

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 :

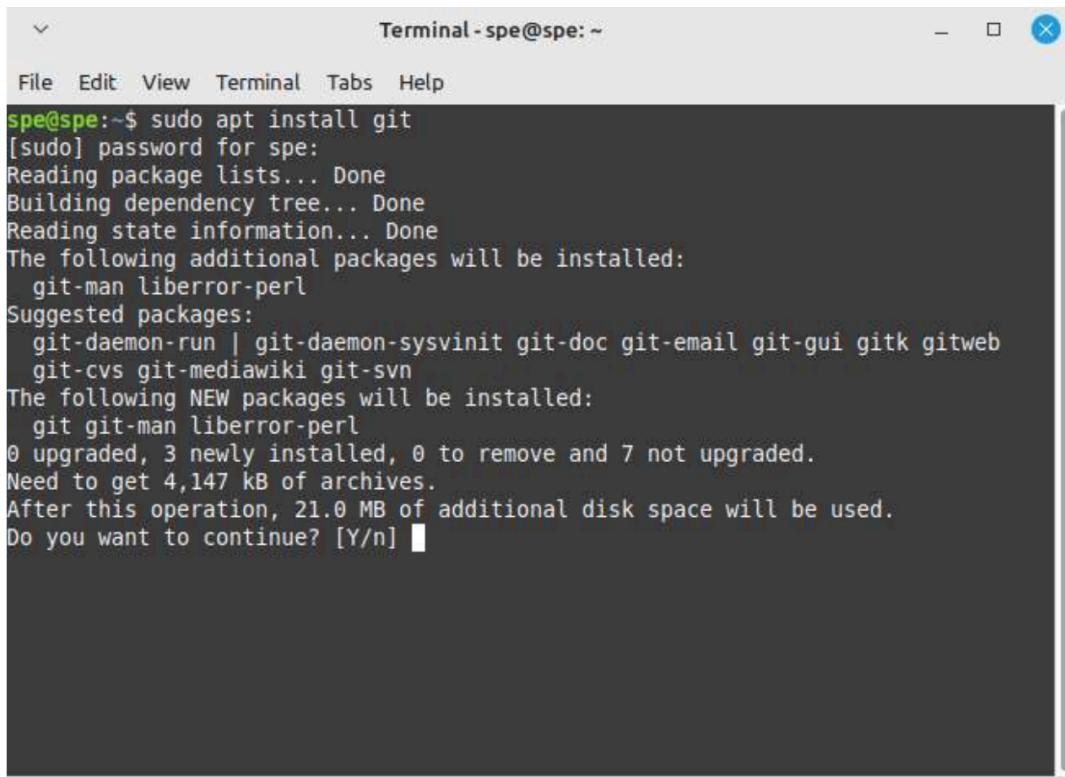
- **Source Code Management (Git and GitHub)** :
Source code management is the practice of tracking modifications to source code. Keeping a running history of the changes made to a codebase helps programmers, developers and testers ensure that they're always working with accurate and up-to-date code and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

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

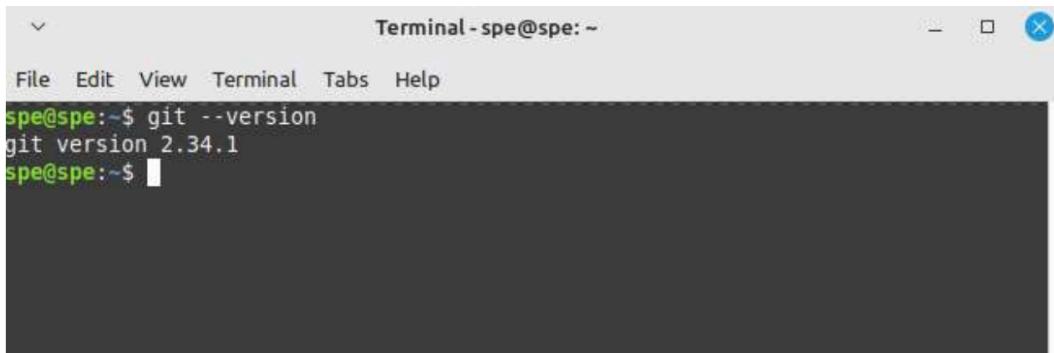
To install Git : type “sudo apt install git”

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.



```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
spe@spe:~$ sudo apt install git
[sudo] password for spe:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb
  git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 7 not upgraded.
Need to get 4,147 kB of archives.
After this operation, 21.0 MB of additional disk space will be used.
Do you want to continue? [Y/n] ■
```



```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
spe@spe:~$ git --version
git version 2.34.1
spe@spe:~$ ■
```

- 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.

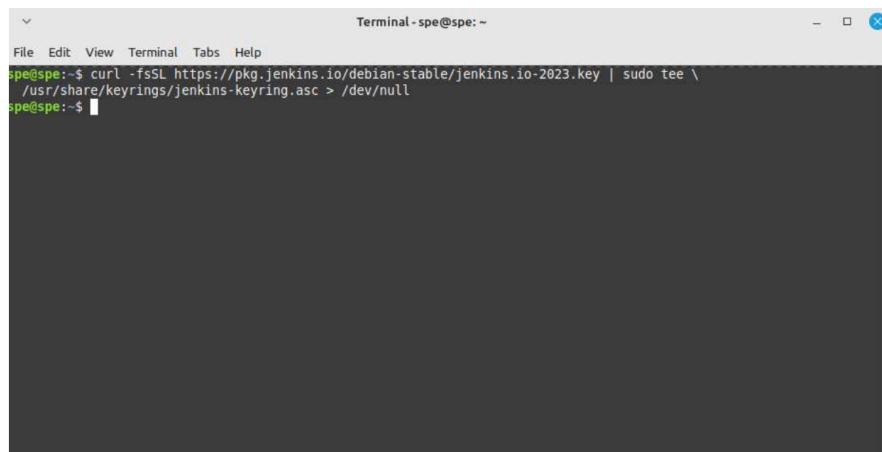
With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.

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
```

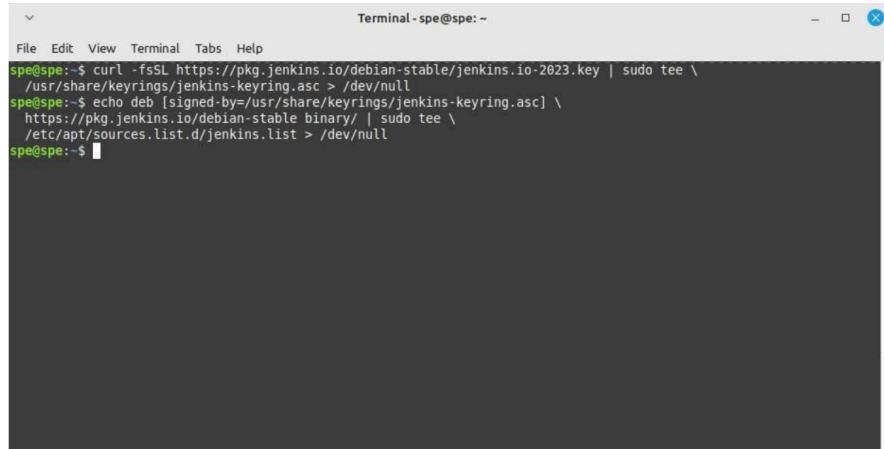


A screenshot of a terminal window titled "Terminal - spe@spe: ~". The window shows a single command being run:

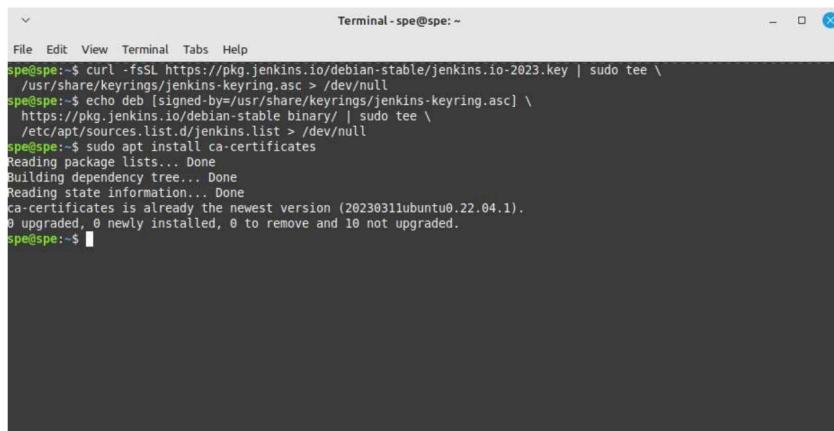
```
spe@spe:~$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
spe@spe:~$
```

The above command fetches the certificates for Jenkins.

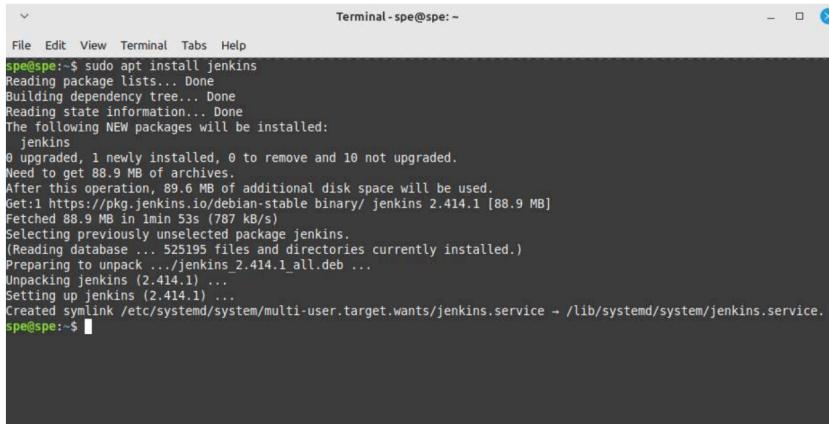
```
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
```

A screenshot of a terminal window titled "Terminal - spe@spe: ~". The window shows a command being run: "curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null". The command is completed successfully, indicated by the prompt "spe@spe:~\$".

