DevOps

JENKINS

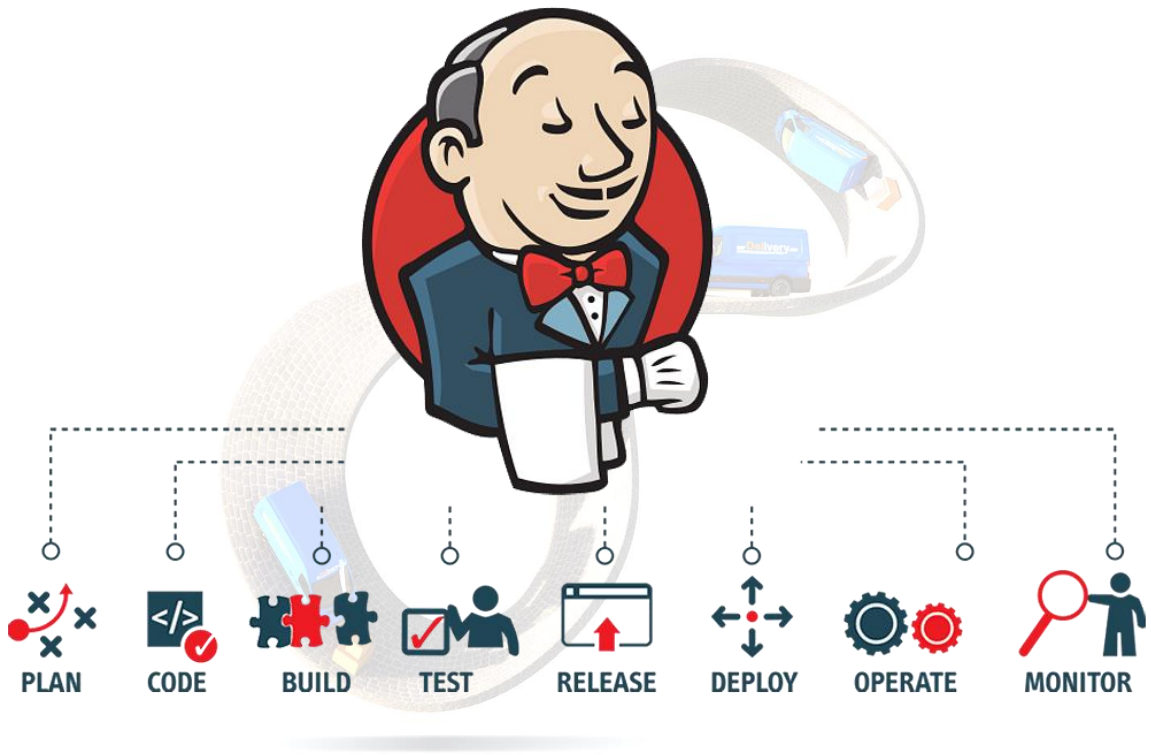
Understanding Jenkins pipelines
Jenkins Master-Slave Architecture
Role-Based Access Control
Jenkins Build Backup
Jenkins Custom workspace

SWIPE LEFT

CHARAN SURYA KILANA

Pipeline:

A pipeline refers to a sequence of events or processes that are interconnected, allowing for the systematic execution of tasks or operations. In the context of software development, a pipeline typically automates the steps involved in building, testing, and deploying code, ensuring a streamlined and efficient workflow.



What is a Jenkins file?

A Jenkins file is a text file that contains the definition of a Jenkins Pipeline, specifying the steps required to execute the pipeline process. It is used to automate the build, test, and deployment processes in a structured and repeatable manner. The Jenkins file utilizes Groovy syntax and can be written in two distinct styles:

1. **Declarative Pipeline**: This is a newer feature in Jenkins that simplifies pipeline syntax. The declarative pipeline is highly structured and easy to write, starting with the keyword pipeline. It allows defining the pipeline in sections like agent, stages, and steps, making it more intuitive and manageable.
2. **Scripted Pipeline**: This is a more traditional way of writing Jenkins pipelines. Scripted pipelines offer a broader range of complex scripting capabilities. It begins with the keyword node, indicating that the pipeline code is scripted inside a node block, providing greater control and flexibility over the pipeline execution.

Declarative

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {

            }
        }
        stage('Test') {
            steps {

            }
        }
        stage('Deploy') {
            steps {

            }
        }
    }
}
```

Scripted

```
node {
    stage('Build') {

    }
    stage('Test') {

    }
    stage('Deploy') {

    }
}
```

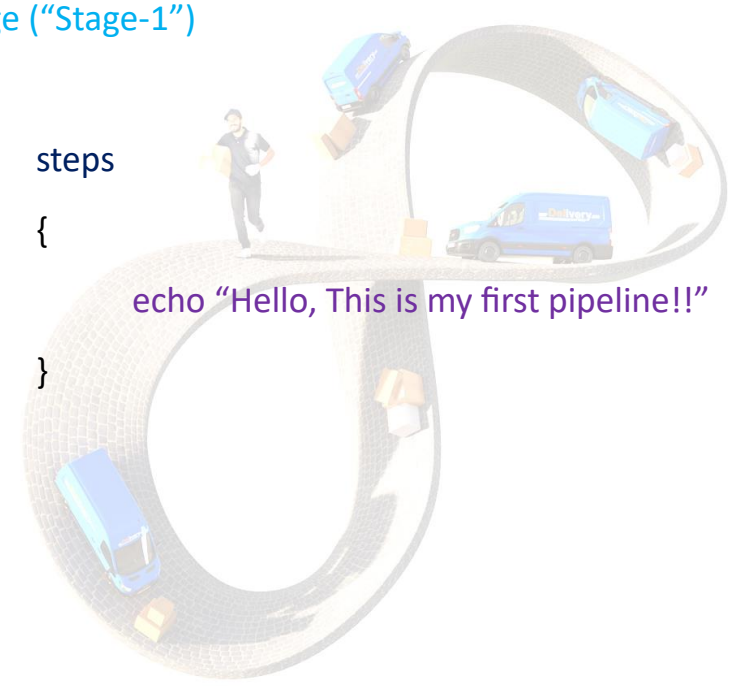

Both types of pipelines are stored in a Jenkins file, which is typically placed in the root of a project's source control repository, making it accessible for Jenkins to use during builds. This setup facilitates continuous integration and delivery by allowing all pipeline configurations to be version controlled along with the application code.

In this series, we will focus on learning more about the Declarative Pipeline approach, exploring how it simplifies the Jenkins automation process with its straightforward and easy-to-understand syntax.

Declarative pipeline:

Single Stage pipeline:

