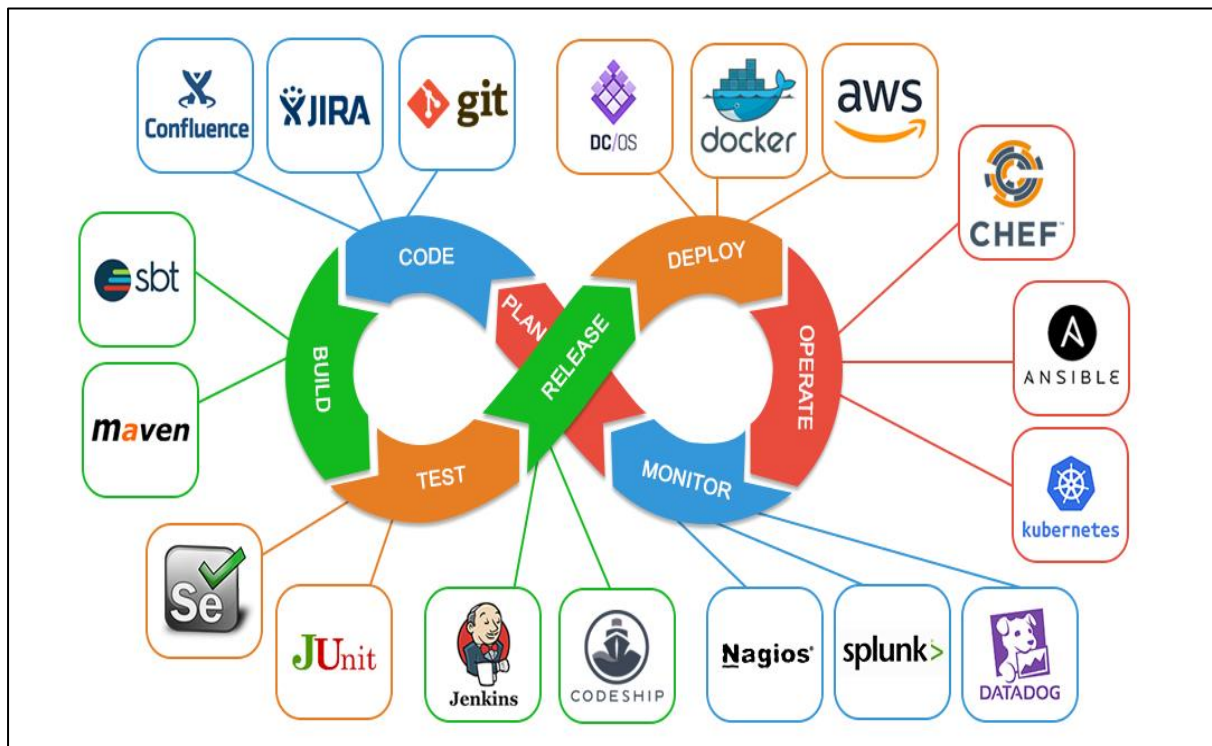


Scientific Calculator – Project Report

(Software Production Engineering)



Harshit Nigam

MT2021052

Project Objective:

The objective of this project is to build a scientific calculator with the help of automation tools, creating an integration and deployment chain. This includes source control management (SCM), Continuous Integration (CI), Continuous Deployment (CD), Configuration Management and Continuous Monitoring (CM). This is achieved by using DevOps tools like Jenkins, Docker, Maven, IntelliJ, and elk. Functionalities like Natural Logarithm, Square root, Power function and factorial are implemented and are taken through the Jenkins pipeline incrementally. The main objective of this project is to learn DevOps concepts CI/CD/CM that is achieved by creating a Jenkins pipeline.

What is DevOps?

DevOps (A shorter word for `Development` and `Operations`) is the combination of practices and tools designed to increase an organization's ability to deliver applications and services faster than traditional software development processes. It is a collaboration of developers and operations engineers participating together in the entire service lifecycle from design through the development process to production.

Why DevOps?

- 1) DevOps has both development and operations team working side by side, which has led to shortening of development cycles.
- 2) Renews focus on customers and their experience of the product. DevOps promotes shorter releases and immediate feedback to enhance user experience.
- 3) Introduces automation to the development process and supports end-to-end responsibility.

Tools used:

- 1) Github: version control system
- 2) Jenkins: CI/CD pipeline
- 3) Maven: Build tool
- 4) Junit: Testing
- 5) Dockerhub: Containerisation/ holds Docker image of our application
- 6) ELK: Continuous Monitoring

Project Structure:

1. Java project with maven on IntelliJ
2. Add some changes to pom.xml
3. Jenkins Pipeline for continuous deployment and integration
4. Create a pipeline and write script :
 - i. git clone
 - ii. mvn clean install
 - iii. Create docker image from docker file
 - iv. Push the created docker image to the docker hub
5. The last stage of pipeline script pulls and runs the docker image on localhost port.

Project Workflow:

- 1) Setup Java (8/ 11) environment for the project.**
- 2) Use IntelliJ to create Maven project with Java CLI program on Scientific Calculator.**
- 3) Linking the project with GitHub repository and using Git for SCM.**
- 4) Install Jenkins and run on 8080 localhost port (Default).**
- 5) Sign up in DockerHub in order to push images to Docker containers.**
- 6) Configure Jenkins with credentials and plugins.**
- 7) Create Jenkins pipeline project and write pipeline script including stages like git clone, Maven Build, create Docker image, pushing image to Docker Hub, Cleaning previously build image and running image on remote server.**
- 8) Build the pipeline and wait for successful running of all stages in the pipeline.**
- 9) At last, the Docker image is pulled and run on the localhost.**

Environment Setup for Cloud/Node machine:

My System Specifications:

OS: Windows 10 Home

Processor: Intel® Core™ i7-6500 CPU @ 2.50GHz 2.59GHz

RAM: 12GB

1. Installing Java:

i) Install Java SE 8 using following link,

<https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html>

ii) Edit Environment variable path and add 'C:\Program Files\Java\jdk1.8.0_311\bin'.

2. Install IntelliJ (Community Edition):

<https://www.jetbrains.com/idea/download/#section=windows>

3. Download Jenkins .war file (LTS version)

<https://www.jenkins.io/download/>

i) Change directory in CMD to the directory where Jenkins.war file is present and Run command: `java -jar jenkins.war -httpPort=9090` (you can put any free port number).

4. Install Docker Desktop for Windows

<https://docs.docker.com/desktop/windows/install/>

5. Download Maven and edit the environment variable 'path' with its bin file.

1. Write Source Code in Java

Functionalities include,

- i) Power Function
- ii) Factorial
- iii) Square root
- iv) Natural Logarithm

Code: https://github.com/NightmareNight-em/devops_calculator/blob/main/src/main/java/Calculator.java

```
---Press a key according to given options---
 1:Square Root  2:Factorial  3:Natural Logarithm  4:Power Funtion  5:Exit
Enter your choice:::1
Enter a number for calculating square-root:::64
12:17:54.524 [main] INFO  Calculator - Calculating square root  of given number:64.0
12:17:54.531 [main] INFO  Calculator - Resultant answer of power operations is : 8.0
The resultant output is: 8.0
---Press a key according to given options---
 1:Square Root  2:Factorial  3:Natural Logarithm  4:Power Funtion  5:Exit
Enter your choice:::2
Enter a number for calculating factorial:::6
12:18:02.556 [main] INFO  Calculator - Calculating factorial  of given number:6
12:18:02.556 [main] INFO  Calculator - Resultant answer of power operations is : 720.0
The resultant output is: 720.0
---Press a key according to given options---
 1:Square Root  2:Factorial  3:Natural Logarithm  4:Power Funtion  5:Exit
Enter your choice:::3
Enter a number for calculating natural-logarithm:::45
12:18:07.788 [main] INFO  Calculator - Calculating natural log  of given number:45.0
12:18:07.789 [main] INFO  Calculator - Resultant answer of natural log operation is : 3.8066624897703196
The resultant output is: 3.8066624897703196
---Press a key according to given options---
 1:Square Root  2:Factorial  3:Natural Logarithm  4:Power Funtion  5:Exit
Enter your choice:::4
Enter two numbers separated by space for calculating power(a^b: a b):::5 4
12:18:16.366 [main] INFO  Calculator - Calculating power function of given numbers:5.0,4.0
12:18:16.366 [main] INFO  Calculator - Resultant answer of power operation is : 625.0
The resultant output is: 625.0
```

Unit Testing – JUnit:

A unit test is a piece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behavior or state.

A JUnit test is a method contained in a class, which is only used for testing. This is called a Test class. To define that a certain method is a test method, annotate it with the @Test annotation. This method executes the code under test. You use an assert method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These calls are typically called asserts or assert statements. Assert statements typically allow to define messages which are shown if the test fails. You should provide here meaningful messages to make it easier for the user to identify and fix the problem.

Code for tests (using JUnit): https://github.com/NightmareNight-em/devops_calculator/blob/main/src/test/java/CalculatorTest.java

Fig.1 shows the number of test cases that passed and the test cases that failed to pass. This enables the developer to work on the issues that arise in the code and makes them easy to be identified to get them fixed.

