CS 816 – Software Production Engineering

Mini Project

Under the guidance of Prof. B Thangaraju

By,

Hemant Kumar

Problem Statement :

Create a scientific calculator program with the following user menu driven operations:

- Square root function - $\sqrt{x}$
- Factorial function - x!
- Natural logarithm (base e) - ln(x)
- Power function - x^n

What is DevOps & Why is it important ?

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

Software and the Internet have transformed the world and its industries, from shopping to entertainment to banking. Software no longer merely supports a business; rather it becomes an integral component of every part of a business. Companies interact with their customers through software delivered as online services or applications and on all sorts of devices. They also use software to increase operational efficiencies by transforming every part of the value chain, such as logistics, communications, and operations. In a similar way that physical goods companies transformed how they design, build, and deliver products using industrial automation throughout the 20th century, companies in today's world must transform how they build and deliver software.

Tools Used :

- **Source Code Management Tool** - Git, GitHub
- **Build Tool** - Maven
- **Test Library** - JUnit 5
- **Continuous Integration Tool** - Jenkins
- **Containerization Tool** - Docker, DockerHub
- **Configuration Management & Continuous Deployment Tool** - Ansible

Steps Followed :

Step 1.) Installation of Tools :

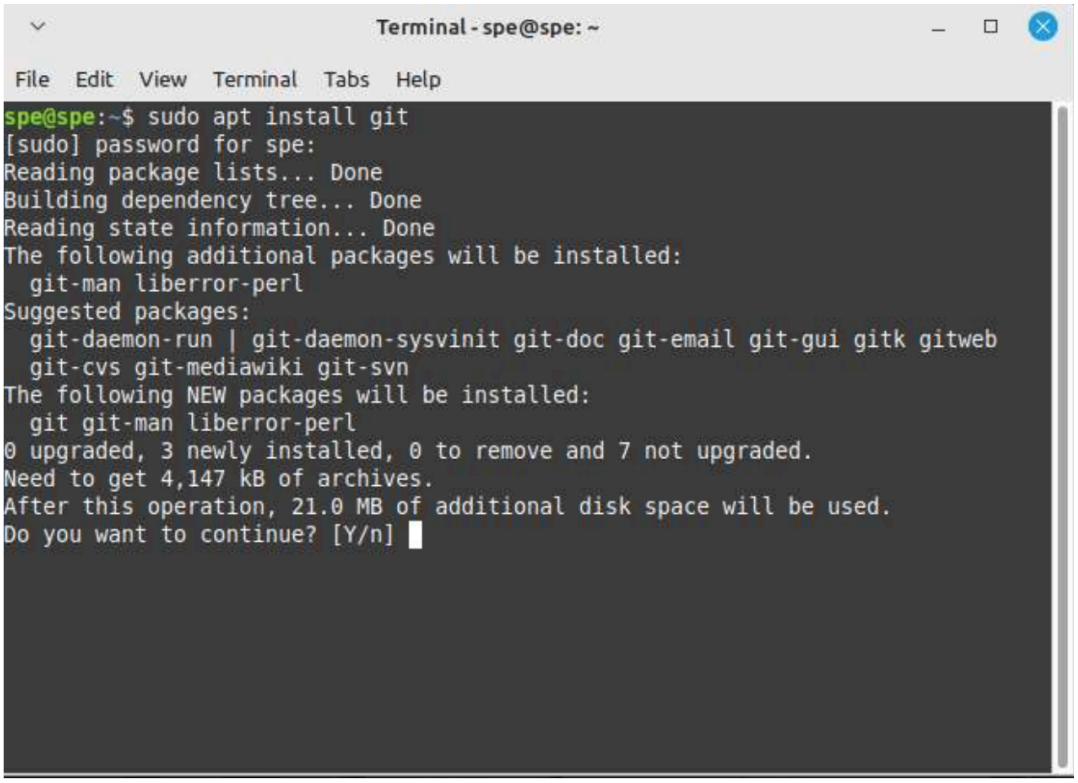
- Source Code Management (Git and GitHub) :
  Source code management is the practice of tracking modifications to source code. Keeping a running history of the changes made to a codebase helps programmers, developers and testers ensure that they're always working with accurate and up-to-date code and helps resolve conflicts when merging code from multiple sources.

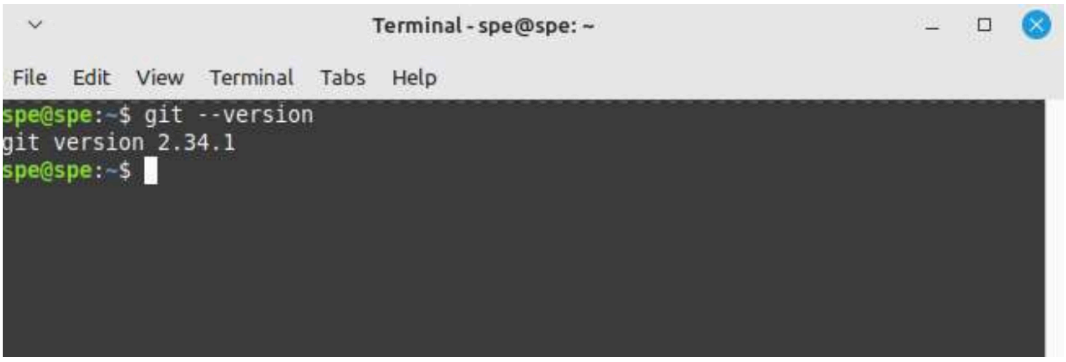
  Git is the version system that I have used in this project.

  To install Git : type "sudo apt install git"

  Verification : After the installation completes, you can verify by typing "git –version". It should show you the git version. If

you receive an error, then it means that the installation was not successful.





- Continuous Integration Tool (Jenkins) :

Jenkins is an open-source automation tool written in Java with plugins built for continuous integration. Jenkins is used to

build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

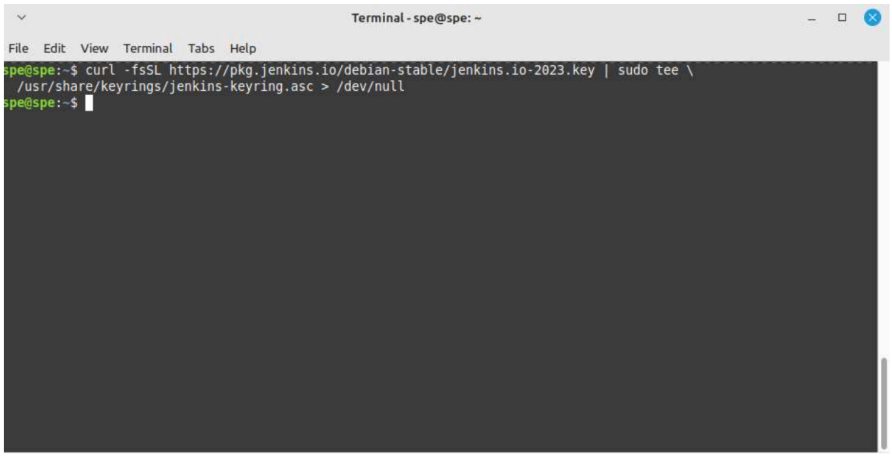
With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example Git, Maven 2 project, Amazon EC2, HTML publisher etc.

Jenkins will be managing the CI/CD pipeline for us

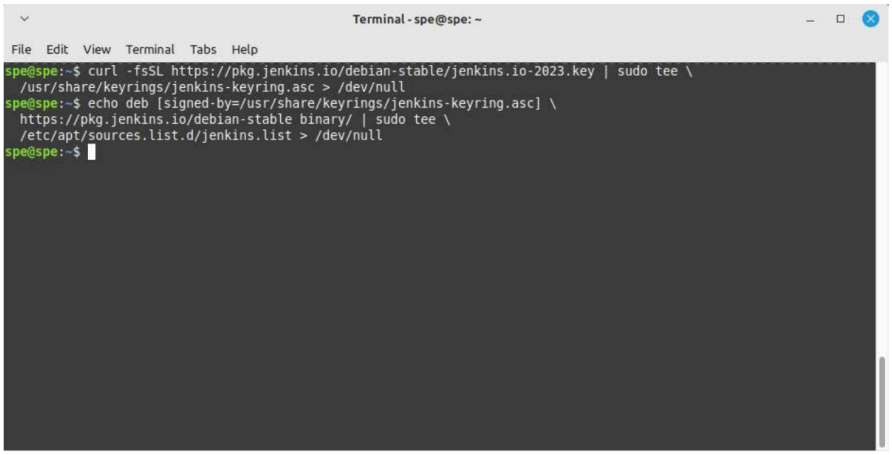
To install Jenkins, type the following commands :

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-
2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

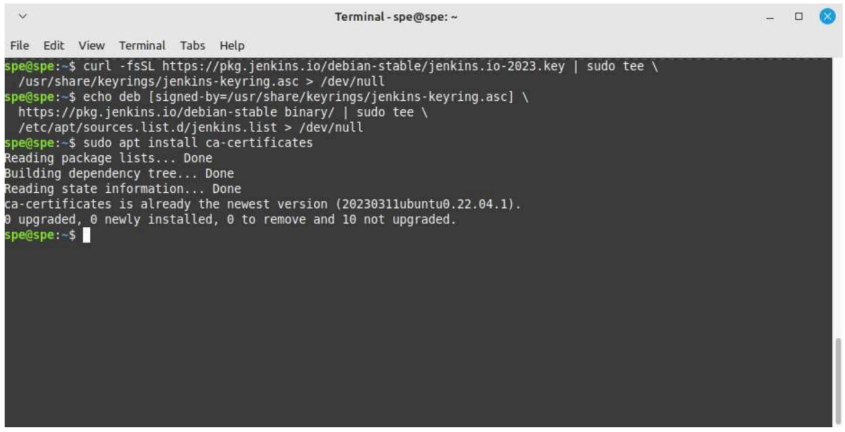


The above command fetches the certificates for Jenkins.

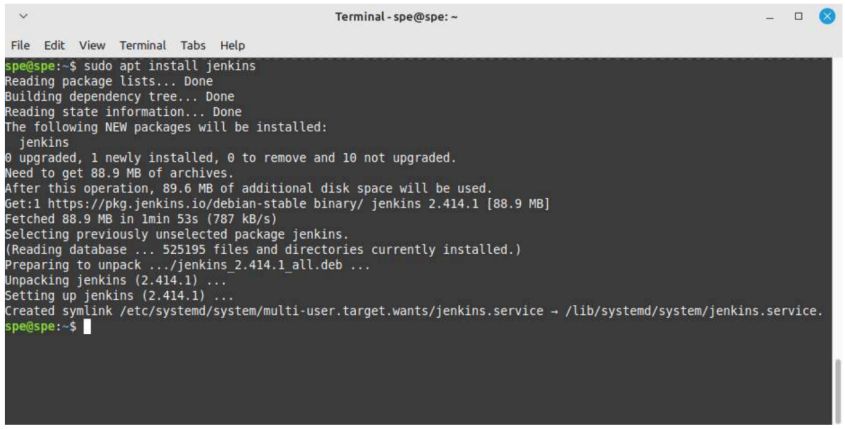
```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```