```
pipeline
{
    agent any
    stages
    {
        stage ("Stage-1")
        {
            steps
            {
                echo "Hello, This is my first pipeline!!"
            }
        }
    }
}
```



Pipeline: A Pipeline in Jenkins is a user-defined model of a continuous delivery (CD) pipeline. The code within a Pipeline defines the entire build process, which typically includes stages for building an application, testing it, and delivering it.

Agent: The agent directive specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment. This determines on which server or node the Pipeline runs.

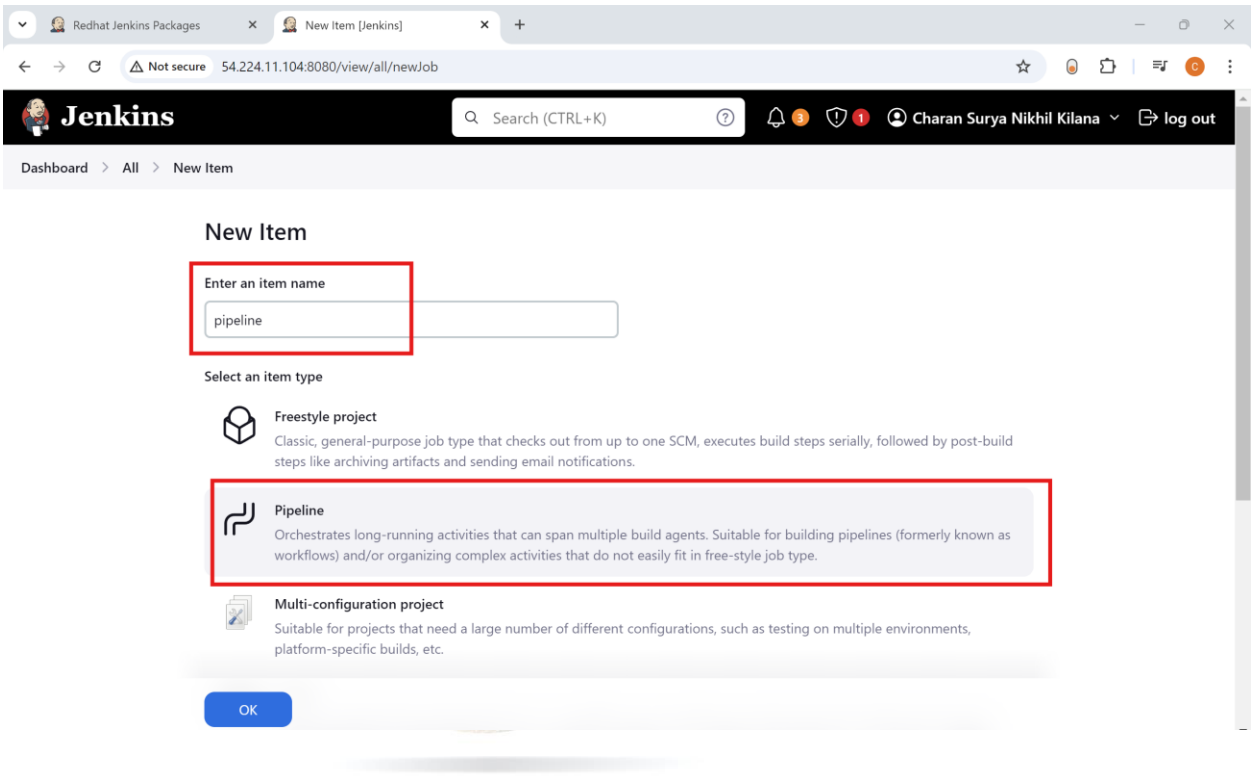
Stage: A stage in a Jenkinsfile represents a logically distinct part of the Pipeline process, such as building, testing, or deploying the application. Each stage contains steps that execute specific tasks required at that point in the Pipeline.

Steps: A step is a single task that tells Jenkins exactly what to do at a particular point in time, such as executing a build script or a Linux command.

Creating a job with pipeline

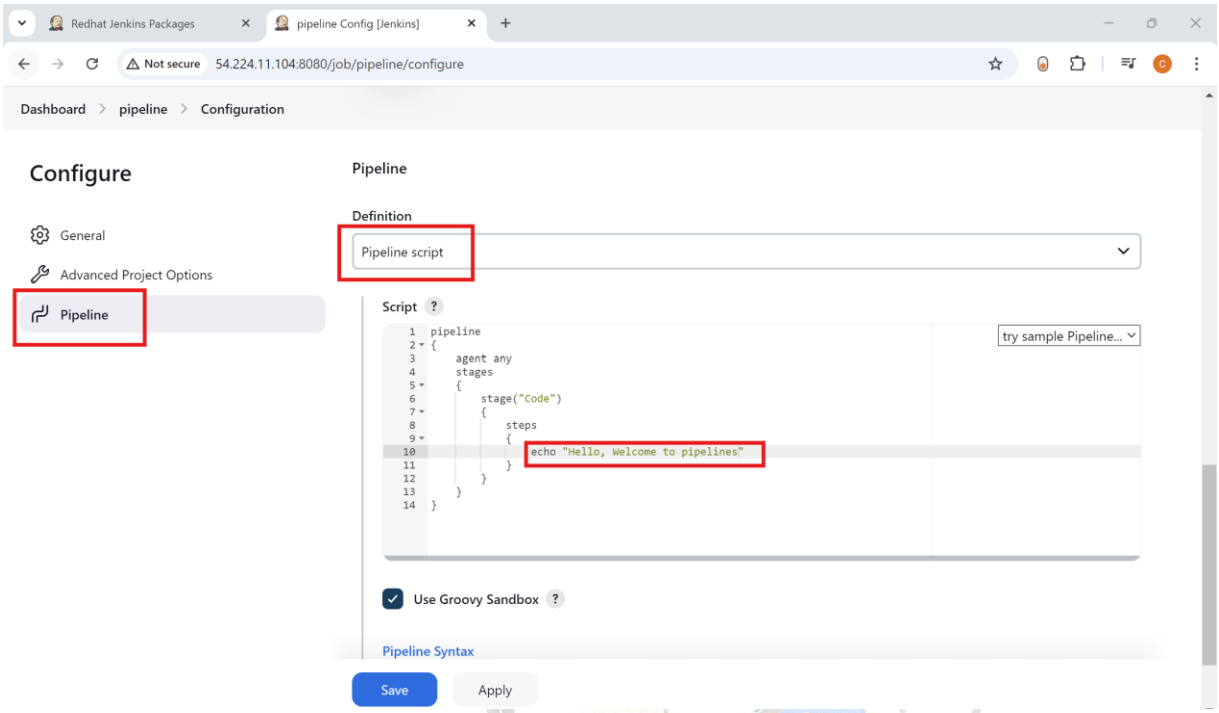
Freestyle is just like a template where you select options and build. However, in the pipeline you're supposed to write a code.

Step-1: Create a **job** with **pipeline**



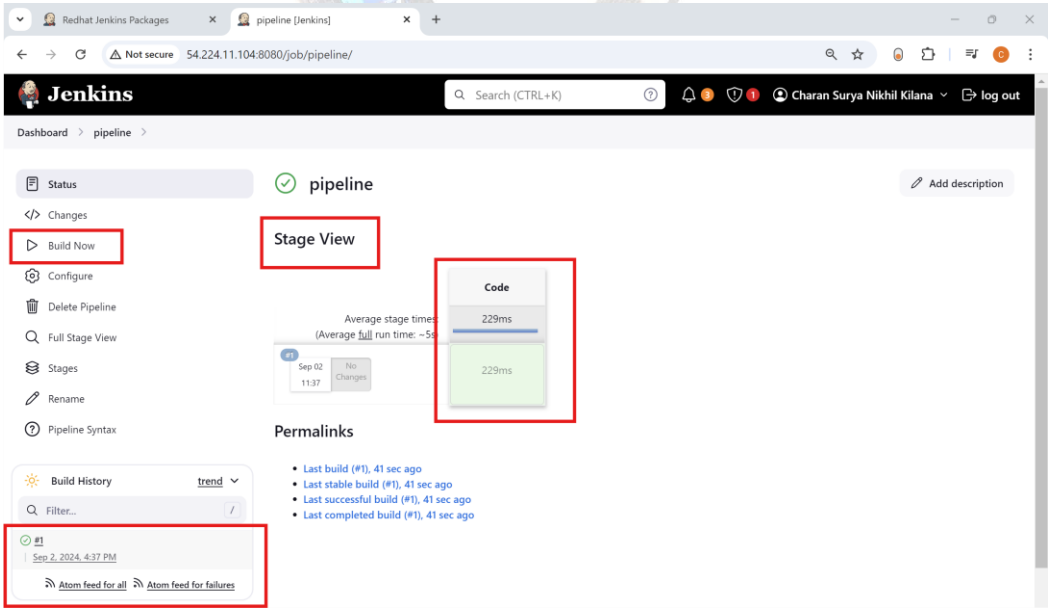
Select Pipeline:

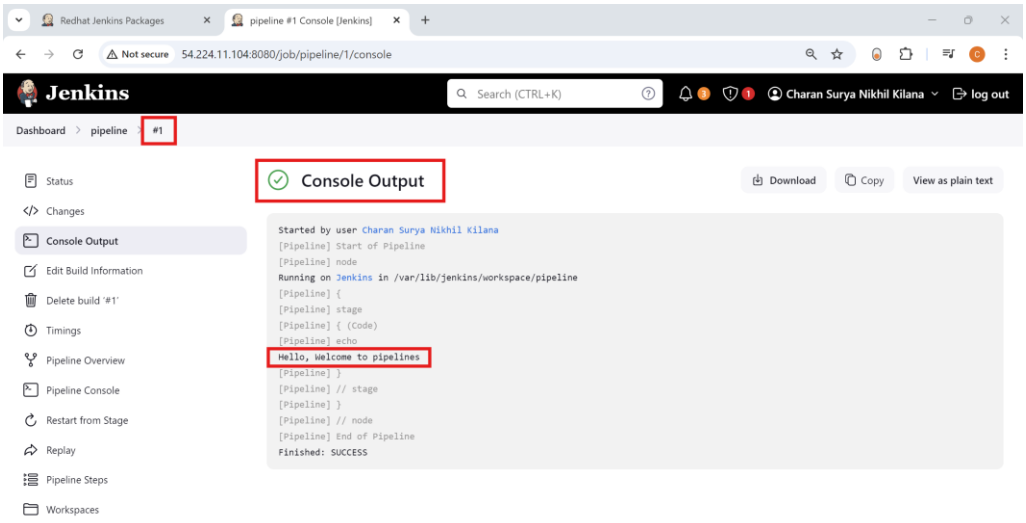
- To print a message within a pipeline, use the echo command in your Jenkins file. This allows for simple text output during the execution of the pipeline, providing status updates or debugging information.