2. Create Maven Project

- i) Open IntelliJ and create Maven Project.
- ii) Add appropriate dependencies in pom.xml to use log4j framework to track what happens in our application.
- iii) Create a .java file in /src/main/java/ with source code and .java file in /src/main/test with test cases to test our application.


```

[INFO] Running calculator.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.053 s - in
calculator.AppTest
[INFO] Running CalculatorTest
19:09:50.279 [main] INFO Calculator - Calculating natural log of given number:10.0
19:09:50.296 [main] INFO Calculator - Resultant answer of natural log operation is :
2.302585092994046
19:09:50.296 [main] INFO Calculator - Calculating natural log of given number:-9.0
19:09:50.297 [main] INFO Calculator - Resultant answer of natural log operation is : -1.0
19:09:50.297 [main] INFO Calculator - Calculating natural log of given number:0.0
19:09:50.298 [main] INFO Calculator - Resultant answer of natural log operation is : -1.0
19:09:50.299 [main] INFO Calculator - Calculating power function of given numbers:3.0,4.0
19:09:50.299 [main] INFO Calculator - Resultant answer of power operation is : 81.0
19:09:50.300 [main] INFO Calculator - Calculating power function of given
numbers:-2.0,-2.0
19:09:50.300 [main] INFO Calculator - Resultant answer of power operation is : 0.25
19:09:50.308 [main] INFO Calculator - Calculating power function of given numbers:-3.0,4.0
19:09:50.309 [main] INFO Calculator - Resultant answer of power operation is : 81.0
19:09:50.311 [main] INFO Calculator - Calculating power function of given numbers:2.0,-2.0
19:09:50.311 [main] INFO Calculator - Resultant answer of power operation is : 0.25
19:09:50.314 [main] INFO Calculator - Calculating natural log of given number:-9.0
19:09:50.317 [main] INFO Calculator - Resultant answer of natural log operation is : -1.0
19:09:50.327 [main] INFO Calculator - Calculating factorial of given number:5
19:09:50.328 [main] INFO Calculator - Resultant answer of power operations is : 120.0
19:09:50.329 [main] INFO Calculator - Calculating factorial of given number:0
19:09:50.329 [main] INFO Calculator - Calculating factorial of given number:-23
19:09:50.330 [main] INFO Calculator - Resultant answer of power operations is : -1.0
19:09:50.331 [main] INFO Calculator - Calculating factorial of given number:1
19:09:50.331 [main] INFO Calculator - Calculating factorial of given number:11
19:09:50.333 [main] INFO Calculator - Resultant answer of power operations is : 3.99168E7
19:09:50.339 [main] INFO Calculator - Calculating square root of given number:100.0
19:09:50.340 [main] INFO Calculator - Resultant answer of power operations is : 10.0
19:09:50.341 [main] INFO Calculator - Calculating square root of given number:100.0
19:09:50.341 [main] INFO Calculator - Resultant answer of power operations is : 10.0
19:09:50.341 [main] INFO Calculator - Calculating square root of given number:-4.0
19:09:50.341 [main] INFO Calculator - Resultant answer of power operations is : -1.0
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.781 s - in
CalculatorTest

```

Fig.1

3. Host the project on GitHub

- i) Create a new repository on GitHub.
- ii) Configure Maven project on local machine and link it to GitHub repository using following commands,

\$ git init // Initialize Maven project as Git local repository

\$ git remote add origin "URL of the GitHub repository"

\$ git add . // stage Maven project in Git environment

\$ git commit -m "Commit Message"

```
$ git push -u origin main
```

NOTE: To run these commands in Windows environment, we need application like `Git Bash`. Alternatively, `IntelliJ` can be used to connect with remote repository, add, commit and push.

4. Create an automated pipeline using Jenkins

Automation is the key principle of DevOps. Automation aims to remove human intervention and execute the entire development cycle from version control to deployment automatically using triggers, scripts etc.

Here, we use Jenkins pipeline, a powerful tool to make a CI/ CD pipeline and create flexible and customized stages to perform tasks involved in software development lifecycle.

1. Download Generic java package Jenkins.war
2. Open CMD with path of the .war file and execute it using following command,

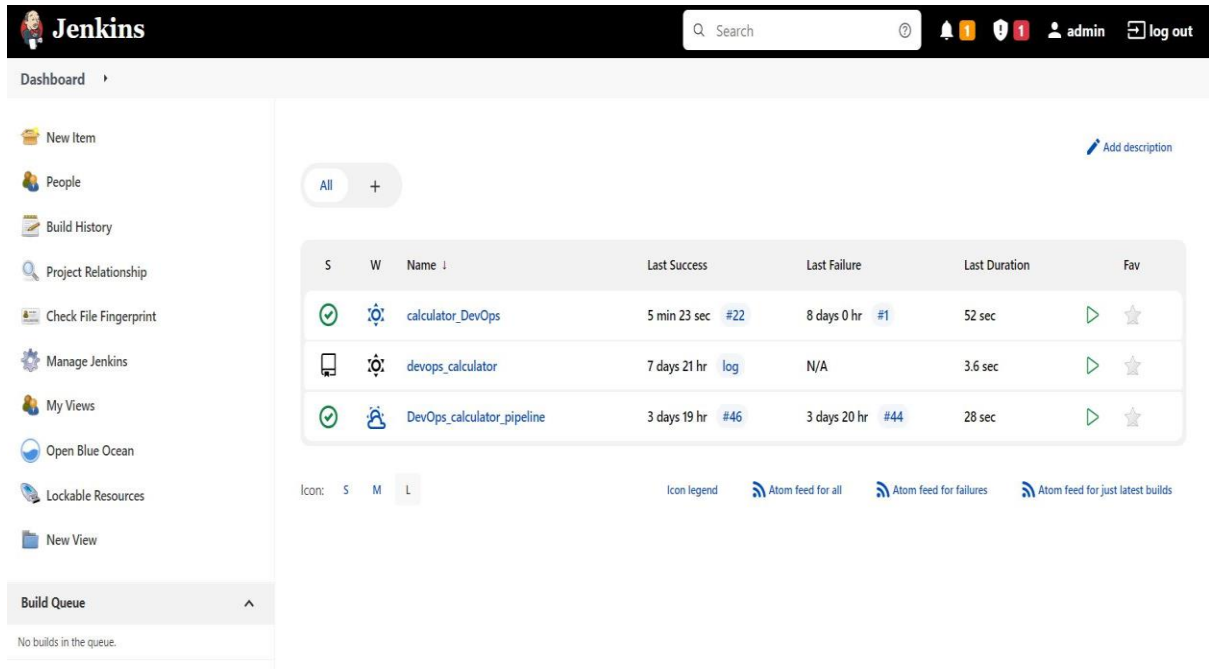
```
> java -jar jenkins.war
```

3. For the interface of Jenkins, go to **localhost:8080** in your browser.

When Jenkins is launched for the first time, it asks for a 32 bit encryption key for authentication which can be found at following path (For Windows OS):

C://Users/{Username}/.jenkins/secrets/master.key

4. Jenkins configuration page opens up. Install suggested plugins, then signup with new user details, which will henceforth, open up the Jenkins dashboard.



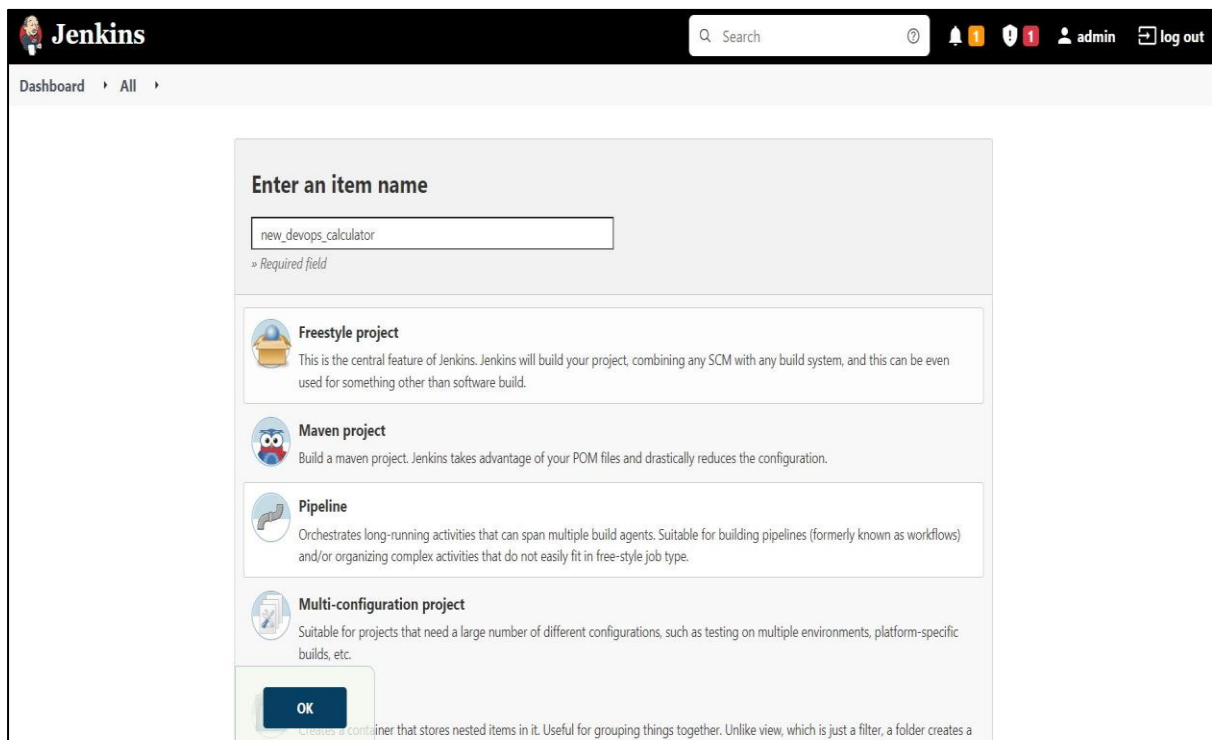
The screenshot shows the Jenkins Dashboard interface. At the top, there's a header with the Jenkins logo, a search bar, and user information (admin) with a log out button. Below the header, the left sidebar contains navigation links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, Open Blue Ocean, Lockable Resources, and New View. The main content area displays a table of builds. The table has columns for status (S), workflow icon (W), name, last success, last failure, last duration, and favorite (Fav). Three builds are listed: 'calculator_DevOps' (success, 5 min 23 sec, #22), 'devops_calculator' (success, 7 days 21 hr, log), and 'DevOps_calculator_pipeline' (success, 3 days 19 hr, #46). Below the table, there's an 'Icon legend' and three Atom feed links: 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. At the bottom, there's a 'Build Queue' section showing 'No builds in the queue.'