```
sudo apt install ca-certificates

sudo apt update

sudo apt install jenkins
```



The above commands refreshes the certificates and installs Jenkins.



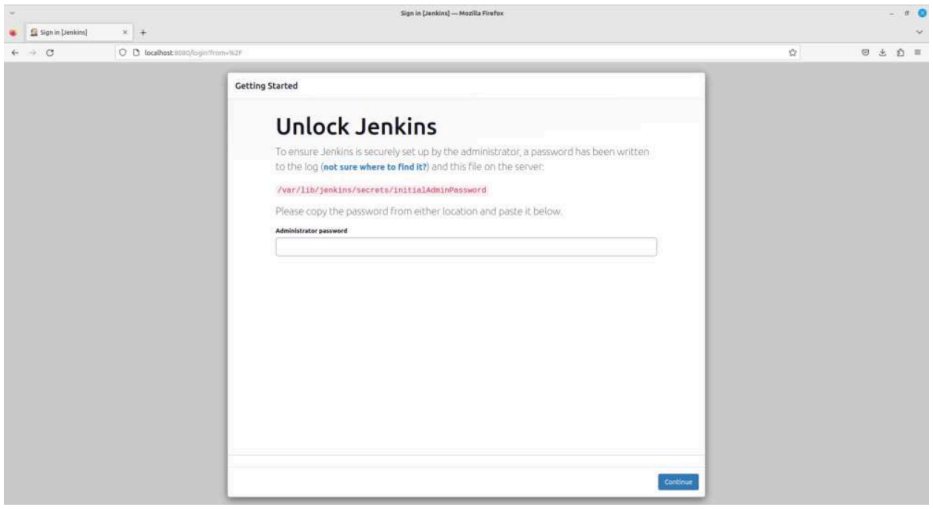
Once you have installed it, we can start it and check its status by typing the following commands :

sudo systemctl start jenkins

sudo systemctl status jenkins

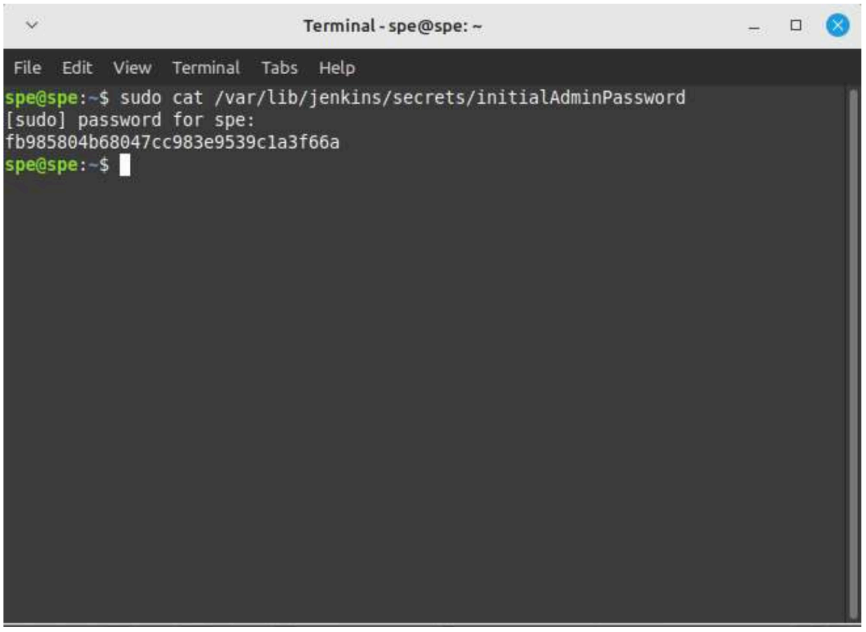
Once we have Jenkins up and running, we will need to configure it. We start by going to port 8080 on the localhost. The URL is as follows : https://localhost:8080

When you visit the URL for the first time, then Jenkins will ask for the password that has been written to the log file :

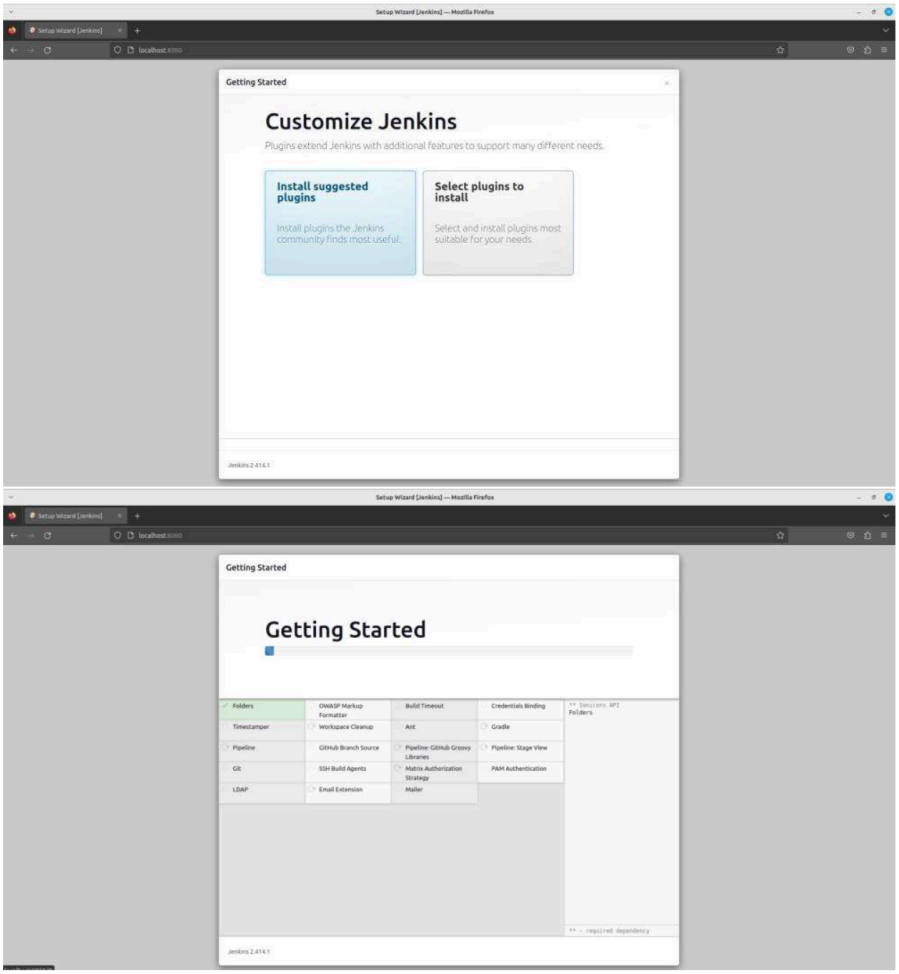


To get this password, we will run the following command :

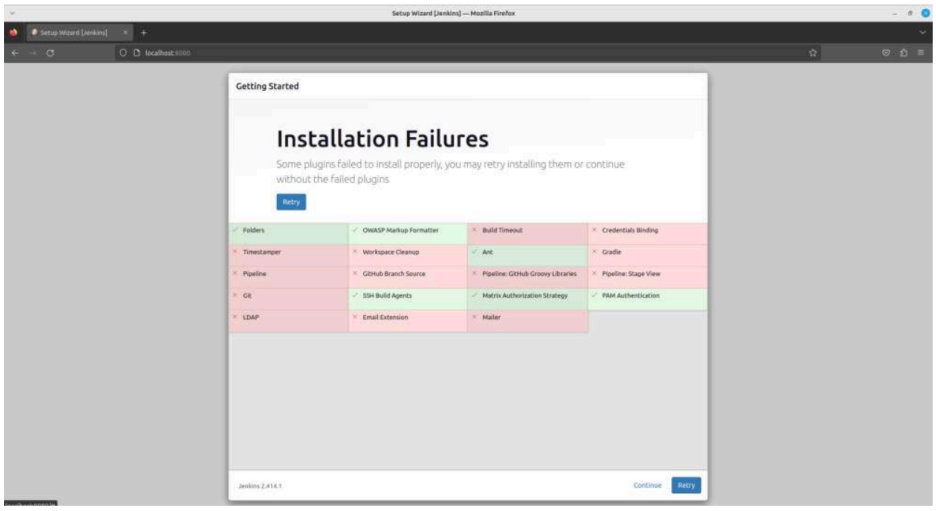
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```



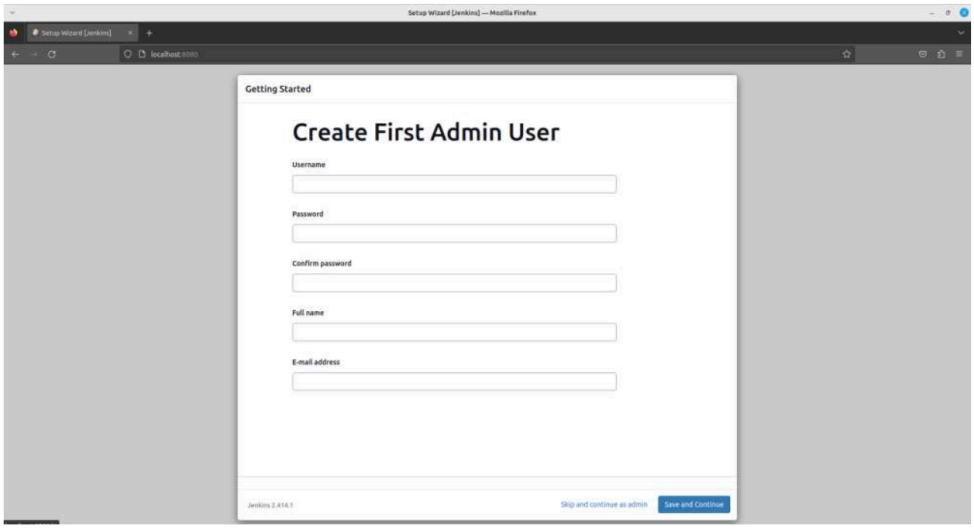
Copy and paste the password that shows on the terminal. Once we do this, Jenkins will ask us for what plugins we want to have installed. We are going to choose the suggested plugins.



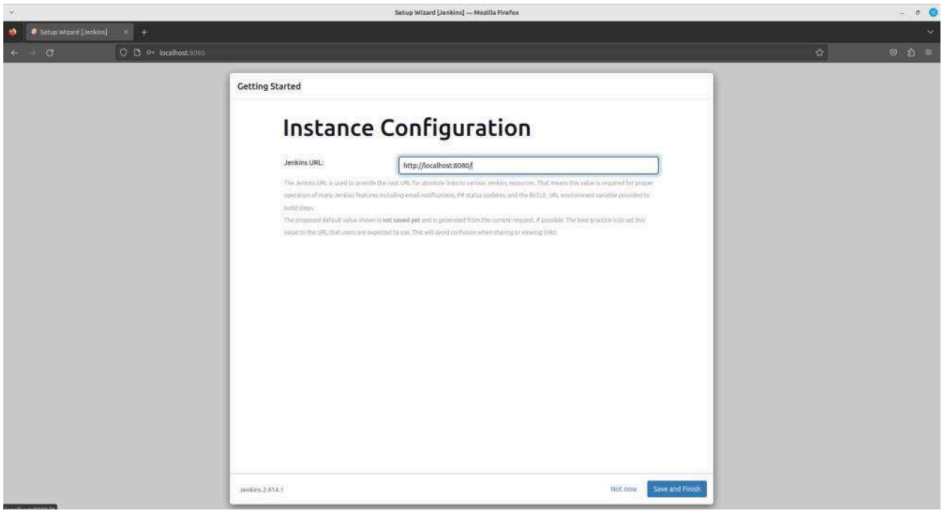
One of the known issues that many face is the failure to install plugins.



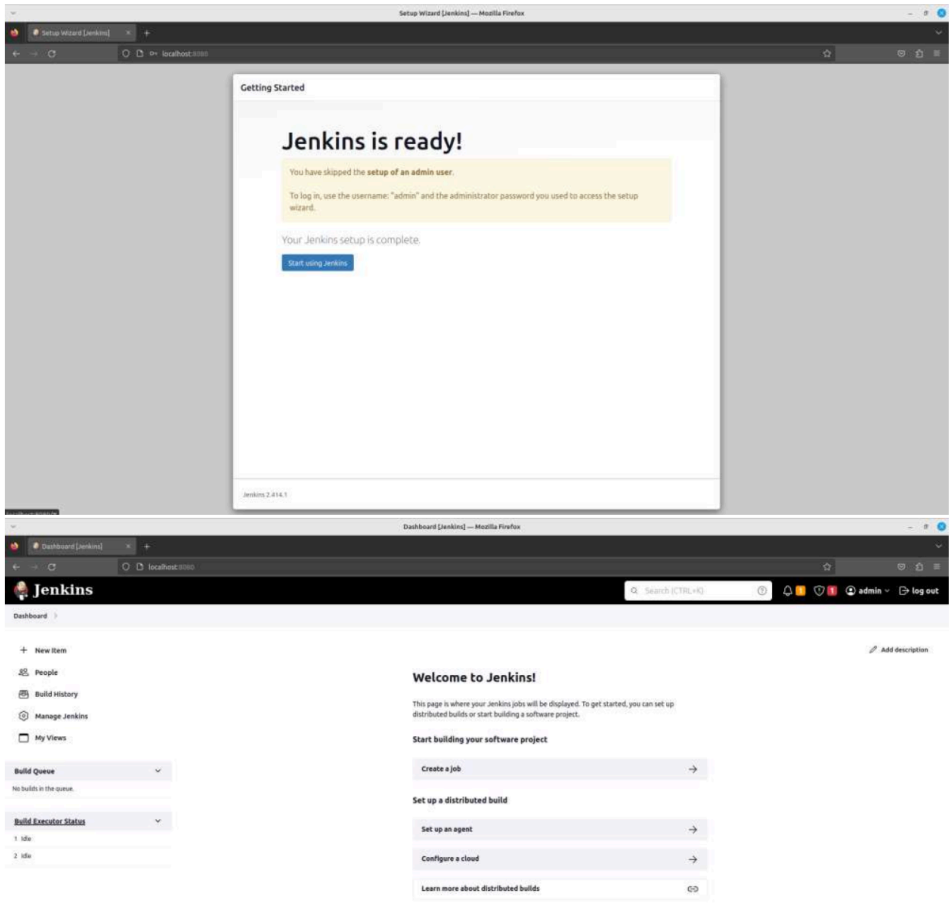
In such cases, keep retrying a few times as many will resolve after a few attempts. Once the errors resolve, the next step is the creation of Jenkins First User creation. You have the option to skip it.We will skip and continue as Jenkins Admin User.



The next step is the instance configuration. Here we keep the default configuration which is to have Jenkins on localhost.



We then click on save and finish. We are now good to go.



- Containerization Tool (Docker, DockerHub) :

  Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

To install docker we are going to first ensure that curl is installed in the system. We do this by typing the command -