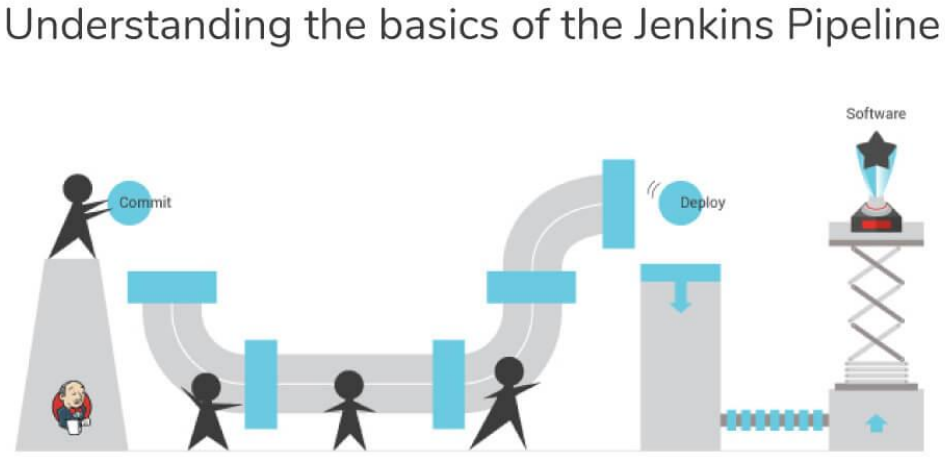
Since we have one stage, we get only code after the build.

If you do not see the stage view in Jenkins, you may need to install the "Pipeline Stage View Plugin." This plugin provides a graphical view of the pipeline stages and their execution status, enhancing the visibility of the pipeline's progress.





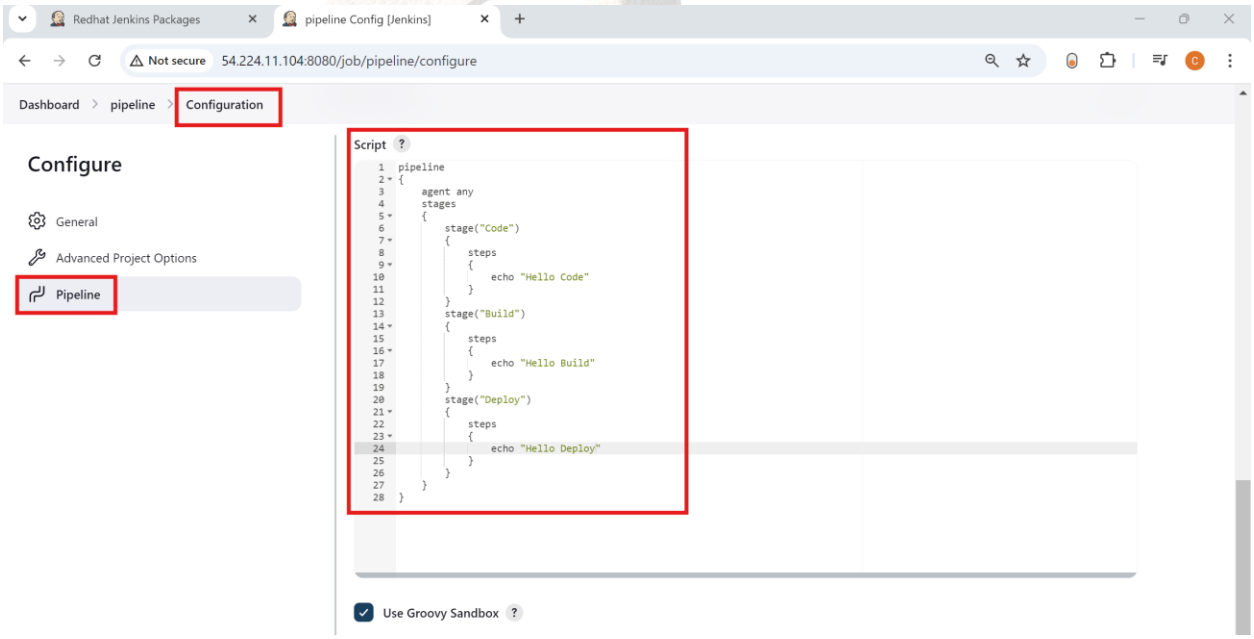
Multi-Stage-Pipeline

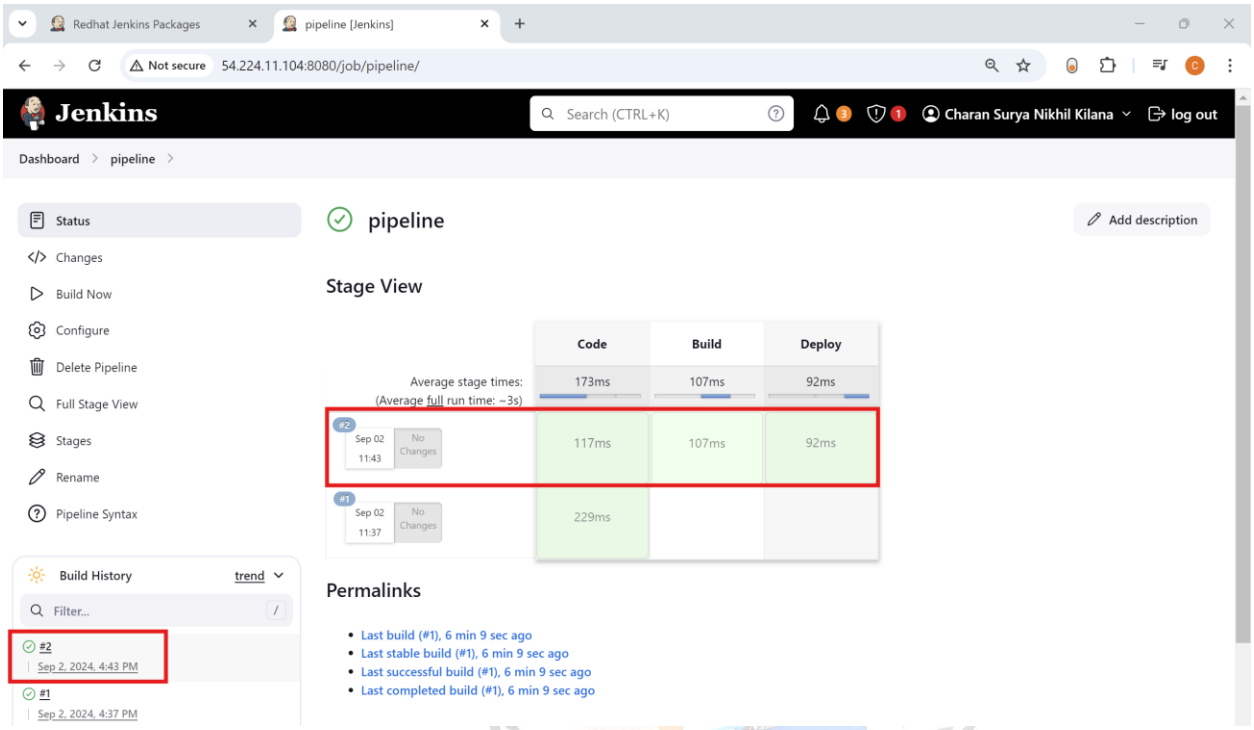


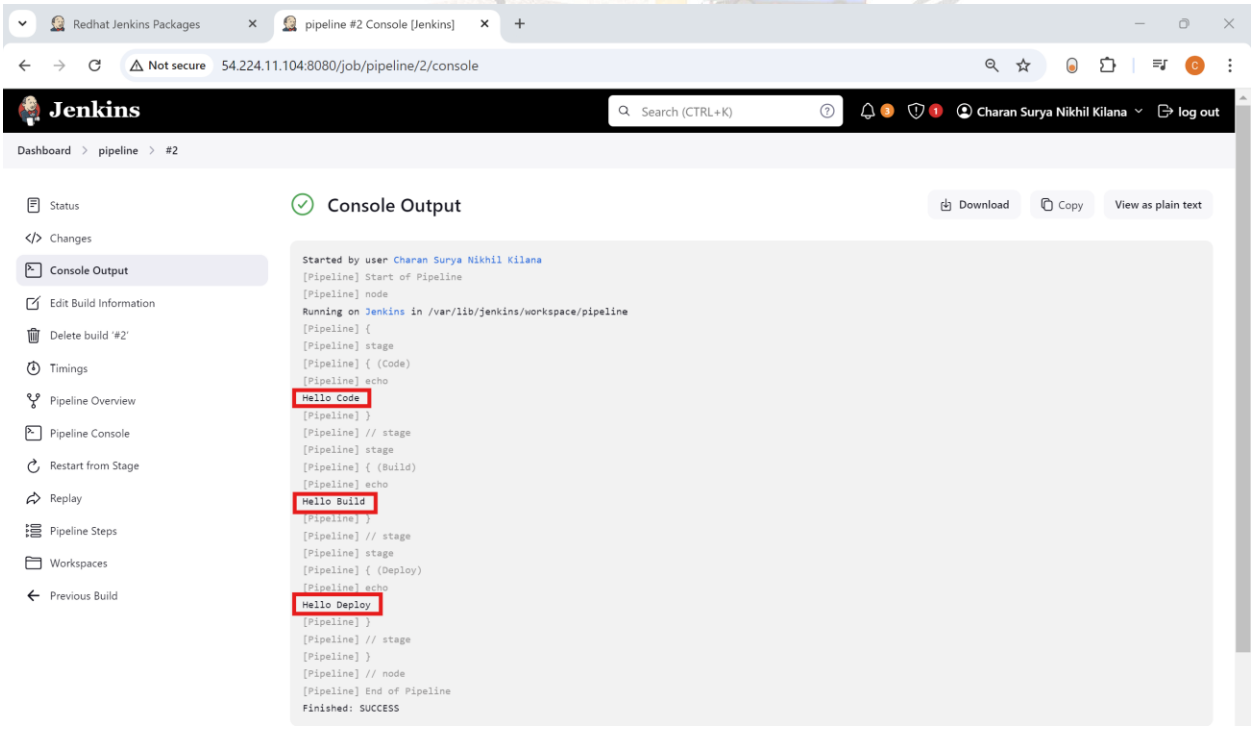
Jenkins

Script ?

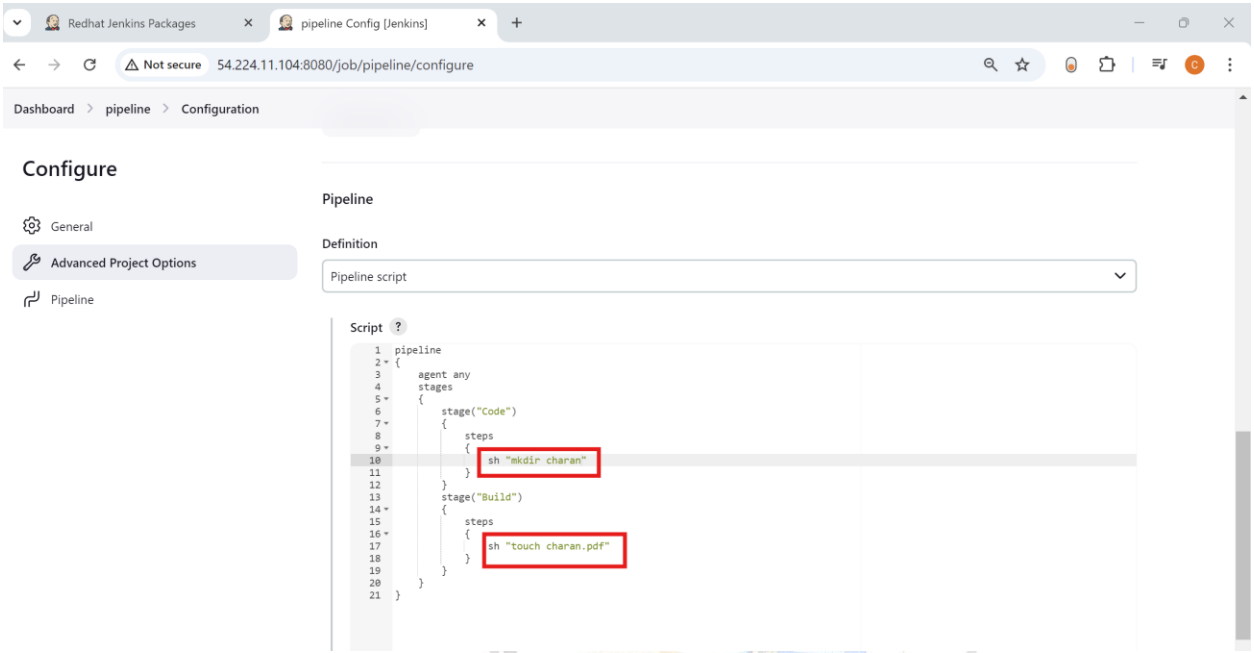
```
pipeline
{
    agent any
    stages
    {
        stage("Code")
        {
            steps
            {
                echo "Hello Code"
            }
        }
        stage("Build")
        {
            steps
            {
                echo "Hello Build"
            }
        }
        stage("Deploy")
        {
            steps
            {
                echo "Hello Deploy"
            }
        }
    }
}
```

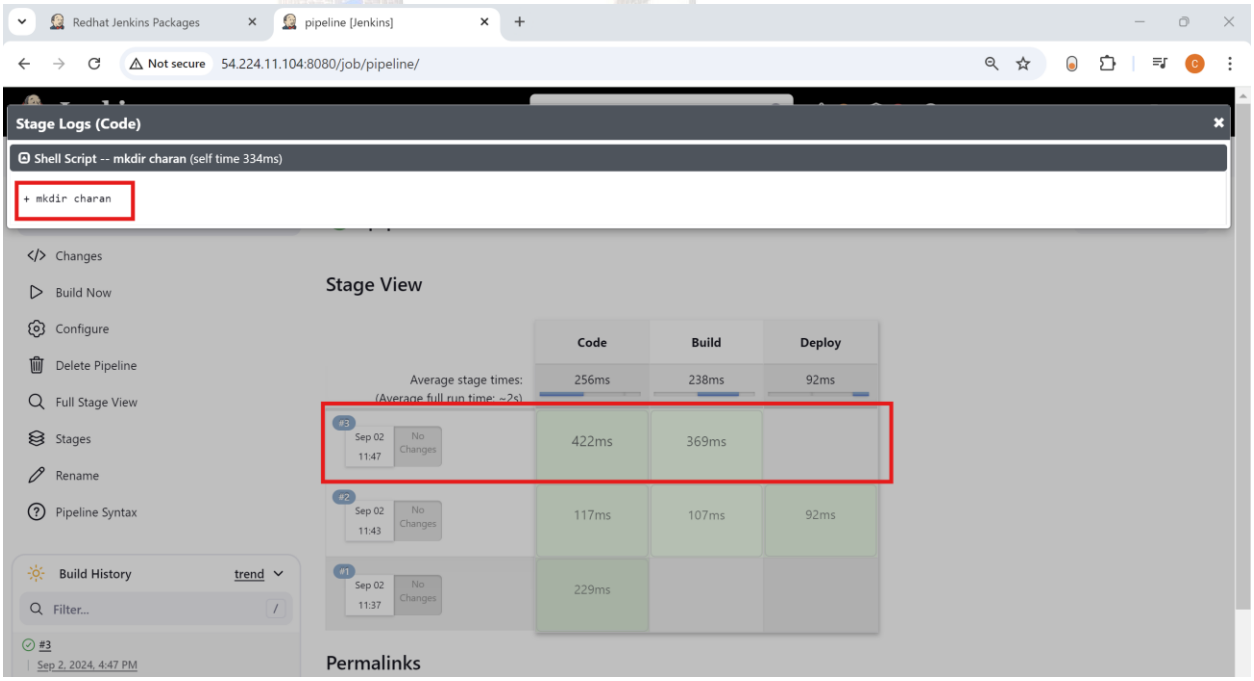

Redhat Jenkins Packages
pipeline [Jenkins]
Not secure
54.224.11.104:8080/job/pipeline/
Jenkins
Search (CTRL+K)
Charan Surya Nikhil Kilana
log out
Dashboard
pipeline
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Stages
Rename
Pipeline Syntax
Build History
trend
Filter...
#2
Sep 2, 2024, 4:43 PM
#1
Sep 2, 2024, 4:37 PM
pipeline
Add description
Stage View
Code
Build