S	W	Name	Last Success	Last Failure	Last Duration	Fav
✓	🔧	calculator_DevOps	5 min 23 sec #22	8 days 0 hr #1	52 sec	▶ ☆
✓	🔧	devops_calculator	7 days 21 hr log	N/A	3.6 sec	▶ ☆
✓	🔧	DevOps_calculator_pipeline	3 days 19 hr #46	3 days 20 hr #44	28 sec	▶ ☆

5. Go to Manage Jenkins -> Manage Plugins -> Under Available section and install the following plugins to write the pipeline and restart Jenkins using above java command.

- i) Blue Ocean
- ii) Docker Pipeline
- iii) Maven Integration Plugin
- iv) Pipeline
- v) Pipeline Maven Integration Plugin
- vi) Pipeline utility steps
- vii) SSH pipeline steps, SSH plugin, Publish over SSH (if using a remote server and connection using SSH).

6. Click on 'new item' on left taskbar. The following page opens up. Select 'Pipeline' and click on 'OK'.



The image shows the Jenkins 'New Item' dialog. At the top, there's a search bar and a user profile 'admin' with a 'log out' button. Below the search bar, the breadcrumb 'Dashboard > All' is visible. The main section is titled 'Enter an item name' and contains a text input field with the value 'new_devops_calculator'. Below the input field, it says '» Required field'. There are four options listed: 'Freestyle project', 'Maven project', 'Pipeline', and 'Multi-configuration project'. Each option has an icon and a brief description. The 'Pipeline' option is highlighted with a blue border. At the bottom, there is an 'OK' button and a 'Cancel' button.

Enter an item name

new_devops_calculator

» 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.

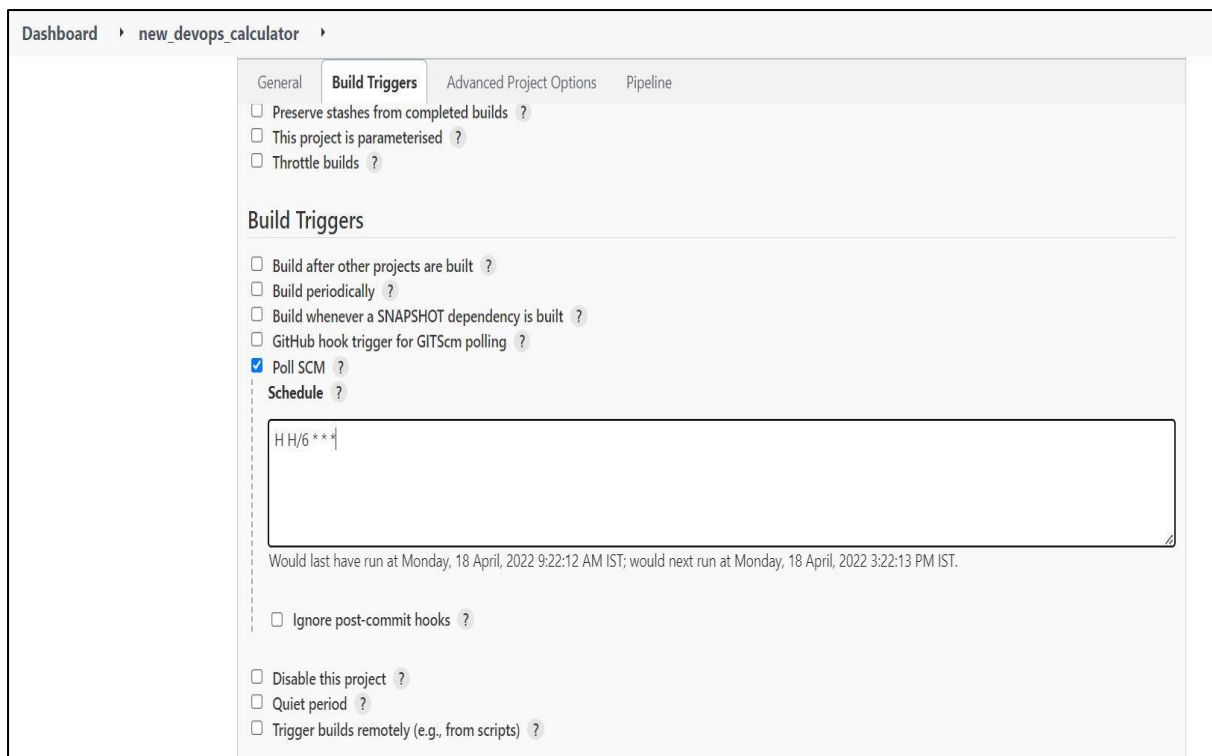
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

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.

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

OK

7. Select 'Poll SCM' in 'Build Triggers' section.



The image shows the Jenkins 'Build Triggers' configuration page for the 'new_devops_calculator' project. The breadcrumb is 'Dashboard > new_devops_calculator'. The page has four tabs: 'General', 'Build Triggers' (selected), 'Advanced Project Options', and 'Pipeline'. Under the 'Build Triggers' tab, there are three checkboxes: 'Preserve stashes from completed builds', 'This project is parameterised', and 'Throttle builds'. Below these is the 'Build Triggers' section with four checkboxes: 'Build after other projects are built', 'Build periodically', 'Build whenever a SNAPSHOT dependency is built', and 'GitHub hook trigger for GITScm polling'. The 'Poll SCM' checkbox is checked. Below the 'Poll SCM' checkbox is a 'Schedule' field with a text input containing 'H H/6 * * *'. Below the schedule field, it says 'Would last have run at Monday, 18 April, 2022 9:22:12 AM IST; would next run at Monday, 18 April, 2022 3:22:13 PM IST.' At the bottom, there are three more checkboxes: 'Ignore post-commit hooks', 'Disable this project', and 'Quiet period'. The 'Trigger builds remotely (e.g., from scripts)' checkbox is also present.

Dashboard > new_devops_calculator

General **Build Triggers** Advanced Project Options Pipeline

☐ Preserve stashes from completed builds ?

☐ This project is parameterised ?

☐ Throttle builds ?

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☐ Build whenever a SNAPSHOT dependency is built ?

☐ GitHub hook trigger for GITScm polling ?

☒ Poll SCM ?

Schedule ?

H H/6 * * *

Would last have run at Monday, 18 April, 2022 9:22:12 AM IST; would next run at Monday, 18 April, 2022 3:22:13 PM IST.

☐ Ignore post-commit hooks ?

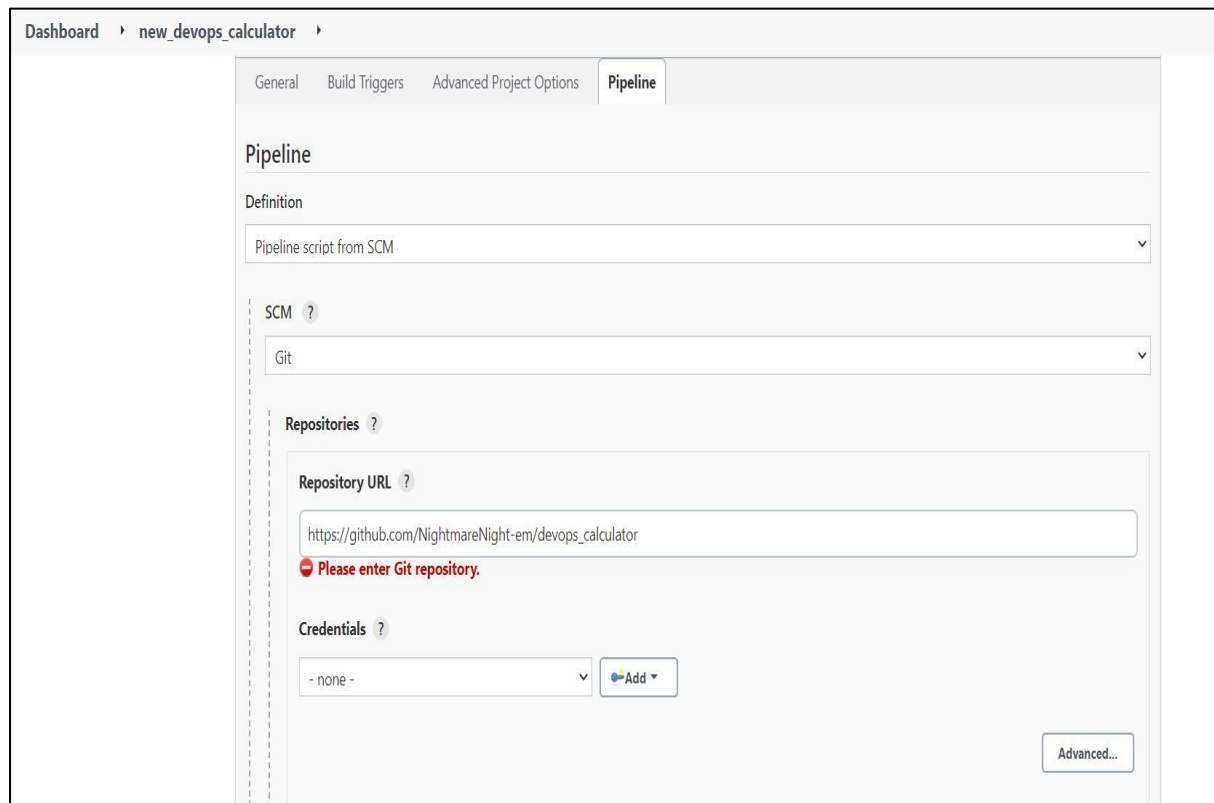
☐ Disable this project ?

☐ Quiet period ?

☐ Trigger builds remotely (e.g., from scripts) ?

This will check for any commits made to our GitHub repository every 6 hours and run the pipeline script automatically if new commits are found.