```
sudo apt install ca-certificates  
sudo apt update  
sudo apt install jenkins
```

A screenshot of a terminal window titled "Terminal - spe@spe: ~". The window shows a sequence of commands: "curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null", "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", and "sudo apt install ca-certificates". The output shows the package manager reading lists, building dependency trees, and determining that "ca-certificates" is already the newest version. The prompt "spe@spe:~\$" is visible at the end.

The above commands refreshes the certificates and installs Jenkins.



A screenshot of a terminal window titled "Terminal - spe@spe:~". The window shows the output of a command to install Jenkins via apt. The text in the terminal is as follows:

```
File Edit View Terminal Tabs Help
spe@spe:~$ sudo apt install jenkins
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  jenkins
0 upgraded, 1 newly installed, 0 to remove and 10 not upgraded.
Need to get 88.9 MB of archives.
After this operation, 89.6 MB of additional disk space will be used.
Get:1 https://pkg.jenkins.io/debian-stable binary/ jenkins 2.414.1 [88.9 MB]
Fetched 88.9 MB in 1min 53s (787 kB/s)
Selecting previously unselected package jenkins.
(Reading database ... 529195 files and directories currently installed.)
Preparing to unpack .../jenkins_2.414.1_all.deb ...
Unpacking Jenkins (2.414.1) ...
Setting up Jenkins (2.414.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/jenkins.service → /lib/systemd/system/jenkins.service.
spe@spe:~$
```

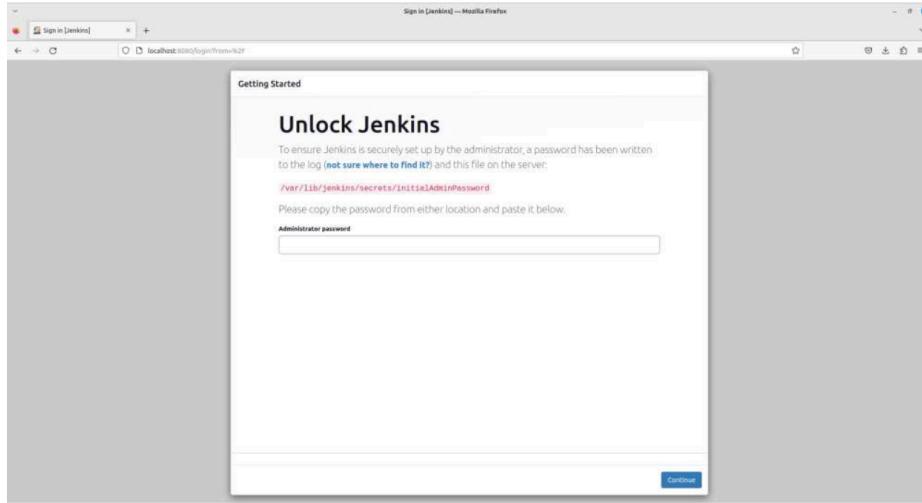
Once you have installed it, we can start it and check its status by typing the following commands :

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 :

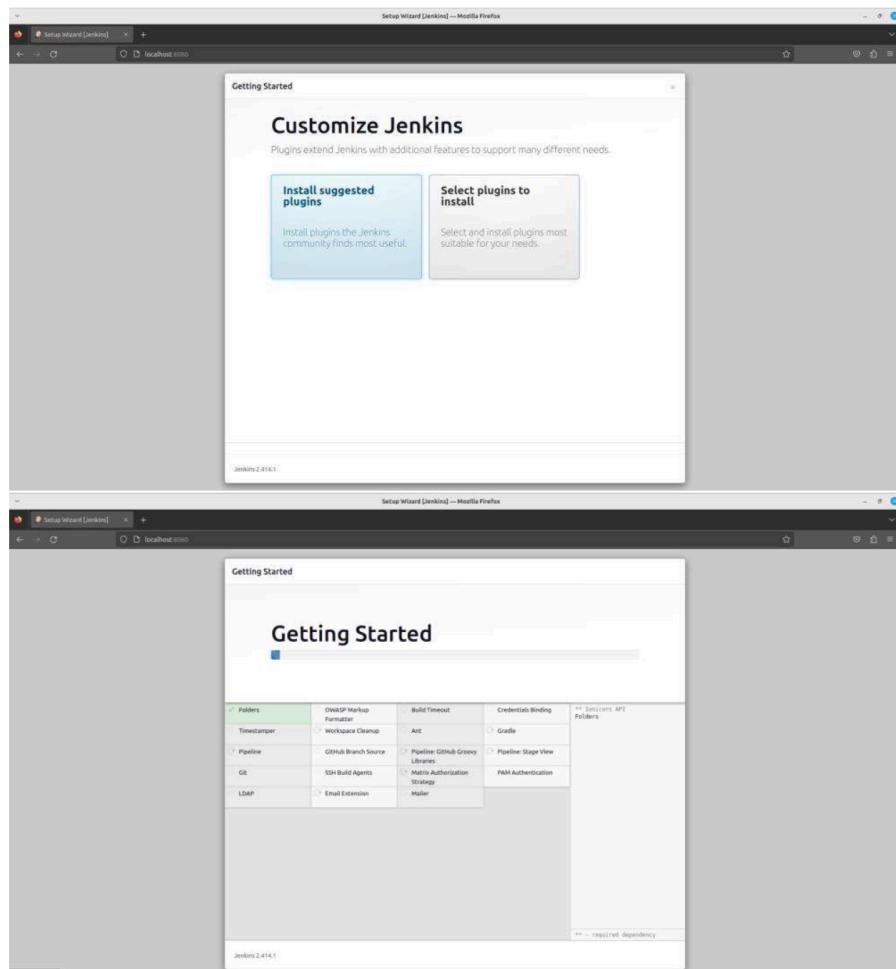


To get this password, we will run the following command :

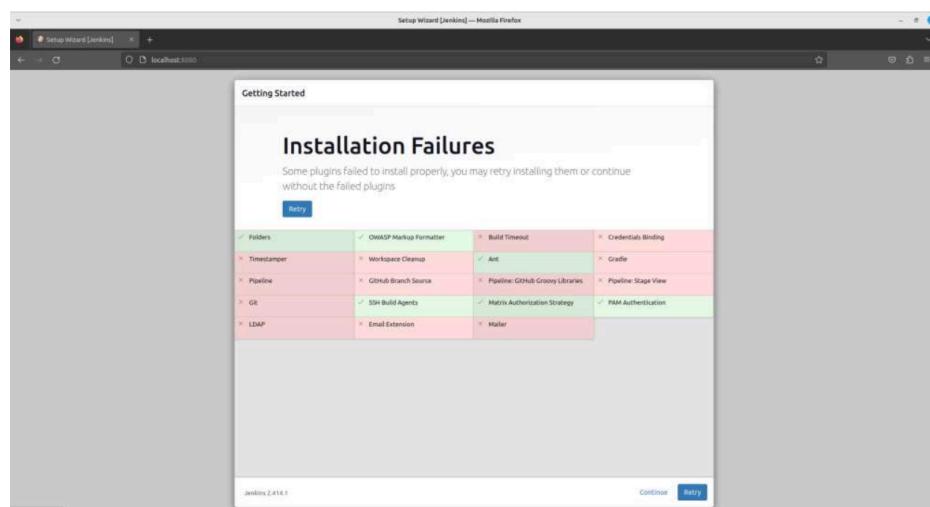
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

A screenshot of a terminal window titled "Terminal - spe@spe: ~". The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The command "spe@spe:~\$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword" is entered, followed by "[sudo] password for spe:". The password "fb985804b68047cc983e9539c1a3f66a" is typed and shown in the terminal. The prompt "spe@spe:~\$" appears again.

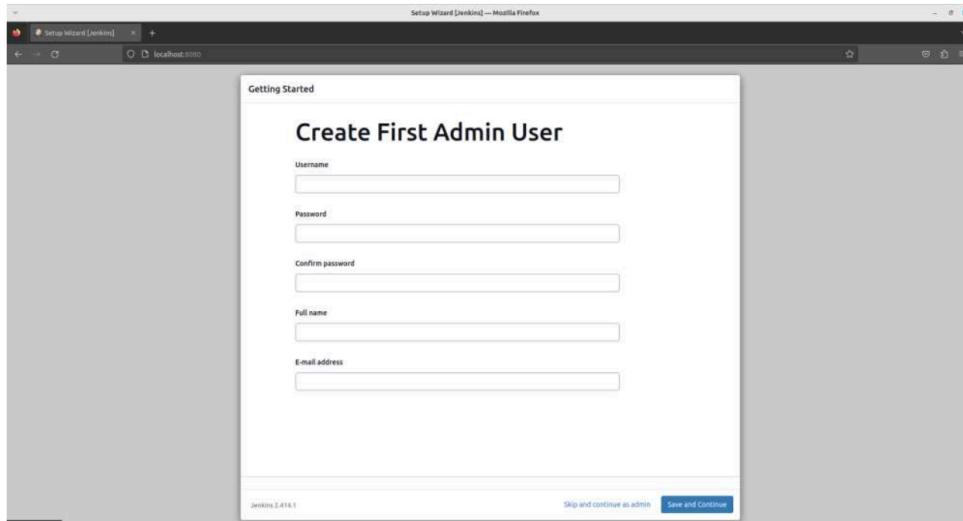
Copy and paste the password that shows on the terminal. Once we do this, Jenkins will ask us for what plugins we want to have installed. We are going to choose the suggested plugins.



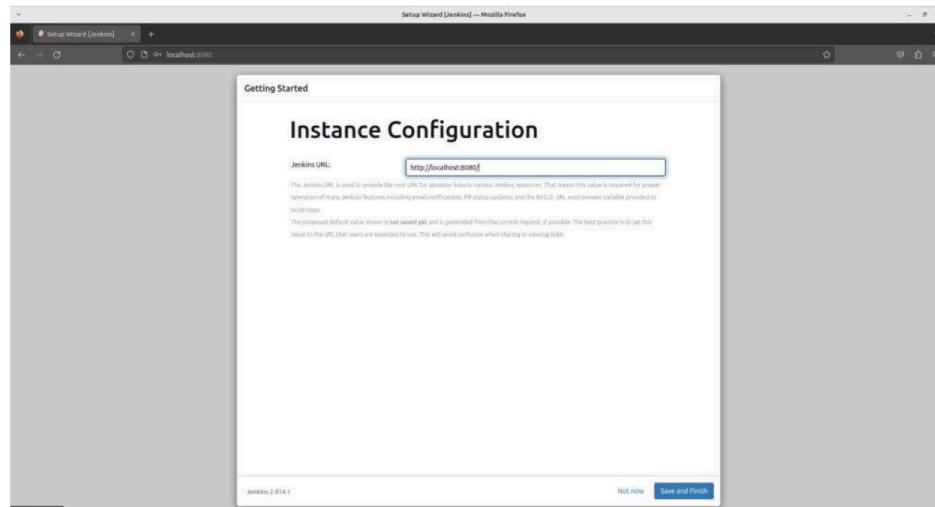
One of the known issues that many face is the failure to install plugins.



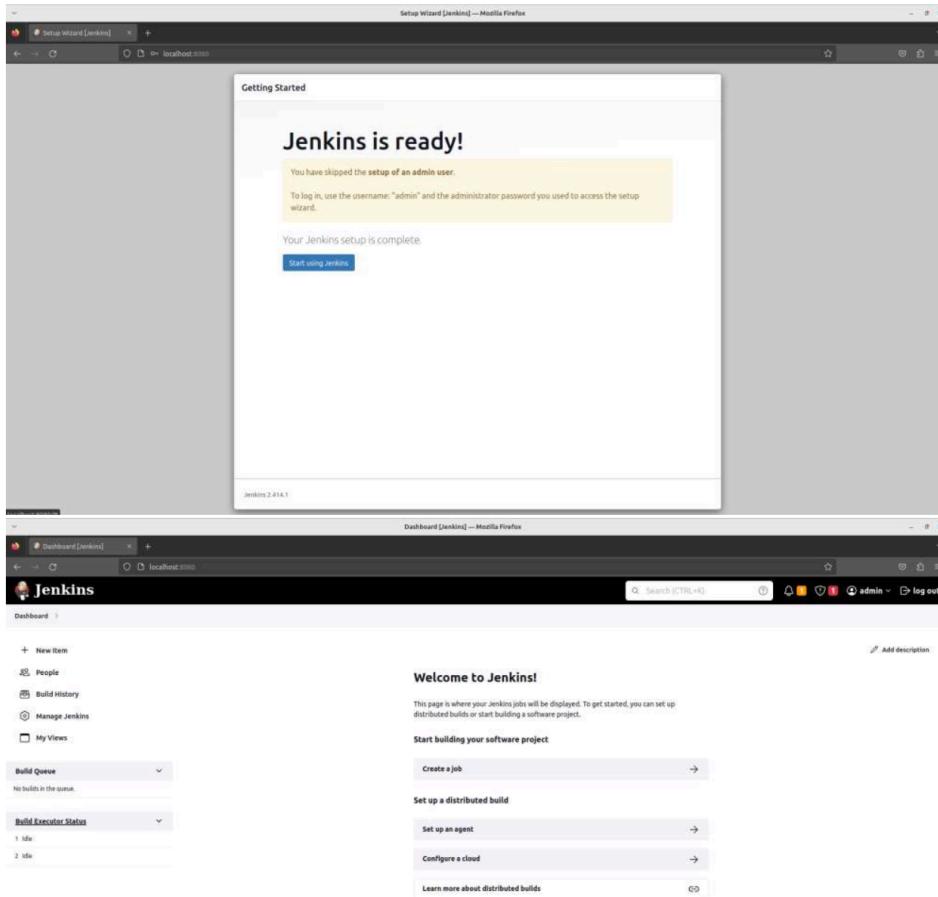
In such cases, keep retrying a few times as many will resolve after a few attempts. Once the errors resolve, the next step is the creation of Jenkins First User creation. You have the option to skip it. We will skip and continue as Jenkins Admin User.



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 -

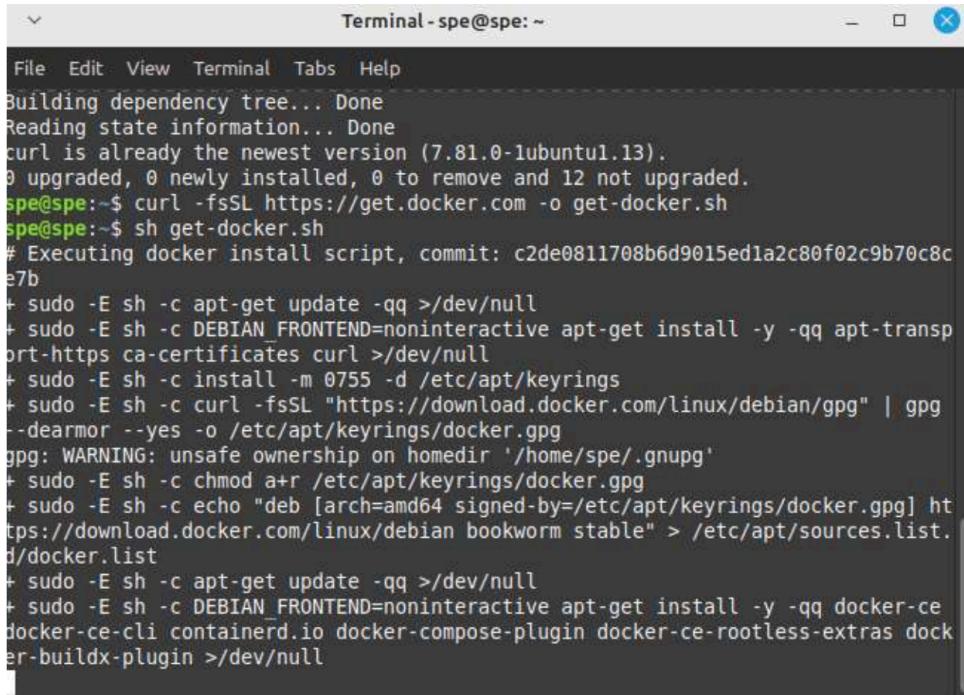
```
curl -fsSL https://get.docker.com -o get-docker.sh
```

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

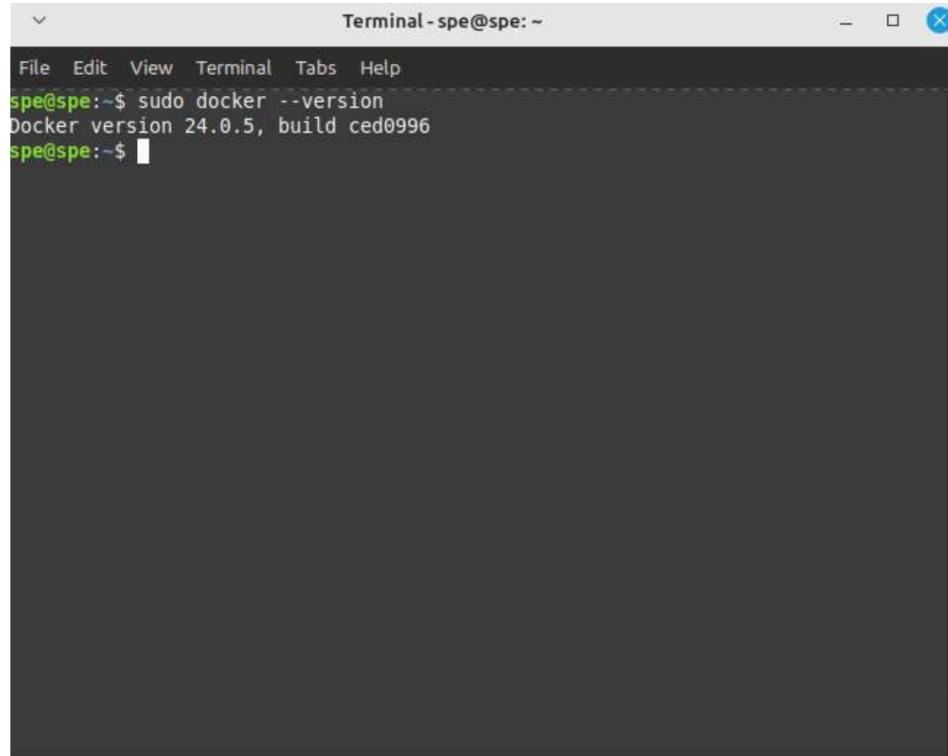
```
sh get-docker.sh
```

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