Deploy
Average stage times:
(Average full run time: ~3s)
173ms
107ms
92ms
#2
Sep 02
11:43
No Changes
117ms
107ms
92ms
#1
Sep 02
11:37
No Changes
229ms
Permalinks
Last build (#1), 6 min 9 sec ago
Last stable build (#1), 6 min 9 sec ago
Last successful build (#1), 6 min 9 sec ago
Last completed build (#1), 6 min 9 sec ago


Redhat Jenkins Packages
pipeline #2 Console [Jenkins]
Not secure
54.224.11.104:8080/job/pipeline/2/console
Jenkins
Search (CTRL+K)
Charan Surya Nikhil Kilana
log out
Dashboard
pipeline
#2
Status
Changes
Console Output
Edit Build Information
Delete build '#2'
Timings
Pipeline Overview
Pipeline Console
Restart from Stage
Replay
Pipeline Steps
Workspaces
Previous Build
Console Output
Download
Copy
View as plain text
Started by user Charan Surya Nikhil Kilana
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Code)
[Pipeline] echo
Hello Code
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
Hello Build
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Hello Deploy
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

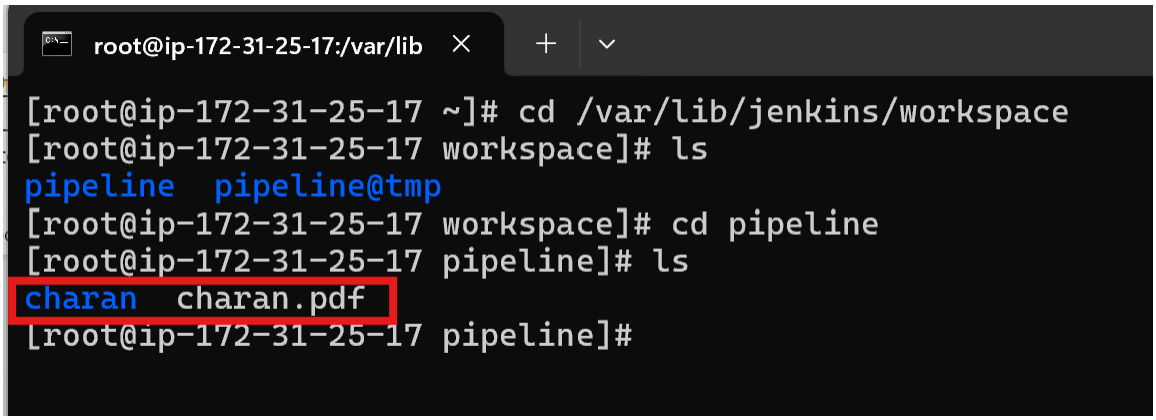
To execute a command instead of **echo,** we use **sh “command”.**



If you check the logs for code, you’ll get mkdir charan.



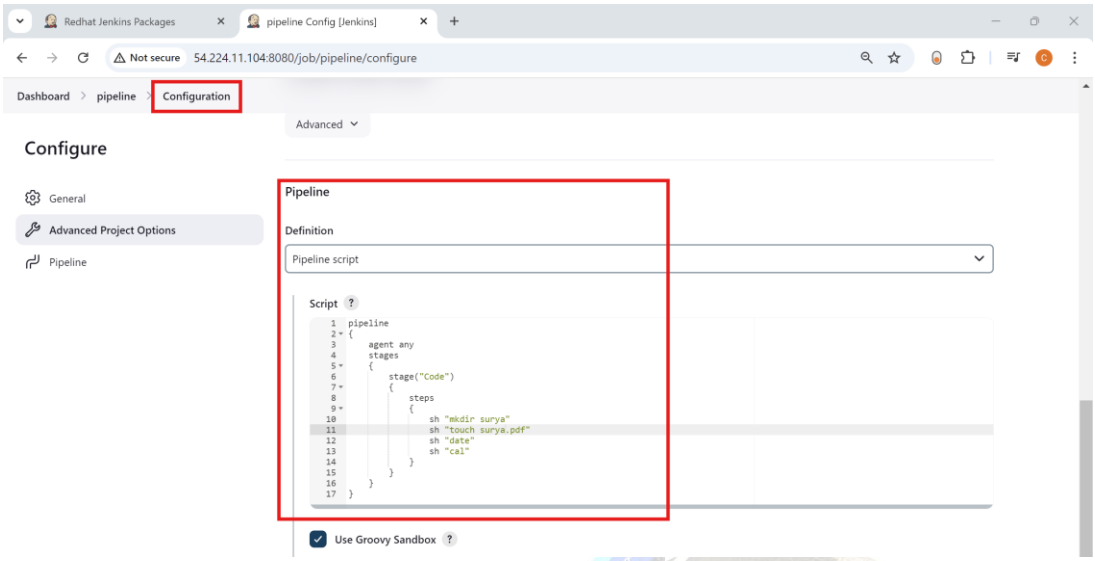
If you check in the terminal in the job, you'll have one file and folder created.

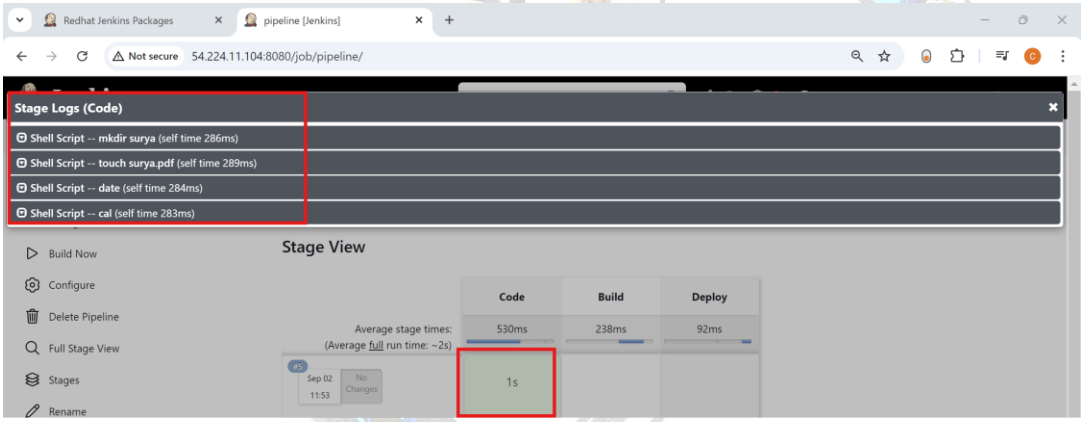


Multiple commands over a single line in a pipeline stage

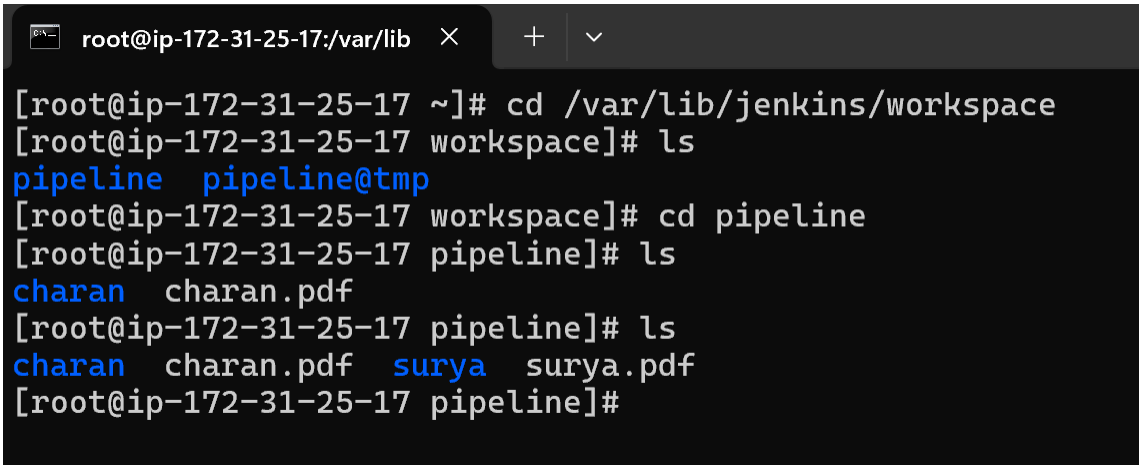
Script ?