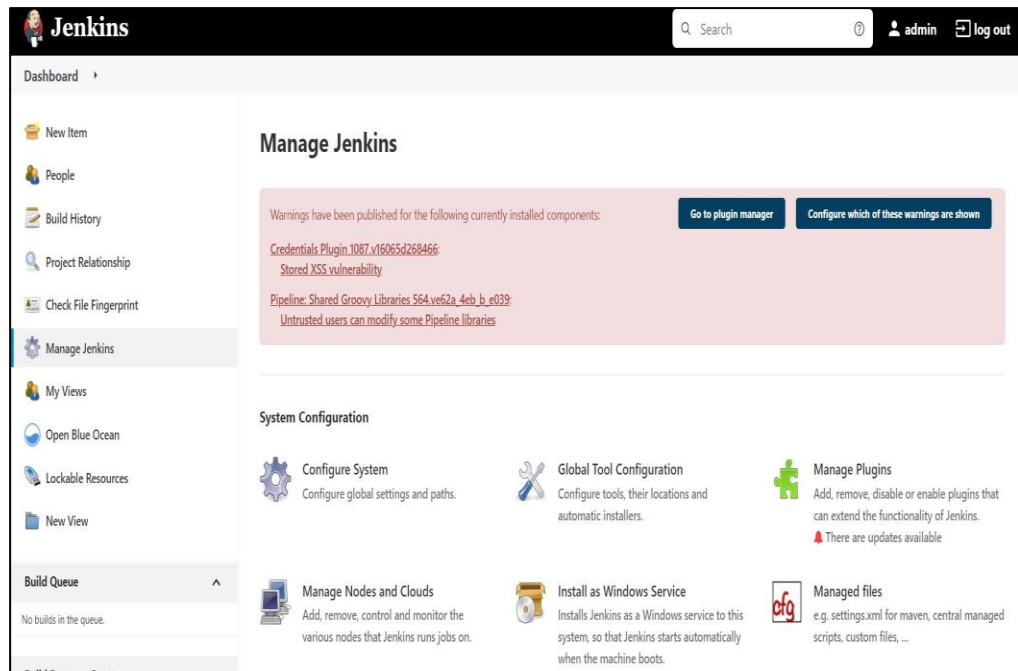
8. Below selections will let Jenkins access pipeline script from our GitHub repository. Paste your GitHub repository link in Repository URL.



The screenshot shows the Jenkins configuration page for a project named 'new_devops_calculator'. The 'Pipeline' tab is selected, showing the 'Definition' section with 'Pipeline script from SCM' chosen. Below this, the 'SCM' is set to 'Git'. In the 'Repositories' section, the 'Repository URL' is filled with 'https://github.com/NightmareNight-em/devops_calculator', but a red error message 'Please enter Git repository.' is displayed below the input field. The 'Credentials' section shows a dropdown menu with '- none -' and an 'Add' button. An 'Advanced...' button is located at the bottom right of the configuration area.

Since our GitHub repository is public, we do not need to add GitHub credentials. However, I am mentioning how to add credentials in Jenkins below.

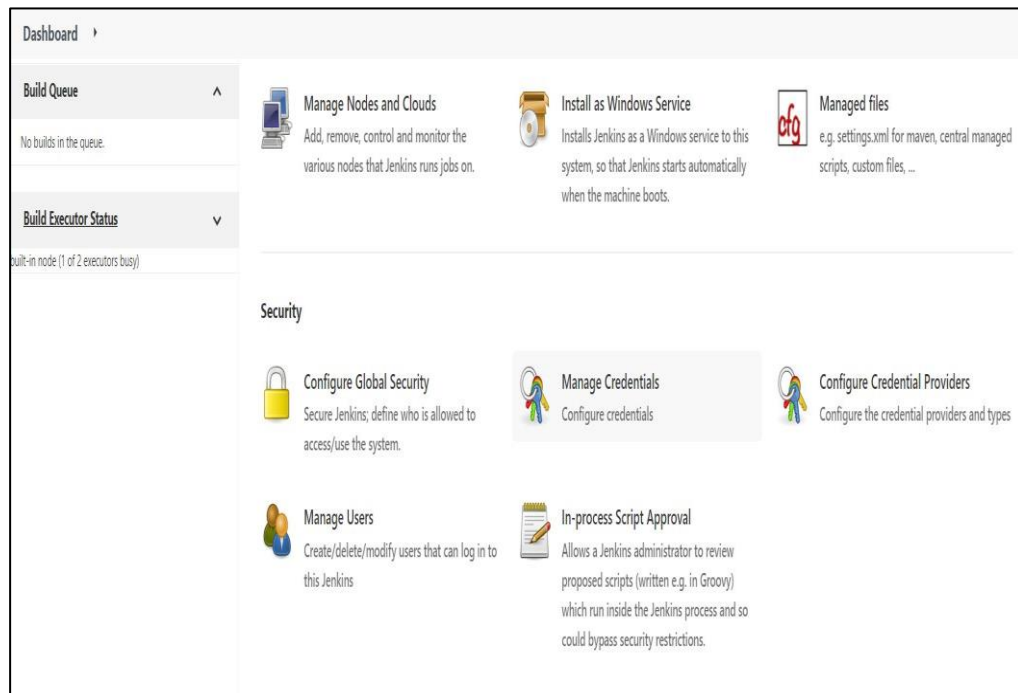
- Click on ‘Manage Jenkins’ on Jenkins Dashboard.



The screenshot shows the Jenkins Dashboard with the 'Manage Jenkins' section highlighted in the left sidebar. The main content area is titled 'Manage Jenkins' and features a warning banner at the top. Below the banner, there is a 'System Configuration' section with several options:

- Configure System**: Configure global settings and paths.
- Global Tool Configuration**: Configure tools, their locations and automatic installers.
- Manage Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins. A red triangle icon indicates that there are updates available.
- Managed files**: e.g. settings.xml for maven, central managed scripts, custom files, ...
- Manage Nodes and Clouds**: Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- Install as Windows Service**: Installs Jenkins as a Windows service to this system, so that Jenkins starts automatically when the machine boots.

- Click on ‘Manage Credentials’.



The screenshot shows the Jenkins Dashboard with the 'Manage Credentials' section highlighted in the left sidebar. The main content area is titled 'Security' and features several options:

- Configure Global Security**: Secure Jenkins; define who is allowed to access/use the system.
- Manage Credentials**: Configure credentials. This option is highlighted with a grey background.
- Configure Credential Providers**: Configure the credential providers and types.
- Manage Users**: Create/delete/modify users that can log in to this Jenkins.
- In-process Script Approval**: Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.

- Click on 'global' in column 'Domain'.

Jenkins Search [?] [1] admin log out

Dashboard > Credentials

New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Open Blue Ocean
Lockable Resources
New View

Build Queue
No builds in the queue.

Credentials

T	P	Store	Domain	ID	Name
[Icon]	[Icon]	Jenkins	(global)	bd76dfcf-5cfb-47b3-8292-f684d72ae500	nightmareNightem/*****
[Icon]	[Icon]	Jenkins	(global)	cb84c965-92d0-4a4f-8897-395b8f6f2bc	nightmareNightem/*****
[Icon]	[Icon]	Jenkins	(global)	docker4harshit	docker4harshit/***** (DockerHub Credentials)
[Icon]	[Icon]	Jenkins	(global)	44.201.235.237	ec2-user (Virtual server)

Icon: [S] [M] [L]

Stores scoped to Jenkins

P	Store	Domains
[Icon]	Jenkins	[Icon] (global)

- Click on 'Add Credentials'.

Jenkins Search [?] [1] admin log out

Dashboard > Credentials > System > Global credentials (unrestricted)

Back to credential domains
Add Credentials

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
bd76dfcf-5cfb-47b3-8292-f684d72ae500	nightmareNightem/*****	Username with password	[Icon]
cb84c965-92d0-4a4f-8897-395b8f6f2bc	nightmareNightem/*****	Username with password	[Icon]
docker4harshit	docker4harshit/***** (DockerHub Credentials)	Username with password	DockerHub Credentials [Icon]
44.201.235.237	ec2-user (Virtual server)	SSH Username with private key	Virtual server [Icon]

Icon: [S] [M] [L]

- Add GitHub credentials if your repository is private.
Note: You need to add DockerHub credentials here. See DockerHub section.

The screenshot shows the Jenkins 'Add Credentials' form for 'Global credentials (unrestricted)'. The breadcrumb trail at the top is 'Dashboard > Credentials > System > Global credentials (unrestricted)'. On the left sidebar, there is a link 'Back to credential domains' and a button 'Add Credentials'. The main form area has the following fields: 'Kind' (a dropdown menu with 'Username with password' selected), 'Scope' (a dropdown menu with 'Global (Jenkins, nodes, items, all child items, etc)' selected), 'Username' (a text input field), a checkbox 'Treat username as secret' (unchecked), 'Password' (a text input field), 'ID' (a text input field), and 'Description' (a text input field). At the bottom right of the form is an 'OK' button.

7. Create Jenkins Pipeline script in a file named **Jenkinsfile** (same name and no extension) and save this file in root directory of Github repository.

Jenkinsfile. It is a simple text file used to write Jenkins pipeline and to automate CI/CD process. It is often termed as `Pipeline as a code` which are of two types, Declarative and Scripted.

Pipeline stages:

- i) Git clone – Checkout SCM
- ii) mvn clean compile test package – Getting executable .jar file
- iii) Building Docker Image

- iv) Pushing docker image into DockerHub repository
- v) Cleaning previously built images
- vi) Running Docker container on development server (localhost)

Pipeline Script:

https://github.com/NightmareNight-em/devops_calculator/blob/main/Jenkinsfile

Note: pom.xml file for Maven project must be in root directory of GitHub repository.

5. Containerization – DockerHub

DockerHub is an open-source project that is promoting cloud-native development through containerization and microservices.