```
sudo docker --version
```



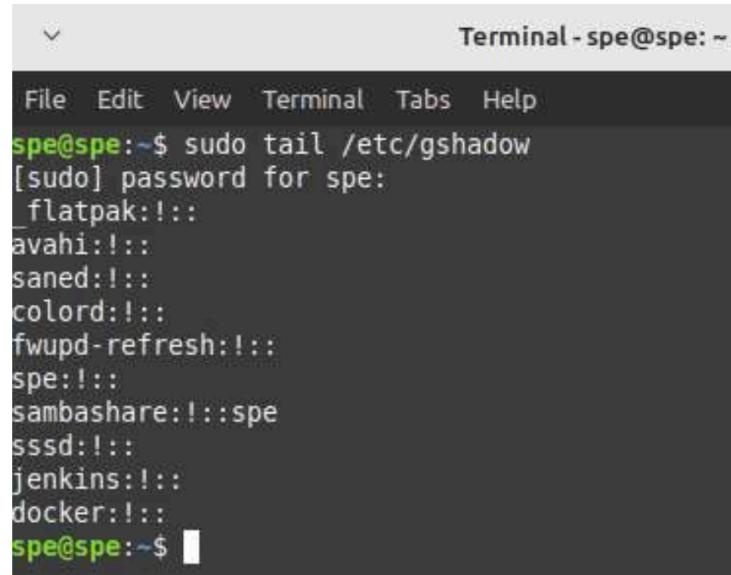
```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.81.0-1ubuntu1.13).
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
spe@spe:~$ curl -fsSL https://get.docker.com -o get-docker.sh
spe@spe:~$ sh get-docker.sh
# Executing docker install script, commit: c2de0811708b6d9015ed1a2c80f02c9b70c8c
e7b
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transp
ort-https ca-certificates curl >/dev/null
+ sudo -E sh -c install -m 0755 -d /etc/apt/keyrings
+ sudo -E sh -c curl -fsSL "https://download.docker.com/linux/debian/gpg" | gpg
--dearmor --yes -o /etc/apt/keyrings/docker.gpg
gpg: WARNING: unsafe ownership on homedir '/home/spe/.gnupg'
+ sudo -E sh -c chmod a+r /etc/apt/keyrings/docker.gpg
+ sudo -E sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] ht
tps://download.docker.com/linux/debian bookworm stable" > /etc/apt/sources.list.
d/docker.list
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce
docker-ce-cli containerd.io docker-compose-plugin docker-ce-rootless-extras dock
er-buildx-plugin >/dev/null
```



```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
spe@spe:~$ sudo docker --version
Docker version 24.0.5, build ced0996
spe@spe:~$
```

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
```



```
File Edit View Terminal Tabs Help
spe@spe:~$ sudo tail /etc/gshadow
[sudo] password for spe:
_flatpak:!::
avahi:!::
saned:!::
colord:!::
fwupd-refresh:!::
spe:!::
sambashare:!:spe
sssd:!::
jenkins:!::
docker:!::
spe@spe:~$ █
```

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.

```
spe@spe:~$ sudo usermod -aG docker jenkins
spe@spe:~$ sudo systemctl restart jenkins
spe@spe:~$ sudo tail /etc/gshadow
_flatpak:!::
avahi:!::
saned:!::
colord:!::
fwupd-refresh:!::
spe:!::
sambashare:!::spe
sssd:!::
jenkins:!::
docker:!::jenkins
spe@spe:~$
```

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.