```
pipeline
{
    agent any
    stages
    {
        stage("Code")
        {
            steps
            {
                sh "mkdir surya"
                sh "touch surya.pdf"
                sh "date"
                sh "cal"
            }
        }
    }
}
```





Now one more file and folder created in the job.



Declaring Variables

To utilize variables within a Jenkins Pipeline, they should be stored in an environment block. Variables declared in this manner are considered global variables, accessible throughout the entire pipeline.

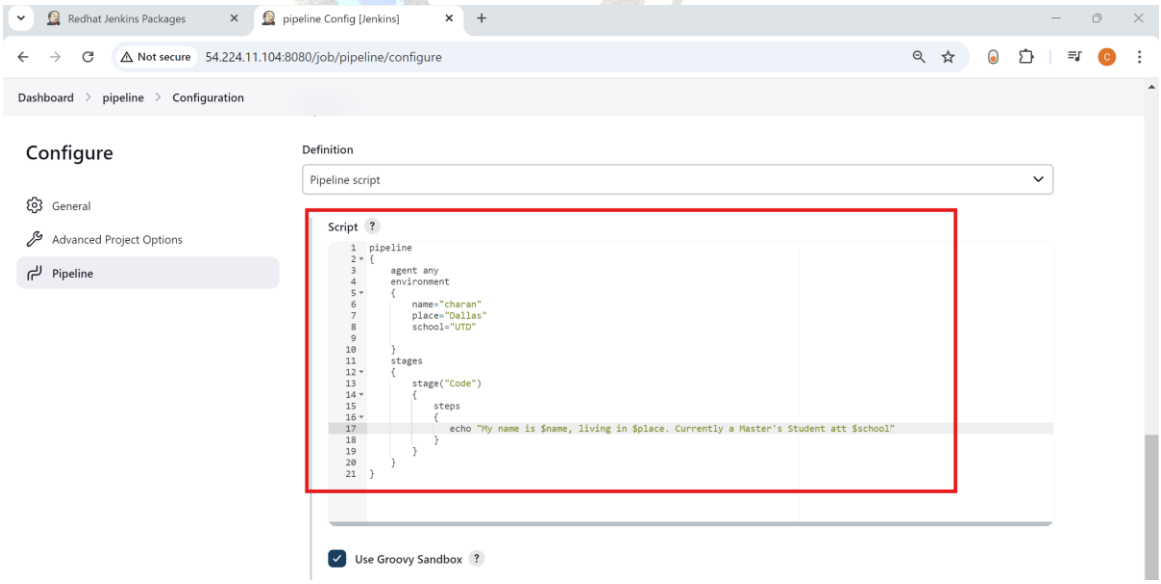
Syntax:

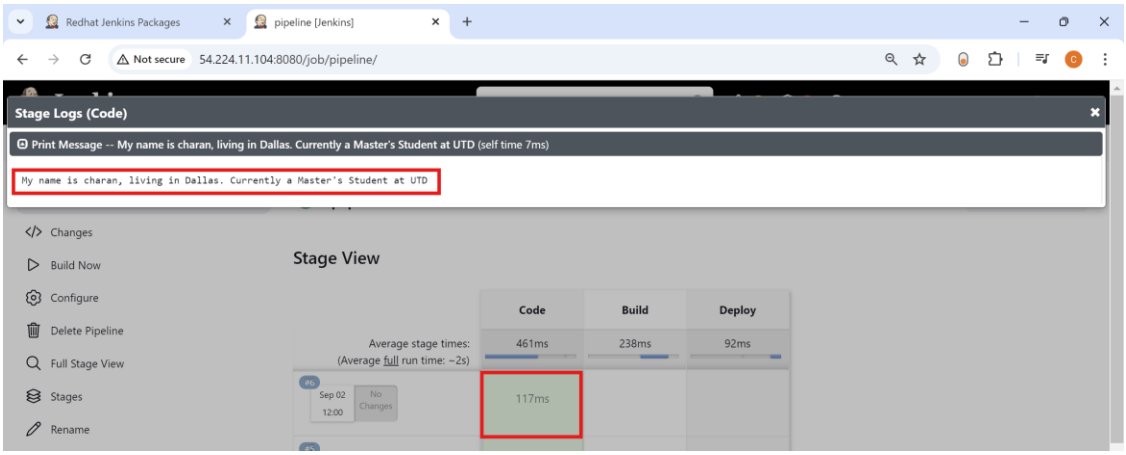
Definition

Pipeline script

Script ?

