.idea	Made Calculator Test-Friendly	3 weeks ago
src	Logging Facility State 5 Code - Logging In Single Line	2 days ago
Dockerfile	Added Dockerfile for building containers with the Calculator Program	2 weeks ago
Jenkinsfile	Added Jenkins Pipeline file	last week
LICENSE.md	Create LICENSE.md	2 weeks ago
README.md	Updated README file	2 weeks ago
ansible.cfg	Modified Inventory and Config files to new ansible slave	2 weeks ago
inventory	Modified Inventory and Config files to new ansible slave	2 weeks ago
playbook.yml	Added ansible playbook for automatic deployment	2 weeks ago
pom.xml	Updated pom.xml to include Logging Dependencies	2 weeks ago

My GitHub repository



Commits on Mar 11, 2023

Logging Facility State 5 Code - Logging In Single Line
dodopool authored and dodopool committed 2 days ago
e817e17

Commits on Mar 10, 2023

Logging Facility Stage 4 Code - Completed Added Logging
dodopool authored and dodopool committed 3 days ago
d0308a5

Commits on Mar 3, 2023

Added Jenkins Pipeline file
dodopool authored and dodopool committed last week
c7d7e74

Logging Facility Stage 3 Code - Fixed logging code so that logs are s... …
dodopool authored and dodopool committed last week
594abfb

Some commits I have made in my repository

Setting up Jenkins

The next tool we are going to use Jenkins, which is used to build a CI/CD pipeline for this project.

We first install Java using the below commands:

```
$ sudo apt-get update
$ sudo apt install -y openjdk-11-jdk
```

Next, we install Jenkins using the below sequence of commands:

```
$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |
sudo apt key add -
$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt install ca-certificates
$ sudo apt-get update
$ sudo apt-get install jenkins
```

Now, we can go to our localhost (port 8080), where Jenkins guides us through the installation process. Initially, it asks us for a password, which can be obtained through the command:

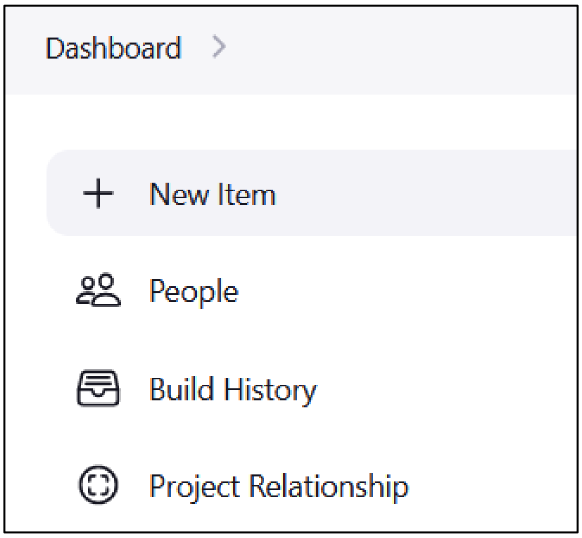
```
$ sudo cat /var/lib/Jenkins/secrets/initialAdminPassword
```

And we are now ready to set up our CI/CD pipeline, which is explained in the next section.

Setting up CI/CD Pipeline using Jenkins

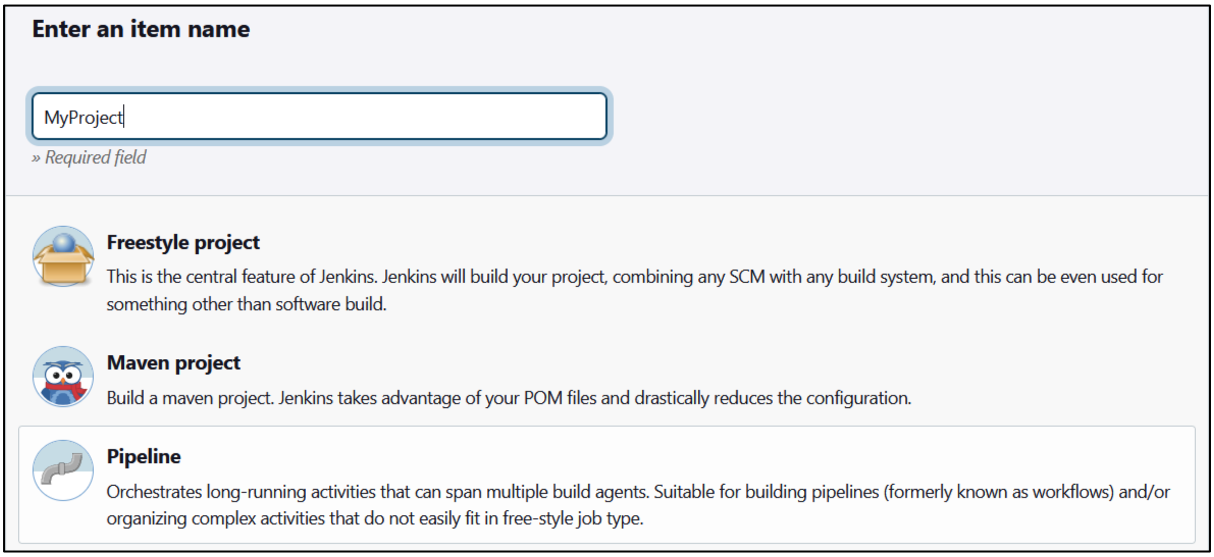
We will create a Pipeline project in Jenkins. The steps to be followed are given below:

1. On the left-hand side of the Jenkins dashboard, we click on New Item to create a new project:



[Step 1] Click on New Item to create a new Project

2. In the screen that follows, we enter out Project name, and choose the project type as pipeline. Finally, we click on OK button to create the pipeline project:



Enter an item name

MyProject

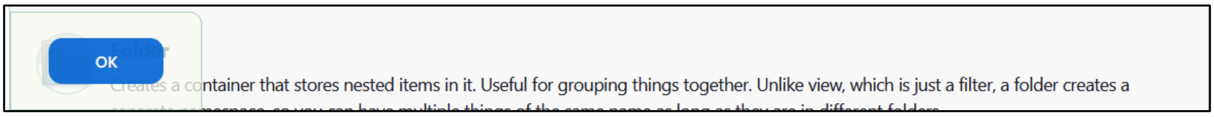
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

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.

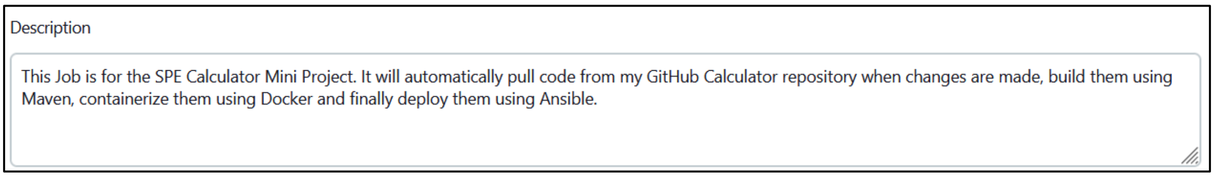
[Step 2] Enter Project Name and choose project type as Pipeline



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

[Step 2] Click on the OK button to create the project

3. We will now be presented with a screen that allows us to configure the pipeline project we created. We now configure this project as follows:
    a. Enter project description:

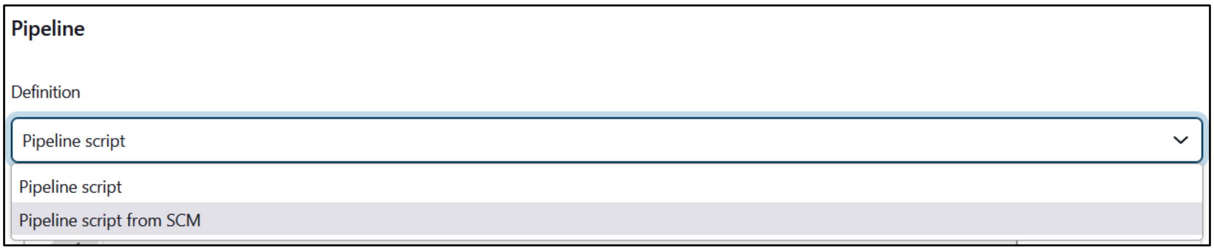


Description

This Job is for the SPE Calculator Mini Project. It will automatically pull code from my GitHub Calculator repository when changes are made, build them using Maven, containerize them using Docker and finally deploy them using Ansible.

[Step 3.a] Enter the Project Description

    b. Now, we need to ensure that the pipeline script will be loaded from our GitHub repository. In order to do so, we scroll all the way down to the Pipeline section, where we choose the Definition option as "Pipeline script from SCM":



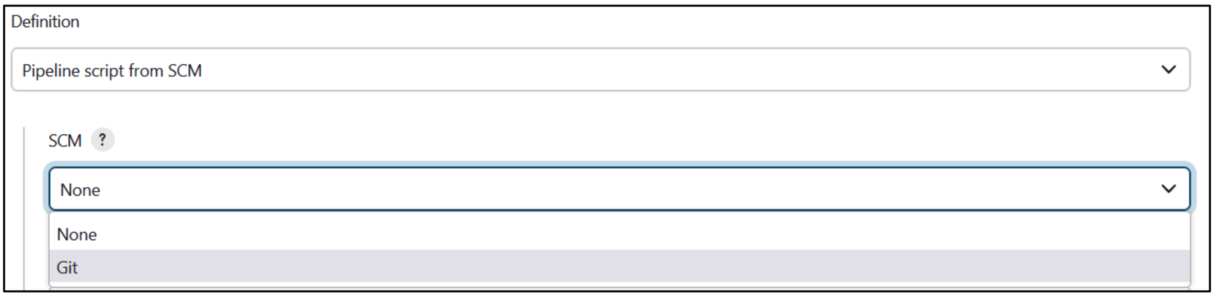
Pipeline

Definition

Pipeline script

[Step 3.b] Choosing the correct Pipeline Definition

    c. We will now be asked to specify our GitHub repository from where we can load the Pipeline script. In the SCM option, we choose the option Git:



Definition

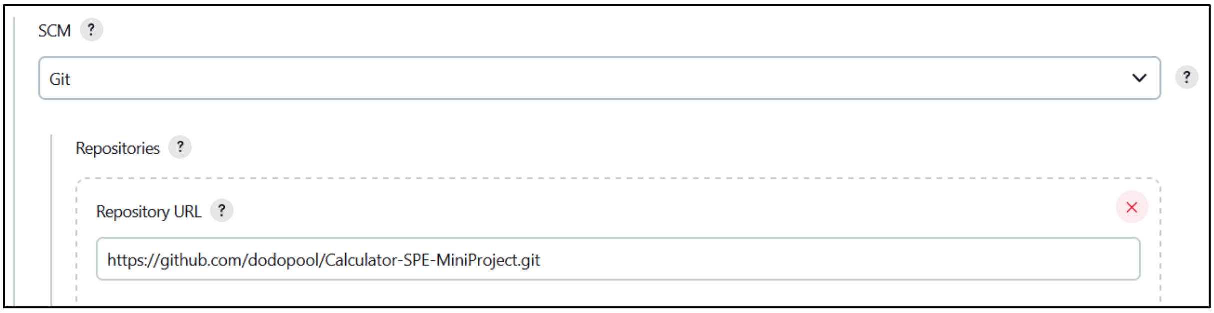
Pipeline script from SCM

SCM ?

None

[Step 3.c] Choose SCM option as Git

d. We now enter the URL of our GitHub repository that will contain the pipeline script (in a file called Jenkinsfile), as follows:



SCM ?

Git

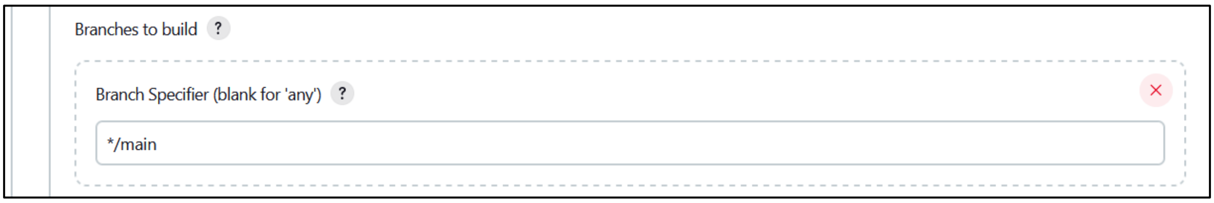
Repositories ?