```
sudo apt install curl
```

Once we do that, we are going to use curl to get script needed to install docker using the following command -

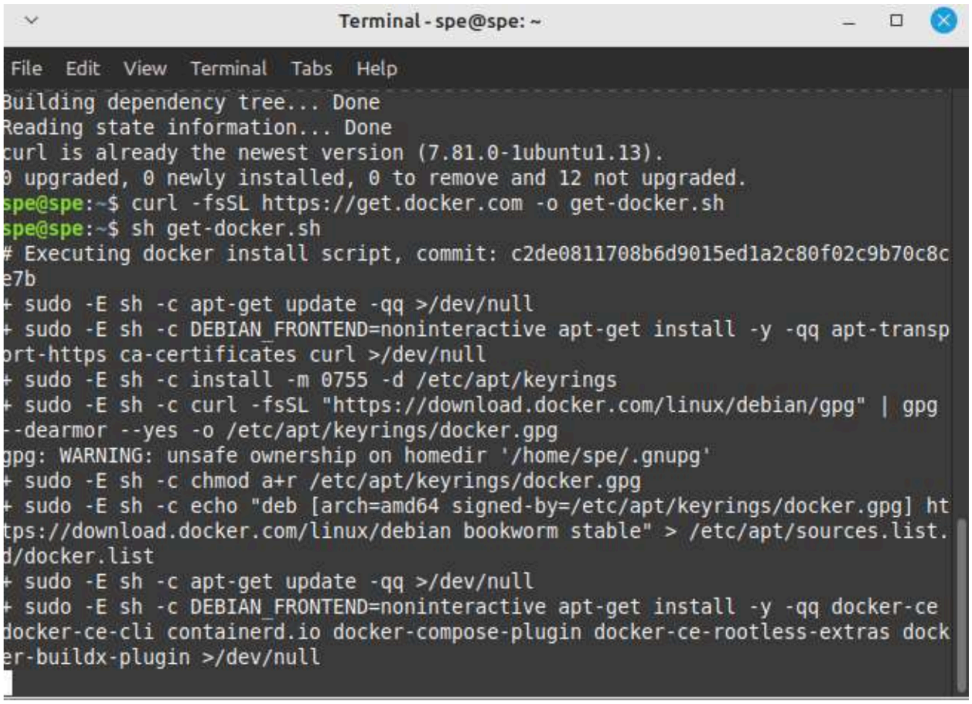
```
curl -fsSL https://get.docker.com -o get-docker.sh
```

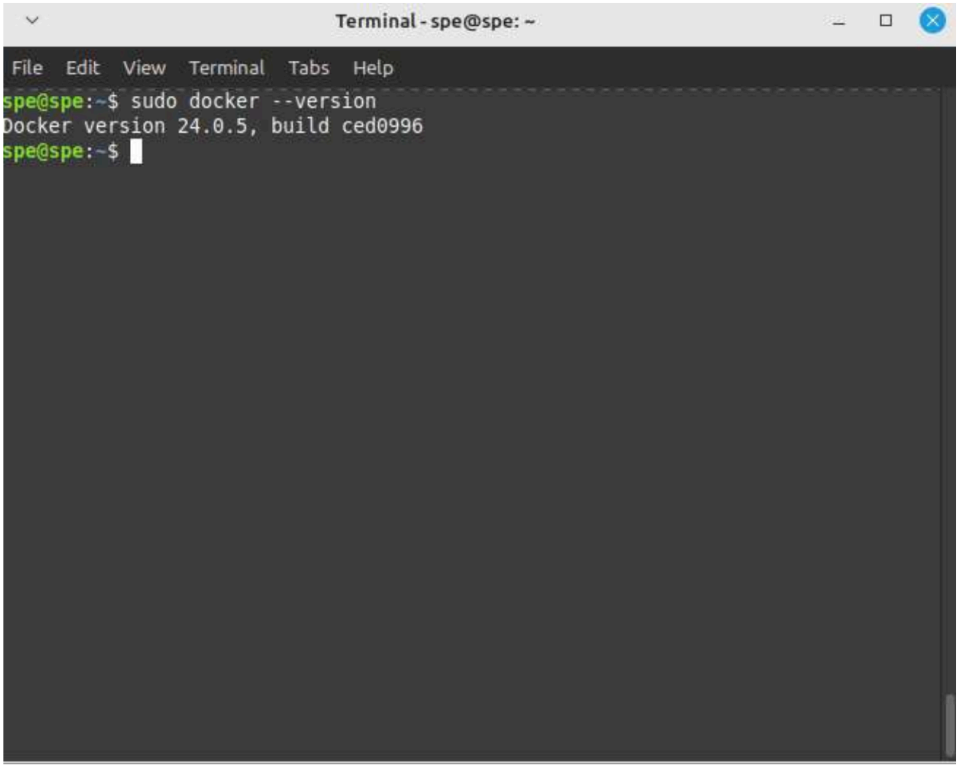
Once we have the script, we are going to run the script by typing the following command.

```
sh get-docker.sh
```

We can verify that Docker is installed by typing the following command.

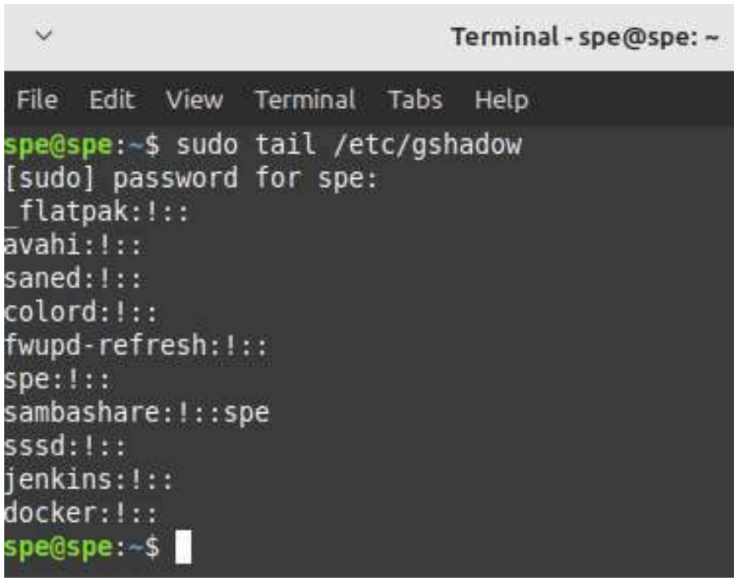
```
sudo docker --version
```





Once Docker is installed, we need to make sure that Docker is in the same user group as Jenkins. We can check it by using the following command.

```
sudo tail /etc/gshadow
```



As you can see, both Jenkins and Docker are not in the same line in the list. We then run the following commands to add it.

sudo usermod -aG docker jenkins

sudo systemctl restart jenkins

After running these commands, try running sudo tail /etc/gshadow to verify that both Jenkins and Docker are in the same line in the list.

Creating an account in DockerHub :

Once we create Docker images, we will want to store them remotely so that they can later be pulled from the remote location to be deployed anywhere. To create an account, visit: https://dockerhub.com and follow the signup procedure to create an account. Remember the username and password as we will need those credentials to be able to push and pull images from DockerHub.

- Continuous Deployment Tool (Ansible) :

Ansible is a software tool that provides simple but powerful automation for cross-platform computer support. It is primarily intended for IT professionals, who use it for application deployment, updates on workstations and servers, cloud provisioning, configuration management, intra-service orchestration, and nearly anything a systems administrator

does on a weekly or daily basis. Ansible doesn't depend on agent software and has no additional security infrastructure, so it's easy to deploy.

To install Ansible, we need to add its repository using the following command.

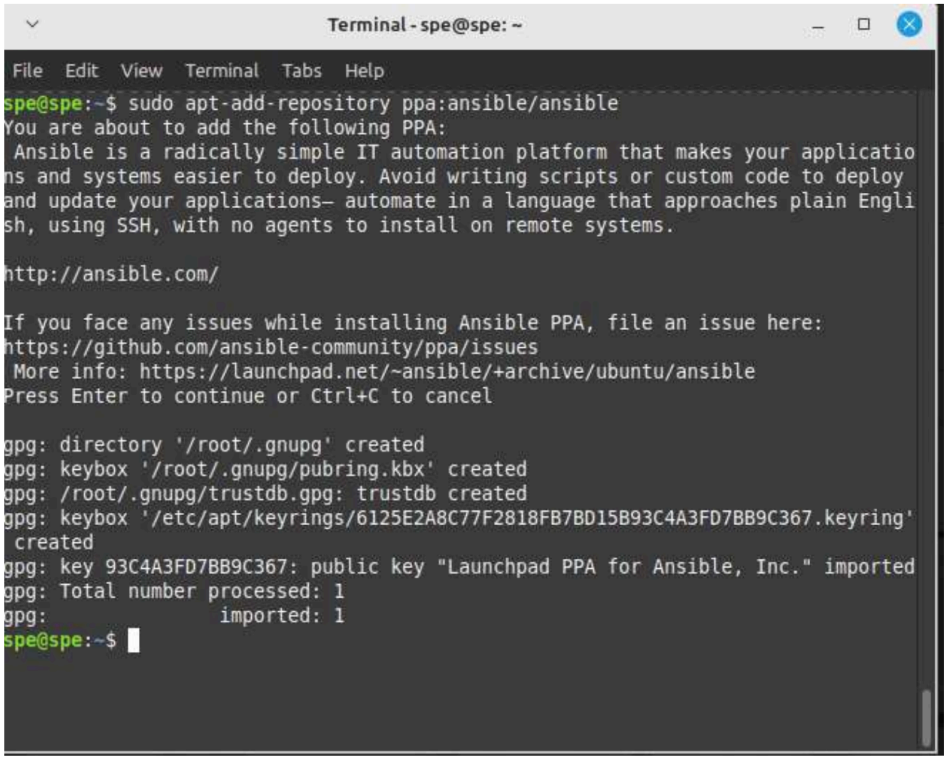
sudo apt-add repository ppa:ansible/ansible

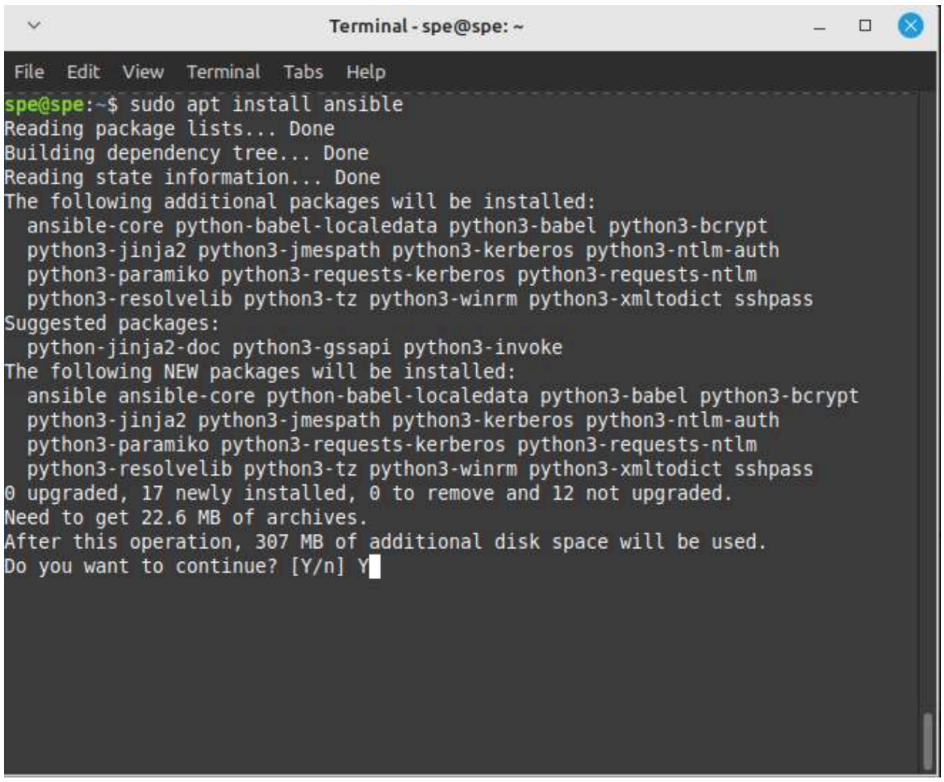
Then we are going to perform apt update to refresh the links that we from the repository :

sudo apt update

Finally, we are going to run the apt install command :

sudo apt install ansible

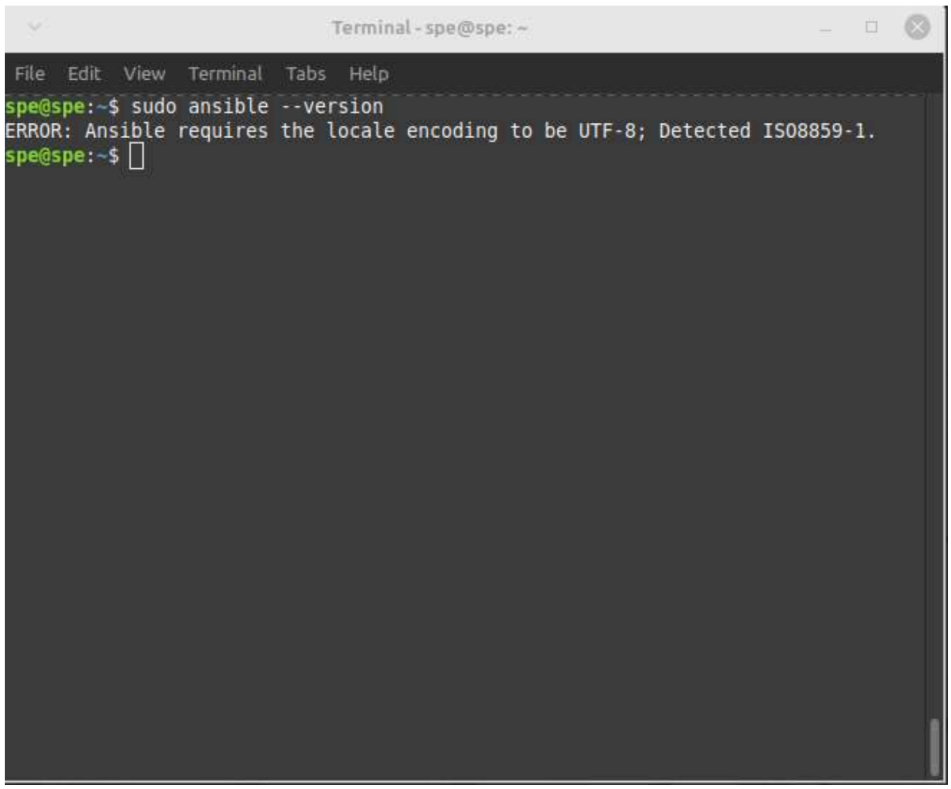




Once we install Ansible, we can check its version by running the following command.

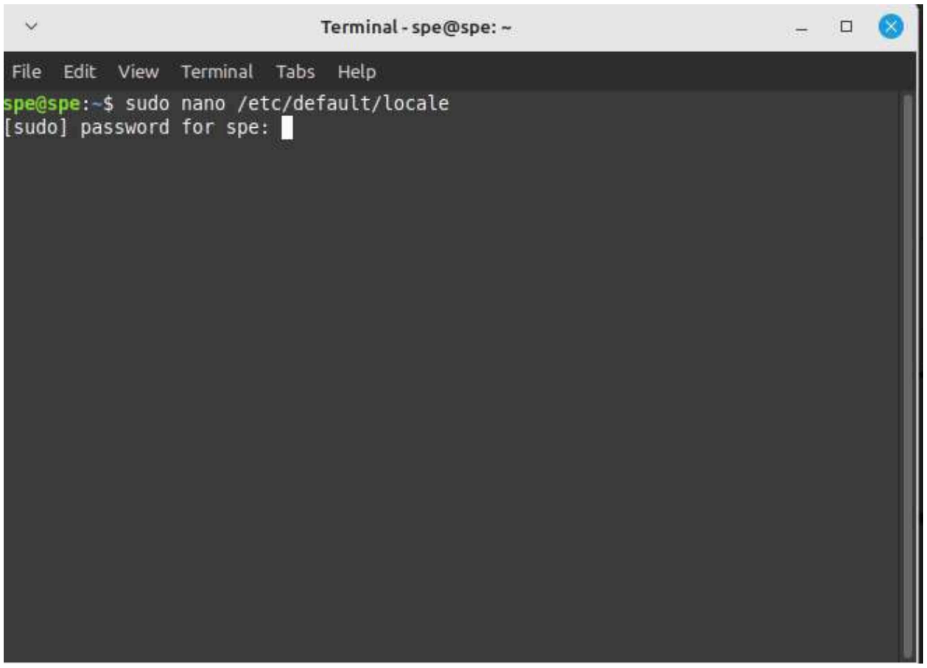
sudo ansible -- version

However, a common error that pops up is the following :



To address this, we are going to change the locale encoding using the following command.

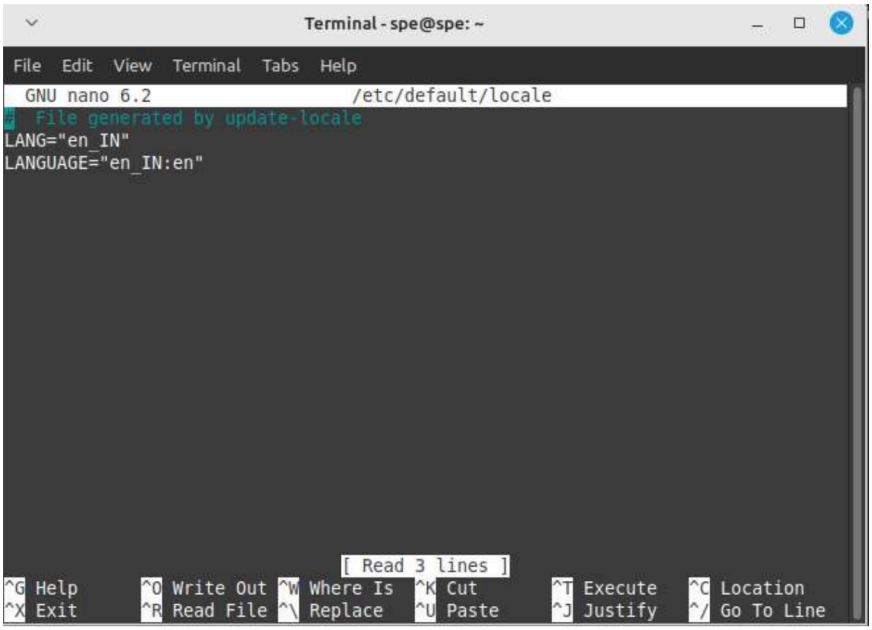
sudo nano /etc/default/locale



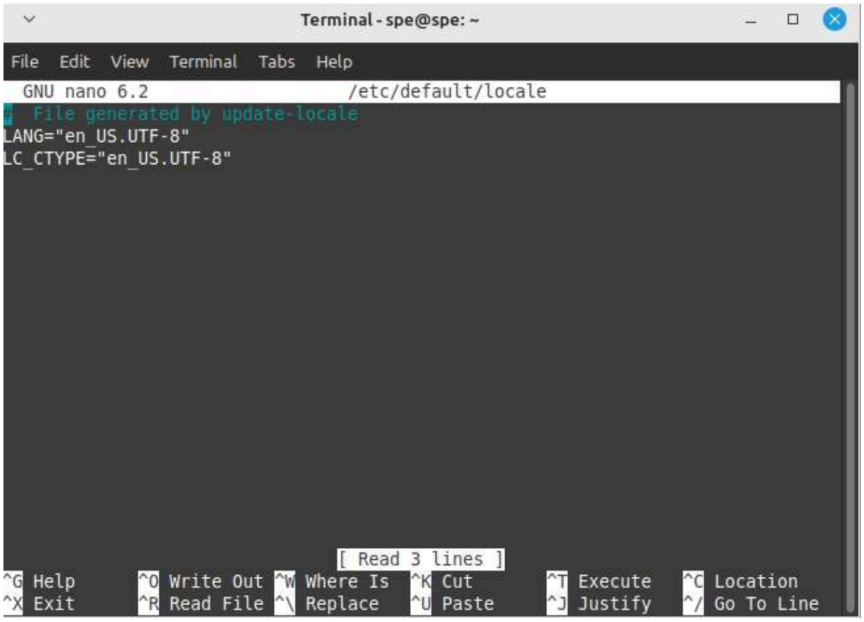
This will open the locale file in the terminal. We are going to replace LANG and LC_TYPE values as follows:

```
LANG="en_US.UTF-8"

LC_CTYPE="en_US.UTF-8"
```

Before:



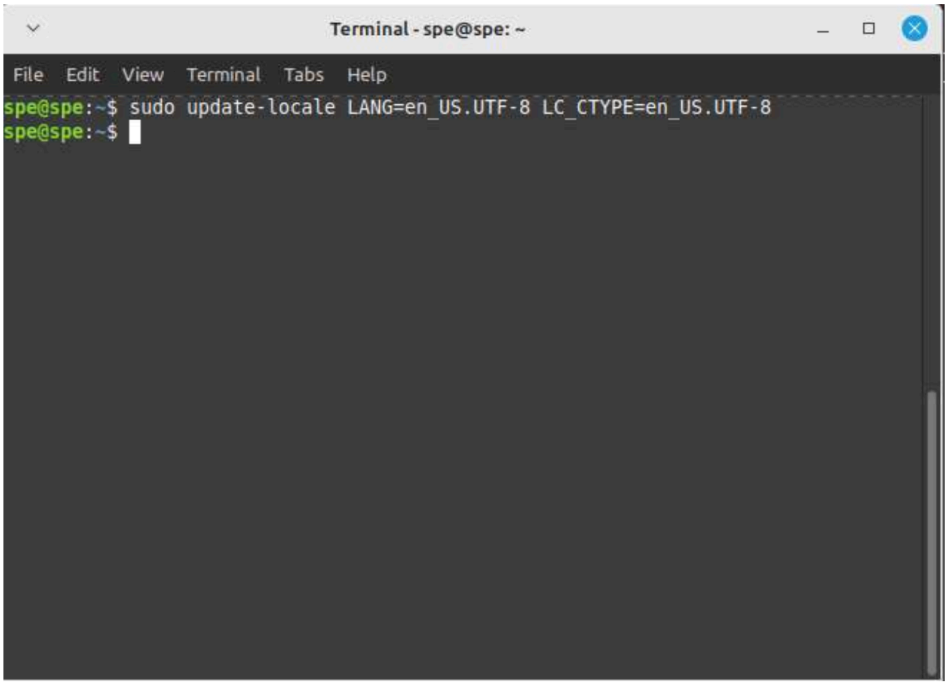
After:



Once we replace it, we perform Ctrl+O and press Enter to write the changes to the file and save it. Then we press Ctrl+X to exit.

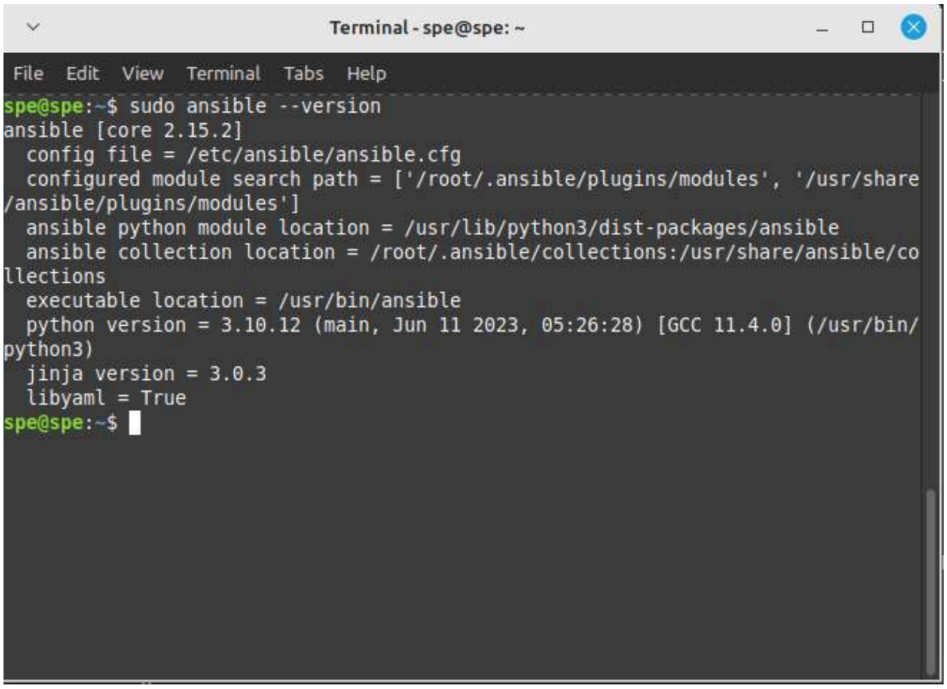
Then we need to run the following command.

sudo update-locale LANG=en_US.UTF-8 LC_CTYPE=en_US.UTF-8



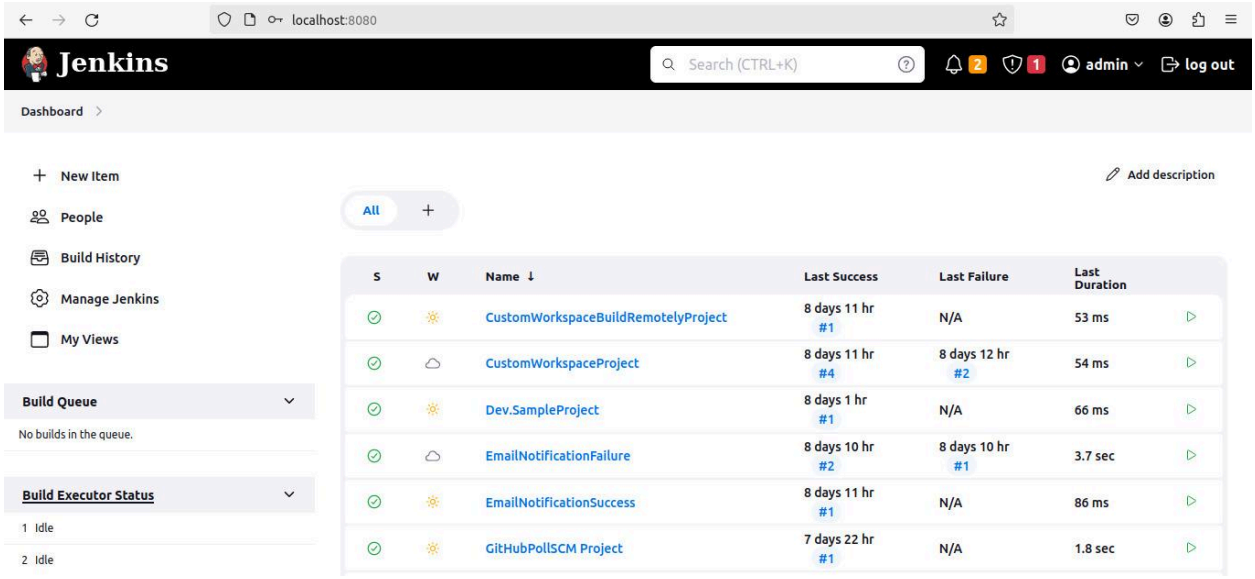
Once the command is executed, we are going to restart our system to let the changes take effect. Then we try to run the command to check for ansible version. We get the following output :