```
pipeline
{
    agent any
    environment
    {
        name="charan"
        place="Dallas"
        school="UTD"

    }
    stages
    {
        stage("Code")
        {
            steps
            {
                echo "My name is $name, living in $place. Currently a Master's Student att $school"
            }
        }
    }
}
```



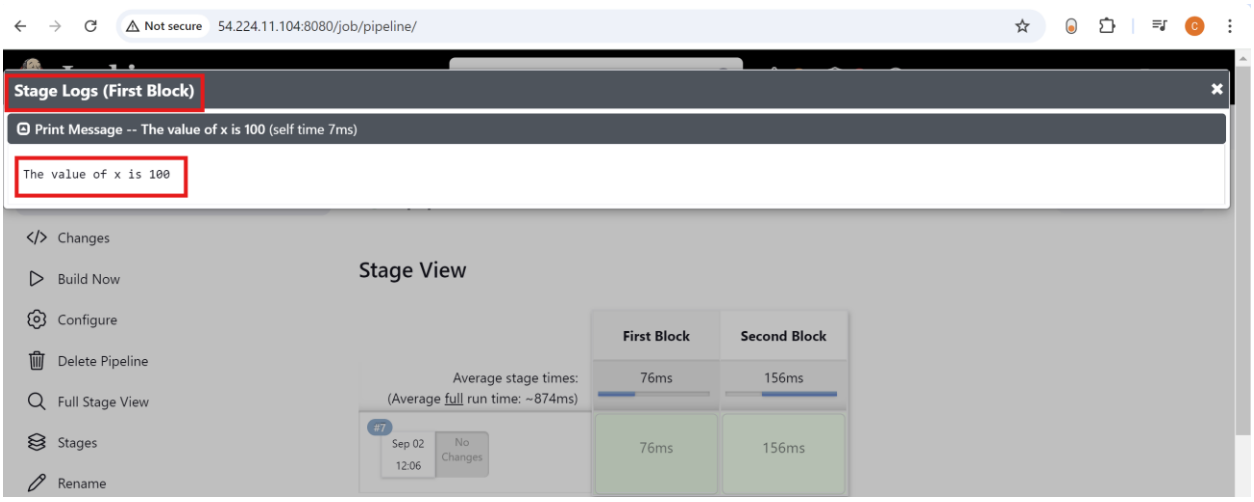


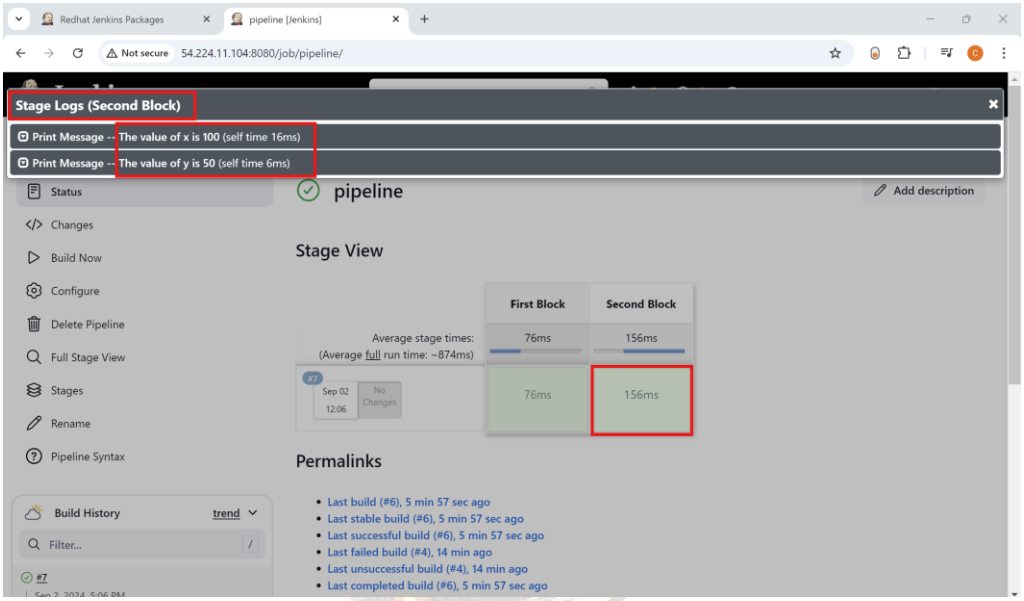
Variables in Jenkins Pipeline are of two types:

1. **Local Variables**: These are defined within a specific stage or block and are only accessible within that scope.
2. **Global Variables**: These are defined at the pipeline level in an environment block and can be accessed throughout the entire pipeline.

Script ?

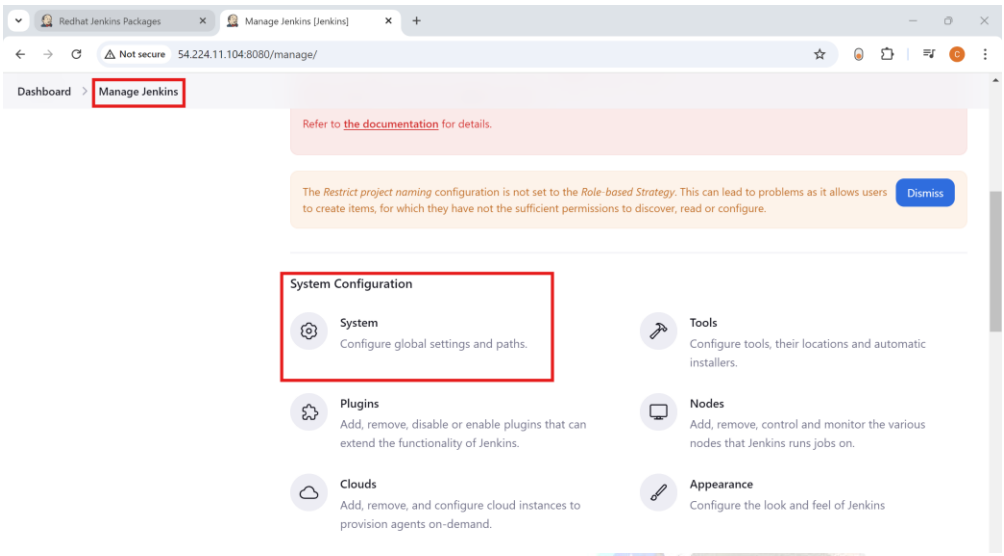
```
pipeline
{
    agent any
    environment
    {
        x=100