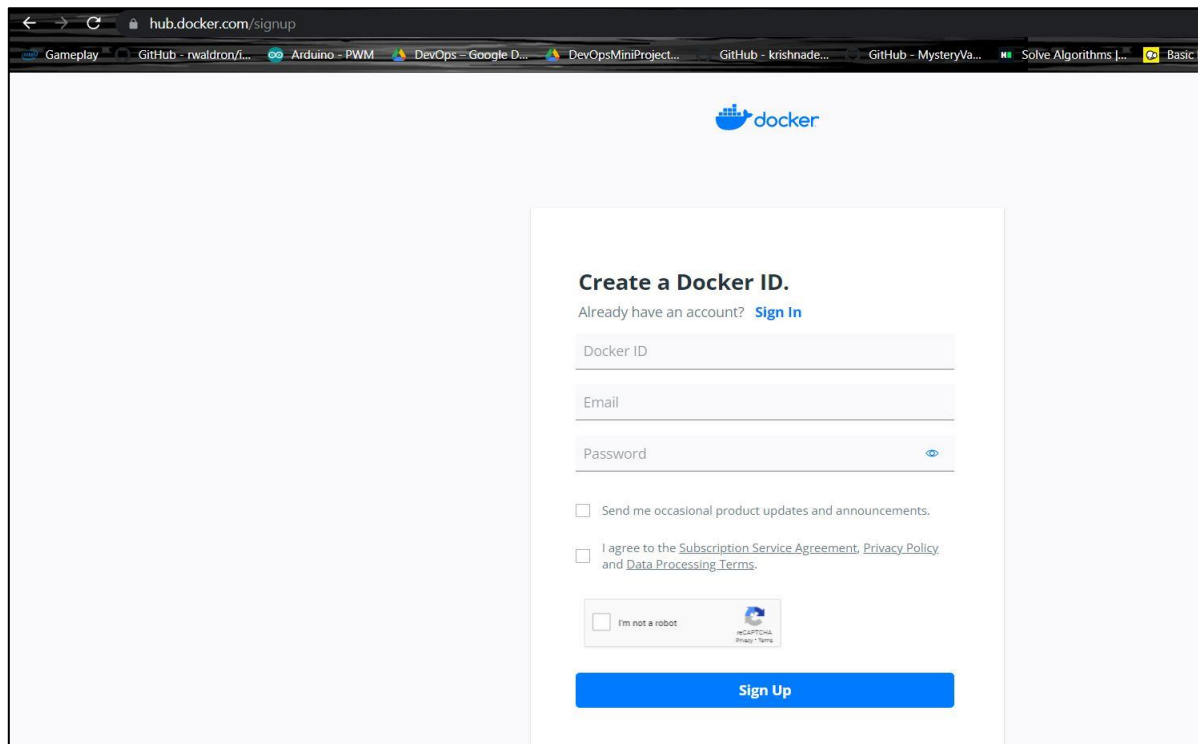
Here, it is worth mentioning the concept of `Virtualization` wherein many guest operating systems can be run on same host OS without the need of extra hardware resources.

Containerization is a special type of virtualization where applications run in independent spaces called containers, all of which share same OS kernel to perform tasks independently.

Herein, we use hub.docker.com to create a repository containing our `container` which holds the docker image of the jar file which came as a result of build process performed by Maven. We first build the docker image and then push it into dockerhub repository in a container.

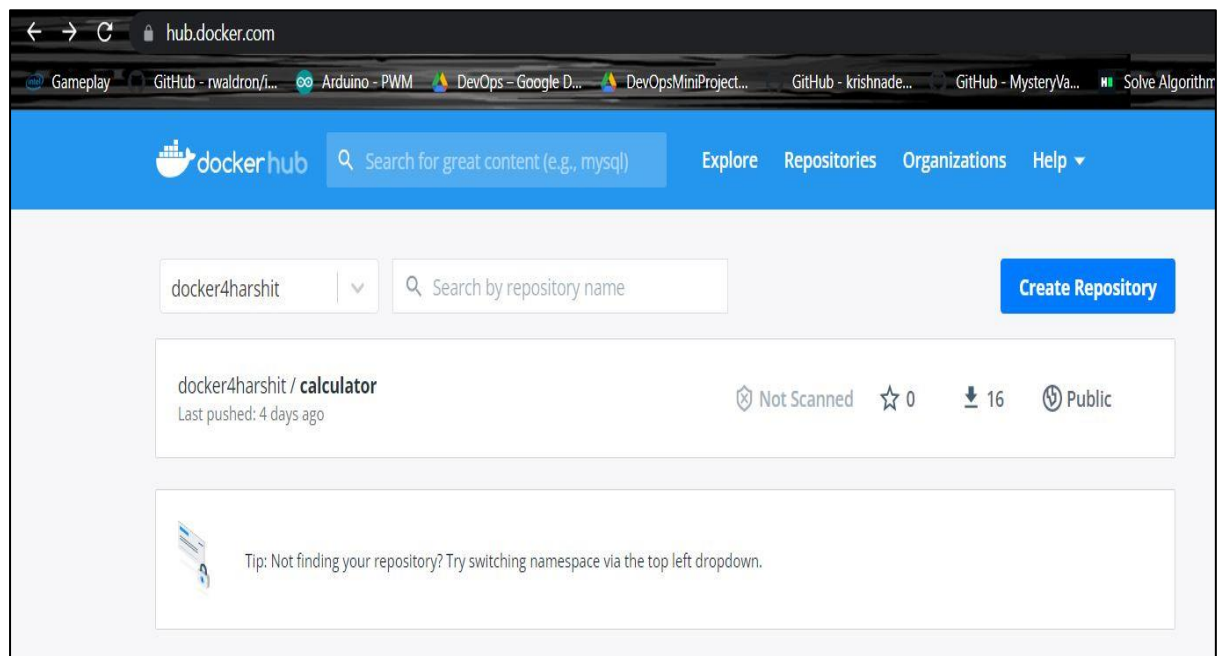
Steps to create repository,

1. Go to hub.docker.com and register with your details.



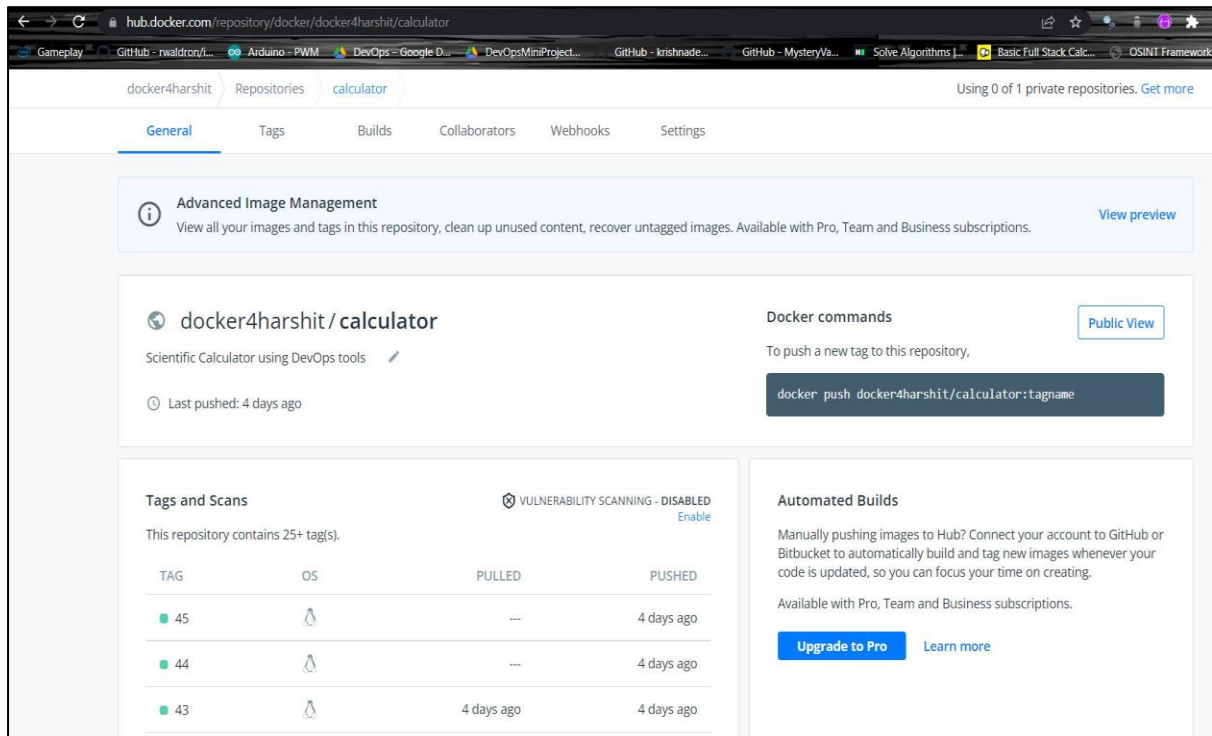
The screenshot shows the Docker Hub signup page. The browser address bar displays 'hub.docker.com/signup'. The page features the Docker logo at the top. The main heading is 'Create a Docker ID.' with a link 'Already have an account? Sign In'. Below this are input fields for 'Docker ID', 'Email', and 'Password'. There are two checkboxes: 'Send me occasional product updates and announcements.' and 'I agree to the Subscription Service Agreement, Privacy Policy and Data Processing Terms.' A CAPTCHA 'I'm not a robot' is also present. A blue 'Sign Up' button is at the bottom.

2. Create Repository.



The screenshot shows the Docker Hub repository creation page. The browser address bar displays 'hub.docker.com'. The page has a blue header with the Docker Hub logo, a search bar, and navigation links: 'Explore', 'Repositories', 'Organizations', and 'Help'. Below the header, there is a dropdown menu showing 'docker4harshit' and a search bar labeled 'Search by repository name'. A blue 'Create Repository' button is on the right. Below this, a repository card for 'docker4harshit / calculator' is shown, with details: 'Last pushed: 4 days ago', 'Not Scanned', '0 stars', '16 downloads', and 'Public'. A tip box at the bottom says: 'Tip: Not finding your repository? Try switching namespace via the top left dropdown.'