```
sudo ansible --version
```

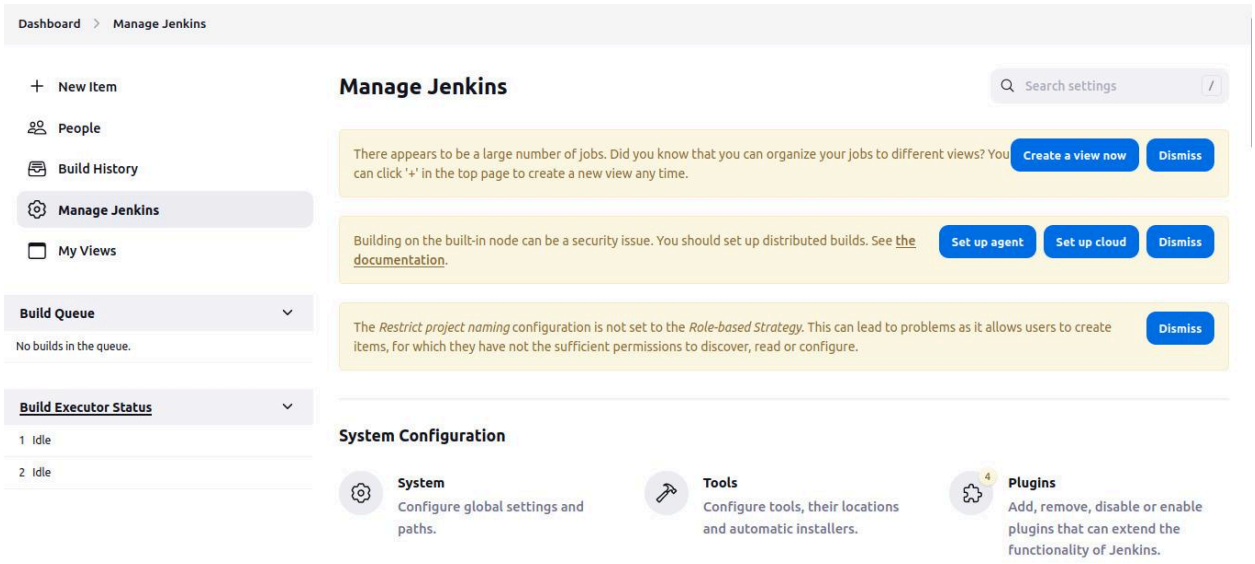


Step 2.) Installing Jenkins Plugins and Creating Credentials in Jenkins

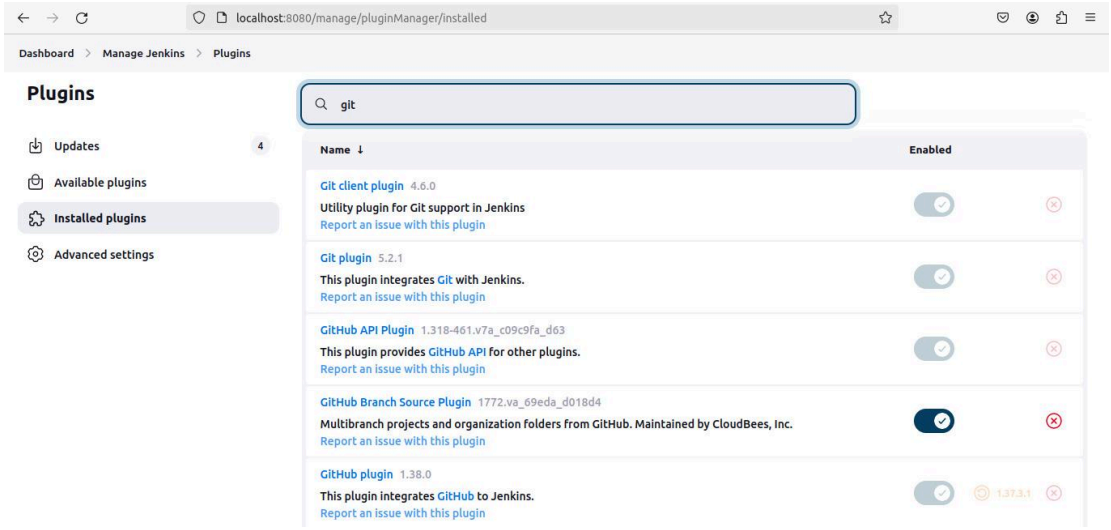
Make sure you have Jenkins up and running. You can verify by typing **sudo systemctl status jenkins.** If it is not up, then we must type **sudo systemctl start jenkins.** Now, go to http://localhost:8080. Type in the username and password for the admin user and login. You will be presented with the following dashboard page.

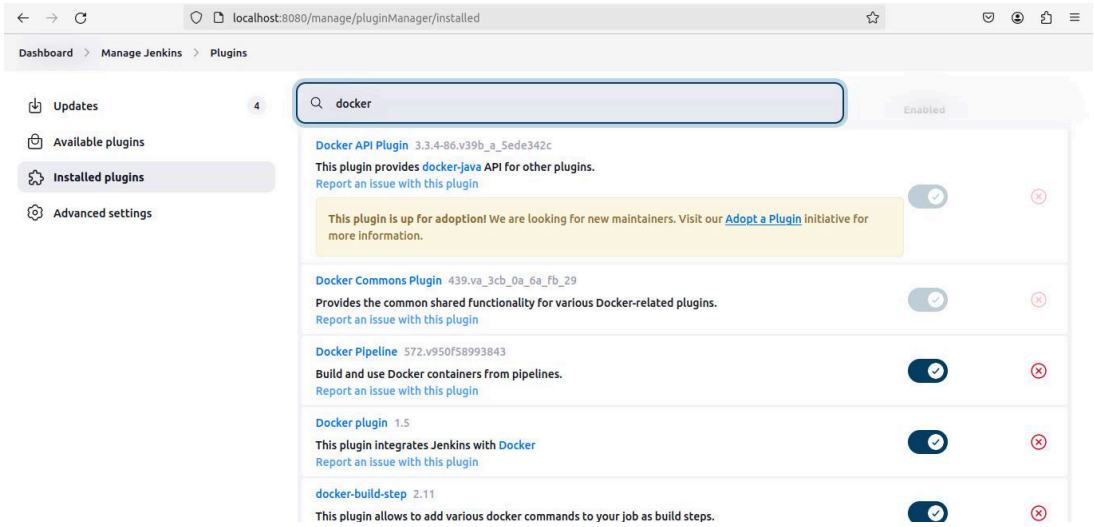


To install the necessary plugins, go to **Manage Jenkins**

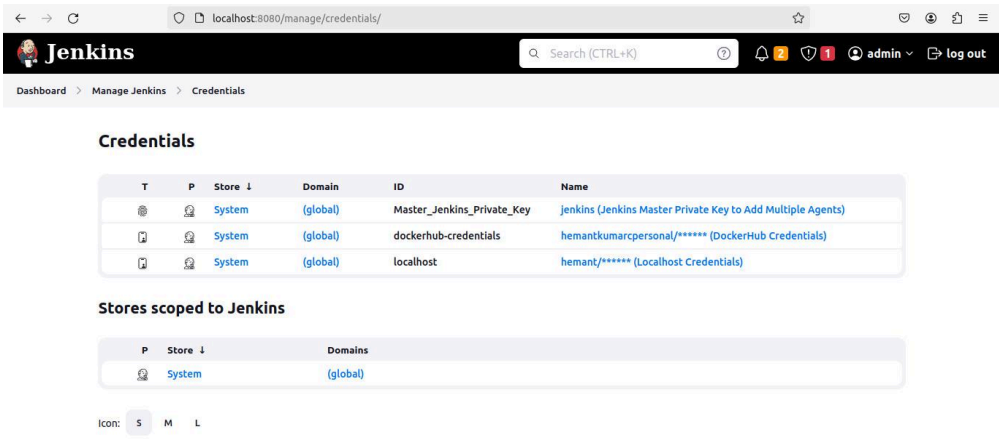


Now go to **Plugins** and install the following plugins for git and docker.

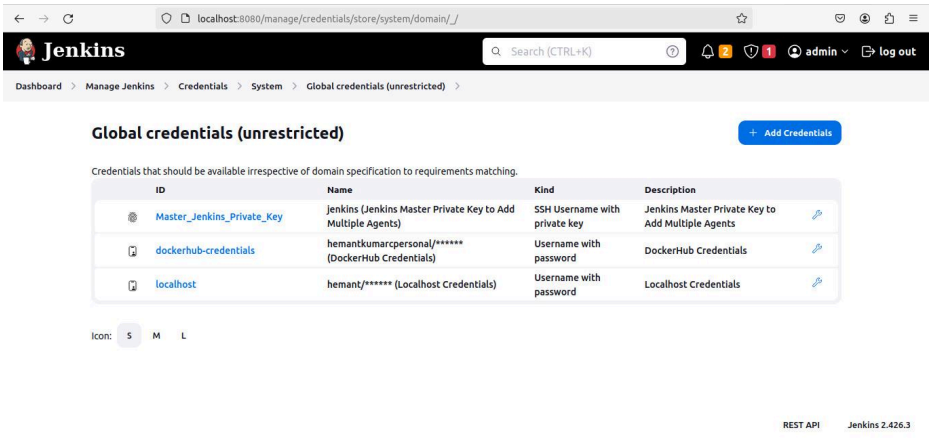




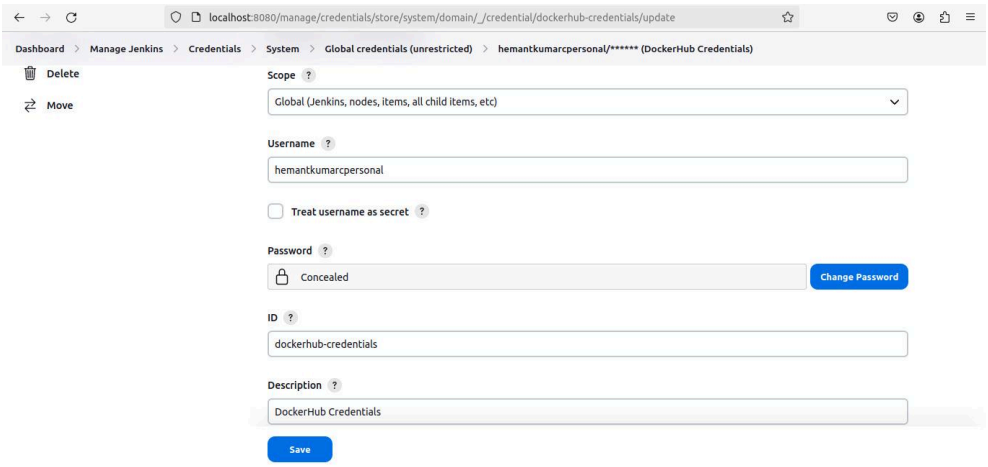
To create the necessary credentials, go to **Manage Jenkins** and go to **Credentials**.



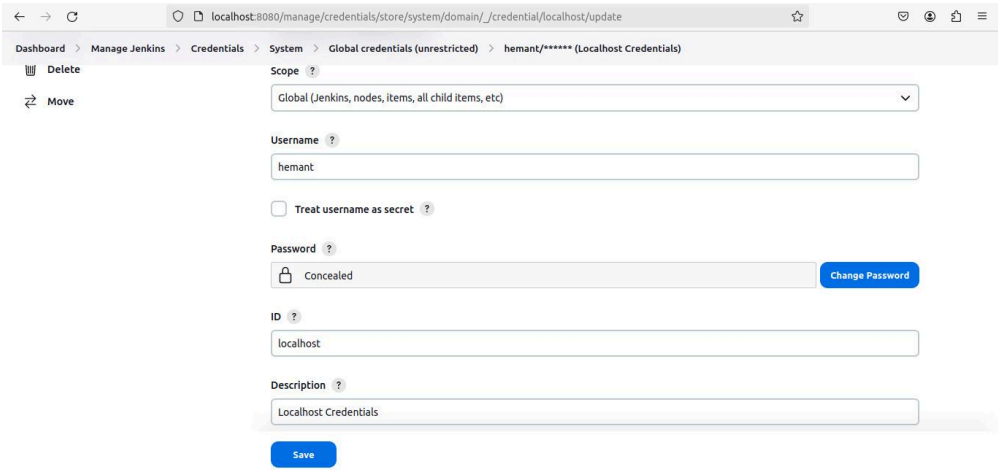
Now, click on **global** under **Stores scoped to Jenkins**. You will be presented with the following window where you can add credentials.



Now click on **Add Credentials** to add DockerHub Credentials. We are presented with the following page. We must populate the fields as given in the page below.



Follow a similar procedure for adding credentials for Ansible in Jenkins. It must look like the following.



Step 3.) Creating and Building the Project

Create a Maven project structure by using any IDE of your choice. Here, I have used IntelliJ Ultimate IDE

src/main/java/org/example/Main.java - Source Code

src/test/java/org/example/MainTest.java - Unit Tests

pom.xml - Project Dependencies

In **Main.java**,

I have created a calculator program that performs 4 functions - factorial, natural logarithm, powers and square root. You can write any application of your choice.

In **MainTest.java**,

I have created all the JUnit tests for my application.

In **pom.xml**,

I have added a manifest attribute, that lets me create a jar file that is runnable by java. Also, I have added dependencies such as JUnit Jupiter API and Mockito to my project.

Jenkinsfile

Jenkinsfile is a Declarative Pipeline script for Jenkins, which orchestrates the build and deployment process.