    }
    stages
    {
        stage("First Block")
        {
            steps
            {
                echo "The value of x is $x"
            }
        }
        stage("Second Block")
        {
            environment
            {
                y=50
            }
            steps
            {
                echo "The value of x is $x"
                echo "The value of y is $y"
            }
        }
    }
}
```

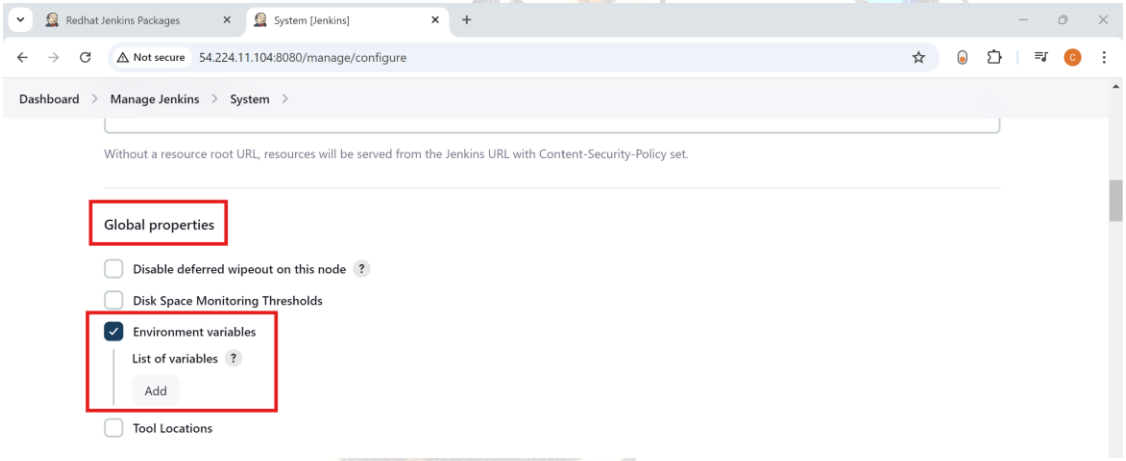


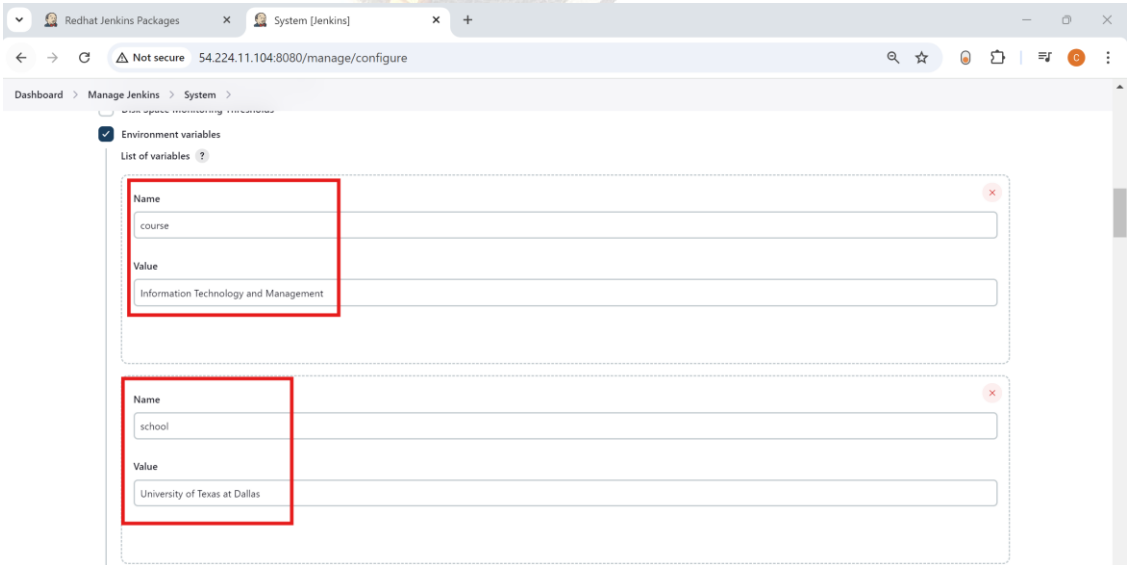


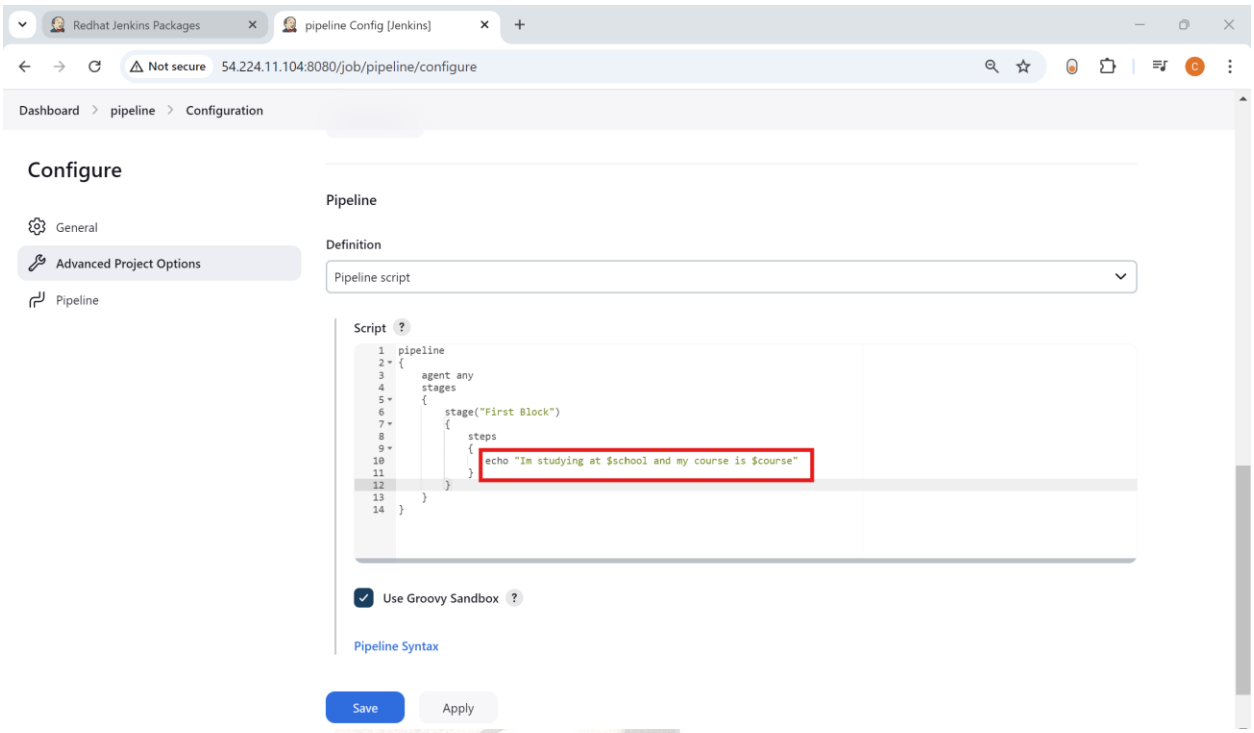
Accessing a Variable in Different Pipelines:

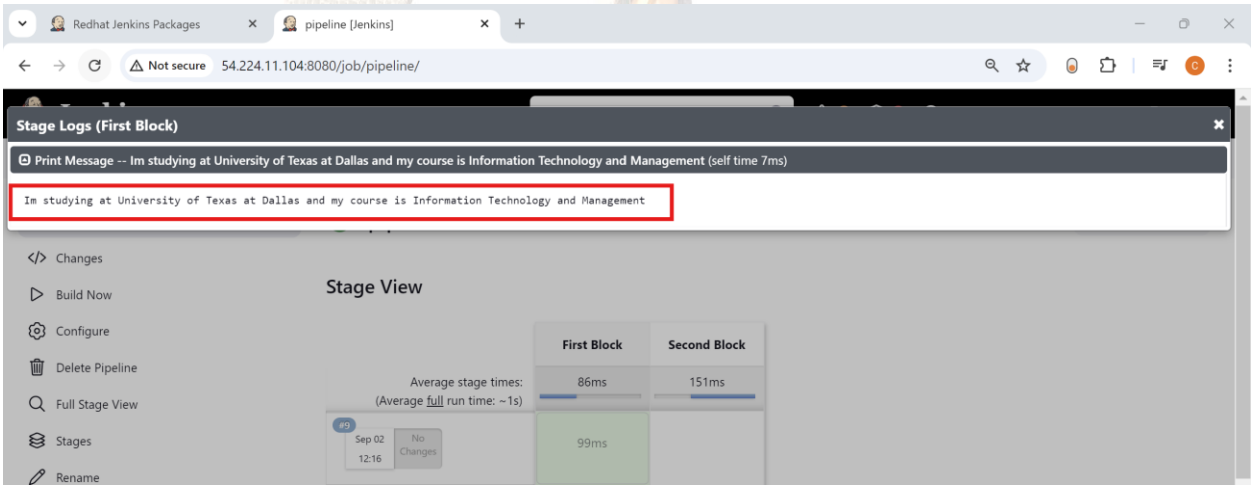
1. Navigate to **Manage Jenkins** from the Jenkins Dashboard.
2. Click on **Configure System**.
3. Scroll down to find **Global properties**.
4. Check the box next to **Environment variables** to enable it.
5. Click **Add** to create a new environment variable. Enter the variable name and its value.
6. Click **Save** to apply the changes.


Redhat Jenkins Packages
Manage Jenkins [Jenkins]
Not secure
54.224.11.104:8080/manage/
Dashboard
Manage Jenkins
Refer to the documentation for details.
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.
Dismiss
System Configuration
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.
Nodes
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Clouds
Add, remove, and configure cloud instances to provision agents on-demand.
Appearance
Configure the look and feel of Jenkins


Redhat Jenkins Packages
System [Jenkins]
Not secure
54.224.11.104:8080/manage/configure
Dashboard
Manage Jenkins
System
Without a resource root URL, resources will be served from the Jenkins URL with Content-Security-Policy set.
Global properties
Disable deferred wipeout on this node
Disk Space Monitoring Thresholds
Environment variables
List of variables
Add
Tool Locations


Redhat Jenkins Packages
System [Jenkins]
Not secure
54.224.11.104:8080/manage/configure
Dashboard
Manage Jenkins
System
Environment variables
List of variables
Name
course
Value
Information Technology and Management
Name
school
Value
University of Texas at Dallas