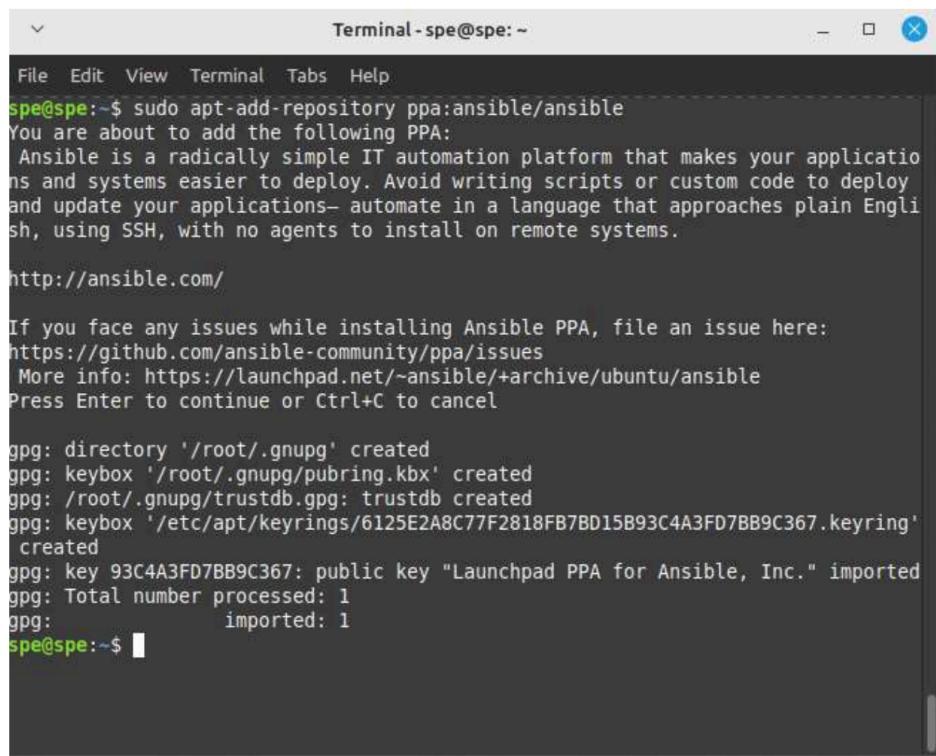
```
sudo apt-add repository ppa:ansible/ansible
```

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
```



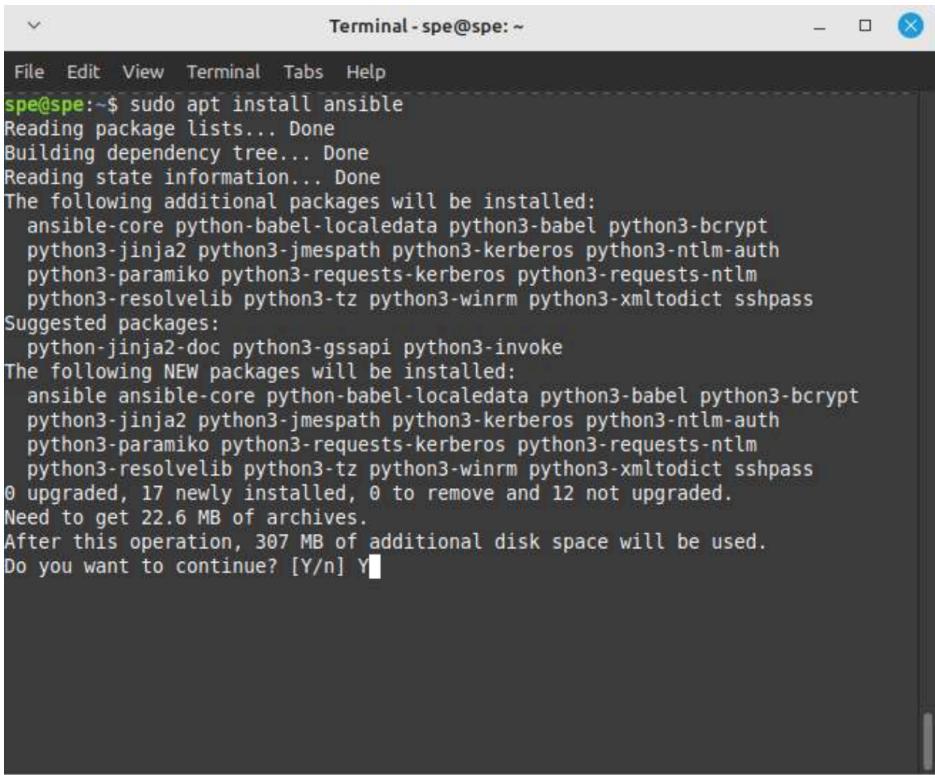
The screenshot shows a terminal window titled "Terminal - spe@spe:~". The user runs the command `sudo apt-add-repository ppa:ansible/ansible`. The terminal displays a warning message about adding a PPA and provides a link to the Ansible website (`http://ansible.com/`). It also gives instructions for reporting issues and provides links to GitHub and Launchpad. The user then presses Enter to continue. The terminal then shows the output of the `gpg` command, which imports a public key from the Launchpad PPA. The key's ID is `93C4A3FD7BB9C367`, and the message indicates it was imported successfully. The session ends with the prompt `spe@spe:~$`.

```
File Edit View Terminal Tabs Help
spe@spe:~$ sudo apt-add-repository ppa:ansible/ansible
You are about to add the following PPA:
Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy. Avoid writing scripts or custom code to deploy and update your applications—automate in a language that approaches plain English, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Press Enter to continue or Ctrl+C to cancel

gpg: directory '/root/.gnupg' created
gpg: keybox '/root/.gnupg/pubring.kbx' created
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: keybox '/etc/apt/keyrings/6125E2A8C77F2818FB7BD15B93C4A3FD7BB9C367.keyring' created
gpg: key 93C4A3FD7BB9C367: public key "Launchpad PPA for Ansible, Inc." imported
gpg: Total number processed: 1
gpg:                         imported: 1
spe@spe:~$
```

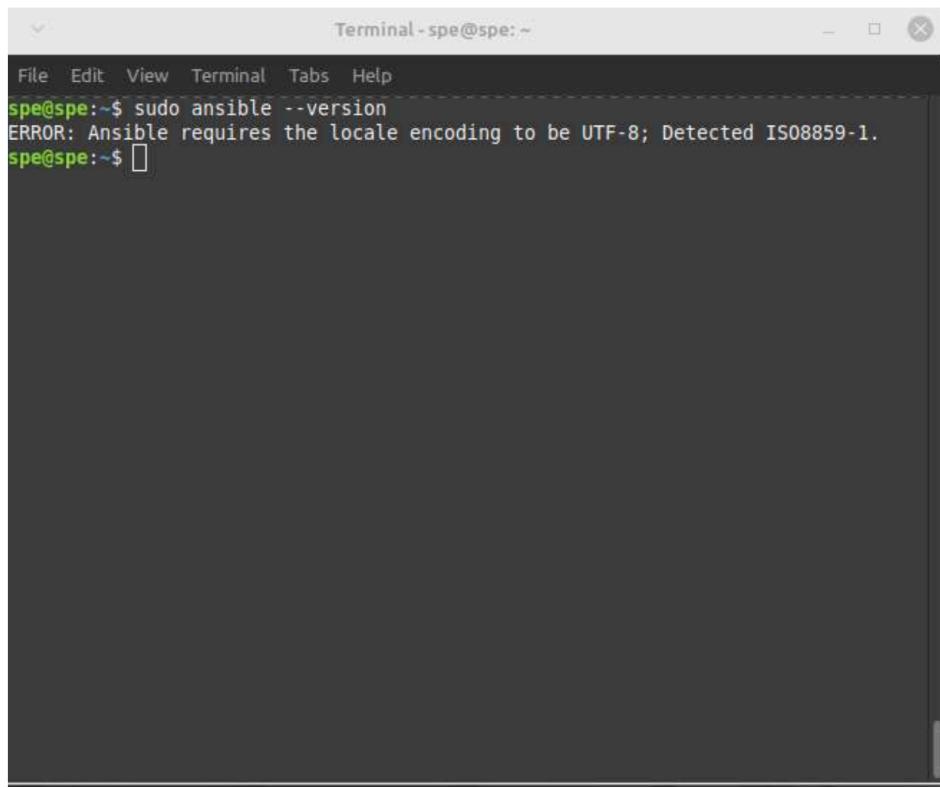


```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
spe@spe:~$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ansible-core python-babel-localedata python3-babel python3-bcrypt
  python3-jinja2 python3-jmespath python3-kerberos python3-ntlm-auth
  python3-paramiko python3-requests-kerberos python3-requests-ntlm
  python3-resolve lib python3-tz python3-winrm python3-xmldict sshpass
Suggested packages:
  python-jinja2-doc python3-gssapi python3-invoke
The following NEW packages will be installed:
  ansible ansible-core python-babel-localedata python3-babel python3-bcrypt
  python3-jinja2 python3-jmespath python3-kerberos python3-ntlm-auth
  python3-paramiko python3-requests-kerberos python3-requests-ntlm
  python3-resolve lib python3-tz python3-winrm python3-xmldict sshpass
0 upgraded, 17 newly installed, 0 to remove and 12 not upgraded.
Need to get 22.6 MB of archives.
After this operation, 307 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