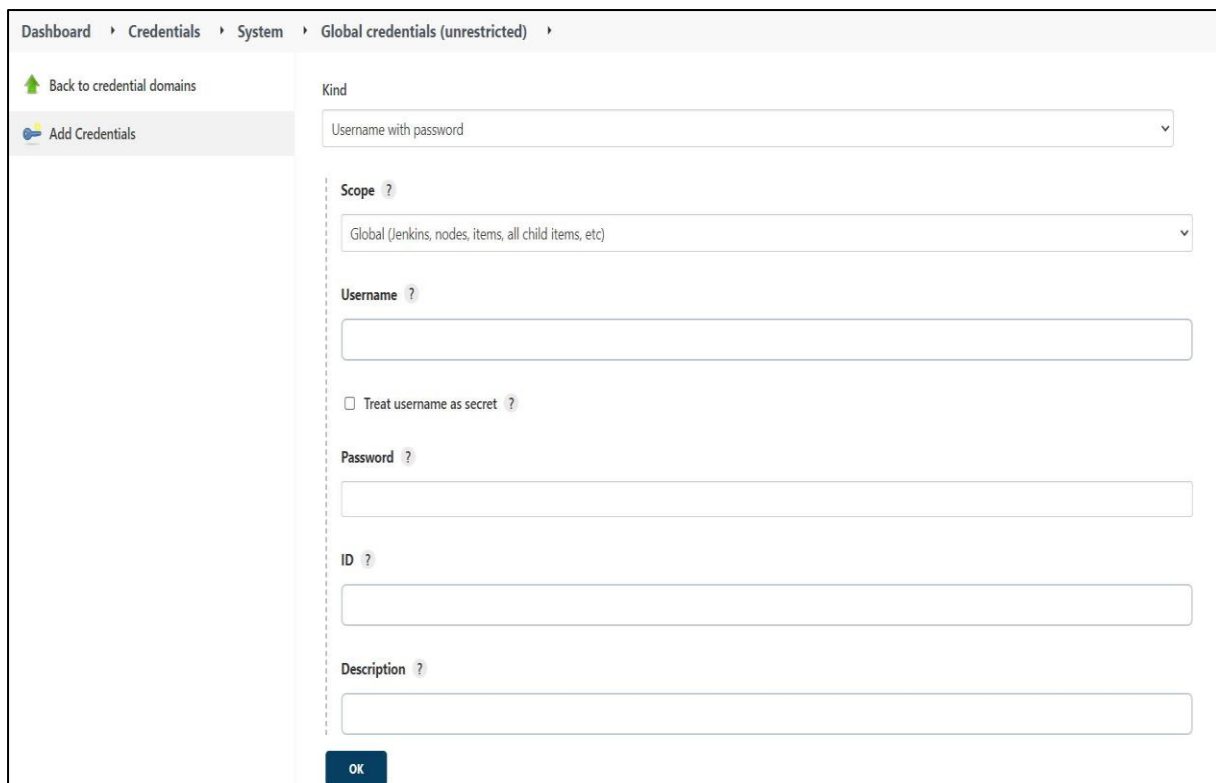
3. Docker images get stored in your repository.



The screenshot shows the Docker Hub interface for the repository 'docker4harshit/calculator'. The page includes a navigation bar with tabs for 'General', 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings'. A banner at the top promotes 'Advanced Image Management'. The repository details section shows the name 'docker4harshit/calculator', a description 'Scientific Calculator using DevOps tools', and the last push time '4 days ago'. A 'Docker commands' section provides the command 'docker push docker4harshit/calculator:tagname'. A 'Tags and Scans' table lists tags 43, 44, and 45, all pushed 4 days ago. A 'VULNERABILITY SCANNING - DISABLED' toggle is visible. An 'Automated Builds' section explains how to connect GitHub or Bitbucket for automatic builds.

TAG	OS	PULLED	PUSHED
45	linux/amd64	---	4 days ago
44	linux/amd64	---	4 days ago
43	linux/amd64	4 days ago	4 days ago

4. Add DockerHub credentials in Jenkins, following the steps given above on adding credentials in Jenkins.



The screenshot shows the Jenkins 'Add Credentials' form. The breadcrumb trail is 'Dashboard > Credentials > System > Global credentials (unrestricted)'. The left sidebar has a 'Back to credential domains' link and an 'Add Credentials' button. The main form has the following fields:

- Kind:** A dropdown menu with 'Username with password' selected.
- Scope:** A dropdown menu with 'Global (Jenkins, nodes, items, all child items, etc)' selected.
- Username:** A text input field.
- Treat username as secret:** An unchecked checkbox.
- Password:** A text input field.
- ID:** A text input field.
- Description:** A text input field.

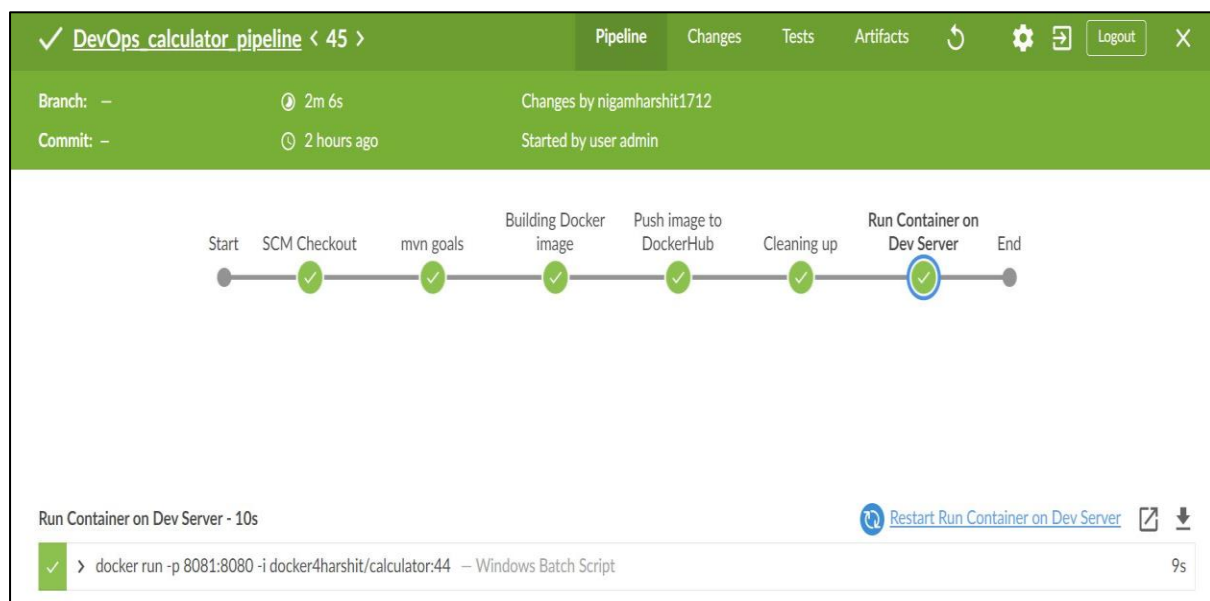
An 'OK' button is at the bottom of the form.

Fill in Repository name and password of your DockerHub account and an ID. Thereafter, write that ID in below code in pipeline script.

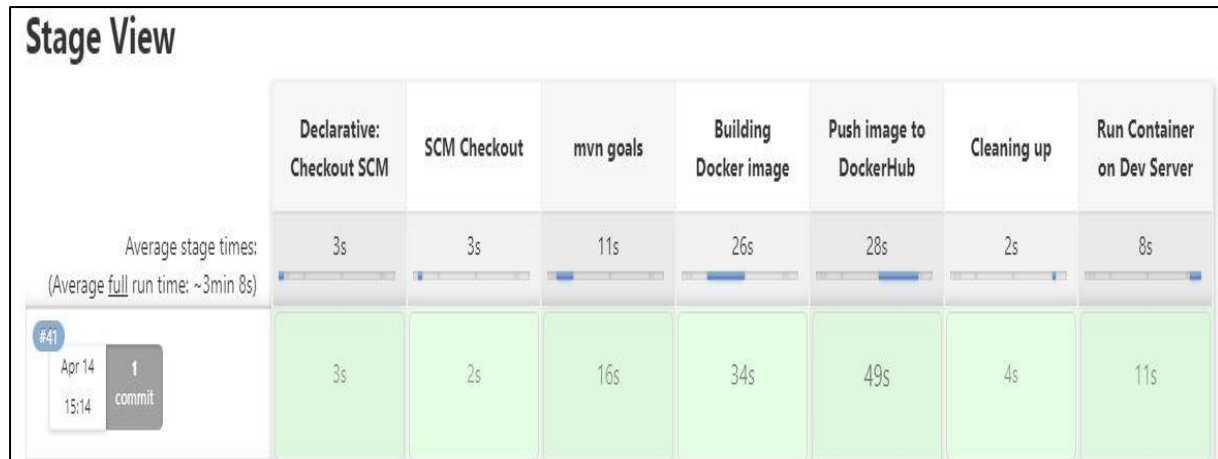
```
enviroment {  
    registry: "docker4harshit/calculator"  
    registryCredential = 'docker'  
    dockerImage = ''  
}
```

Here, registry include DockerHub account name and registryCredential include ID that you put when you add credentials of DockerHub in Jenkins.

6. Working of Pipeline



Stages displayed using BlueOcean (Plugin) dashboard



Stage 1:

Cloning from our GitHub repository.