```
pipeline {

    agent any

    tools {
      maven 'maven_3_9_6'
    }

    environment {
        DOCKER_IMAGE_NAME = 'calculator'
        GITHUB_REPO_URL = 'https://github.com/hemantkumarc/SPE_MiniProject.git'
    }

    stages {

        stage('Clone Project')
        {
            steps {

                script{
                    git branch: 'main', url: "${GITHUB_REPO_URL}"
                }

            }
        }

        stage('Build Code')
        {
            steps {
                sh "mvn clean install"
            }
        }


        stage('Test Code')
        {
            steps {
                sh "mvn test"
            }
        }

        stage('Building Docker Image')
        {
            steps {

                script {
                    docker.build("${DOCKER_IMAGE_NAME}", '.')
                }

            }
        }

        stage('Pushing Docker Image')
        {
            steps {

                script {

                    docker.withRegistry('', 'dockerhub-credentials') {

                        sh "docker image tag calculator hemantkumarcpersonal/calculator:latest"
                        sh "docker push hemantkumarcpersonal/calculator:latest"

                    }
                }

            }
        }

        stage('Run Ansible Playbook')
        {
            steps {

                script {

                    ansiblePlaybook (

                        playbook: 'playbook.yml',
                        inventory: 'inventory'
```

Jenkins provides the ability to perform distributed builds by delegating them to “agent” nodes. Doing this allows you to execute several projects with only one instance of the Jenkins server, while the workload is distributed to its agents.

Specifying **agent any** means that Jenkins will run the job on any of the available nodes.

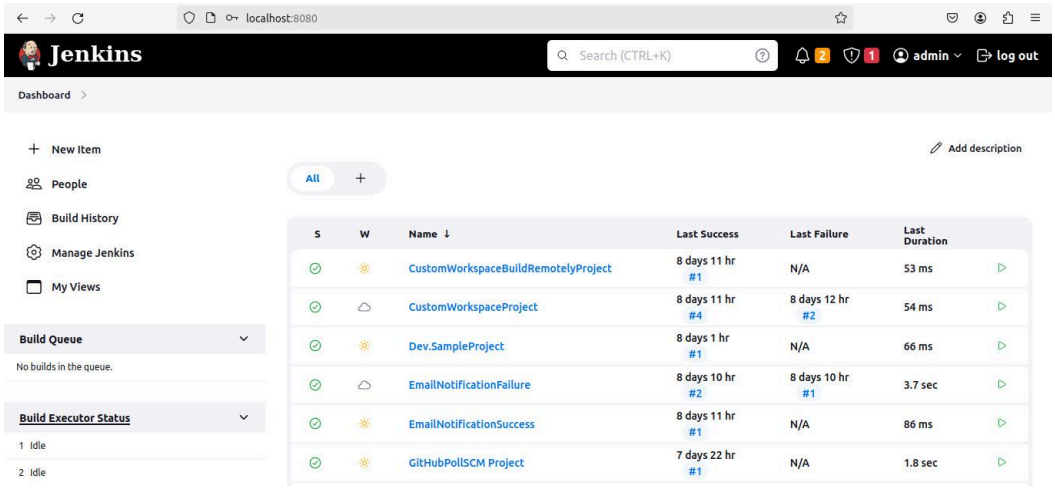
Then we configure the **tools** that we are going to use in the pipeline script.

In order for the pipeline to be able to build maven projects and perform JUnit tests, we install maven directly as part of the Jenkins server.

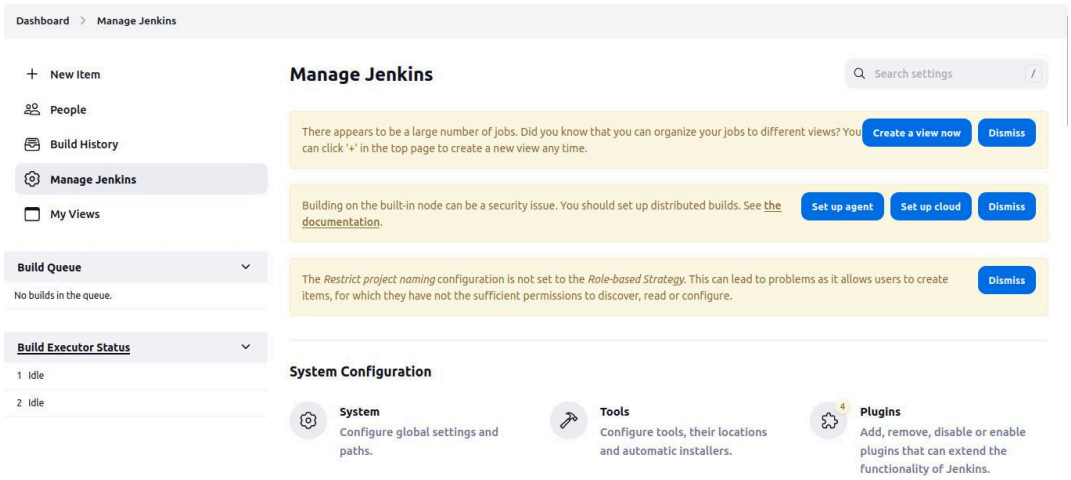
For this, we perform the following steps. Before we start, make sure that Jenkins is up and running. For this type,

sudo systemctl start jenkins

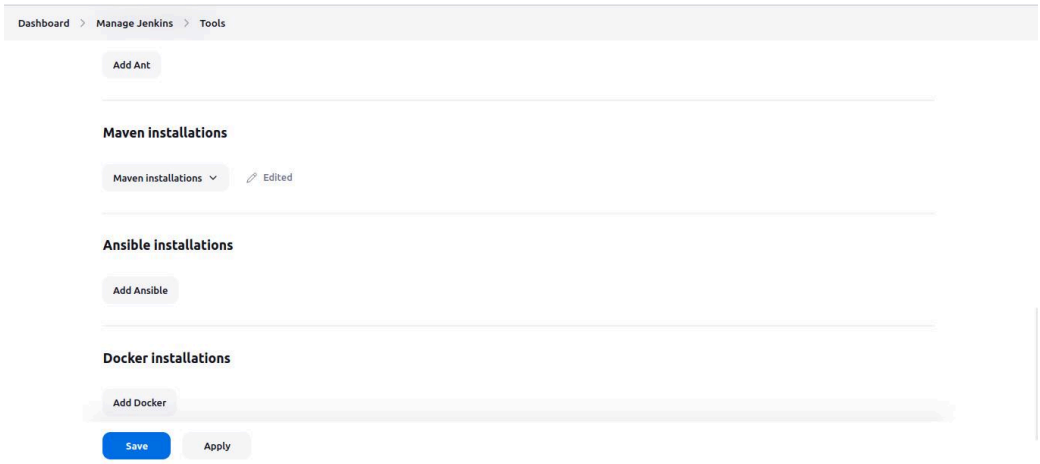
Go to https://localhost:8080. Type in the username and password for the Jenkins admin user. When the login is successful, you will be presented with the dashboard page.



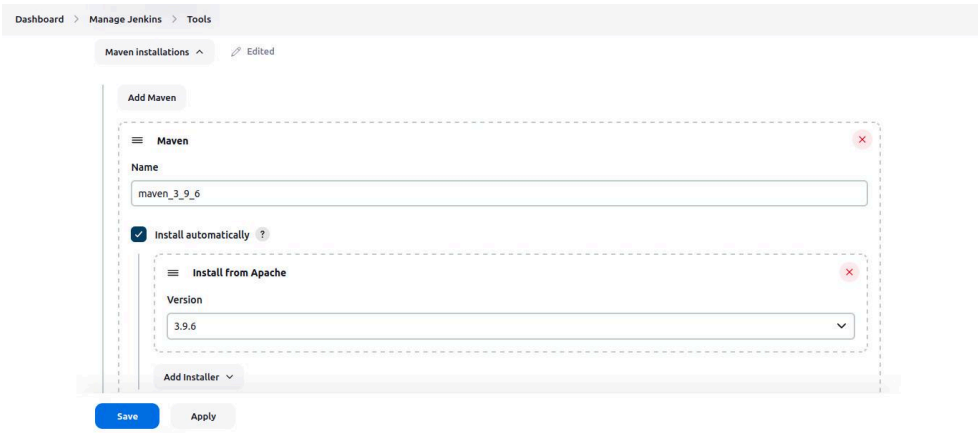
Then go to **Manage Jenkins**,



Then go to **Tools**, and scroll down to **Maven Installations**



Click on **Add Maven**, and give it an appropriate **Name** and select the **Version** of Maven that you want to install from the Apache repository.



Then click on **Apply** and **Save**. This will save the settings for later use.

Coming to the pipeline script, we then specify the tool that we are going to use as part of this pipeline, here **maven** and specify the **name** that we earlier configured in Jenkins.

Now as part of **environment**, we define environment variables used through the pipeline.

DOCKER_IMAGE_NAME is set to 'calculator' and **GITHUB_REPO_URL** is set to the GitHub repository URL.

The **stages** block contains the individual stages of the pipeline. Each stage represents a phase in the build and deployment process.

The first stage is '**Clone Project**', which uses the Git plugin to clone the code from the specified GitHub repository (GITHUB_REPO_URL) and branch ('**main**')

The second stage is '**Build Code**', which uses the maven installed locally as part of the Jenkins server, to clean the project and remove all the files generated by the previous build and deploy the packaged JAR file under the target folder.

The third stage is '**Test Code**', which runs JUnit tests.

The fourth stage is '**Building Docker Image**', where a Docker image named 'calculator' is built from the code in the current directory (.)

The fifth stage is '**Pushing Docker Image**', which is responsible for tagging the built Docker image and pushing it to a Docker registry (in this case, DockerHub). It utilizes Docker credentials (in my case '**dockerhub-credentials**') for authentication.

The sixth stage is '**Run Ansible Playbook**', which executes the Ansible playbook ('**playbook.yml**') using the inventory file ('**inventory**').