Repository URL ?

https://github.com/dodopool/Calculator-SPE-MiniProject.git

[Step 3.d] Specifying the GitHub repository

e. We scroll down a bit, and also change the “Branches to build” as main, which will ensure that we build the main branch whenever any changes are made to the repository:



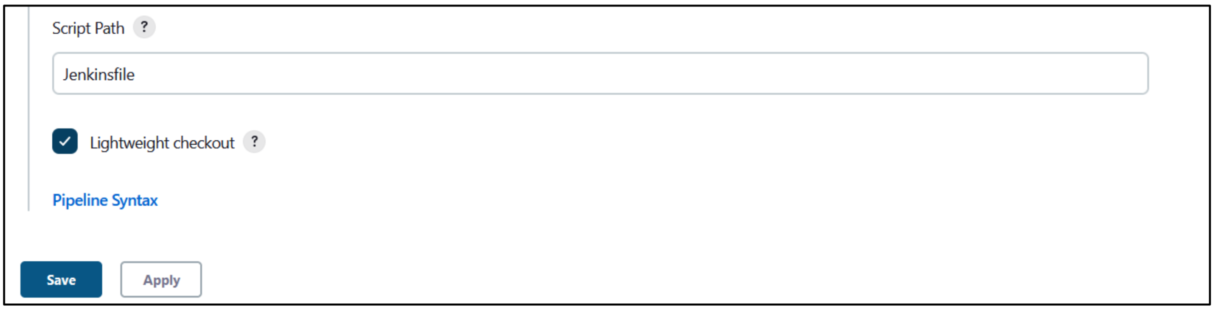
Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

[Step 3.e] Change the “Branch to build” to main

f. We finally save our project by clicking on the Apply and then on the Save button to save the project:



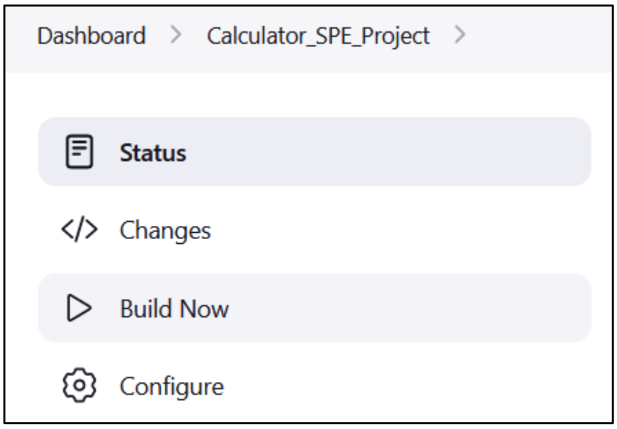
Script Path ?

Jenkinsfile

☑ Lightweight checkout ?

[Step 3.f] Click on Apply and then Save to save the project

4. We can manually build our project by clicking on our Build Now button:



[Step 4] Project can be built manually by clicking on Build Now button

There are a few things to note at this point:

1. We can avoid building our project manually by using Ngrok, so that our project gets built automatically whenever we make changes to our GitHub repository. The settings needed to enable this will be described in the Ngrok section.
2. We will also need to add credentials to access DockerHub, which will be described in the Docker section.
3. Also, the pipeline script will be explained in details in the upcoming sections.

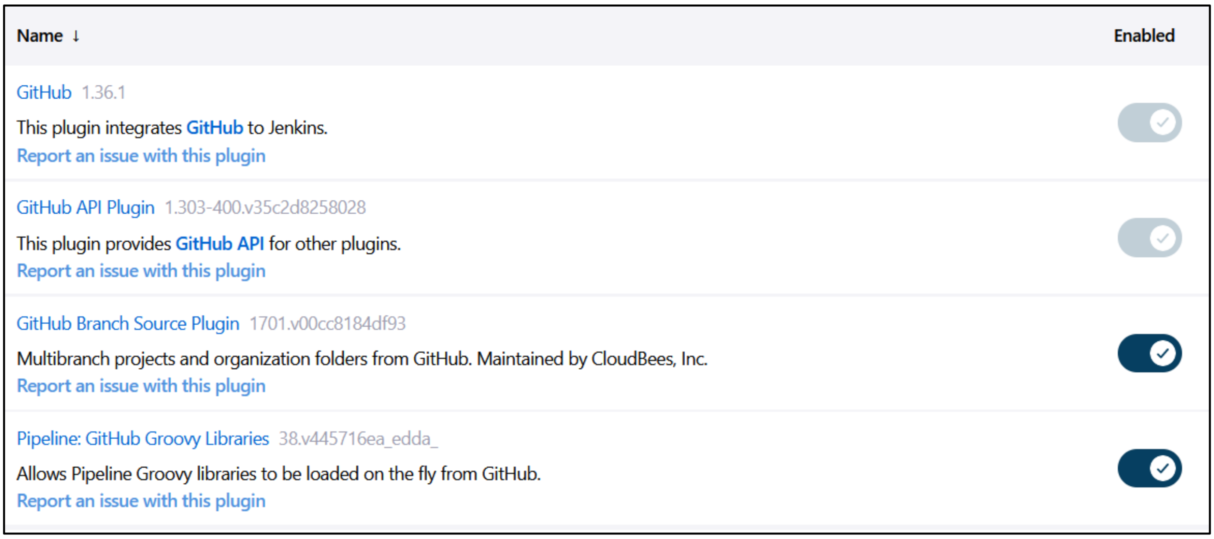
Pipeline Script for Pulling Repository

The first stage in our pipeline is to pull the GitHub repository. The script used for doing so is:

```
stage(‘Pull GitHub Repository’) {
        steps {
                // Get code from GitHub Repository
                git branch: ‘main’, url: ‘https://github.com/dodopool/Calculator-SPE-MiniProject.git'
        }
}
```

It is pretty easy to understand the above script. It asks Jenkins to pull the main branch of the repository mentioned in the URL.

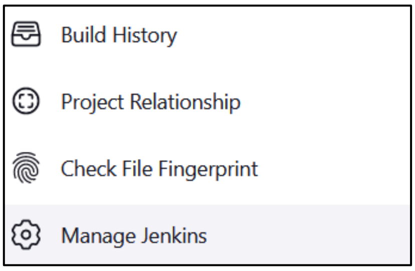
Note that we need to install the following plugins in Jenkins to make the above script work:

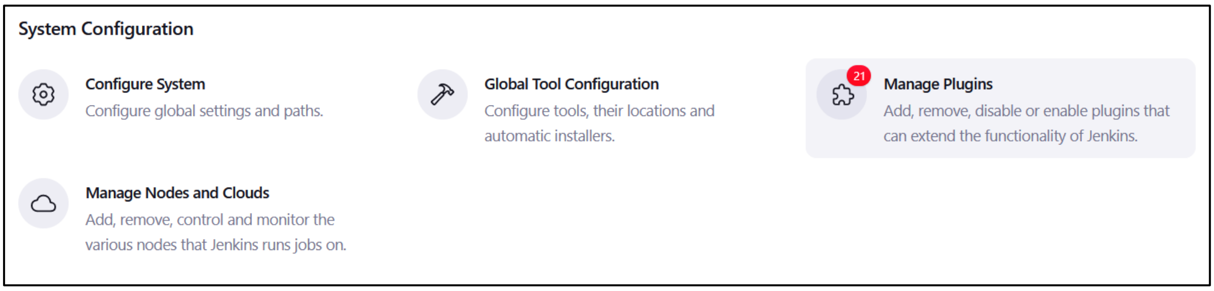


Name ↓	Enabled
GitHub 1.36.1 This plugin integrates **GitHub** to Jenkins. Report an issue with this plugin	✓
GitHub API Plugin 1.303-400.v35c2d8258028 This plugin provides **GitHub API** for other plugins. Report an issue with this plugin	✓
GitHub Branch Source Plugin 1701.v00cc8184df93 Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc. Report an issue with this plugin	✓
Pipeline: GitHub Groovy Libraries 38.v445716ea_edda_ Allows Pipeline Groovy libraries to be loaded on the fly from GitHub. Report an issue with this plugin	✓

Plugins required for the GitHub Pipeline script

Note that we can install plugins by clicking Manage Jenkins (on the left of Dashboard), then scrolling down and then clicking (under System Configuration) Manage Plugins, thereby allowing us to manage plugins used in Jenkins:





System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

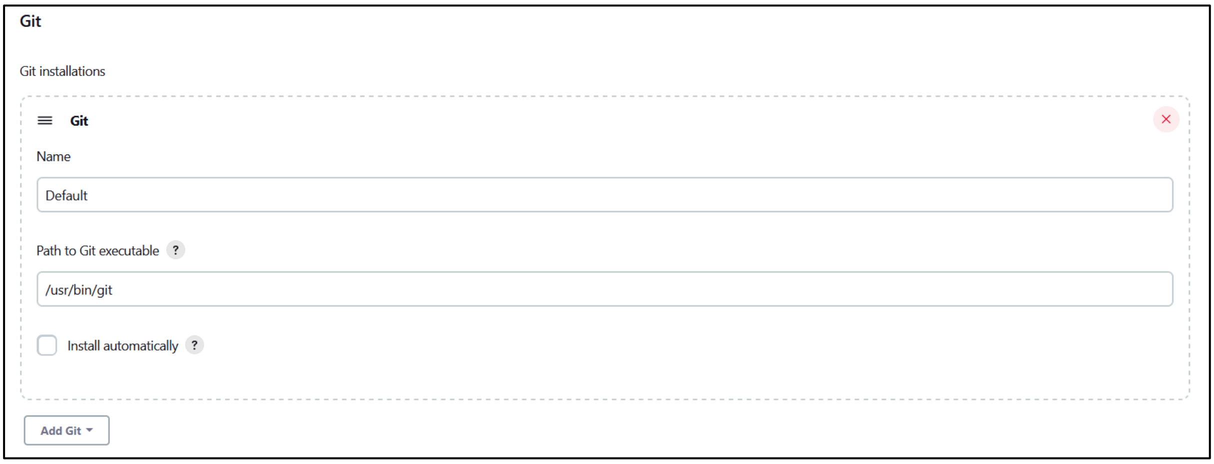
Manage Plugins 21
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Images showing how to Manage and Install plugins in Jenkins

We also need to add the PATH of Git to Jenkins. In order to do so, we follow the below steps:

1. Go to Manage Jenkins → Global Tool Configuration
2. We scroll down to the Git section, where we specify the Path to Git:



Git

Git installations

Git

Name

Default

Path to Git executable ?

/usr/bin/git

□ Install automatically ?

[Step 2] Specifying Git Path on Jenkins

3. That's it, we have successfully specified path to Git.

Setting up Maven

We firstly install the IntelliJ IDEA Community version using the below command:

```
$ sudo snap install intellij-idea-community --classic
```

We can start IntelliJ IDEA using Desktop or the command line:

```
$ intellij-idea-community
```

Now, we install Maven using:

```
$ sudo apt update
$ sudo apt install maven
```

In order to create a Maven project, one can follow the steps mentioned in this GeeksForGeeks article.

Maven Project Structure for Calculator Program

My code is developed in Java 11. I used Log4j for enabling logging and JUnit for doing automated testing. The tree view of my project is shown below:

```
Calculator-SPE-MiniProject/
├── ansible.cfg
├── Dockerfile
├── inventory
├── Jenkinsfile
├── LICENSE.md
├── playbook.yml
├── pom.xml
├── README.md
└── src
    ├── main
    │   ├── java
    │   │   └── org
    │   │       └── example
    │   │           ├── Calculator.java
    │   │           └── Main.java
    │   └── resources
    │       └── log4j2.xml
    └── test
        └── java
            └── CalculatorTest.java
```

The project files containing code are:

1. **Calculator.java:** This file implements a Calculator class that supports the following functionalities:
    a. Computing Square Root
    b. Computing Factorial
    c. Computing Natural Logarithm
    d. Computing Power
2. **Main.java:** This file provides a menu-driven interface through which the user can interact with the program.
3. **CalculatorTest.java:** This file contains test cases for each of the functions that are implemented in the Calculator class.

In order to enable logging and testing, we need to add the relevant dependencies in the pom.xml file. Because this file is too big to be included in the report, I am not putting its screenshot here. However, the file can be viewed on my GitHub repository here.