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

sudo ansible -- version

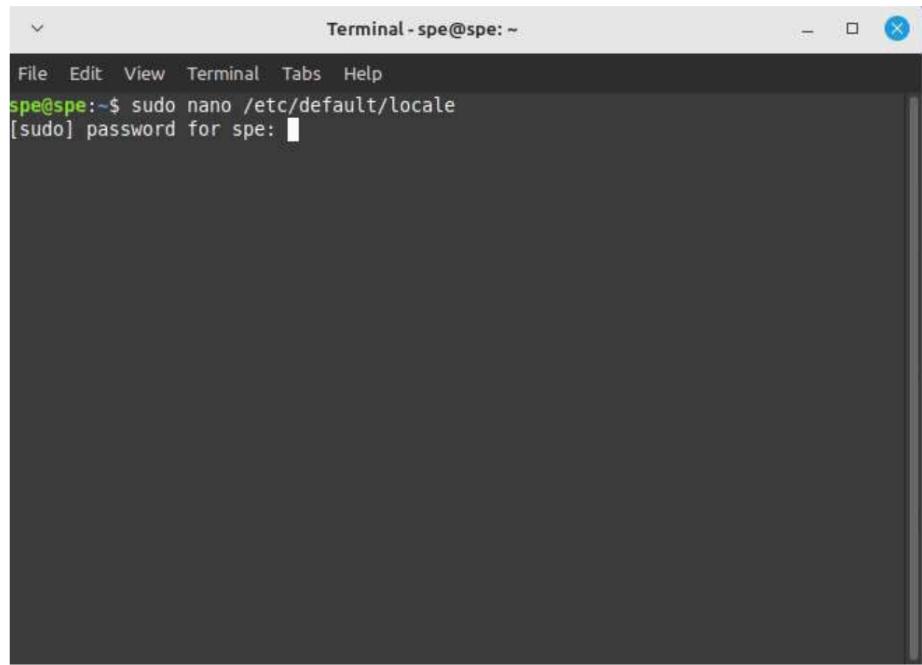
However, a common error that pops up is the following :



A screenshot of a terminal window titled "Terminal - spe@spe:~". The window has a dark gray background. At the top, there is a menu bar with options: File, Edit, View, Terminal, Tabs, Help. Below the menu, the command "sudo ansible --version" is entered, followed by its output: "ERROR: Ansible requires the locale encoding to be UTF-8; Detected ISO8859-1." The prompt "spe@spe:~\$ " is visible at the bottom left.

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

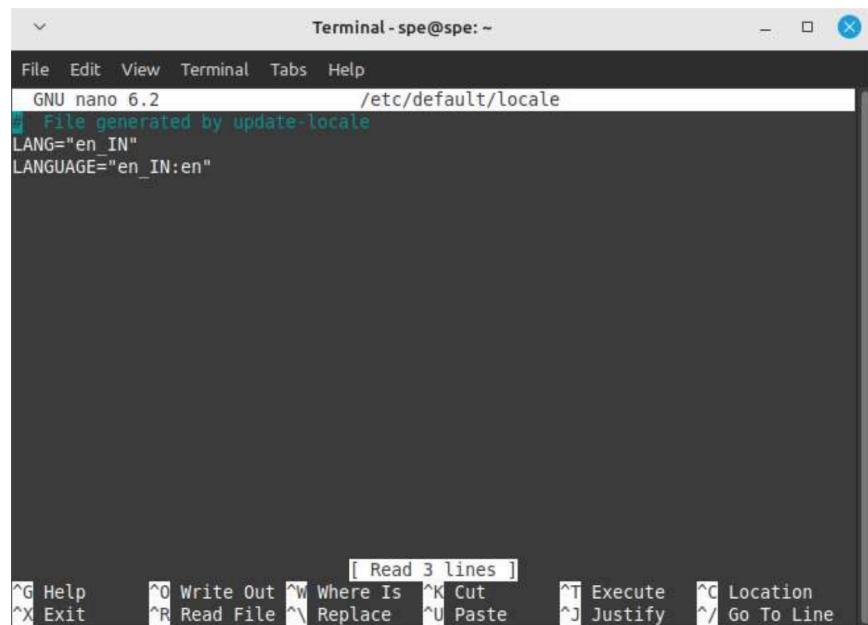
```
sudo nano /etc/default/locale
```



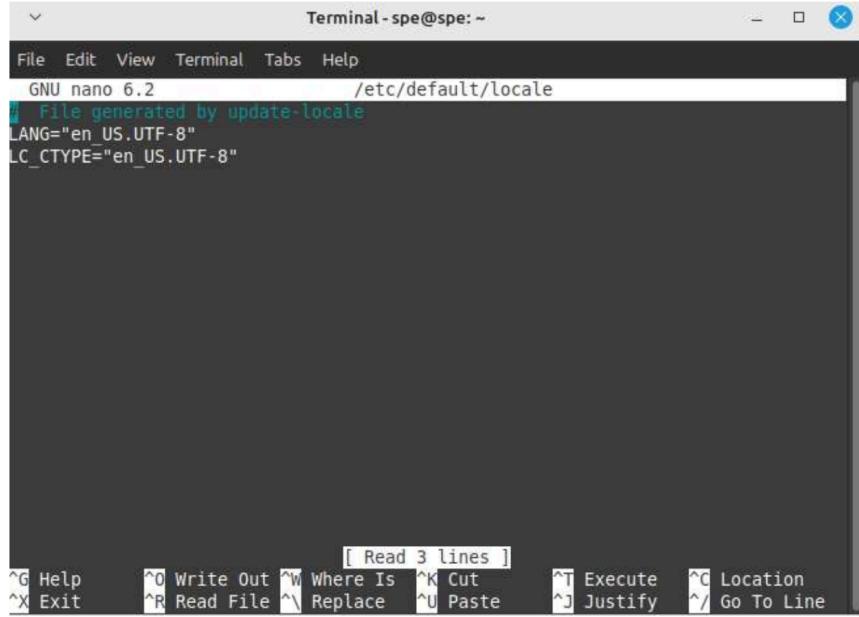
This will open the locale file in the terminal. We are going to replace LANG and LC_TYPE values as follows:

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

Before:



After:



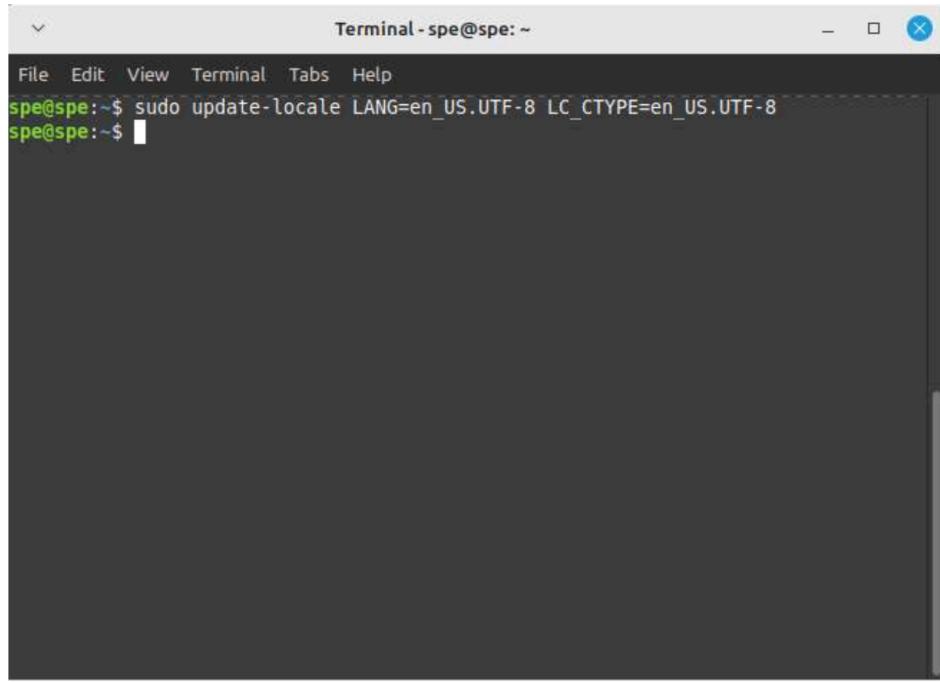
```
Terminal - spe@spe: ~
File Edit View Terminal Tabs Help
GNU nano 6.2          /etc/default/locale
↓ File generated by update-locale
LANG="en_US.UTF-8"
LC_CTYPE="en_US.UTF-8"

[ Read 3 lines ]
^G Help      ^O Write Out ^W Where Is ^K Cut      ^T Execute   ^C Location
^X Exit      ^R Read File ^N Replace   ^U Paste    ^J Justify   ^/ Go To Line
```

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.

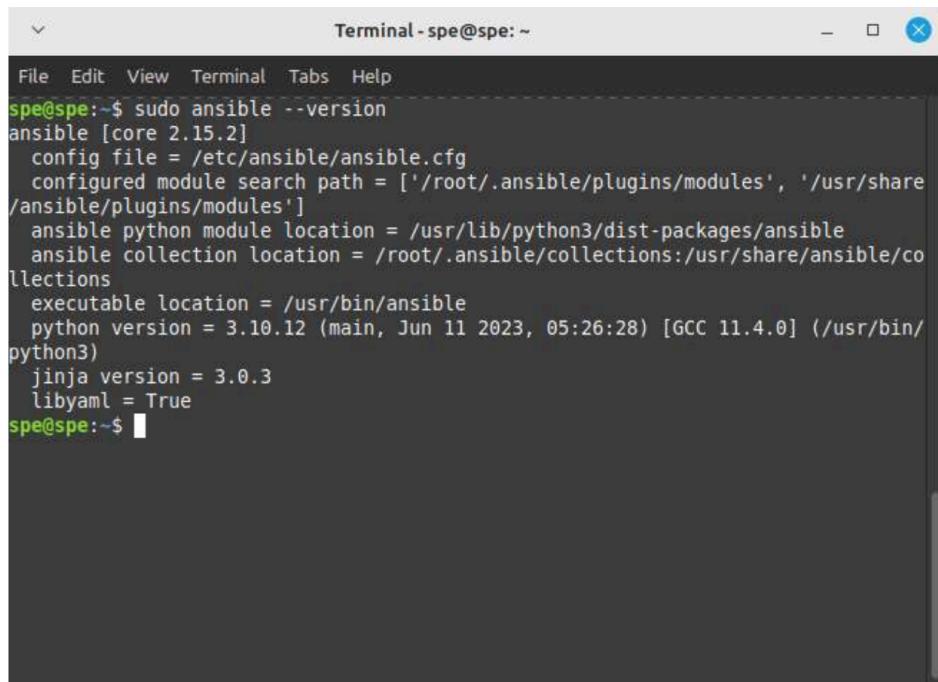
```
sudo update-locale LANG=en_US.UTF-8 LC_CTYPE=en_US.UTF-8
```



A screenshot of a terminal window titled "Terminal - spe@spe:~". The window has a dark theme. The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The command line shows the user running the command "sudo update-locale LANG=en_US.UTF-8 LC_CTYPE=en_US.UTF-8". The prompt "spe@spe:~\$" is visible at the bottom.