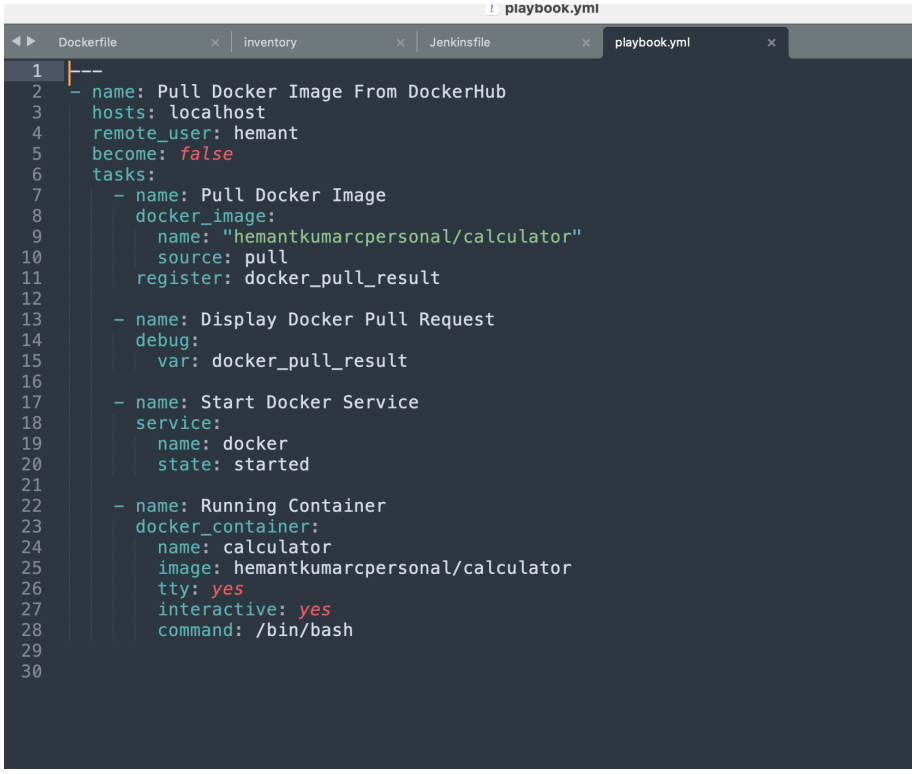
Ansible

Ansible consists of several key components and modules that work together to enable automation and configuration management. These components collectively facilitate the creation of playbooks and the execution of tasks on managed nodes.

playbook.yml

Playbooks are YAML field that define a series of tasks, along with the desired state and conditions. Playbooks combine modules to automate complex workflows and configurations. They provide a structured way to describe the steps required to achieve a specific outcome.

Taks are individual actions or operations performed by modules. Tasks within a playbook define what needs to be accomplished, such as installing a package, restarting a service, or copying a file.



The Ansible playbook here performs several tasks related to Docker :

Pull Docker Image From DockerHub - The playbook starts by defining a task to pull a Docker image from DockerHub. THe

specified image is '**hemantkumarcpersonal/calculator**'. The result of the Docker image pull operation is stored in the variable **docker_pull_result** for later reference.

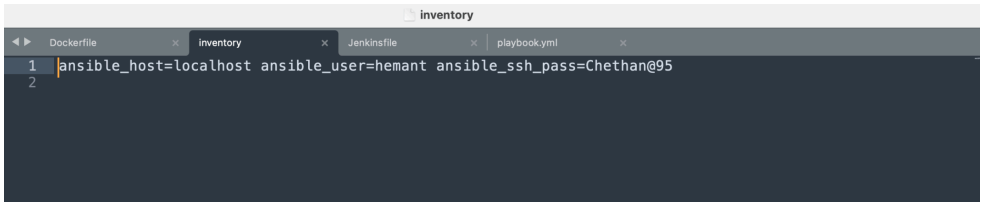
Display Docker Pull Request - This task utilizes the Ansible debug module to display the result of the Docker image pull operation. It helps provide insights into whether the image was successfully pulled or if there were any errors.

Start Docker Service - The playbook includes a task to ensure that the Docker service on the localhost is in the "started" state. This ensures that Docker is running and ready to manage containers.

Running Container - The final task runs a Docker container named "**calculator**" based on the previously pulled image ("**hemantkumarcpersonal/calculator**"). The container is started in the detached mode, ensuring it runs in the background. To execute commands in the Docker container, we say that we want a terminal ("**tty**") and start the container in the "**interactive**" mode, allowing us to execute commands while the container is in the running state.

inventory

Inventories are files that list the hosts (managed nodes) and group them into categories. This categorization allows for easier management of resources and targeted execution of tasks on specific groups.



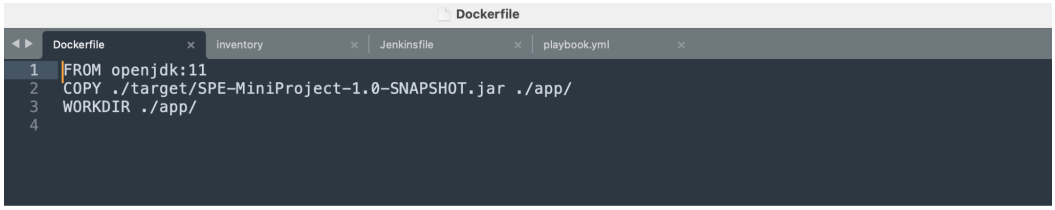
Here, the inventory file contains information about a single host, “**localhost**” and includes specific connection details using Ansible variables.

ansible_host = localhost - This is the name or IP Address of the host that Ansible will manage. In this case, it is set to “**localhost**”, indicating that Ansible will be interacting with the local machine.

ansible_user = hemant - This variable specifies the remote user that Ansible should use when connecting to the host. In this example, the remote user is set to “**hemant**”. Ansible will use this username to establish the SSH connection to the specified host.

ansible_ssh_pass = Chethan@95 - This variable sets the SSH password for the specified remote user (“**hemant**”). It is used for authentication when connecting to the host. However, note that specifying passwords directly in the inventory file is not recommended for security reasons. It’s preferable to use SSH keys for authentication.

Dockerfile



FROM openjdk:11 - This line specifies the base image for the Docker container. Here, since we want to execute Java in the Docker container, for that purpose, we use the official openjdk image from DockerHub. This image serves as the foundation for building our java calculator program.

COPY ./target/SPE-MiniProject-1.0-SNAPSHOT.jar ./app/ - This line copies the snapshot jar file from the target folder that was created while building the maven project and it places this file in the /app/ directory inside the image.

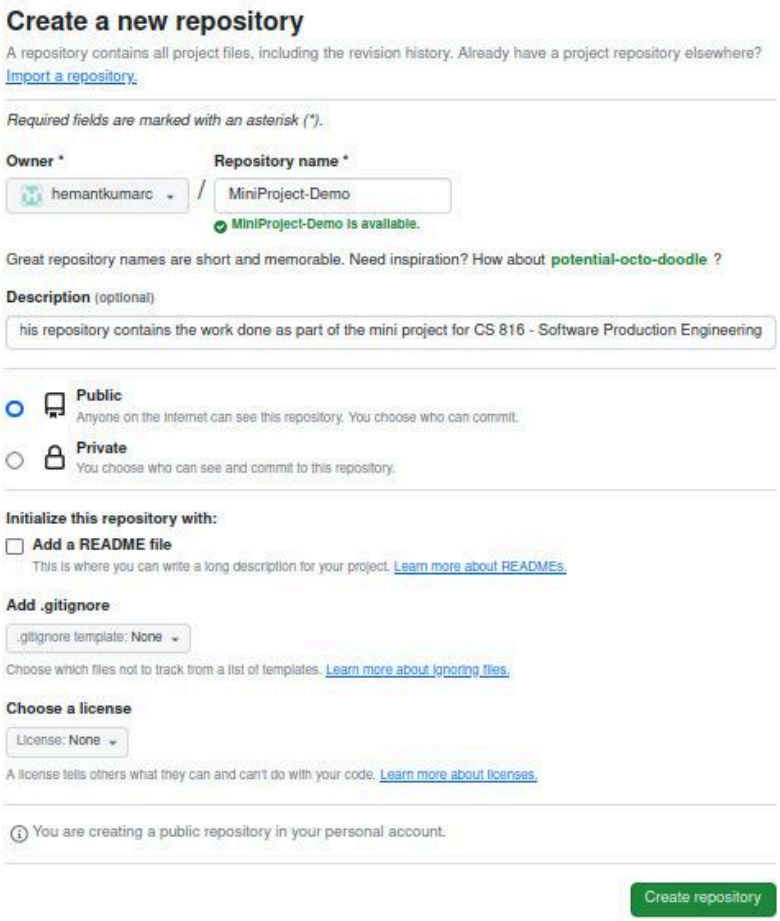
WORKDIR ./app/ - This line sets the current working directory for subsequent instructions in the Dockerfile to the /app directory.

Once these files are added, we initialize the project root structure to be a git repository by typing **git init**.

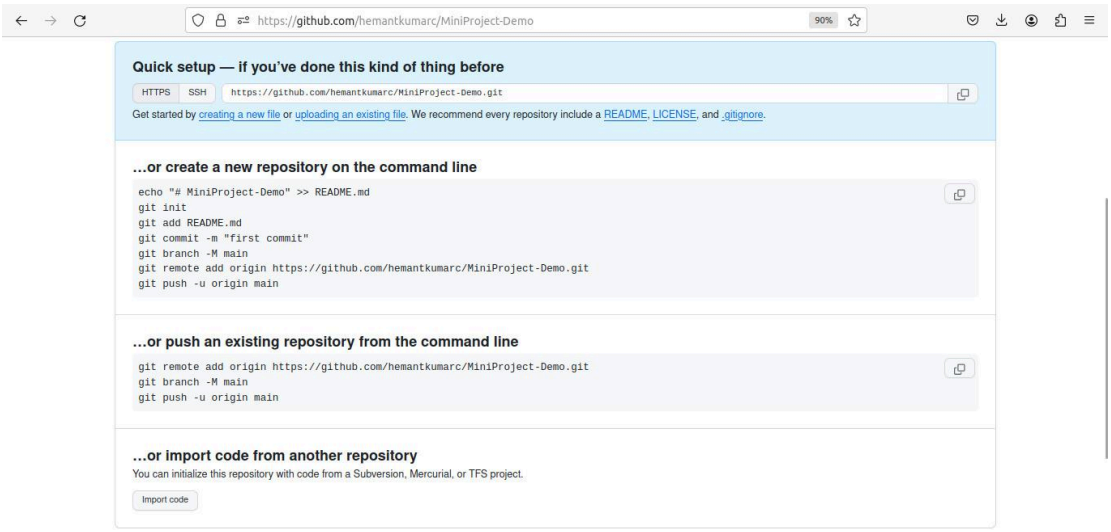
```
hemant@hemant-Latitude-E5470:~/Documents$ mkdir MiniProject
hemant@hemant-Latitude-E5470:~/Documents$
hemant@hemant-Latitude-E5470:~/Documents$ cd MiniProject/
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/hemant/Documents/MiniProject/.git/
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
```