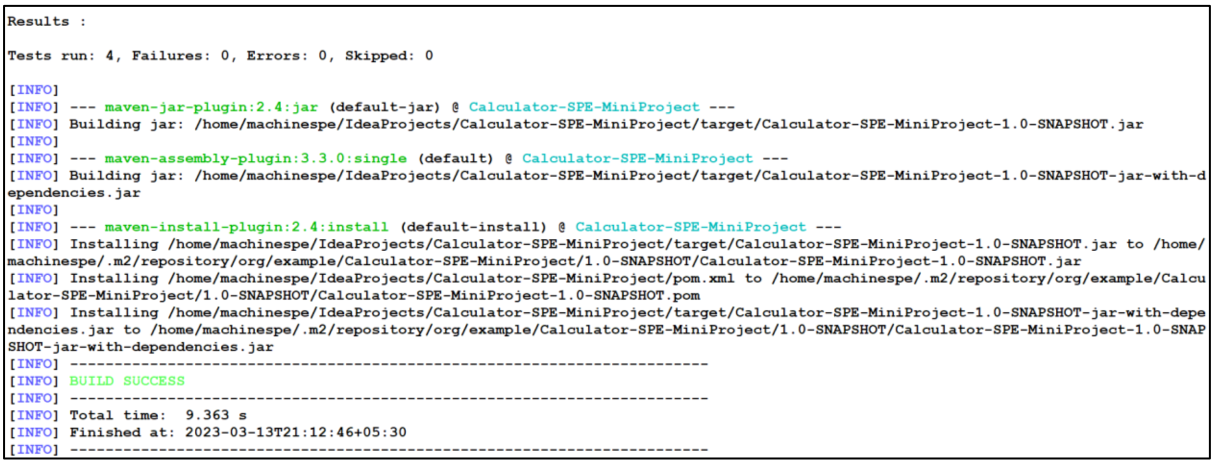
Pipeline Script for Building Projects using Maven

After pulling of the GitHub repository, the second stage of my pipeline script is to build the code using Maven. The relevant pipeline script for this part is:

```
stage('Build Pulled Code using Maven') {
        steps {
                sh 'mvn clean install'
        }
}
```

The above script simply executes the mvn clean install shell command to build the code that was pulled from the GitHub repository. It will also perform testing automatically using the test cases provided in the repository.

An example run of mvn clean install is shown in the below screenshot:



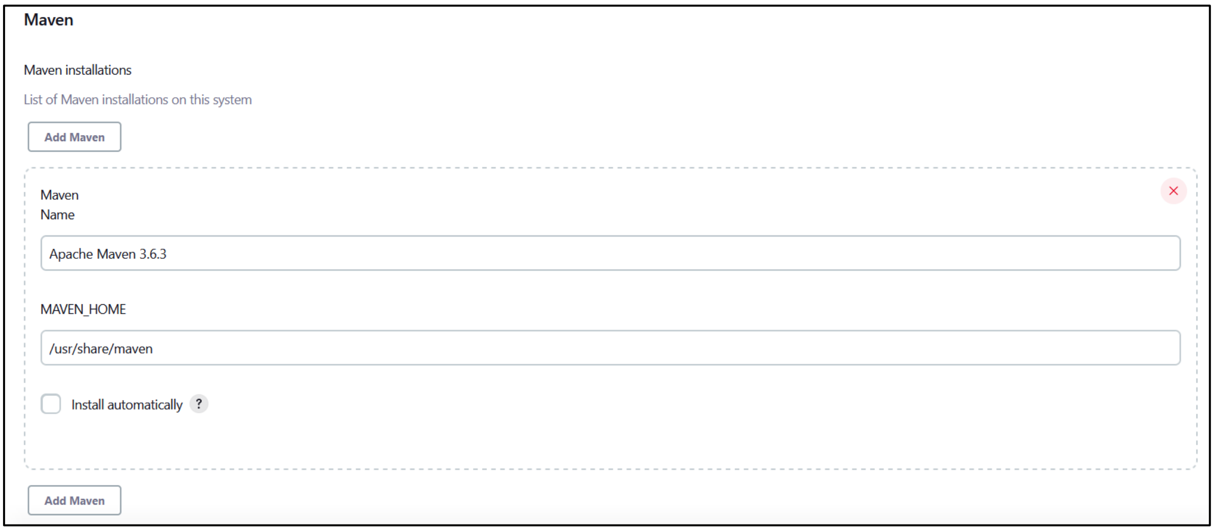
```
Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ Calculator-SPE-MiniProject ---
[INFO] Building jar: /home/machinespe/IdeaProjects/Calculator-SPE-MiniProject/target/Calculator-SPE-MiniProject-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-assembly-plugin:3.3.0:single (default) @ Calculator-SPE-MiniProject ---
[INFO] Building jar: /home/machinespe/IdeaProjects/Calculator-SPE-MiniProject/target/Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-d
ependencies.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ Calculator-SPE-MiniProject ---
[INFO] Installing /home/machinespe/IdeaProjects/Calculator-SPE-MiniProject/target/Calculator-SPE-MiniProject-1.0-SNAPSHOT.jar to /home/
machinespe/.m2/repository/org/example/Calculator-SPE-MiniProject/1.0-SNAPSHOT/Calculator-SPE-MiniProject-1.0-SNAPSHOT.jar
[INFO] Installing /home/machinespe/IdeaProjects/Calculator-SPE-MiniProject/pom.xml to /home/machinespe/.m2/repository/org/example/Calcu
lator-SPE-MiniProject/1.0-SNAPSHOT/Calculator-SPE-MiniProject-1.0-SNAPSHOT.pom
[INFO] Installing /home/machinespe/IdeaProjects/Calculator-SPE-MiniProject/target/Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-depe
ndencies.jar to /home/machinespe/.m2/repository/org/example/Calculator-SPE-MiniProject/1.0-SNAPSHOT/Calculator-SPE-MiniProject-1.0-SNAP
SHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.363 s
[INFO] Finished at: 2023-03-13T21:12:46+05:30
[INFO] ------------------------------------------------------------------------
```

Results of running mvn clean install

Our built program can be found in the target directory, where we can run it using:

```
$ java -jar <Jar File Created With Dependencies>
```

Note that we need to specify the PATH to Maven in Jenkins, this can be done in a way similar to that for Git. The relevant screenshots showing adding Maven's PATH is shown below:



Maven

Maven installations

List of Maven installations on this system

Maven Name: Apache Maven 3.6.3

MAVEN_HOME: /usr/share/maven

□ Install automatically

Specifying Maven’s PATH in Jenkins

Installing Docker

Docker is an operating system virtualization platform that allows applications to be delivered in containers. As a result, rather than just supplying the software, the full environment is provided as a Docker image, including all the software dependencies.

Since we now have a code base, we need to put it inside a container so that it can be deployed easily. In order to do so, we install docker using:

```
$ sudo apt update
$ sudo apt-get install docker.io
```

We can also check the status of docker service using:

```
$ service docker status
```

Dockerfile for Creating Images

Now that we have docker installed, let us create a Dockerfile that will help us in creating images containing the Calculator program, which can be pushed to Docker Hub (see the next section).

The contents of the Dockerfile are shown below:

```
# This Dockerfile uses the image @ https://hub.docker.com/_/openjdk

FROM openjdk:11
COPY ./target/Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java", "-jar", "Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

The above Dockerfile:

1. **COPY:** Copies the built Calculator JAR file to the home directory of the container.
2. **WORKDIR:** Changes the working directory of the container to its home directory.
3. **CMD:** Executes the command required for running the Calculator Program.

Note that we use the OpenJDK-11 image to create our docker image containing the calculator program.

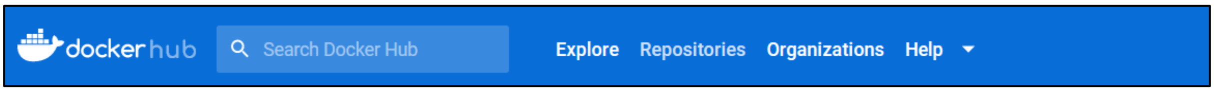
Now, we can use this Dockerfile to package our Calculator program.

Creating Docker Hub Account and Creating Repository

Now that we have docker installed on our system and have a Dockerfile ready, we need to create a Docker Hub account so that we can store our created images there. In order to do so, we head over to Docker Hub and create an account there.

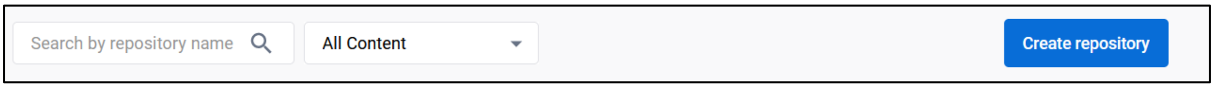
Once we have created an account on Docker Hub, we need to create a repository that will hold our images. In order to do so, we:

1. Click on the Repository link in the top of the Docker Hub webpage:



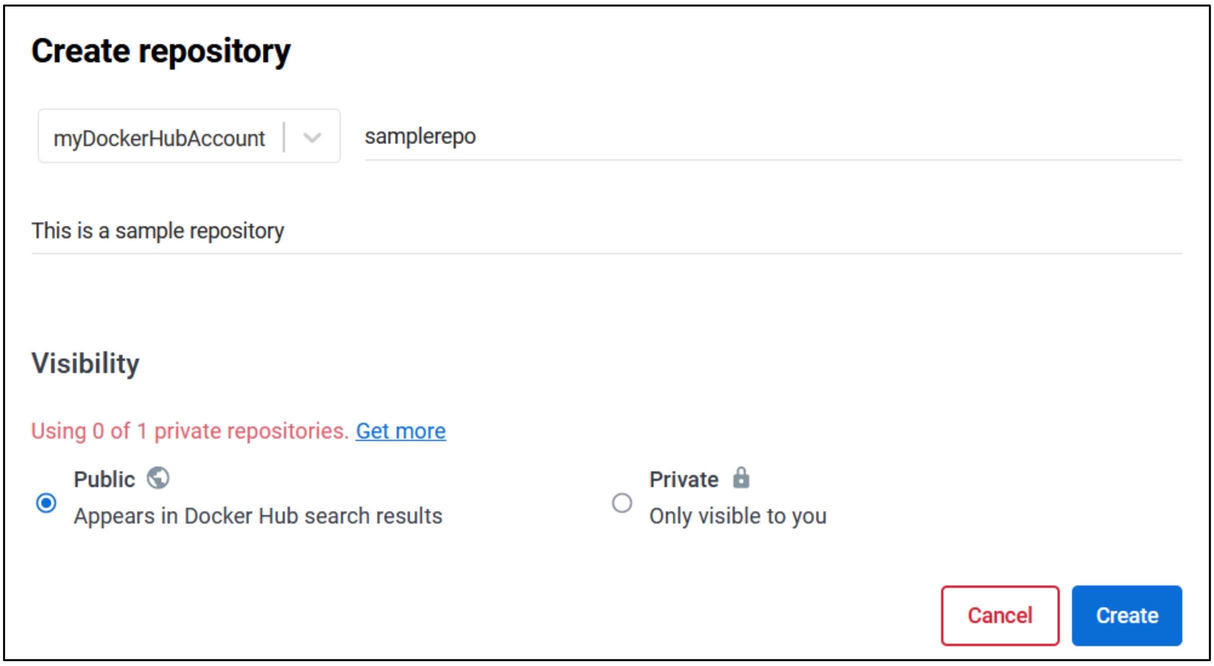
[Step 1] Click on the Repositories link

2. In the new webpage, on the right, we click on Create repository:



[Step 2] Click on Create repository

3. Now, enter a repository name, and click on Create button to create the repository:



Create repository

myDockerHubAccount / samplerepo

This is a sample repository

Visibility

Using 0 of 1 private repositories. Get more

● Public
Appears in Docker Hub search results

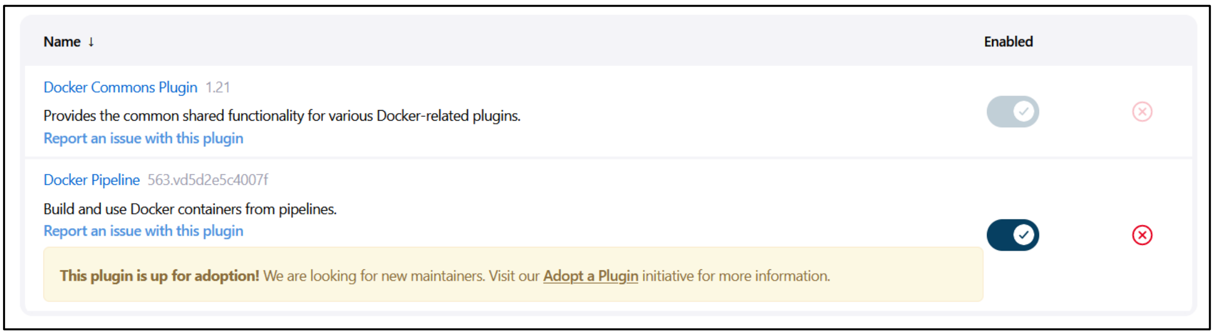
○ Private
Only visible to you

[Step 3] Entering Repository Name, and creating repository

4. That’s it, we have successfully created out Docker Hub repository!

Pipeline Script for Pushing Docker Images to Docker Hub

Firstly, let us install plugins required for this step. We go to Dashboard → Manage Jenkins → Manage Plugins, and install the following plugins:



Name ↓	Enabled	
Docker Commons Plugin 1.21 Provides the common shared functionality for various Docker-related plugins. Report an issue with this plugin	✓	⊗
Docker Pipeline 563.vd5d2e5c4007f Build and use Docker containers from pipelines. Report an issue with this plugin **This plugin is up for adoption!** We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.	✓	⊗

Plugins required for the Docker Pipeline Script

Now, before we create images and push them to Docker Hub, we need to declare a few environment variables in our pipeline script, as shown below:

```
environment {
        registry = 'rangoota/basicrepo'
        registryCredential = 'dockerhubconnect'
        dockerImage = ''
}
```

The above script:

1. Creates an environment variable ‘registry’ with value ‘rangoota/basicrepo’. This would be the name of our Docker Hub repository to which we would be pushing our images to.
2. Creates an environment variable ‘registryCredential’ with value ‘dockerhubconnect’. This value is the name of credential that we will use for authenticating ourselves to Docker Hub. The steps needed to setup this credential is explained near the end of this section.
3. Creates and environment variable ‘dockerImage’, which will hold our image name. Initially, it is empty.

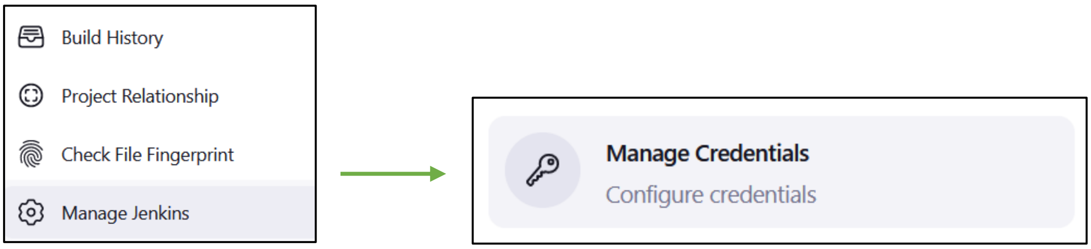
With environment variables set up, we can create the Docker image. Below is the pipeline script that I used for creating the Docker image from Dockerfile:

```
stage('Creating Image using Docker') {
        steps {
                script {
                   dockerImage = docker.build registry + ":latest"
                }
        }
}
```

The above script simply builds the Docker image, with the ‘latest’ tag on it.

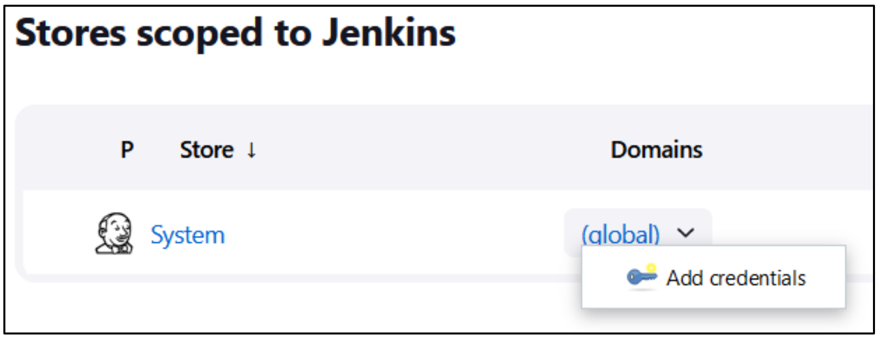
Now, we need to set up our Docker Hub credentials in Jenkins so that we can push images to our Docker Hub repository. In order to do so:

1. Go to Jenkins Dashboard → Manage Jenkins → Manage Credentials (under Security section)



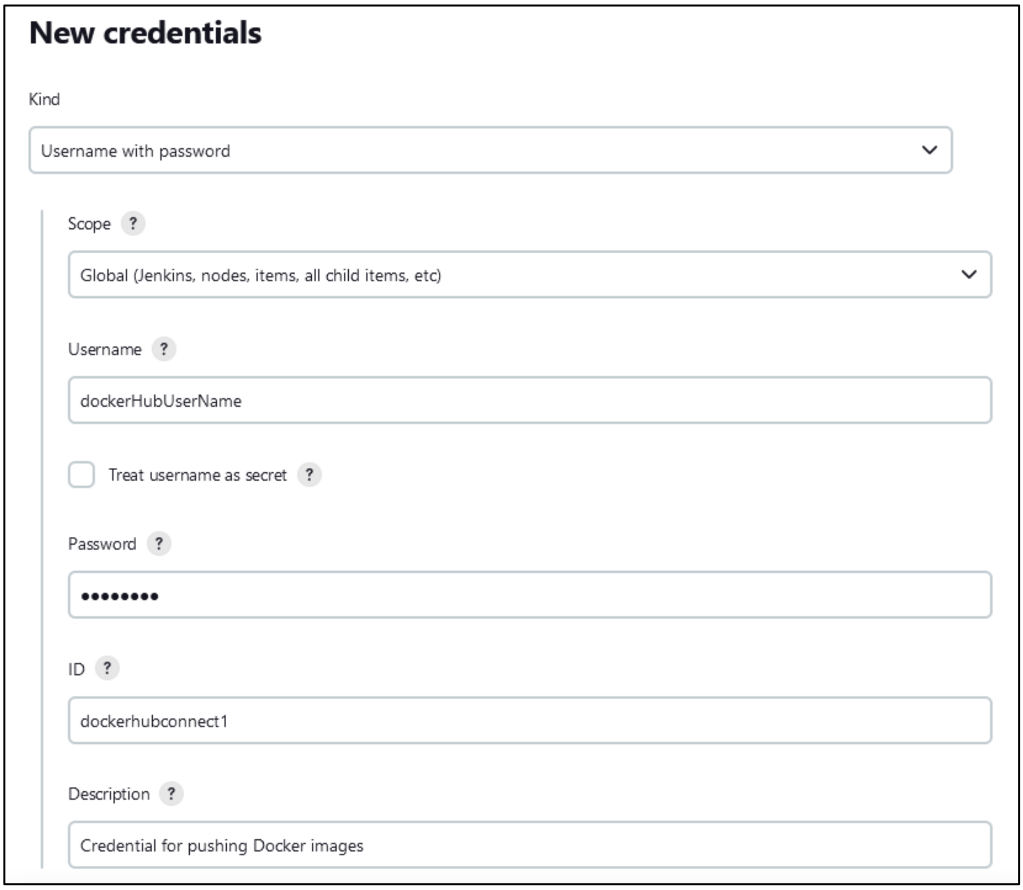
[Step 1] Going to Manage Credentials in Jenkins

2. Under the "Stores scoped to Jenkins", we click on (global), which presents us with a drop-down menu to add credentials:



[Step 2] Adding credentials to Jenkins

3. Now, in the screen that follow, we choose:
   - Kind: Username with password
   - Username: Your Docker Hub account username
   - Password: Your Docker Hub account password
   - ID: dockerhubconnect
   - Description: Credential for pushing Docker images



New credentials

Kind

Username with password

Scope ?

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

Username ?

dockerHubUserName

□ Treat username as secret ?

Password ?

••••••••

ID ?

dockerhubconnect1

Description ?

Credential for pushing Docker images

Adding Docker Hub credentials

4. Now, we click on the OK button to save our credentials.
5. That’s it! Our Docker Hub credentials are ready to be used by Jenkins.

With all these steps done, below is the pipeline script used for pushing Docker images to Docker Hub:

```
stage('Pushing the Image to Docker Repository') {
            steps {
                script {
                    docker.withRegistry('', registryCredential) {
                        dockerImage.push()
                    }
             }
     }
}
```

The above script simply pushed the Docker Image created in the previous step using the Docker Hub credentials we created earlier.

After pushing the image, we can delete the image on our machine to save disk space. The pipeline script for this is shown below:

```
stage('Remove Docker Image from Local System to free space') {
            steps {
                sh 'docker rmi $registry:latest'
            }
}
```

The above script simply removes the Docker image file we created on our machine.

Setting Up Ansible

Ansible is used for Infrastructure as Code (IaC) service provider. It is very useful for automatically creating environments. We can install Ansible using the following commands:

```
$ sudo apt update
$ sudo apt install ansible
```

Now, we need to install OpenSSH server on the client and the Ansible server, so that Ansible can communicate to other nodes using SSH. We can install OpenSSH-server using:

```
$ sudo apt update
$ sudo apt-get install openssh-server
```

Now, we need to ensure that ansible server can securely login to other client nodes. In order to do so, we will use PKI infrastructure, which works using Public Key and Private Key. We firstly generate a Public Key/Private Key pair in the machine running Ansible using:

```
$ ssh-keygen -t rsa
```

Now, we need to copy our server's Public Key to other nodes. We can do this by executing the following command in the server containing Ansible:

```
$ ssh-copy-id REMOTEUSER@<REMOTE_IP_ADDRESS>
```

While executing the above command, we will be asked the username and password of the remote system, which we will have to provide.

Now, we are ready to make Ansible execute commands on our client nodes.

Firstly, we need to create an inventory file, which tells Ansible which all client nodes can be used by it. My inventory file is as follows:

```
[SlaveNodes]
speminiprojectslave ansible_host=172.16.138.79
```

The above inventory file is explained below:

1. [SlaveNodes] tells Ansible that the client nodes in the following lines belong to the SlaveNodes group

2. The second line specifies the ansible host, which is 172.16.138.79.

We need to instruct ansible to use the above inventory file. This we can specify in a configuration file called ansible.cfg, whose contents are shown below:

```
[defaults]
inventory=./inventory
remote_user=speminiprojectslave
ask_pass=false

[privilege_escalation]
become=yes
become_user=root
become_method=sudo
```

The configuration file above mentions the inventory file and the name of the user through which Ansible will login to the client nodes.

The next few lines tells Ansible how it can escalate privileges in the client nodes. It does so my using the sudo command, which is assumed to be installed on the client nodes. Also, for the purposes of this project, I made sure that the client user on my machine (i.e. speminiprojectslave) can execute sudo command without password. We can do this by following the instructions on the 3rd answer at this link. The reason I did so is to ensure that I do not push password required my Ansible to GitHub repository (that contains all these files).

We also need to ensure that correct version of Python is installed in the client nodes, as Ansible communicates to client nodes using Python and SSH.

With all these things out of the way, we can finally use Ansible for deployment. We can specify the commands required to be executed by Ansible in files called playbooks.

The playbook file I used is as follows:

```
---
- name: Pull Docker Image and Deploy It
  hosts: SlaveNodes
  vars:
          ansible_python_interpreter: /usr/bin/python3
  become: yes
  become_method: sudo
  tasks:
          - name: Ensure that Docker service is enabled
            service:
                    name: docker
                    state: started

          - name: Pull the Docker Image
            docker_image:
                    name: rangoota/basicrepo:latest
                    source: pull