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
```



The screenshot shows a terminal window titled "Terminal - spe@spe: ~". The window has a dark background and white text. It displays the output of the command "sudo ansible --version". The output shows the following details:

```
File Edit View Terminal Tabs Help
spe@spe:~$ sudo ansible --version
ansible [core 2.15.2]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
spe@spe:~$
```

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.

The screenshot shows the Jenkins dashboard at localhost:8080. The left sidebar includes links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Under 'Build Queue', it says 'No builds in the queue.' Under 'Build Executor Status', there are 1 Idle and 2 Idle executors. The main area displays a table of build jobs:

S	W	Name	Last Success	Last Failure	Last Duration
Green	Sun	CustomWorkspaceBuildRemotelyProject	8 days 11 hr #1	N/A	53 ms
Green	Cloud	CustomWorkspaceProject	8 days 11 hr #4	8 days 12 hr #2	54 ms
Green	Sun	Dev.SampleProject	8 days 1 hr #1	N/A	66 ms
Green	Cloud	EmailNotificationFailure	8 days 10 hr #2	8 days 10 hr #1	3.7 sec
Green	Sun	EmailNotificationSuccess	8 days 11 hr #1	N/A	86 ms
Green	Sun	GitHubPollSCM Project	7 days 22 hr #1	N/A	1.8 sec

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

The screenshot shows the 'Manage Jenkins' page. The left sidebar has a link for 'Manage Jenkins'. The main area features three informational boxes:

- A box about organizing jobs: "There appears to be a large number of jobs. Did you know that you can organize your jobs to different views? You can click '+' in the top page to create a new view any time." It includes 'Create a view now' and 'Dismiss' buttons.
- A box about security: "Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation." It includes 'Set up agent', 'Set up cloud', and 'Dismiss' buttons.
- A box about project naming: "The Restrict project naming configuration is not set to the Role-based Strategy. This can lead to problems as it allows users to create items, for which they have not the sufficient permissions to discover, read or configure." It includes a 'Dismiss' button.

Below these boxes is a 'System Configuration' section with three categories:

- System**: Configure global settings and paths.
- Tools**: Configure tools, their locations and automatic installers.
- Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins. A badge indicates 4 available updates.

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

The screenshot shows the Jenkins plugin manager interface at `localhost:8080/manage/pluginManager/installed`. The left sidebar has tabs for 'Updates', 'Available plugins', 'Installed plugins' (which is selected), and 'Advanced settings'. A search bar at the top contains the query 'git'. The main table lists five installed plugins related to Git:

Name	Version	Enabled
Git client plugin	4.6.0	<input checked="" type="checkbox"/>
Git plugin	5.2.1	<input checked="" type="checkbox"/>
GitHub API Plugin	1.318-461.v7a_c09c9fa_d63	<input checked="" type="checkbox"/>
GitHub Branch Source Plugin	1772.va_69eda_d018d4	<input checked="" type="checkbox"/>
GitHub plugin	1.38.0	<input checked="" type="checkbox"/>

The screenshot shows the Jenkins plugin manager interface at `localhost:8080/manage/pluginManager/installed`. The left sidebar has tabs for 'Updates', 'Available plugins', 'Installed plugins' (selected), and 'Advanced settings'. A search bar at the top contains the query 'docker'. The main table lists five installed Docker-related plugins:

Name	Version	Enabled
Docker API Plugin	3.3.4-86.v39b_a_Sede342c	<input checked="" type="checkbox"/>
Docker Commons Plugin	439.va_3cb_0a_6a_fb_29	<input checked="" type="checkbox"/>
Docker Pipeline	572.v950f58993843	<input checked="" type="checkbox"/>
Docker plugin	1.5	<input checked="" type="checkbox"/>
docker-build-step	2.11	<input checked="" type="checkbox"/>

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

The screenshot shows the Jenkins 'Credentials' page at localhost:8080/manage/credentials/. The page title is 'Jenkins'. The navigation bar includes 'Dashboard', 'Manage Jenkins', and 'Credentials'. The main section is titled 'Credentials' and displays a table with three rows:

T	P	Store ↓	Domain	ID	Name
SSH	System	(global)		Master_Jenkins_Private_Key	Jenkins (Jenkins Master Private Key to Add Multiple Agents)
Username	System	(global)		dockerhub-credentials	hemantkumarcpersonal/***** (DockerHub Credentials)
Username	System	(global)		localhost	hemant/***** (Localhost Credentials)

Below the table, a section titled 'Stores scoped to Jenkins' shows a single row:

P	Store ↓	Domains
SSH	System	(global)

At the bottom left, there are icons for 'S' (Small), 'M' (Medium), and 'L' (Large).

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

The screenshot shows the 'Global credentials (unrestricted)' page at localhost:8080/manage/credentials/store/system/domain/_/. The page title is 'Jenkins'. The navigation bar includes 'Dashboard', 'Manage Jenkins', 'Credentials', 'System', and 'Global credentials (unrestricted)'. The main section is titled 'Global credentials (unrestricted)' and displays a table with three rows:

ID	Name	Kind	Description
Master_Jenkins_Private_Key	Jenkins (Jenkins Master Private Key to Add Multiple Agents)	SSH Username with private key	Jenkins Master Private Key to Add Multiple Agents
dockerhub-credentials	hemantkumarcpersonal/***** (DockerHub Credentials)	Username with password	DockerHub Credentials
localhost	hemant/***** (Localhost Credentials)	Username with password	Localhost Credentials

At the top right, there is a blue button labeled '+ Add Credentials'. At the bottom left, there are icons for 'S' (Small), 'M' (Medium), and 'L' (Large). At the bottom right, it says 'REST API Jenkins 2.426.3'.

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.

The screenshot shows the Jenkins 'Manage Jenkins' interface under 'Credentials'. A specific credential named 'hemantkumarcpersonal/***** (DockerHub Credentials)' is being edited. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'hemantkumarcpersonal'. There is an unchecked checkbox for 'Treat username as secret'. The 'Password' field is concealed. The 'ID' field is 'dockerhub-credentials'. The 'Description' field is 'DockerHub Credentials'. A blue 'Save' button is at the bottom.

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

The screenshot shows the Jenkins 'Manage Jenkins' interface under 'Credentials'. A specific credential named 'hemant/***** (Localhost Credentials)' is being edited. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'hemant'. There is an unchecked checkbox for 'Treat username as secret'. The 'Password' field is concealed. The 'ID' field is 'localhost'. The 'Description' field is 'Localhost Credentials'. A blue 'Save' button is at the bottom.

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.

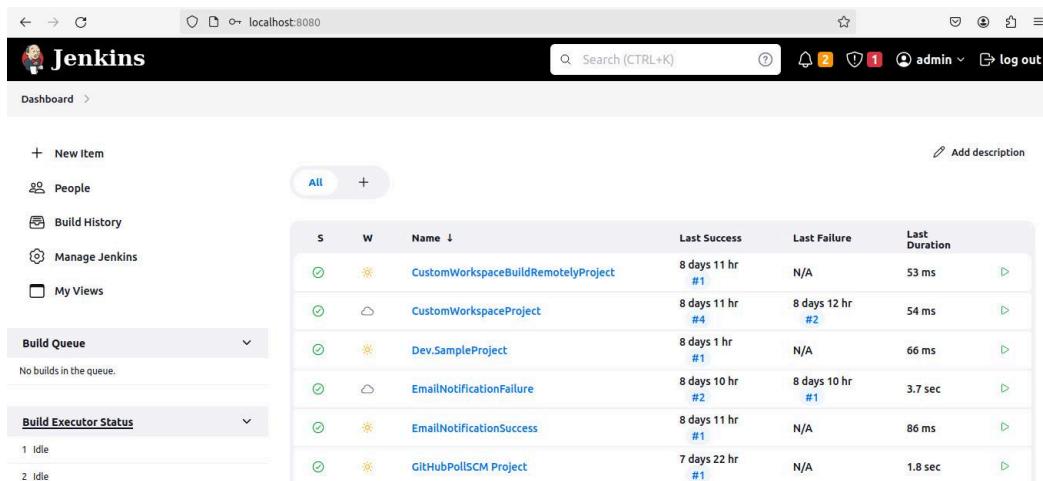
Then we configure the **tools** that we are going to use in the pipeline script.

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.



The screenshot shows the Jenkins dashboard at <https://localhost:8080>. The top navigation bar includes links for New Item, People, Build History, Manage Jenkins, and My Views. The main content area displays a table of build projects:

S	W	Name ↓	Last Success	Last Failure	Last Duration
○	☀️	CustomWorkspaceBuildRemotelyProject	8 days 11 hr #1	N/A	53 ms
○	☁️	CustomWorkspaceProject	8 days 11 hr #4	8 days 12 hr #2	54 ms
○	☀️	Dev.SampleProject	8 days 1 hr #1	N/A	66 ms
○	☁️	EmailNotificationFailure	8 days 10 hr #2	8 days 10 hr #1	3.7 sec
○	☀️	EmailNotificationSuccess	8 days 11 hr #1	N/A	86 ms
○	☀️	GitHubPollSCM Project	7 days 22 hr #1	N/A	1.8 sec

Below the table, there are sections for Build Queue (empty) and Build Executor Status (1 Idle, 2 Idle).

Then go to **Manage Jenkins**,

The screenshot shows the Jenkins 'Manage Jenkins' interface. On the left, there's a sidebar with links for 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted), and 'My Views'. Below these are dropdown menus for 'Build Queue' (empty) and 'Build Executor Status' (showing 1 Idle and 2 Idle). The main content area has a heading 'Manage Jenkins' and a search bar. It displays several status messages and configuration options:

- A message about organizing jobs into views with buttons to 'Create a view now' or 'Dismiss'.
- A message about building on the built-in node being a security issue, with buttons to 'Set up agent', 'Set up cloud', or 'Dismiss'.
- A message about the 'Restrict project naming' configuration, with a 'Dismiss' button.

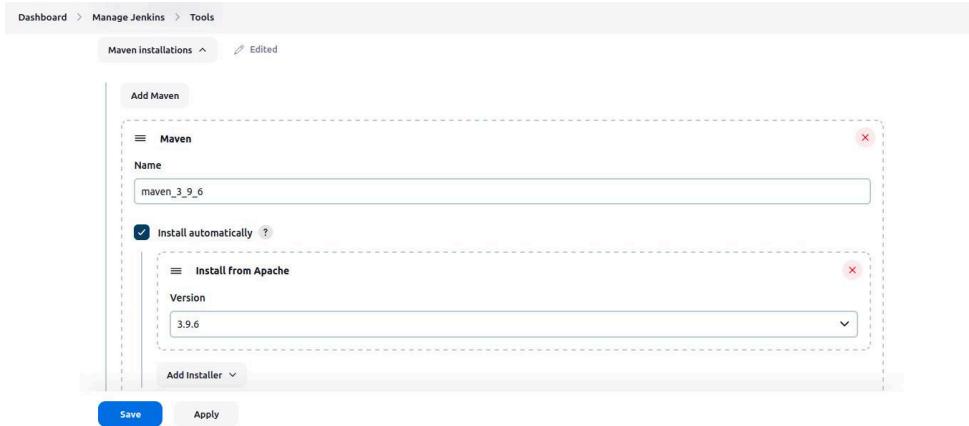
Below these messages is a section titled 'System Configuration' with three items:

- 'System' (with a gear icon): Configure global settings and paths.
- 'Tools' (with a wrench icon): Configure tools, their locations and automatic installers.
- 'Plugins' (with a gear icon): Add, remove, disable or enable plugins that can extend the functionality of Jenkins. This section shows 4 available plugins.

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

The screenshot shows the Jenkins 'Tools' configuration page. At the top, it says 'Dashboard > Manage Jenkins > Tools'. There are sections for 'Maven installations', 'Ansible installations', and 'Docker installations'. Each section has an 'Add' button (e.g., 'Add Ant', 'Add Ansible', 'Add Docker'). At the bottom of the page are 'Save' and 'Apply' buttons.

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’).

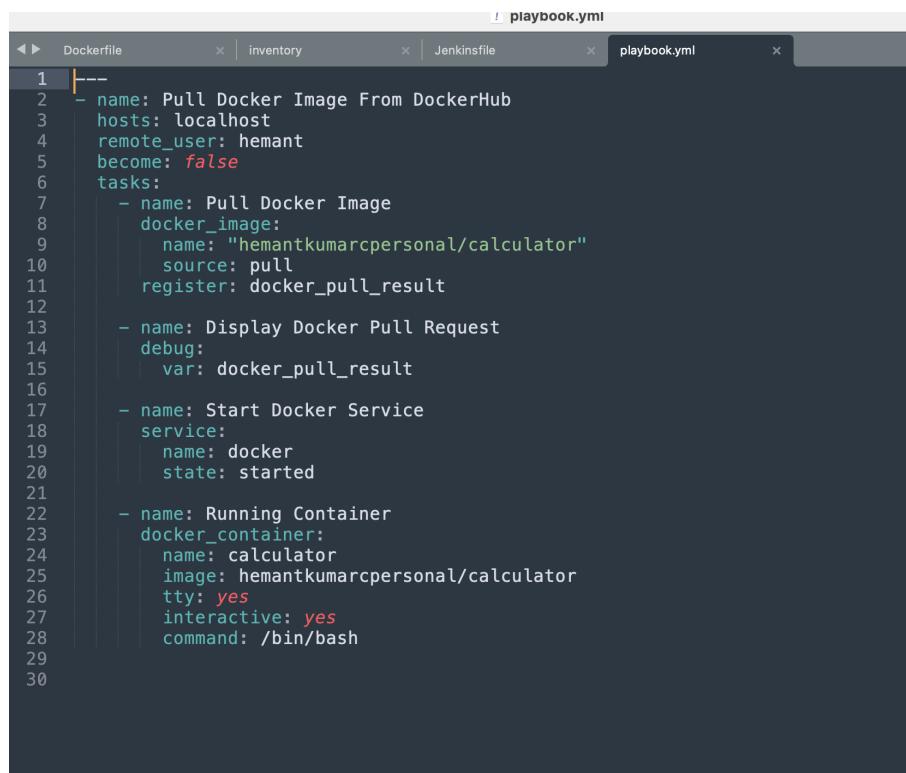
Ansible

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 files 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.

Tasks 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.