Stage 2:

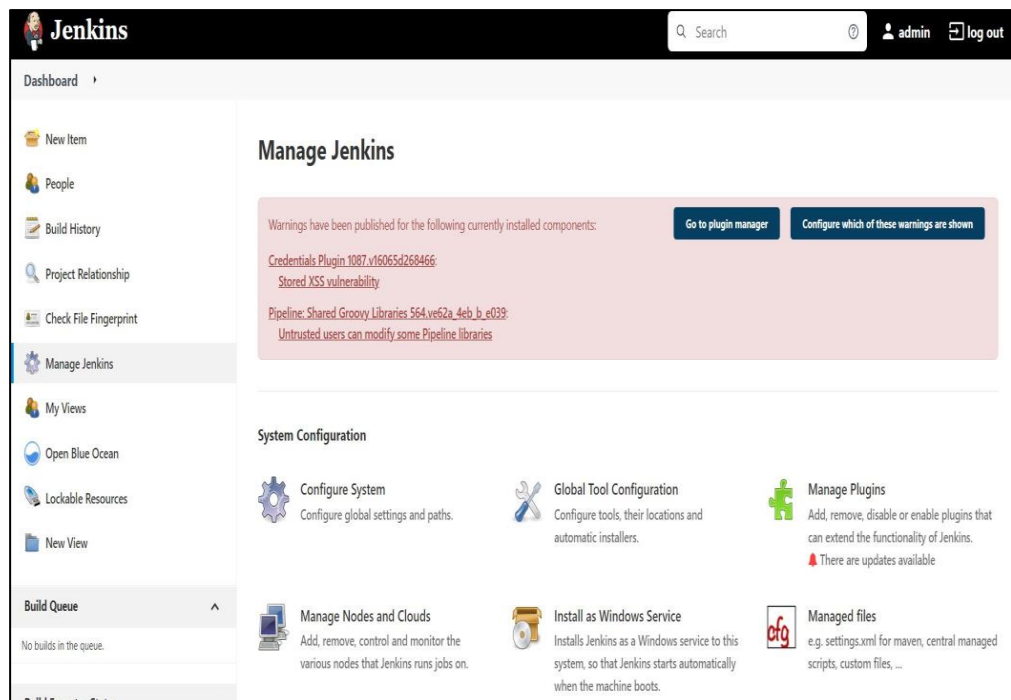
Command ``mvn clean compile test package`` is run on our local terminal by Jenkins. This results in a `.jar` executable file in `/target` folder.

Note: Ensure that Maven and JDK is installed in your local machine and you have added bin files of both libraries in the 'PATH' in system variables in Environment variables.

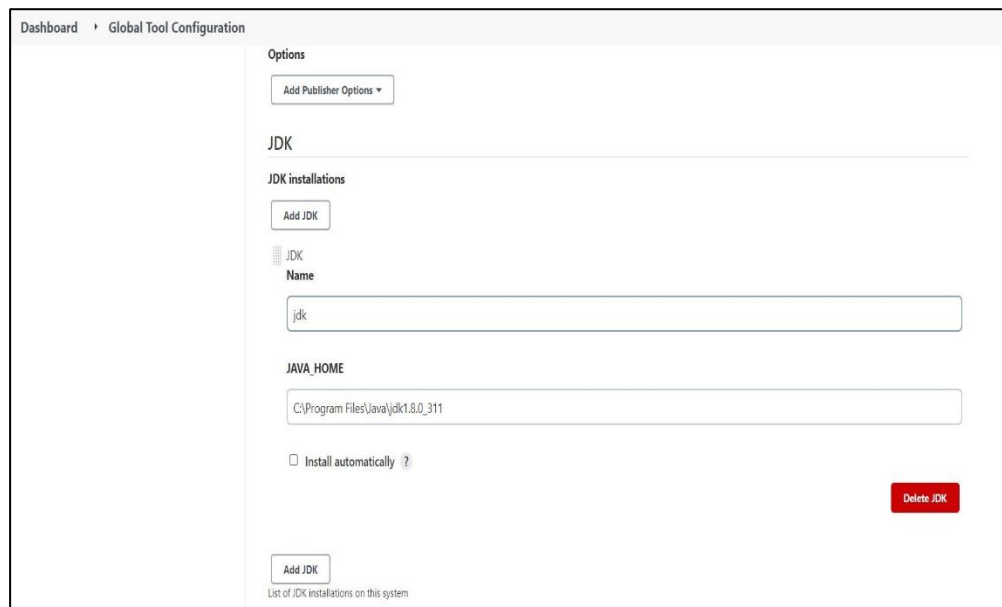
If you still get an error 'mvn is not found as internal or external command', try out below steps.

- Click 'Manage Jenkins' on Jenkins dashboard.

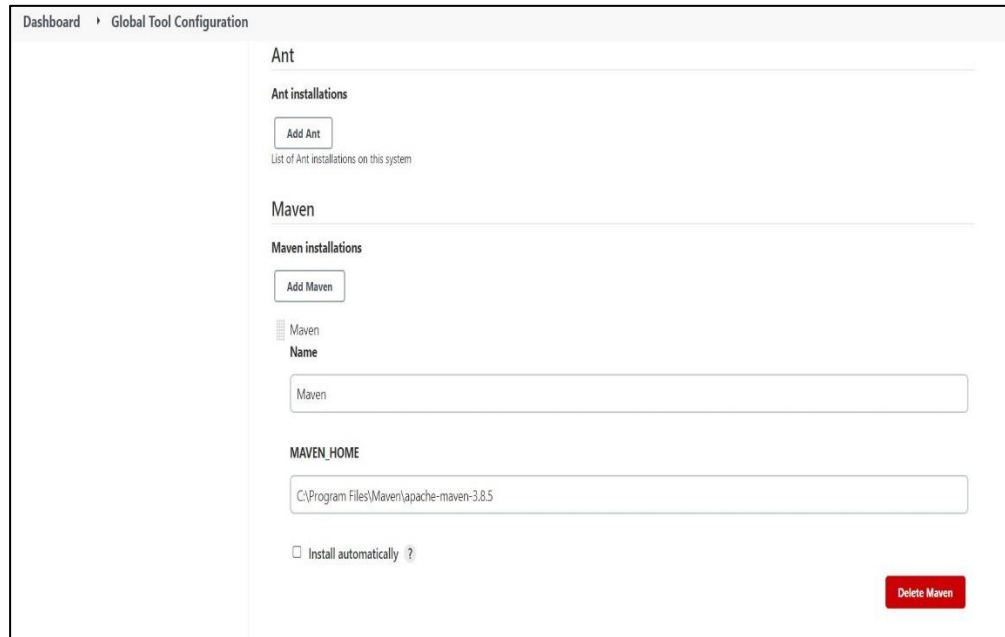
- Click on 'Global Tool Configuration'.



- Add JDK installations and give a local path to the JDK directory.



- Add Maven installations and give a local path to the Apache Maven directory.



The screenshot shows the 'Global Tool Configuration' page. It has two main sections: 'Ant' and 'Maven'. The 'Ant' section has an 'Add Ant' button and a note 'List of Ant installations on this system'. The 'Maven' section has an 'Add Maven' button, a table with one row for 'Maven' with a 'Name' column, a text input field for 'MAVEN_HOME' containing 'C:\Program Files\Maven\apache-maven-3.8.5', a checkbox for 'Install automatically' which is unchecked, and a red 'Delete Maven' button.

Stage 3, 4 and 5:

Docker image of the jar file is built using Dockerfile (containing information about the location and execution command of the jar file among other details).

Dockerfile:

https://github.com/NightmareNight-em/devops_calculator/blob/main/Dockerfile

This image is pushed into DockerHub repository with a unique tag attached.


Henceforth, previously build docker images are deleted from the repository.

Stage 6:

The below docker images are pulled from DockerHub repository.

```
C:\Users\Harshit's PC>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker4harshit/calculator	44	170f851af65f	2 hours ago	530MB
docker4harshit/calculator	43	b6657d6bb7f2	2 hours ago	530MB
docker4harshit/calculator	41	8e2f07846211	3 hours ago	530MB
docker4harshit/calculator	34	3274c6c8650e	3 hours ago	528MB
docker4harshit/calculator	33	de698e13a48a	4 hours ago	471MB
docker4harshit/calculator	32	8e5a641ecfa3	5 hours ago	471MB
docker4harshit/calculator	31	9619eaba2bbc	5 hours ago	471MB
docker4harshit/calculator	30	235ad0ce35a6	6 hours ago	471MB
docker4harshit/calculator	29	8f4e41bcfd53	8 hours ago	471MB



docker4harshit/calculator:45
DIGEST: sha256:e17da9c92b35924b9621fecadc615afdc32319ba211eae51be02d2874ecc626b

OS/ARCH
linux/amd64

COMPRESSED SIZE
229.56 MB

LAST PUSHED
14 minutes ago by docker4harshit

IMAGE LAYERS

1	ADD file ... in /	52.39 MB
2	CMD ["bash"]	0 B
3	/bin/sh -c set -eux; apt-get	4.92 MB
4	/bin/sh -c set -eux; if	10.37 MB
5	/bin/sh -c apt-get update &&	52.05 MB
6	/bin/sh -c set -eux; apt-get	5.17 MB

Command
ADD file:e8d512b08fe2ddc6f2c85831c73e4c72b9c850fa428913d19da4bb1a34f84cf2 in /

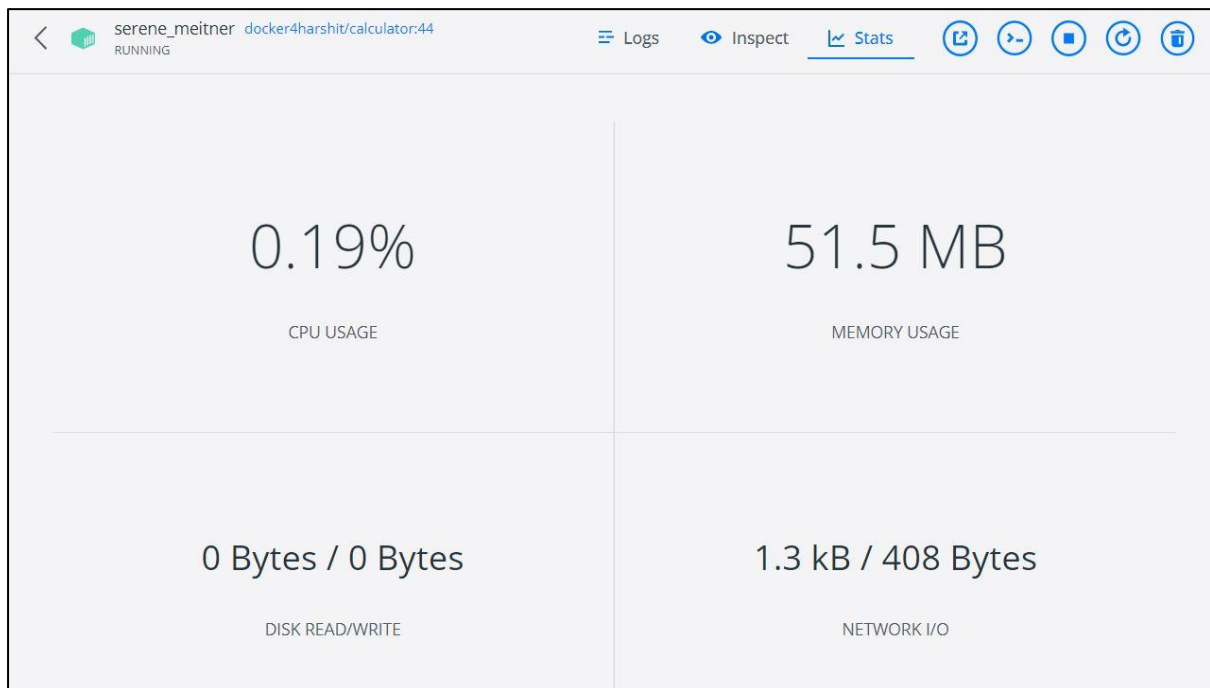
Docker image with tag:45 in repository `docker4harshit`


```
>docker run -p 8081:8080 -I docker4harshit/calculator:44
```

The above command is run in this stage which binds local port 8081 to the container port 8080 and runs the docker image `docker4harshit:44` on this port.