          - name: Run the Container
            shell: docker run -it -d rangoota/basicrepo:latest
```

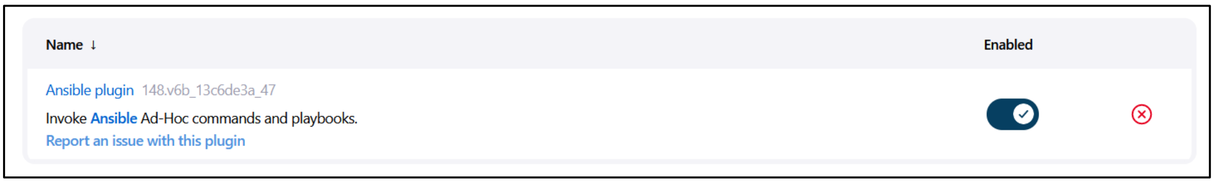
The above playbook file:

1. Instructs Ansible to become the root user
2. Starts the docker service in the first task
3. Pulls the Docker image from our Docker Hub repository.
4. Runs the container created from the pulled Docker image.

Now, we are ready to include Ansible in our Jenkins pipeline.

Pipeline Script for Ansible Deployment

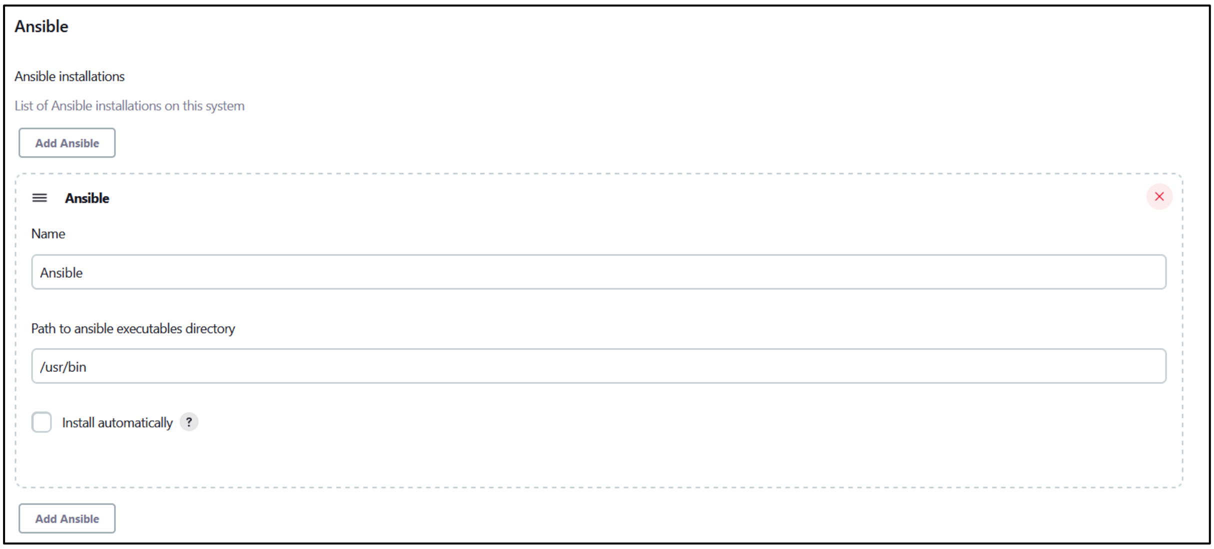
Before we include Ansible in our Jenkins pipeline, we need to install the following plugins for Ansible (by going to Dashboard → Manage Jenkins → Manage Plugins):



Name ↓	Enabled	
Ansible plugin 148.v6b_13c6de3a_47 Invoke Ansible Ad-Hoc commands and playbooks. Report an issue with this plugin	✓	⊗

Plugins required for Ansible Pipeline script

We also need to add the Ansible PATH in Jenkins by going to Dashboard → Manage Jenkins → Global Tool Configuration → Scroll Down to Ansible, and add its PATH:



Ansible

Ansible installations

List of Ansible installations on this system

Ansible

Name: Ansible

Path to ansible executables directory: /usr/bin

□ Install automatically

Adding Ansible PATH to Jenkins

With all these things done, below is the Jenkins Pipeline script we use for Ansible:

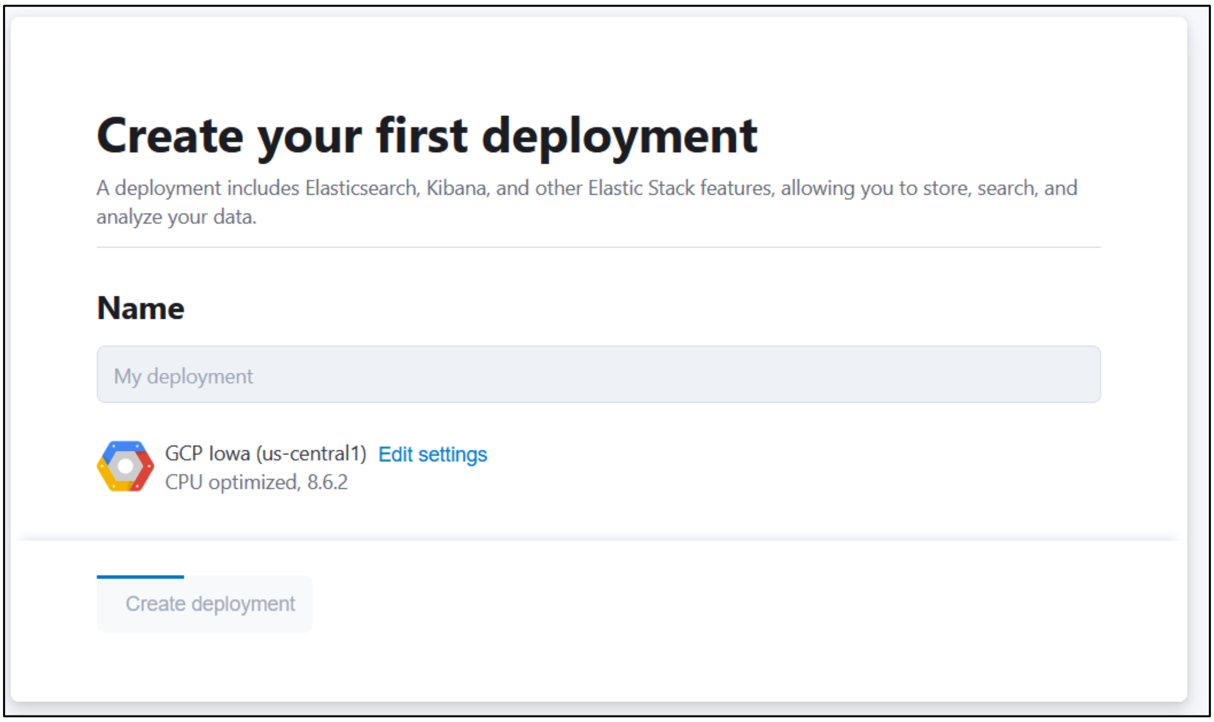
```
stage('Run ansible for deployment') {
            steps {
                ansiblePlaybook colorized: true, disableHostKeyChecking: true, installation: 'Ansible',
inventory: './inventory', playbook: 'playbook.yml'
            }
}
```

The above pipeline script simply executes the playbook file ‘playbook.yml’, while specifying the inventory to use as well. One might note that we did not use the Ansible Configuration file that we created earlier. This is because Ansible automatically checks for the presence of a configuration file in the local directory before moving on to defaults, therefore, it would automatically take into account the configuration file that we have used.

Monitoring using ELK Stack

Now that we’ve set up the whole pipeline, we need to monitor it for any errors. In order to do so, we will use ELK Stack. Firstly, we create an account here.

After we have logged in successfully, it will create a deployment for us. We can click on “Create Deployment” to begin the deployment creation process:



Create your first deployment

A deployment includes Elasticsearch, Kibana, and other Elastic Stack features, allowing you to store, search, and analyze your data.

Name

My deployment

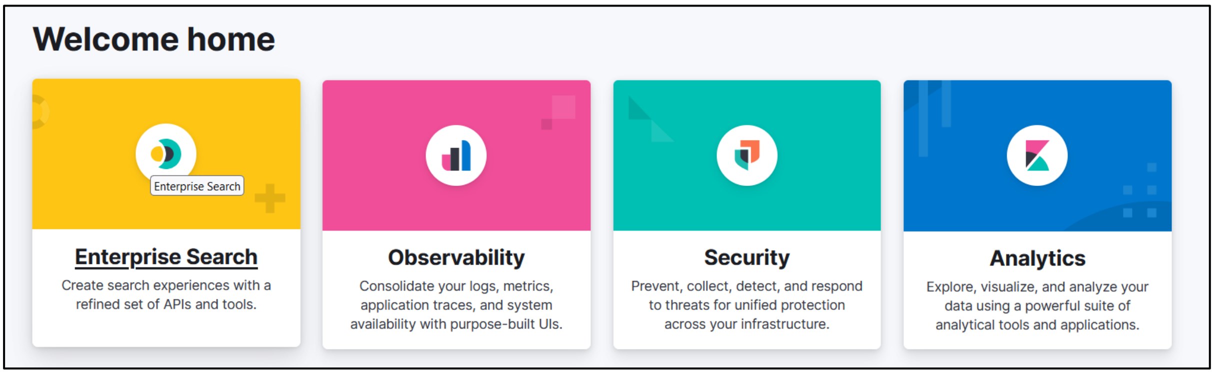
GCP Iowa (us-central1) Edit settings
CPU optimized, 8.6.2

Create deployment

Creating a Deployment in ELK Stack

After this, it will show us a username and password that can be used for authentication purposes. It is recommended to keep it safely stored somewhere.

We will now be presented with our Dashboard:



Welcome home

Enterprise Search
Create search experiences with a refined set of APIs and tools.

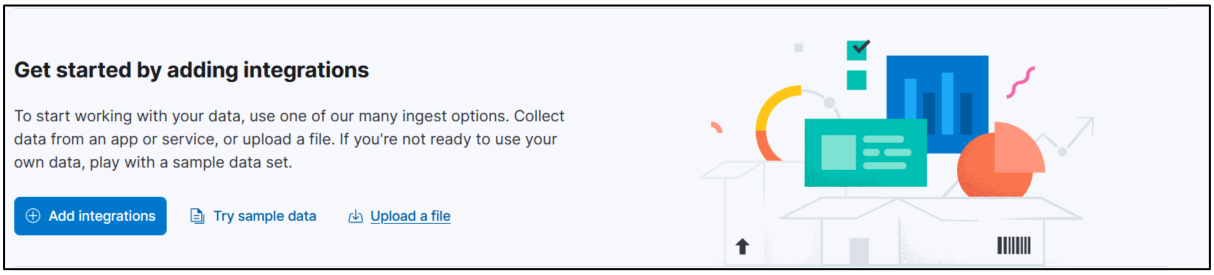
Observability
Consolidate your logs, metrics, application traces, and system availability with purpose-built UIs.

Security
Prevent, collect, detect, and respond to threats for unified protection across your infrastructure.

Analytics
Explore, visualize, and analyze your data using a powerful suite of analytical tools and applications.

Dashboard of ELK Stack

We scroll down a bit in the Dashboard and below the section “Get started by adding integrations”, we click on “Upload a file”:

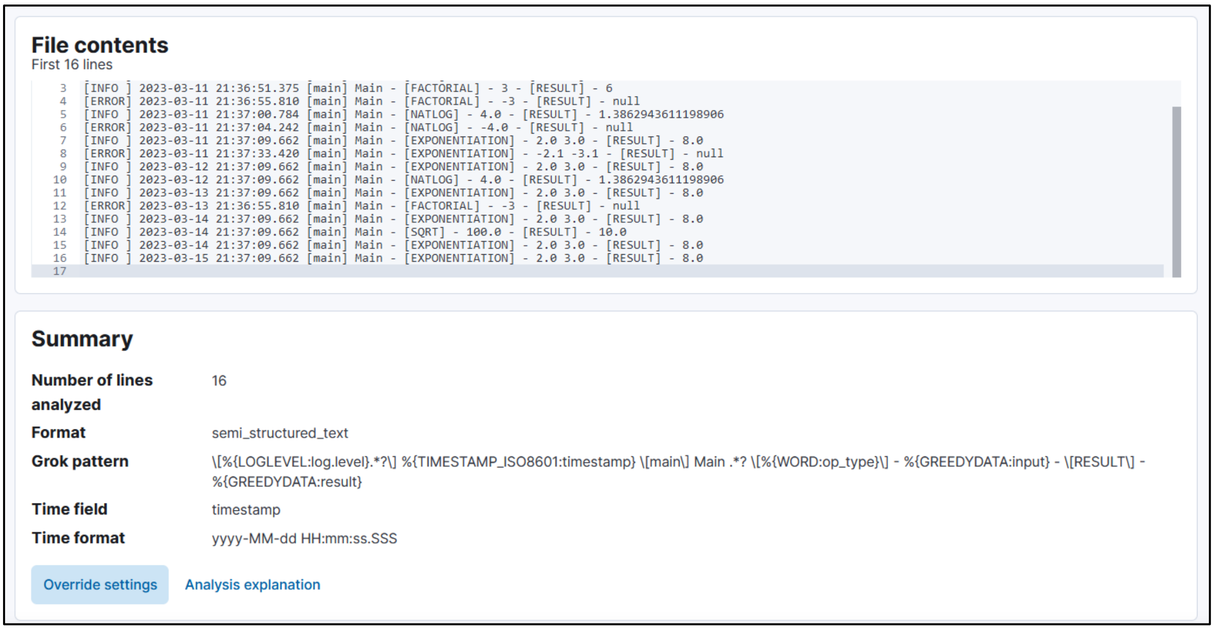


Get started by adding integrations

To start working with your data, use one of our many ingest options. Collect data from an app or service, or upload a file. If you're not ready to use your own data, play with a sample data set.

Click on “Upload a file” to upload logs

In the screen that follows, we will be asked to upload our file. I upload my log file:



File contents

First 16 lines