```
! playbook.yml
Dockerfile | inventory | Jenkinsfile | playbook.yml
1 ---  
2   - name: Pull Docker Image From DockerHub  
3     hosts: localhost  
4     remote_user: hemant  
5     become: false  
6     tasks:  
7       - name: Pull Docker Image  
8         docker_image:  
9           name: "hemantkumarcpersonal/calculator"  
10          source: pull  
11          register: docker_pull_result  
12  
13       - name: Display Docker Pull Request  
14         debug:  
15           var: docker_pull_result  
16  
17       - name: Start Docker Service  
18         service:  
19           name: docker  
20           state: started  
21  
22       - name: Running Container  
23         docker_container:  
24           name: calculator  
25           image: hemantkumarcpersonal/calculator  
26           tty: yes  
27           interactive: yes  
28           command: /bin/bash  
29  
30
```

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.

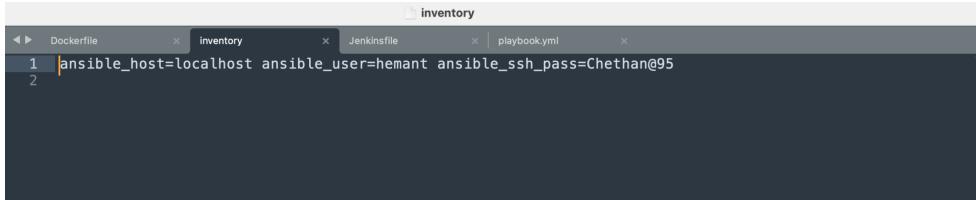
Display Docker Pull Request - This task utilizes the Ansible debug module to display the result of the Docker image pull operation. It helps provide insights into whether the image was successfully pulled or if there were any errors.

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.



The screenshot shows a terminal window with four tabs open: Dockerfile, inventory, Jenkinsfile, and playbook.yml. The inventory tab is active and displays the following content:

```
1 |ansible_host=localhost ansible_user=hemant ansible_ssh_pass=Chethan@95
2
```

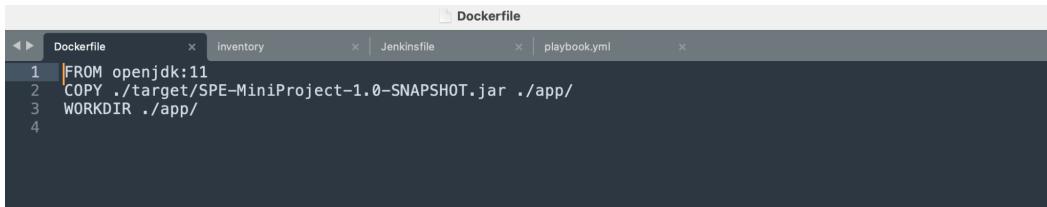
Here, the inventory file contains information about a single host, “localhost” and includes specific connection details using Ansible variables.

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



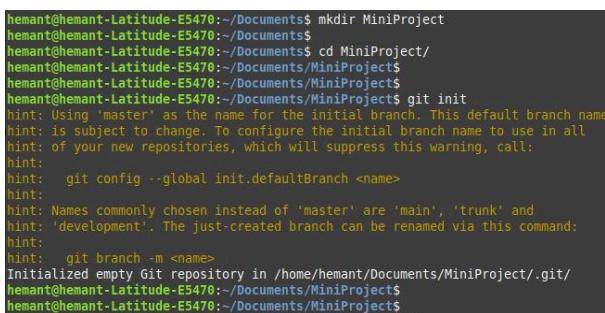
```
Dockerfile
FROM openjdk:11
COPY ./target/SPE-MiniProject-1.0-SNAPSHOT.jar ./app/
WORKDIR ./app/
```

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/MiniProjects
hemant@hemant-Latitude-E5470:~/Documents/MiniProjects$ 
hemant@hemant-Latitude-E5470:~/Documents/MiniProjects$ 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/MiniProjects
hemant@hemant-Latitude-E5470:~/Documents/MiniProjects$
```

```

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)
    .backgroundfile
    Jenkinsfile
    inventory
    playbook.yaml
    vars.yaml
    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**.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * <input type="text" value="hemantkumarc"/>	Repository name * <input type="text" value="MiniProject-Demo"/> ✓ MiniProject-Demo is available.
---	--

Great repository names are short and memorable. Need inspiration? How about [potential-octo-doodle](#) ?

Description (optional)

Visibility

- 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.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

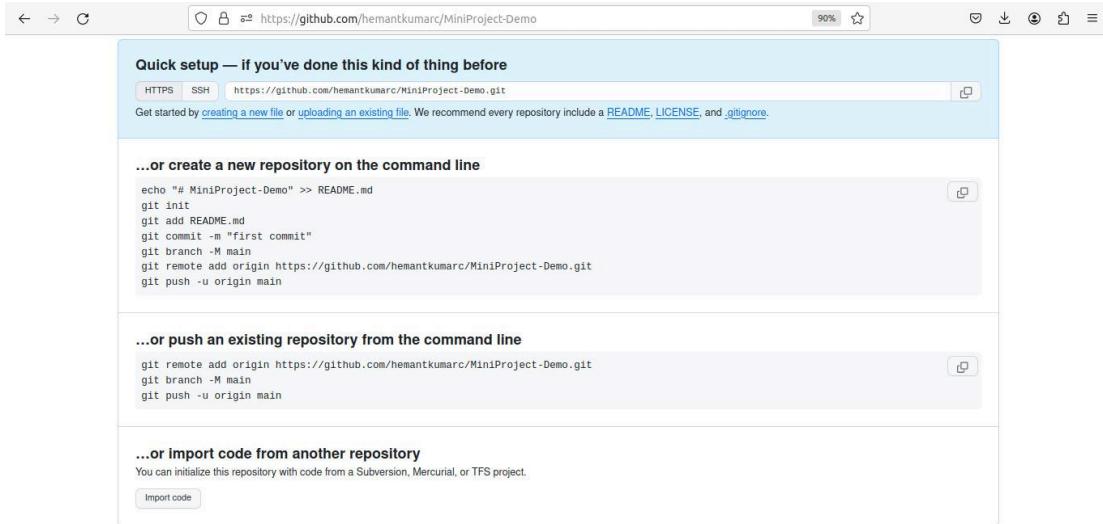
Choose a license

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

① You are creating a public repository in your personal account.

Create repository

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/MiniProjects$ 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/MiniProjects$ hemant@hemant-Latitude-E5470:~/Documents/MiniProjects$ git branch -M main
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ hemant@hemant-Latitude-E5470:~/Documents/MiniProjects$ 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
```