```
TCP    0.0.0.0:8081          Nightmare:0            LISTENING
[com.docker.backend.exe]
```

Image running on port 8081



Container started – Image running

Below screenshots describe image (our Java application) running on terminal in local machine.

```
C:\WINDOWS\system32>docker run -i docker4harshit/calculator:44
Select the appropriate option from the following list:
1. Square root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Enter you choice: 1
=====
Enter the number: 4
11:00:45.580 [main] INFO  Calculator - SQUARE_ROOT - Input:4.0 - Output:2.0
Square root of 4.0 is: 2.0
```

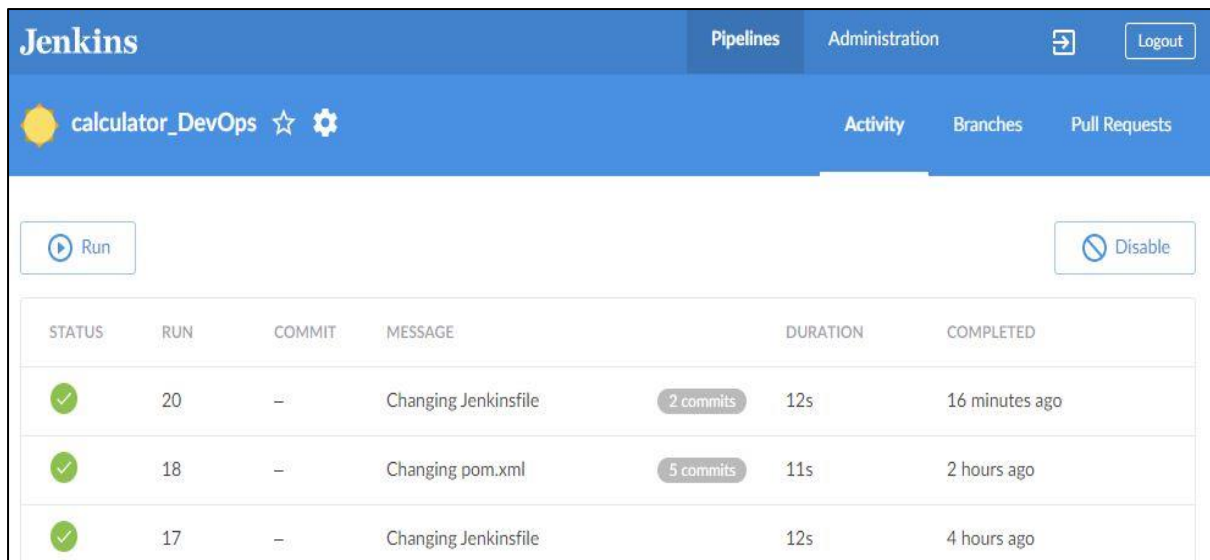
```
C:\WINDOWS\system32>docker run -i docker4harshit/calculator:44
Select the appropriate option from the following list:
1. Square root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Enter you choice: 3
=====
Enter the number: 0
11:01:17.372 [main] INFO  Calculator - LOGARITHM - Input:0.0 - Output:-Infinity
Natural Logarithmic value of 0.0 is: -Infinity
```

```
C:\WINDOWS\system32>docker run -i docker4harshit/calculator:44
Select the appropriate option from the following list:
1. Square root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Enter you choice: 2
=====
Enter the number: 4
11:01:00.561 [main] INFO  Calculator - FACTORIAL - Input:4 - Output:24
Factorial of 4 is: 24
```

```
C:\WINDOWS\system32>docker run -i docker4harshit/calculator:44
Select the appropriate option from the following list:
1. Square root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Enter you choice: 4
=====
Enter the base number: 3 4
Enter the power: 11:01:36.030 [main] INFO  Calculator - POWER_FUNCTION - Input:3.0^4.0 - Output:81.0
Power function: 3.0^4.0 = 81.0
```

Jenkins pipeline helps to make an automatic flow for the DevOps and helps us in Continuous Integration and Deployment on cloud platforms. This Jenkins job for now is manually triggered but for other SCM it can be triggered based on events such as a new push or new pull but for GitHub, you might have set up a webhook. This can be done via port forwarding; you have to make your Jenkins port 8080 open on the wide web so it can be triggered based on the event. For now, we will skip it and we will build it manually.

Pipelines and detailed pipeline steps can be viewed in BlueOcean dashboard. BlueOcean tab appears in the left sided taskbar on Jenkins dashboard.



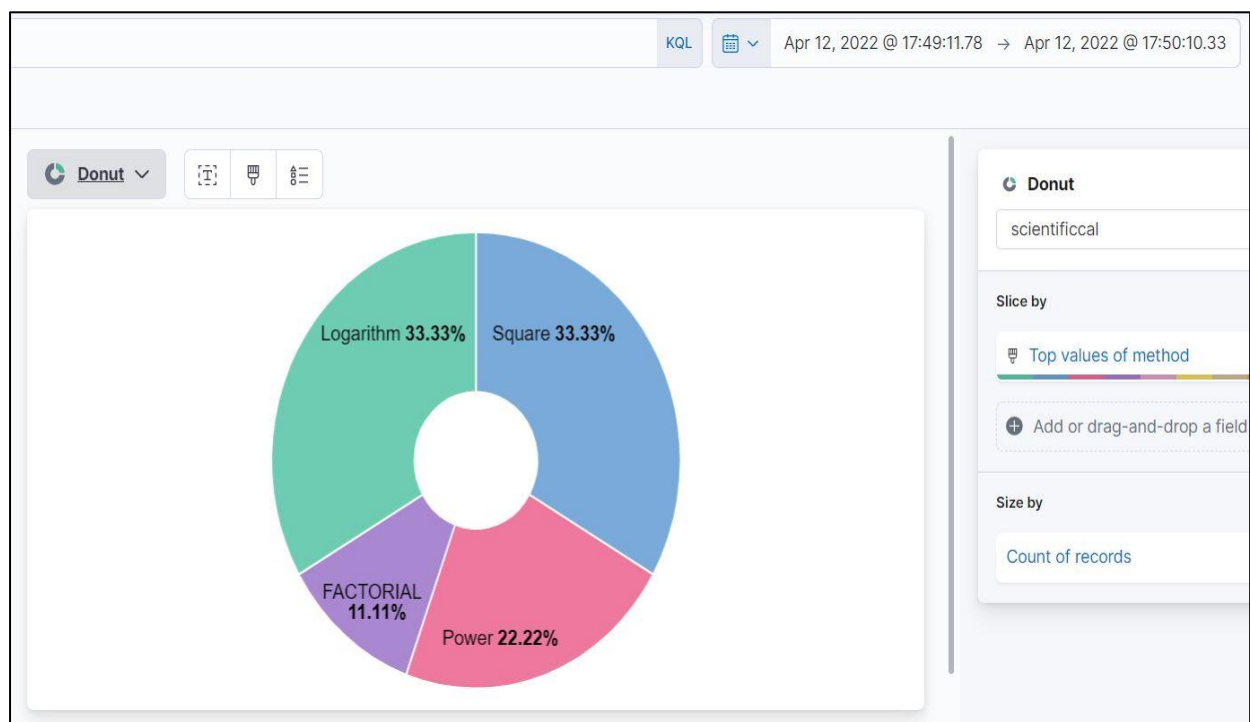
The screenshot shows the Jenkins BlueOcean dashboard. At the top, there's a blue header with the Jenkins logo, the name 'calculator_DevOps', and navigation tabs for 'Pipelines', 'Administration', 'Activity', 'Branches', and 'Pull Requests'. Below the header, there are 'Run' and 'Disable' buttons. The main content is a table of pipeline runs.

STATUS	RUN	COMMIT	MESSAGE	DURATION	COMPLETED
✓	20	–	Changing Jenkinsfile	2 commits 12s	16 minutes ago
✓	18	–	Changing pom.xml	5 commits 11s	2 hours ago
✓	17	–	Changing Jenkinsfile	12s	4 hours ago

BlueOcean pipeline plugin

7. Continuous Monitoring – ELK

The ELK Stack is a log management platform – collection of three open-source products – Elasticsearch, Logstash and Kibana- from Elastic.



Problems faced while working on the project

- 1) Docker within a docker official image doesn't support SSH. I got below error with all the ways I tried out.

[ssh-agent] Could not find a suitable ssh-agent provider.

To resolve this issue, I had option of running docker image locally or change docker image build of the code over Ubuntu docker image and run our created docker image on it.

- 2) Docker run command wasn't working until I discovered that it has to be used with `-i` options to work for applications with user inputs so that image is run in interactive mode.
- 3) pom.xml needed multiple updated dependencies and plugins to work correctly.

My GitHub Repository:

https://github.com/NightmareNight-em/devops_calculator

My DockerHub Repository:

<https://hub.docker.com/r/docker4harshit/calculator>

THANK YOU.