```
[INFO ] 2023-03-11 21:36:51.375 [main] Main - [FACTORIAL] - 3 - [RESULT] - 6
[ERROR] 2023-03-11 21:36:55.810 [main] Main - [FACTORIAL] - -3 - [RESULT] - null
[INFO ] 2023-03-11 21:37:00.784 [main] Main - [NATLOG] - 4.0 - [RESULT] - 1.3862943611198906
[ERROR] 2023-03-11 21:37:04.242 [main] Main - [NATLOG] - -4.0 - [RESULT] - null
[INFO ] 2023-03-11 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
[ERROR] 2023-03-11 21:37:33.420 [main] Main - [EXPONENTIATION] - -2.1 -3.1 - [RESULT] - null
[INFO ] 2023-03-12 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
[INFO ] 2023-03-12 21:37:09.662 [main] Main - [NATLOG] - 4.0 - [RESULT] - 1.3862943611198906
[INFO ] 2023-03-13 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
[ERROR] 2023-03-13 21:36:55.810 [main] Main - [FACTORIAL] - -3 - [RESULT] - null
[INFO ] 2023-03-14 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
[INFO ] 2023-03-14 21:37:09.662 [main] Main - [SQRT] - 100.0 - [RESULT] - 10.0
[INFO ] 2023-03-14 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
[INFO ] 2023-03-15 21:37:09.662 [main] Main - [EXPONENTIATION] - 2.0 3.0 - [RESULT] - 8.0
```

Summary

Number of lines analyzed 16

Format semi_structured_text

Grok pattern \[%{LOGLEVEL:log.level}.*?\] %{TIMESTAMP_ISO8601:timestamp} \[main\] Main .*? \[%{WORD:op_type}\] - %{GREEDYDATA:input} - \[RESULT\] - %{GREEDYDATA:result}

Time field timestamp

Time format yyyy-MM-dd HH:mm:ss.SSS

Uploaded Log File

As we can see, ELK Stack automatically tries to infer a grok pattern – which is the pattern that is followed by the uploaded file. However, this pattern need not always be correct, and it can be changed by clicking on the Override settings button, where we can enter our grok pattern. For the logs that were generated by my program, the grok pattern is:

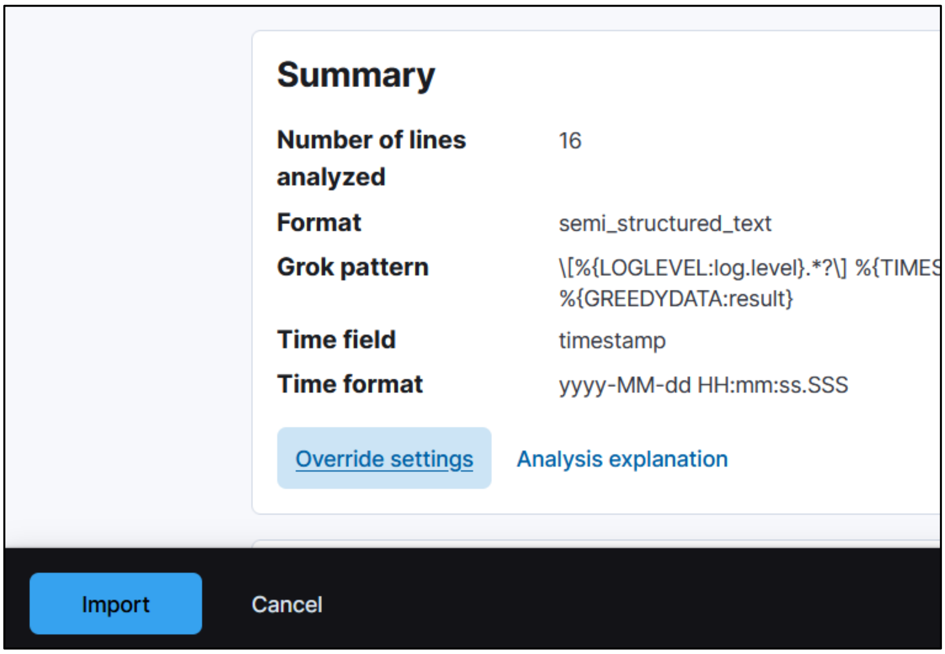
```
\[%{LOGLEVEL:log.level}.*?\] %{TIMESTAMP_ISO8601:timestamp} \[main\]
Main .*? \[%{WORD:op_type}\] - %{GREEDYDATA:input} - \[RESULT\] -
%{GREEDYDATA:result}
```

As we can see from the above grok pattern:

1. It searches for the log level – It can be either of INFO or ERROR
2. It searches for the timestamp – In the ISO8601 timestamp format

3. It searches for the operation type – It can be either of SQRT (Square Root), FACTORIAL (Factorial), NATLOG (Natural Logarithm) or EXPONENTIATION (Power).
4. It searches for the input and result

Once we have entered the above grok pattern, we click Apply and then click on Import button:



Summary

Number of lines analyzed 16

Format semi_structured_text

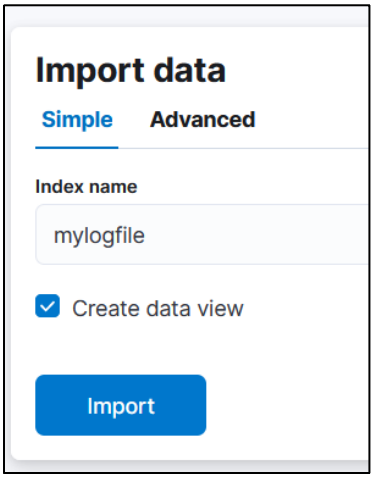
Grok pattern \[%{LOGLEVEL:log.level}.*?\] %{TIMES
%{GREEDYDATA:result}

Time field timestamp

Time format yyyy-MM-dd HH:mm:ss.SSS

Click on the Import button to import the log

It then asks us to provide an index name, and then click on import:



Import data

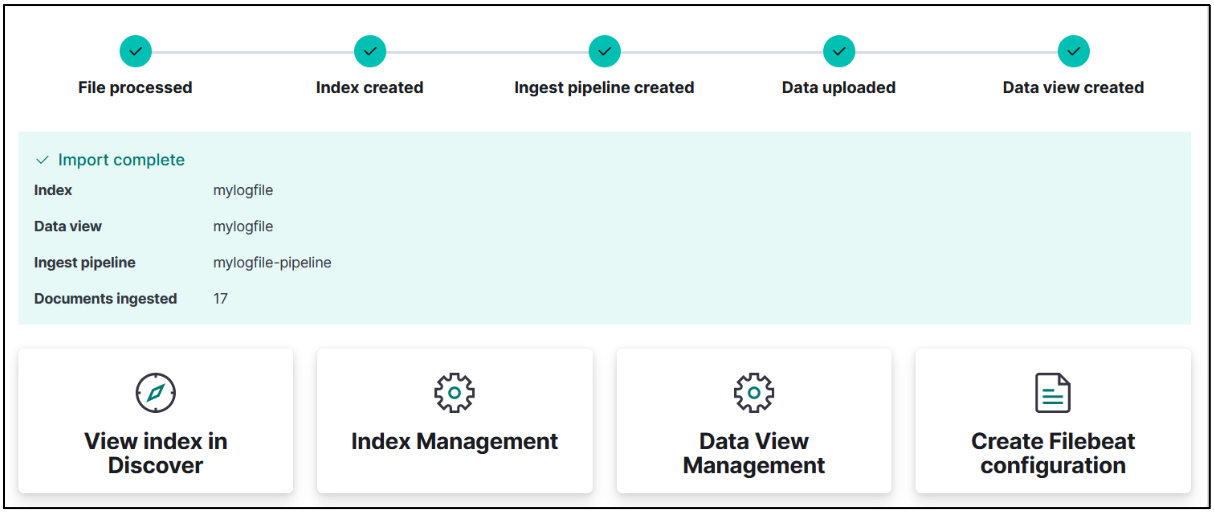
Index name

mylogfile

☑ Create data view

Providing index name

Next, we click on View index in Discover:



File processed

Index created

Ingest pipeline created

Data uploaded

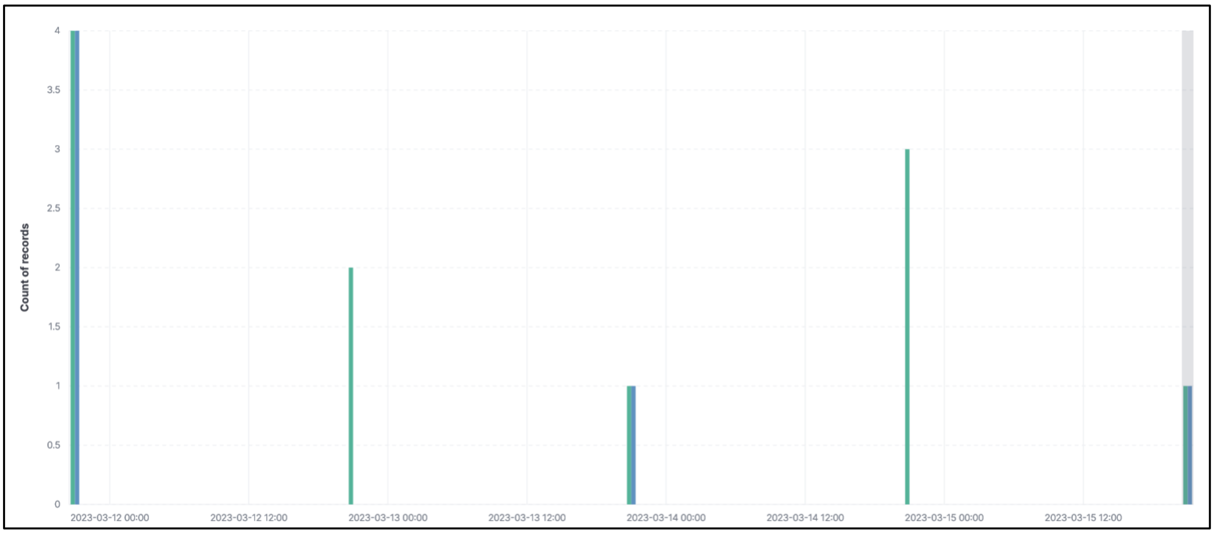
Data view created

Import complete

Index	mylogfile
Data view	mylogfile
Ingest pipeline	mylogfile-pipeline
Documents ingested	17

Log file successfully processed

We can now edit our visualization to our needs, and we get the following Dashboard:



Blue bars indicate ERROR Log level, while Green Bars indicate INFO log level

As we add more and more data, our visualizations will become a lot denser and better. This is how we can visualize our logs using ELK stack.

Setting up Ngrok for Automatic Build Triggers

Instead of manually building jobs in Jenkins, we can build jobs automatically as and when our code repository gets updated. In order to do so, we first need to install Ngrok, which exposes the Jenkins server running on our local machine to the internet.

We follow the below steps to setup Ngrok:

1. Sign up at Ngrok
2. Download the Ngrok file from here.
3. Execute the following command:

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

4. Copy the authentication token from Ngrok dashboard.
5. Add the authentication token to Ngrok by executing the below command:

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

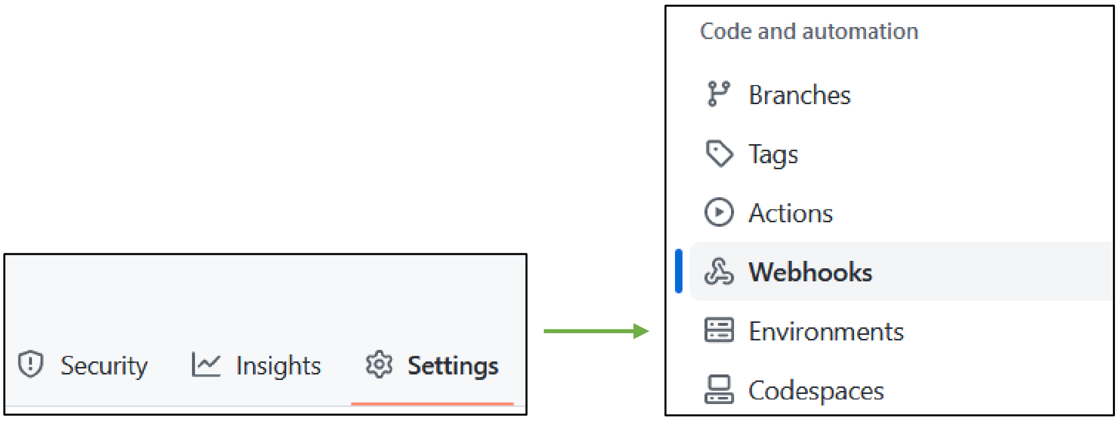
6. Now, we can expose the relevant port by using:

```
$ nrgrok http <Port number to expose>
```

Using this, we can expose our Jenkins Port number 8080 to the HTTP Port 80. We will use the new HTTP URL on which our Jenkins is exposed as the target URL for delivering webhooks.

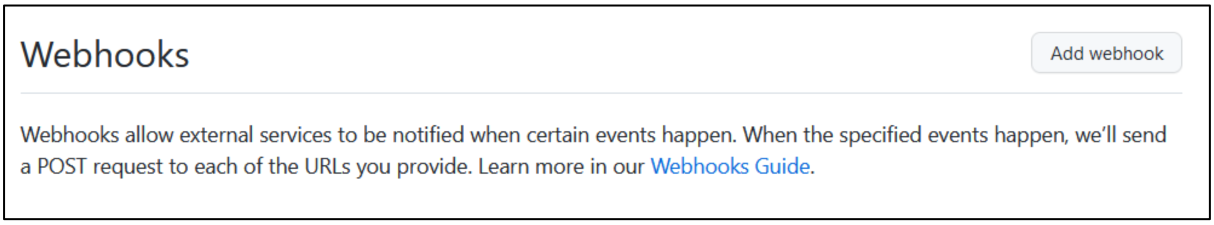
Now, we need to setup webhooks for our repository. Firstly, we create an authentication token with the permissions of admin:repo_hook (the steps are similar to the one shown for Git, except for the permissions).

Next, we go to settings of our GitHub repository, scroll down a bit and click on "Webhooks" under the Code and automation section:



Going to Webhooks settings in GitHub Repository

Now, we see an Add webhook button on the right, we click that and it takes us to a page where we can configure the webhook:

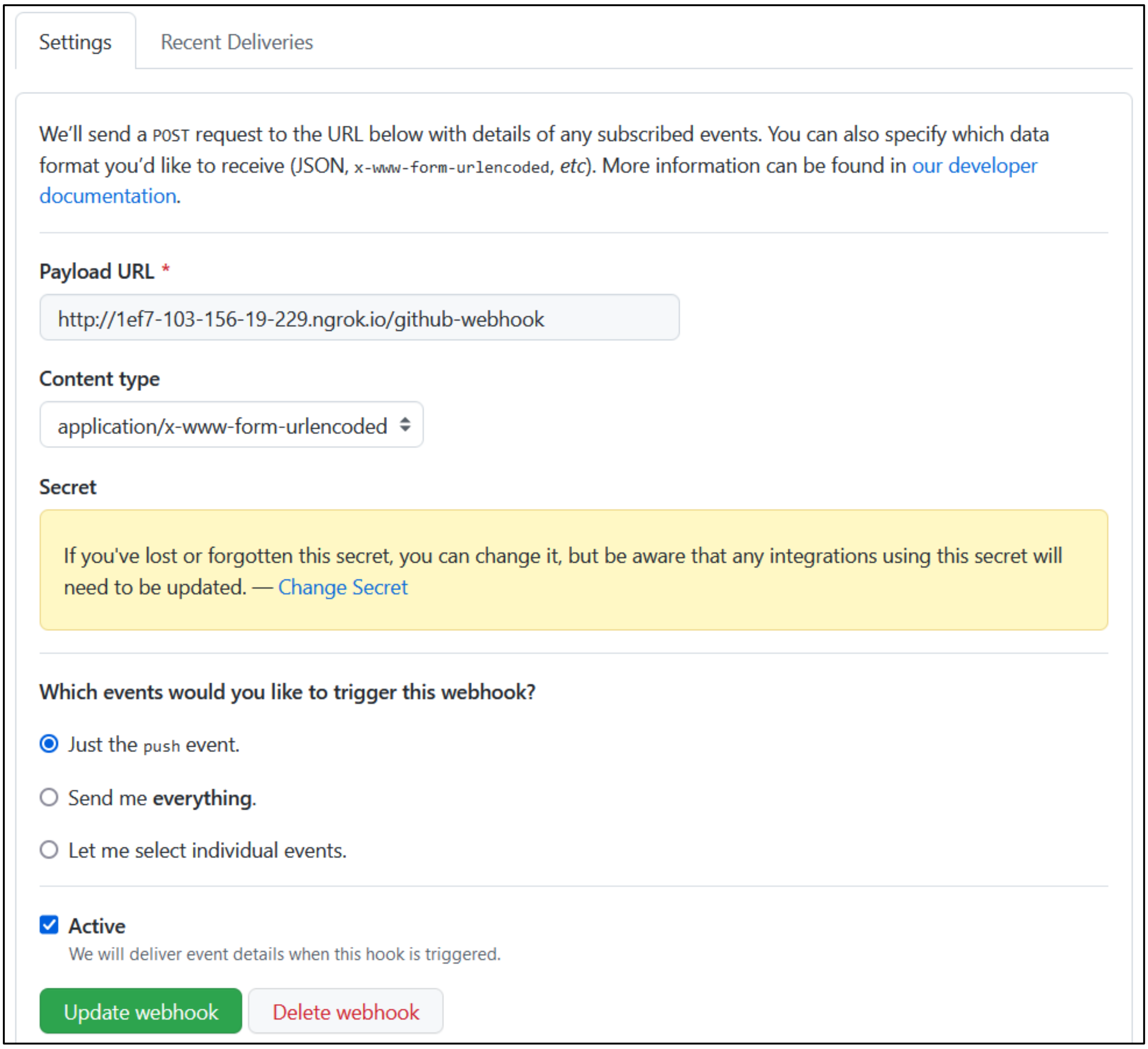


Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

Adding Webhooks to GitHub repository

Now, we enter our Webhook details, and click on Update webhook/Create webhook as follows:



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 *

http://1ef7-103-156-19-229.ngrok.io/github-webhook

Content type

application/x-www-form-urlencoded

Secret

If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated. — Change Secret

Which events would you like to trigger this webhook?

● Just the `push` event.

○ Send me **everything**.

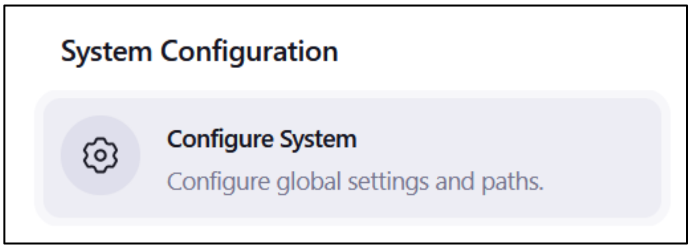
○ Let me select individual events.

☑ **Active**

We will deliver event details when this hook is triggered.

Setting up Webhooks

Now, we need to go to Jenkins, and configure the necessary settings required for the webhook. We go to Dashboard → Manage Jenkins → Configure System (under System Configuration Section)



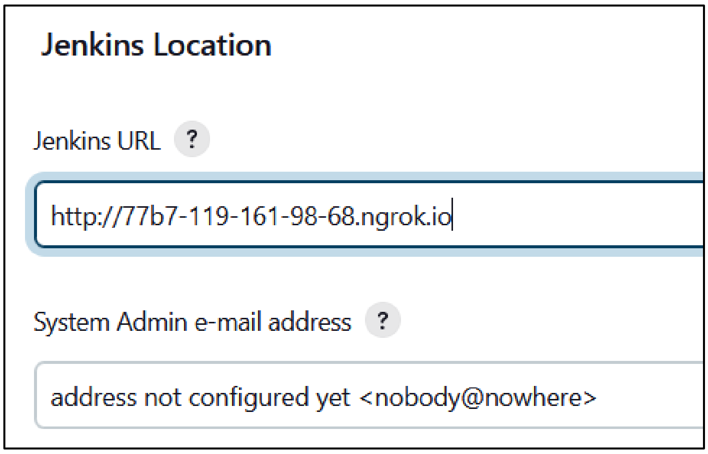
System Configuration

Configure System

Configure global settings and paths.

Going to Configure System Settings

Now, we scroll all the way down until we find Jenkins URL, and change it to the Internet IP Address given by Ngrok, and click on Apply, and then Save:



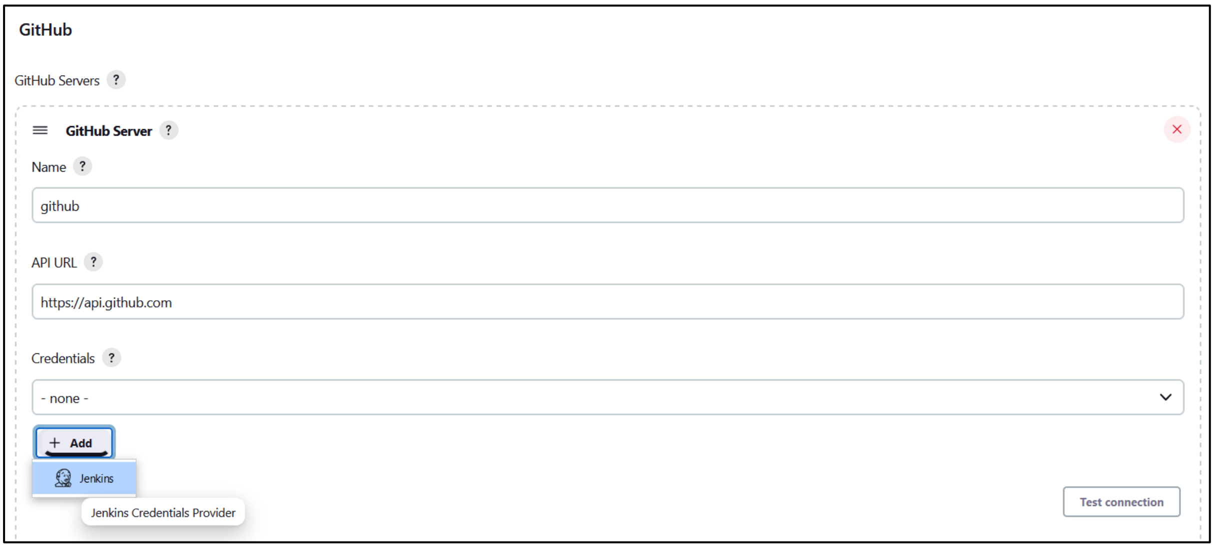
Jenkins Location

Jenkins URL: http://77b7-119-161-98-68.ngrok.io

System Admin e-mail address: address not configured yet <nobody@nowhere>

Setting the Jenkins URL

We further scroll down and under the GitHub section, we click on Add Credentials:



GitHub

GitHub Servers

GitHub Server

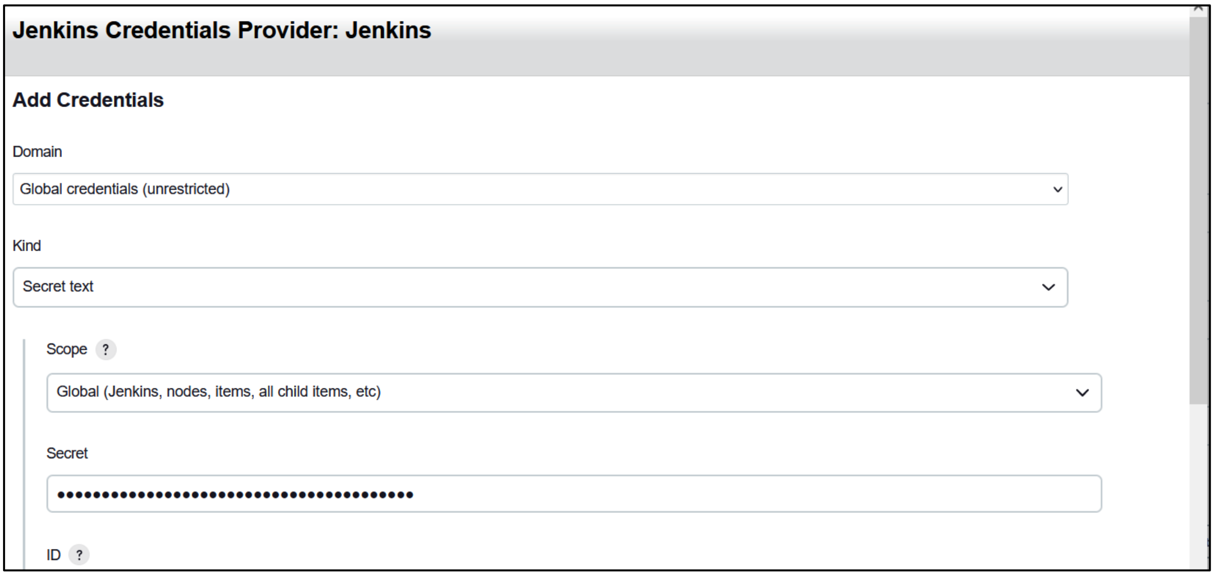
Name: github

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

Credentials: - none -

Adding Credentials for GitHub

We choose the kind as Secret Text and enter our authentication token, and click on Add:



Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: Secret text

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

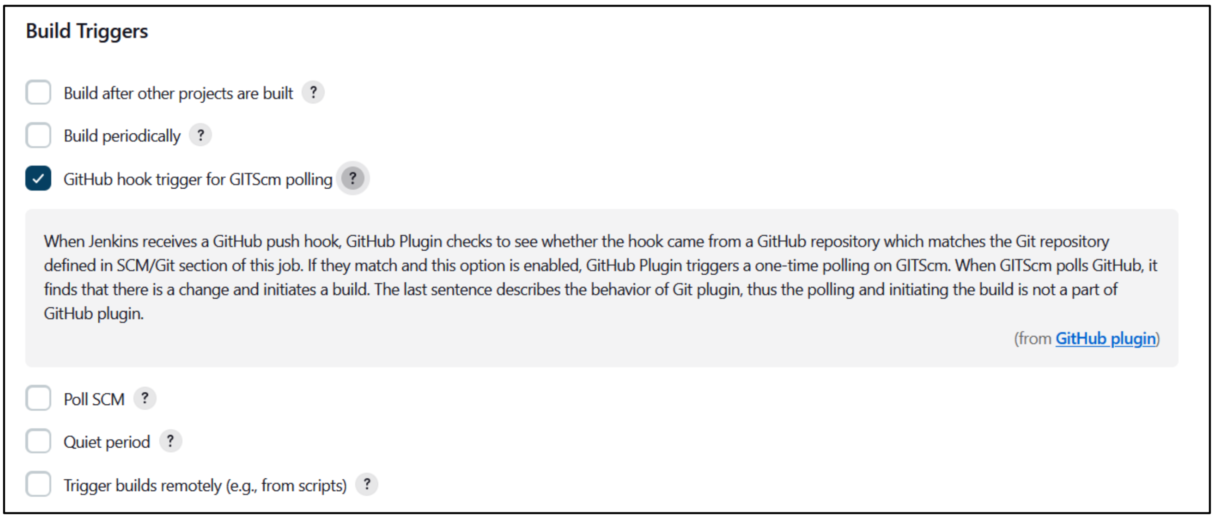
Secret: ••

ID

Adding Credentials

We finally click on Apply, and then Save.

Finally, there is one more step that we need to change, in the Jenkins Project, we need to tick the 'GitHub hook trigger for GITScm polling' option, and then click on Apply, then Save:



Build Triggers