```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Dockerfile
        Jenkinsfile
        inventory
        playbook.yml
        pom.xml
        src/

nothing added to commit but untracked files present (use "git add" to track)
```

Meanwhile on GitHub, we create a new repository with the name **MiniProject-Demo.**



Now, we are presented with the following screen.



Now, back in the repository, add all the project files into the staging area by typing the command **git add .**

```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git add .
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git commit -m "Add all application files"
[master (root-commit) 32c3574] Add all application files
 7 files changed, 285 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 Jenkinsfile
 create mode 100644 inventory
 create mode 100644 playbook.yml
 create mode 100644 pom.xml
 create mode 100644 src/main/java/org/example/Main.java
 create mode 100644 src/test/java/org/example/MainTest.java
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git branch -M main
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git remote add origin https://github.com/hemantkumarc/MiniProject-Demo.git
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
```

In order to make commits to the local git repository, we commit the changes that we have made, by typing **git commit** and the **commit message** that we want.

Now to rename the local default branch from **master -> main**, type **git branch -M main**. After this, we connect our local git repository with the remote repository that is present on GitHub by typing the following command.

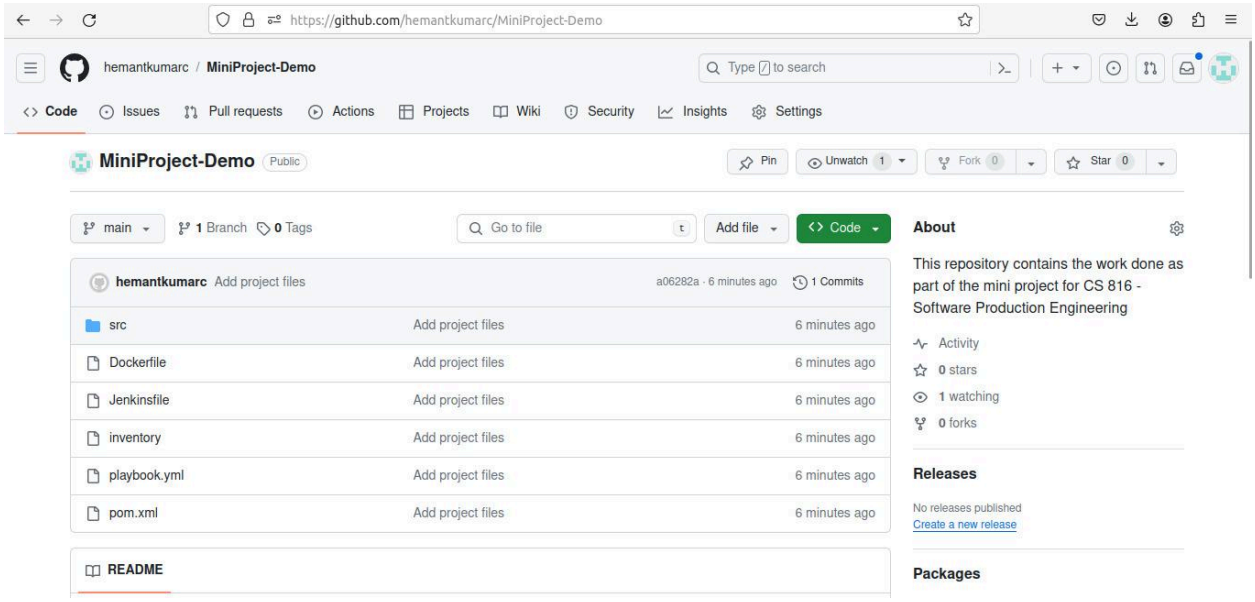
git remote add origin https://github.com/hemantkumarc/MiniProject-Demo.git

We are now ready to push our changes to GitHub, by typing

git push -u origin main

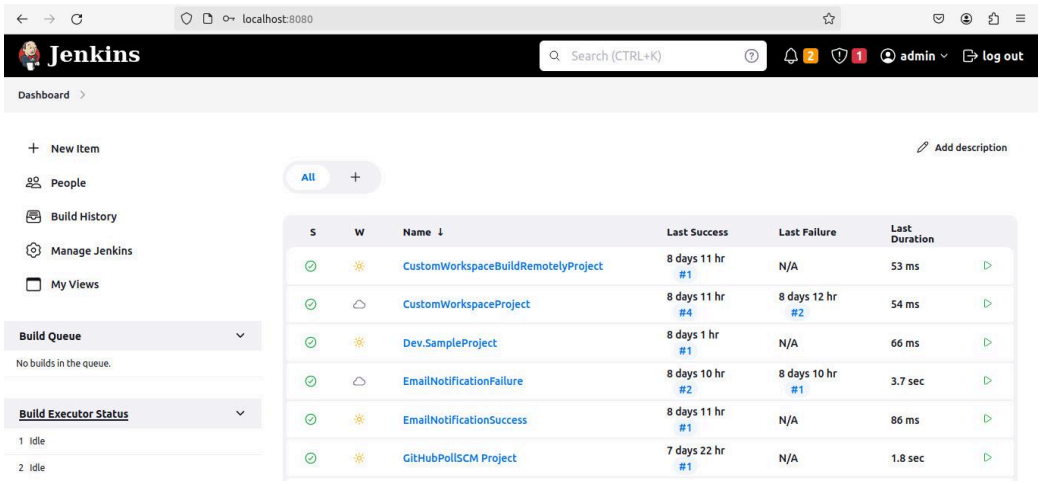
```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git push -u origin main
Username for 'https://github.com': hemantkumarc
Password for 'https://hemantkumarc@github.com':
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (18/18), 3.21 KiB | 469.00 KiB/s, done.
Total 18 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/hemantkumarc/MiniProject-Demo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
```

Now, we can see that our remote repository is updated with the following files.

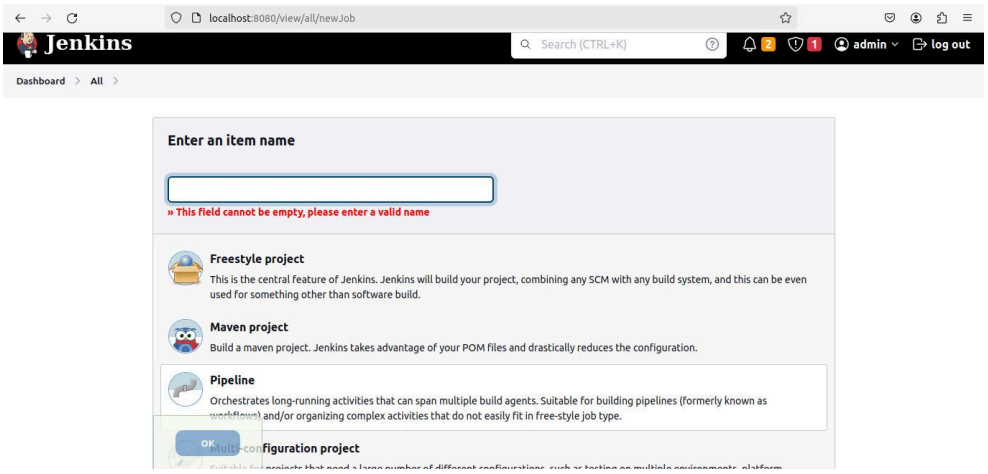


Step 4.) Creating the Jenkins CI/CD Pipeline

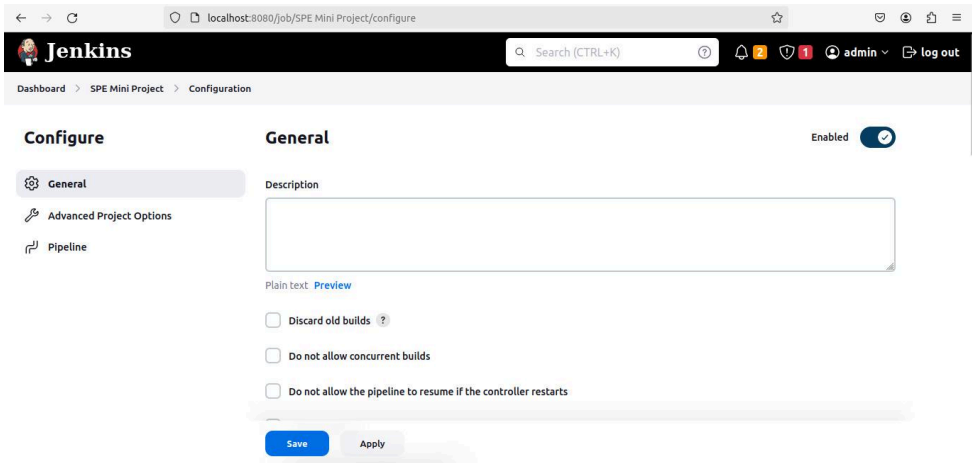
Now go to the Jenkins Dashboard page.



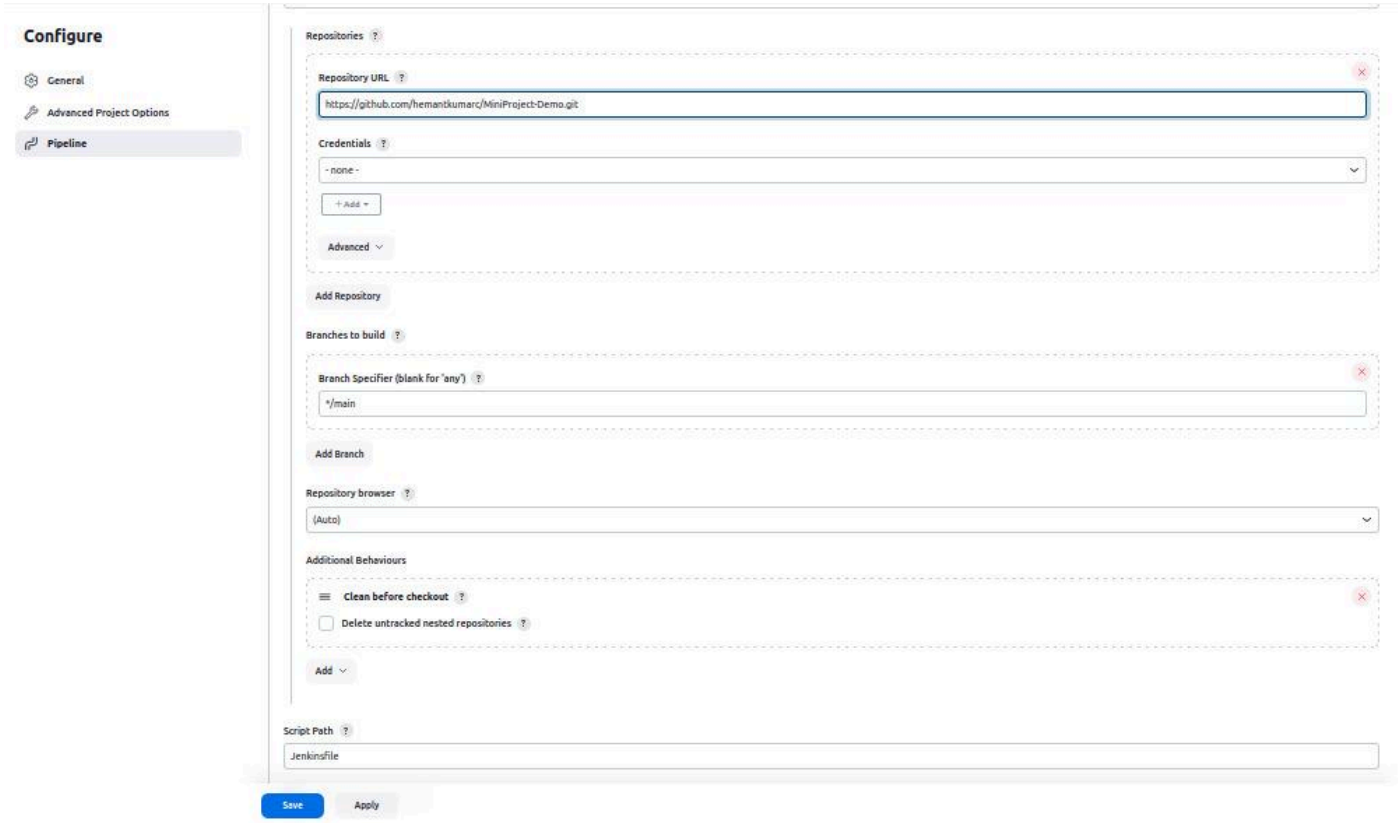
Now click on **New Item**,



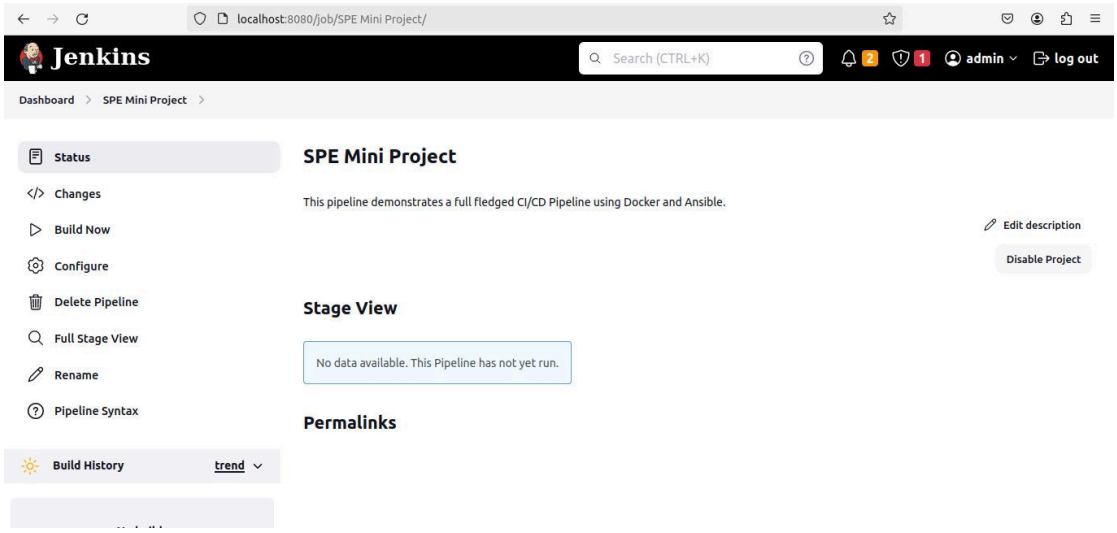
We add an appropriate name for the project and select **Pipeline** and then select **OK**. We are presented with the following page.



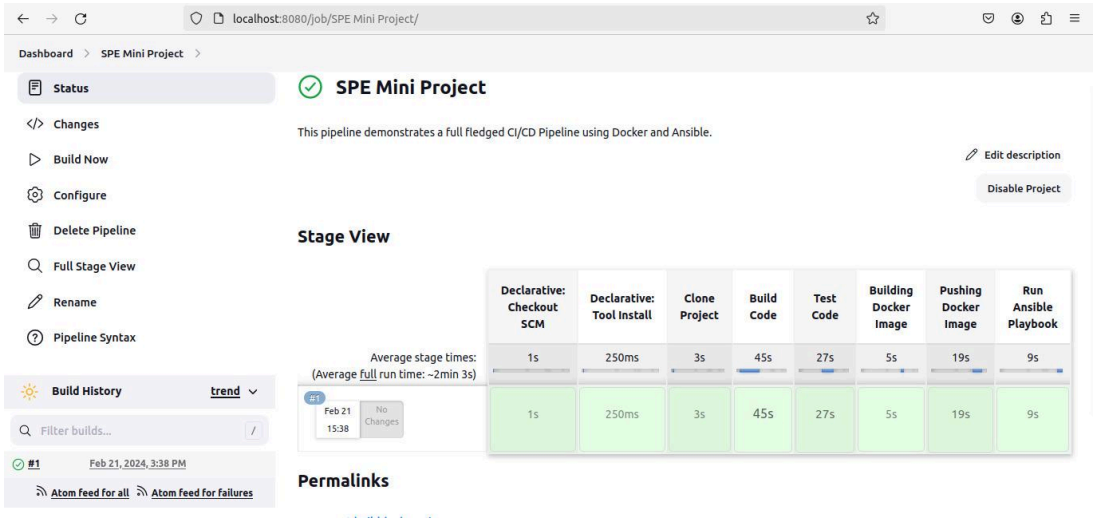
We update this page by giving an appropriate description for the project and under **Pipeline -> Definition**, we select **Pipeline script from SCM**. We select the **Source Code Management (SCM)** as **Git**. And we fill the other fields in the page accordingly as given below.



We then click on **Apply** and **Save**. We then get the following page.



We then click on **Build Now**. We get the following build console page where the build is successful.



Now go to the terminal and type the following commands to see if everything is working as expected.

sudo docker images - this must now show the images that we built locally as part of the '**Building Docker Image**' with the name '**calculator**' and the one that we pulled from DockerHub during the '**Run Ansible Playbook**' stage of the Jenkins CI/CD Pipeline '**hemantkumarcpersonal/calculator**'.

```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ sudo docker images
REPOSITORY                        TAG       IMAGE ID       CREATED             SIZE
hemantkumarcpersonal/calculator   latest    ff339c55976f   About a minute ago   654MB
calculator                        latest    ff339c55976f   About a minute ago   654MB
openjdk                           11        47a932d998b7   18 months ago        654MB
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
```

sudo docker ps -a - this must show a container that is up and running as part of the '**Running Container'** task. This container will have the name '**calculator**'.

```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ sudo docker ps -a
CONTAINER ID   IMAGE                             COMMAND       CREATED             STATUS             PORTS     NAMES
bc15c44966f0   hemantkumarcpersonal/calculator   "/bin/bash"   About a minute ago   Up About a minute            calculator
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$
```

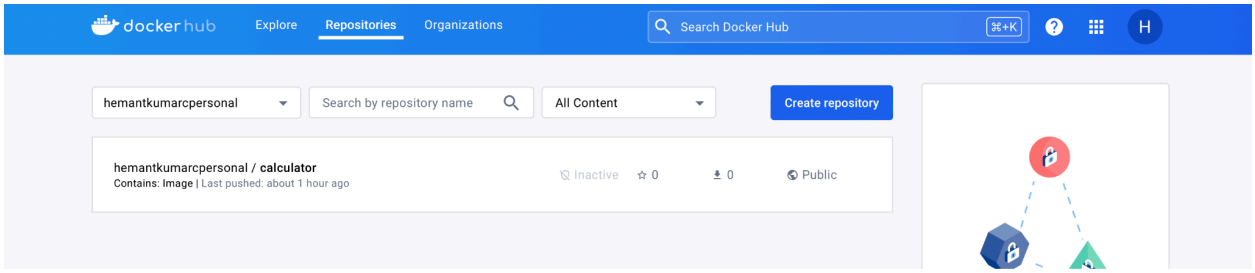
sudo docker start -i calculator - this starts the container by first executing it in the interactive mode so that we can run commands to execute the jar file.

```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ sudo docker start -i calculator
root@bc15c44966f0:/app# ls
SPE-MiniProject-1.0-SNAPSHOT.jar
root@bc15c44966f0:/app#
root@bc15c44966f0:/app#
root@bc15c44966f0:/app# java -jar SPE-MiniProject-1.0-SNAPSHOT.jar

::: Calculator :::
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power Function
5. Exit

Enter Your Choice : 1
Enter a number : 4
Result = 2.0
--------------------------------------

::: Calculator :::
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power Function
5. Exit

Enter Your Choice : 5
root@bc15c44966f0:/app#
```

Then we execute the jar file by typing the command,
java -jar SPE-MiniProject-1.0-SNAPSHOT.jar

And we get the calculator program up and running.

Meanwhile in DockerHub, we have created the following repository that anyone can download and use.



References：

1.) https://aws.amazon.com/devops/what-is-devops/

2.)https://www.splunk.com/en_us/blog/learn/source-code-management.html

3.) https://opensource.com/resources/what-ansible

4.) https://www.edureka.co/blog/what-is-jenkins/

5.) https://docs.docker.com/get-started/overview/

6.) GitHub Link :
https://github.com/hemantkumarc/MiniProject-Demo

7.) DockerHub Link :
https://hub.docker.com/search?q=hemantkumarcpersonal