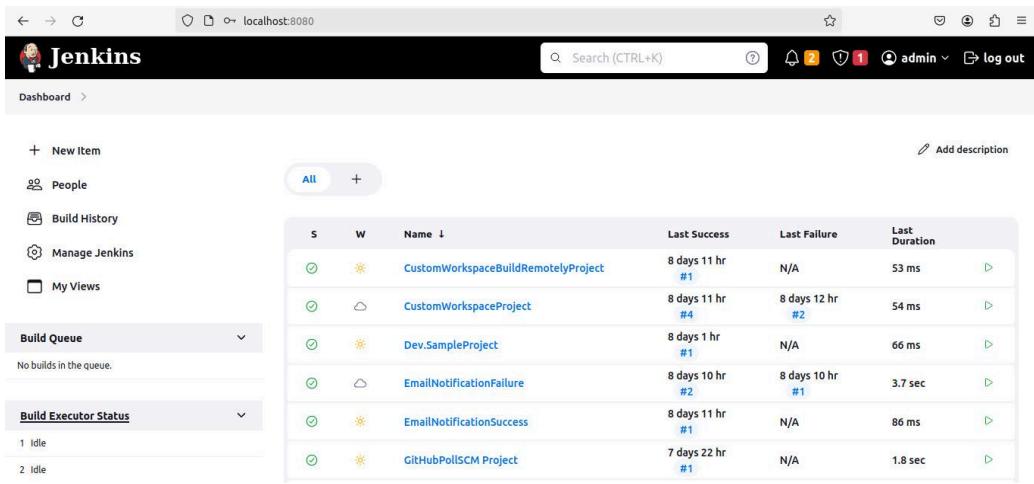
```
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$  
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$  
hemant@hemant-Latitude-E5470:~/Documents/MiniProject$ git push -u origin main  
Username for 'https://github.com': hemantkumarc  
Password for 'https://hemantkumarc@github.com':  
Enumerating 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.

The screenshot shows a GitHub repository page for 'MiniProject-Demo'. The 'Code' tab is selected. The repository has 1 branch and 0 tags. The 'Code' tab is active, displaying a list of files added by user 'hemantkumarc'. The files listed are: src, Dockerfile, Jenkinsfile, inventory, playbook.yml, and pom.xml. Each file entry includes a timestamp of '6 minutes ago'. On the right side of the page, there is an 'About' section stating 'This repository contains the work done as part of the mini project for CS 816 - Software Production Engineering'. Below it are sections for 'Activity', 'Releases', and 'Packages'.

Step 4.) Creating the Jenkins CI/CD Pipeline

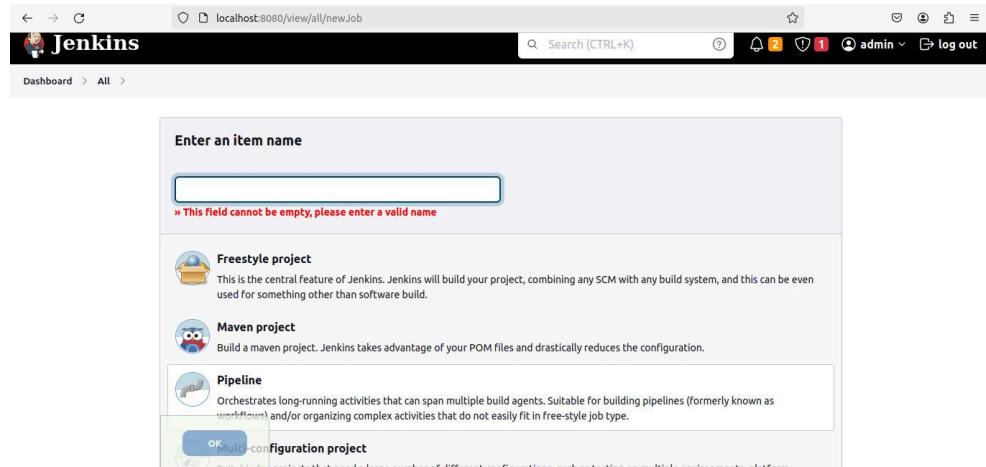
Now go to the Jenkins Dashboard page.



The screenshot shows the Jenkins dashboard at localhost:8080. The left sidebar includes links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Under 'Build Queue', it says 'No builds in the queue.' Below that is 'Build Executor Status' with 1 Idle and 2 Idle entries. The main area displays a table of build projects:

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	CustomWorkspaceBuildRemotelyProject	8 days 11 hr #1	N/A	53 ms
✓	☁️	CustomWorkspaceProject	8 days 11 hr #4	8 days 12 hr #2	54 ms
✓	☀️	Dev.SampleProject	8 days 1 hr #1	N/A	66 ms
✓	☁️	EmailNotificationFailure	8 days 10 hr #2	8 days 10 hr #1	3.7 sec
✓	☀️	EmailNotificationSuccess	8 days 11 hr #1	N/A	86 ms
✓	☀️	GithubPollSCM Project	7 days 22 hr #1	N/A	1.8 sec

Now click on **New Item**,



The screenshot shows the 'Enter an item name' dialog in Jenkins. It has a red error message: »This field cannot be empty, please enter a valid name«. Below the input field are three project types: 'Freestyle project', 'Maven project', and 'Pipeline'. The 'Pipeline' option is selected, with a note: '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.' At the bottom are 'OK' and 'Cancel' buttons.

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

The screenshot shows the Jenkins configuration interface for a project named "SPE Mini Project". The "General" tab is active. The "Enabled" checkbox is checked. The "Description" field is empty. Under "Advanced Project Options", there are three checkboxes: "Discard old builds", "Do not allow concurrent builds", and "Do not allow the pipeline to resume if the controller restarts". At the bottom are "Save" and "Apply" buttons.

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.

The screenshot shows the Jenkins Pipeline configuration interface. The "Pipeline" tab is active. In the "Repositories" section, the "Repository URL" is set to "https://github.com/hemantkumarc/MiniProject-Demo.git". In the "Branches to build" section, the "Branch Specifier" is set to "/main". In the "Additional Behaviours" section, "Clean before checkout" is selected. In the "Script Path" section, the "Script Path" is set to "Jenkinsfile". At the bottom are "Save" and "Apply" buttons.

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

The screenshot shows the Jenkins interface for the 'SPE Mini Project'. The top navigation bar includes links for 'Dashboard' and 'SPE Mini Project'. The main content area has a 'Status' tab selected, which displays the message: 'This pipeline demonstrates a full fledged CI/CD Pipeline using Docker and Ansible.' Below this, there are several actions: 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. To the right of these actions are buttons for 'Edit description' and 'Disable Project'. A 'Stage View' section shows a message: 'No data available. This Pipeline has not yet run.' At the bottom left is a 'Build History' section with a single entry: '#1 Feb 21, 2024, 3:38 PM' and 'No Changes'. The 'trend' dropdown is set to 'trend'.

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

The screenshot shows the Jenkins interface for the 'SPE Mini Project' build history. The top navigation bar includes links for 'Dashboard' and 'SPE Mini Project'. The main content area has a 'Status' tab selected, which displays the message: 'This pipeline demonstrates a full fledged CI/CD Pipeline using Docker and Ansible.' Below this, there are several actions: 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. To the right of these actions are buttons for 'Edit description' and 'Disable Project'. A 'Stage View' section shows a message: 'Average stage times: (Average full run time: ~2min 3s)'. It includes a table with columns for Declarative: Checkout SCM, Declarative: Tool Install, Clone Project, Build Code, Test Code, Building Docker Image, Pushing Docker Image, and Run Ansible Playbook. The first row shows times: 1s, 250ms, 3s, 45s, 27s, 5s, 19s, and 9s. The second row shows a timestamp: Feb 21 15:38 and 'No Changes'. Below the table is a 'Permalinks' section with links for 'Atom feed for all' and 'Atom feed for failures'. The 'Build History' section at the bottom shows a single entry: '#1 Feb 21, 2024, 3:38 PM' and 'No Changes'.

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/MiniProjects$ 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/MiniProjects$
```

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/MiniProjects$ 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/MiniProjects$
```

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/MiniProjects$ sudo docker start -i calculator
root@bc15c44966f0:/app# ls
SPE-MiniProject-1.0-SNAPSHOT.jar
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

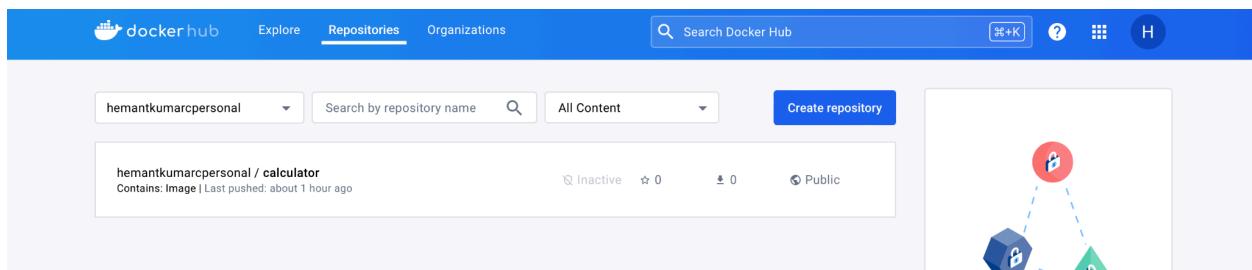
Enter Your Choice : 1
Enter a number : 4
Result = 2.0
-----
::: Calculator :::
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power Function
5. Exit

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>