□ Build after other projects are built

□ Build periodically

☑ GitHub hook trigger for GITScm polling

When Jenkins receives a GitHub push hook, GitHub Plugin checks to see whether the hook came from a GitHub repository which matches the Git repository defined in SCM/Git section of this job. If they match and this option is enabled, GitHub Plugin triggers a one-time polling on GITScm. When GITScm polls GitHub, it finds that there is a change and initiates a build. The last sentence describes the behavior of Git plugin, thus the polling and initiating the build is not a part of GitHub plugin.

(from GitHub plugin)

□ Poll SCM

□ Quiet period

□ Trigger builds remotely (e.g., from scripts)

Checking the GitHub hook trigger in the Jenkins Project Settings

That's it! Our Project now will build automatically whenever any new changes are pushed to the repository!

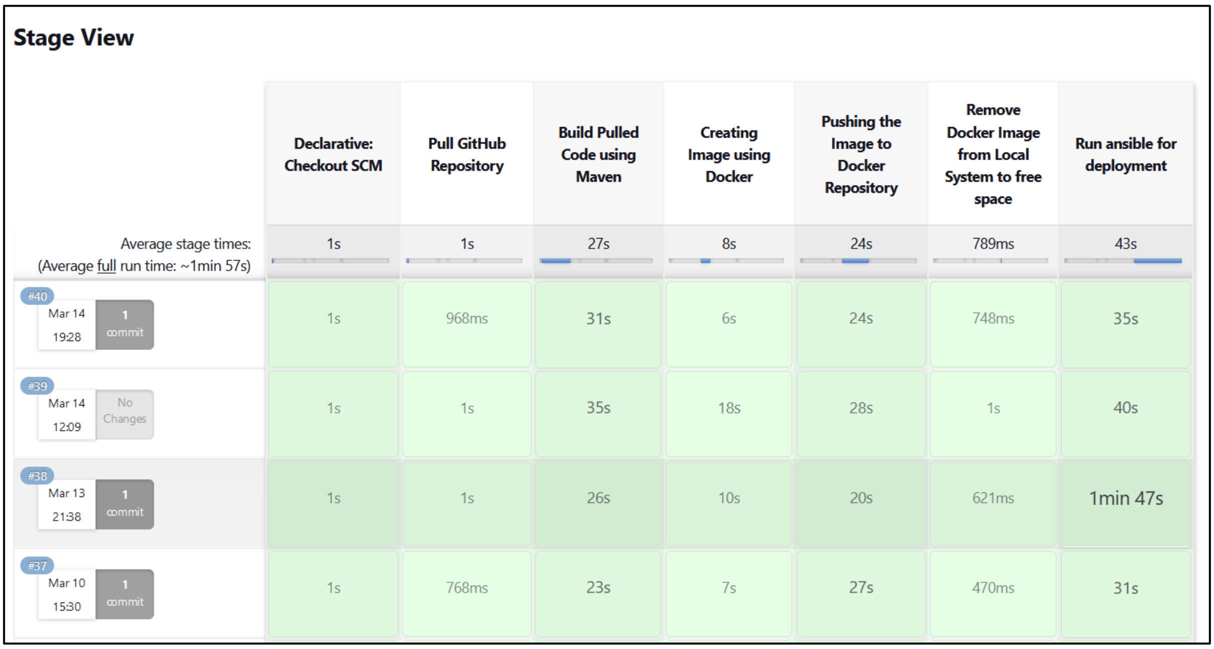
Adding Apache Log4j for Logging Facility

I used Apache Log4j for logging. It is a popular logging framework that is used in many projects. In order to enable logging in my application, I had to:

1. Include the dependencies for log4j in my pom.xml file
2. Create a new file called log4j.xml, through which I specified the logging pattern.

I followed the steps given in this tutorial and this tutorial to enable logging in my application.

Screenshot of Pipeline

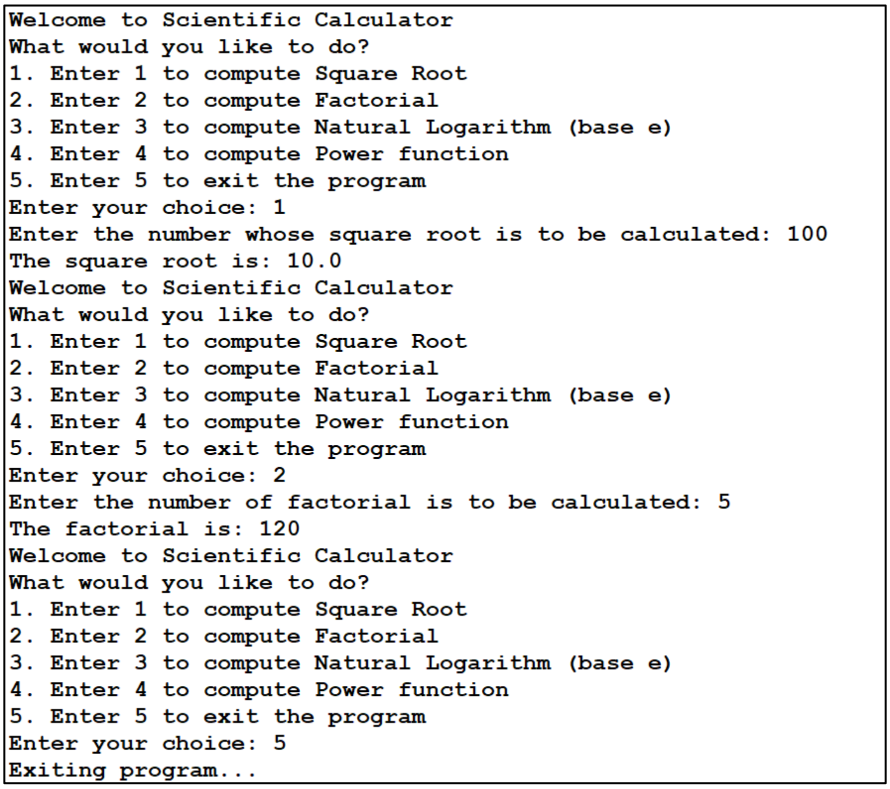


Stage View

	Declarative: Checkout SCM	Pull GitHub Repository	Build Pulled Code using Maven	Creating Image using Docker	Pushing the Image to Docker Repository	Remove Docker Image from Local System to free space	Run ansible for deployment
Average stage times: (Average full run time: ~1min 57s)	1s	1s	27s	8s	24s	789ms	43s
#40 Mar 14 19:28 1 commit	1s	968ms	31s	6s	24s	748ms	35s
#39 Mar 14 12:09 No Changes	1s	1s	35s	18s	28s	1s	40s
#38 Mar 13 21:38 1 commit	1s	1s	26s	10s	20s	621ms	1min 47s
#37 Mar 10 15:30 1 commit	1s	768ms	23s	7s	27s	470ms	31s

Pipeline Screenshot (All Success 😊)

Screenshot of Program



```
Welcome to Scientific Calculator
What would you like to do?
1. Enter 1 to compute Square Root
2. Enter 2 to compute Factorial
3. Enter 3 to compute Natural Logarithm (base e)
4. Enter 4 to compute Power function
5. Enter 5 to exit the program
Enter your choice: 1
Enter the number whose square root is to be calculated: 100
The square root is: 10.0
Welcome to Scientific Calculator
What would you like to do?
1. Enter 1 to compute Square Root
2. Enter 2 to compute Factorial
3. Enter 3 to compute Natural Logarithm (base e)
4. Enter 4 to compute Power function
5. Enter 5 to exit the program
Enter your choice: 2
Enter the number of factorial is to be calculated: 5
The factorial is: 120
Welcome to Scientific Calculator
What would you like to do?
1. Enter 1 to compute Square Root
2. Enter 2 to compute Factorial
3. Enter 3 to compute Natural Logarithm (base e)
4. Enter 4 to compute Power function
5. Enter 5 to exit the program
Enter your choice: 5
Exiting program...
```

Screenshot of Program

With this, we are done and have successfully built an end-to-end CI/CD pipeline.