Note: To list all environment variables, you can use the command sh "printenv" in your pipeline script. This will display all available variables, including the defaults set by Jenkins. Look through the list to identify and confirm the variables you have defined.

Pipeline with Parameters:

To use parameters in a Jenkins pipeline:

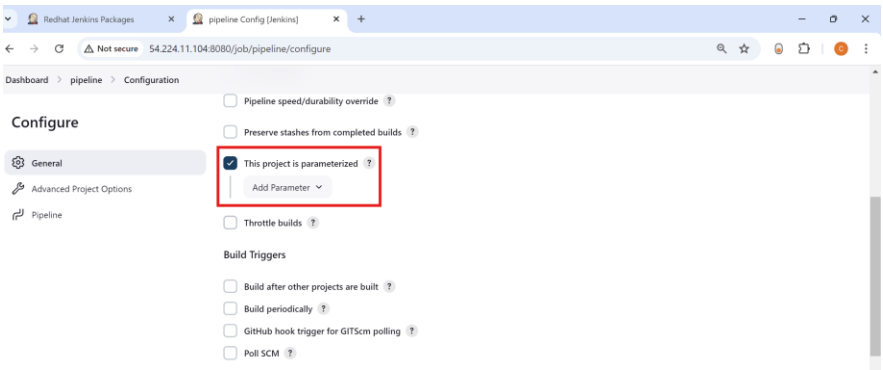
1. **Define Parameters**: After the agent any line in your pipeline script, add parameters block to specify the types of parameters you need.
2. **UI Setup**: For configuration via the Jenkins UI, navigate to **Configure** on your job page, and under **This project is parameterized**, add and configure your parameters.

String Parameter:

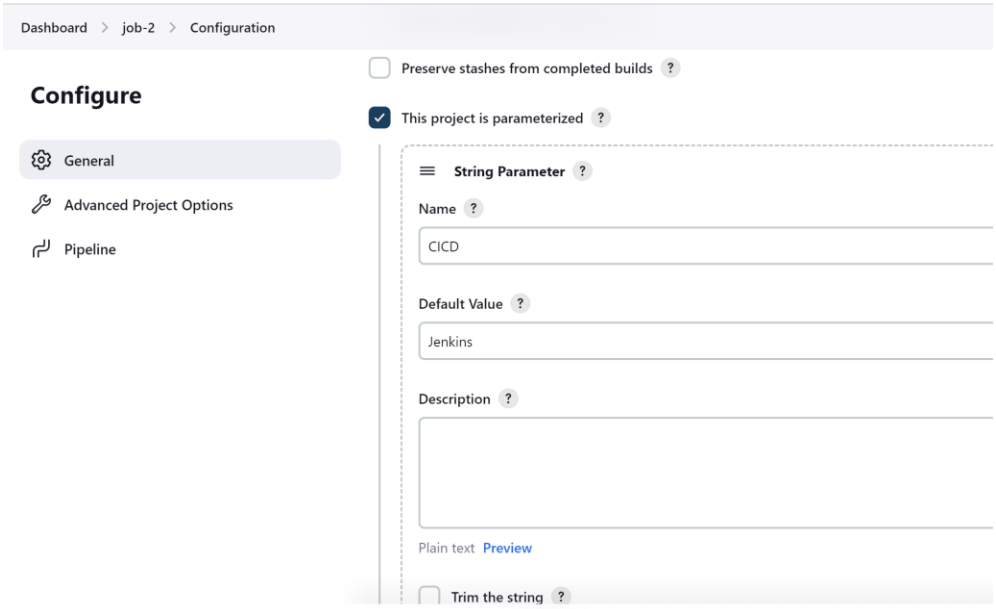
- **name:** This is the key used to access the value of the parameter within the pipeline scripts.
- **Declaration**: When declaring a string parameter, ensure the property for the default value is capitalized (i.e., defaultValue).
- **Description**: The description is optional. If no description is needed, use empty quotes.

Script ?

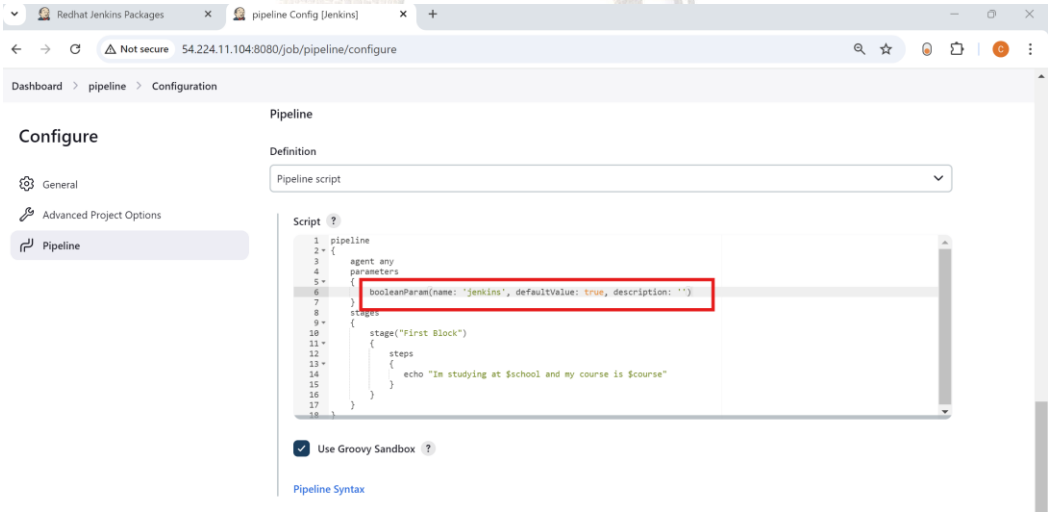
```
pipeline
{
    agent any
    parameters
    {
        string(name: 'CICD', defaultValue: 'Jenkins', description: '')
    }
    stages
    {
        stage ("Code")
        {
            steps
            {
                sh "cal"
            }
        }
    }
}
```



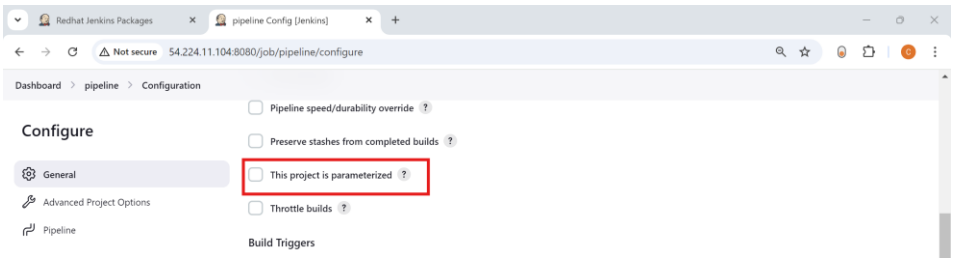
Before it was empty, later got filled with parameters.



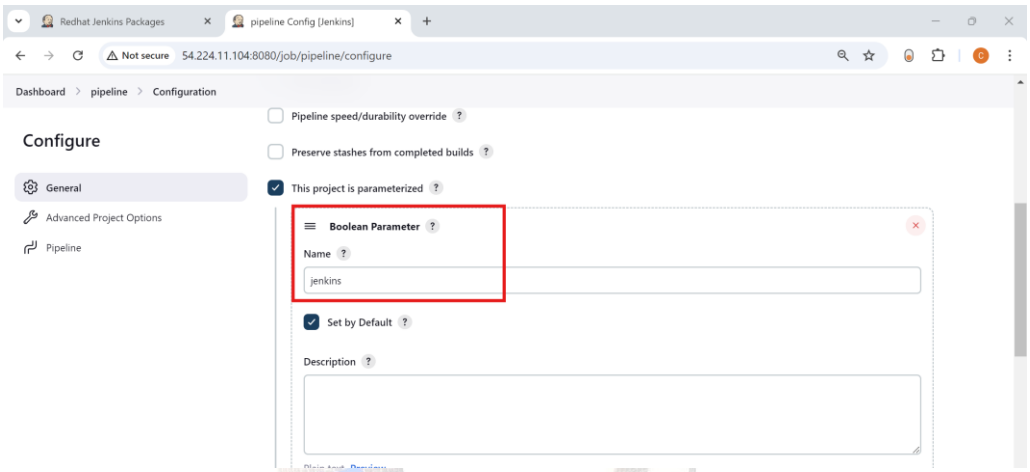
Boolean Parameter



Before build



After Build



Choice Parameter:

When declaring a choice parameter, use square brackets to list the options. Each option should be enclosed in quotes and separated by commas.

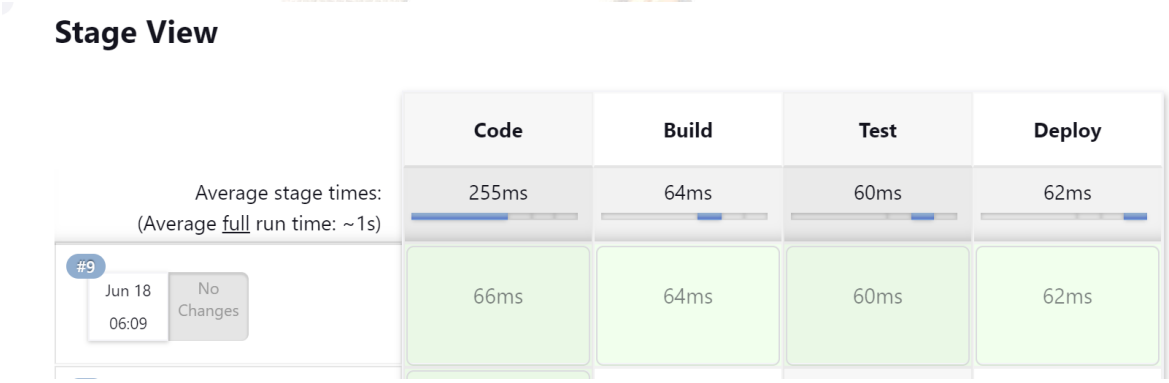
Script ?

```
pipeline
{
    agent any
    parameters
    {
        choice(name: 'CICD', choices: ['Jenkins', 'bamboo', 'circle-ci', 'aws code pipeline'], description: '')
    }
    stages
    {
        stage ("Code")
        {
            steps
            {
                sh "cal"
            }
        }
    }
}
```

After Build

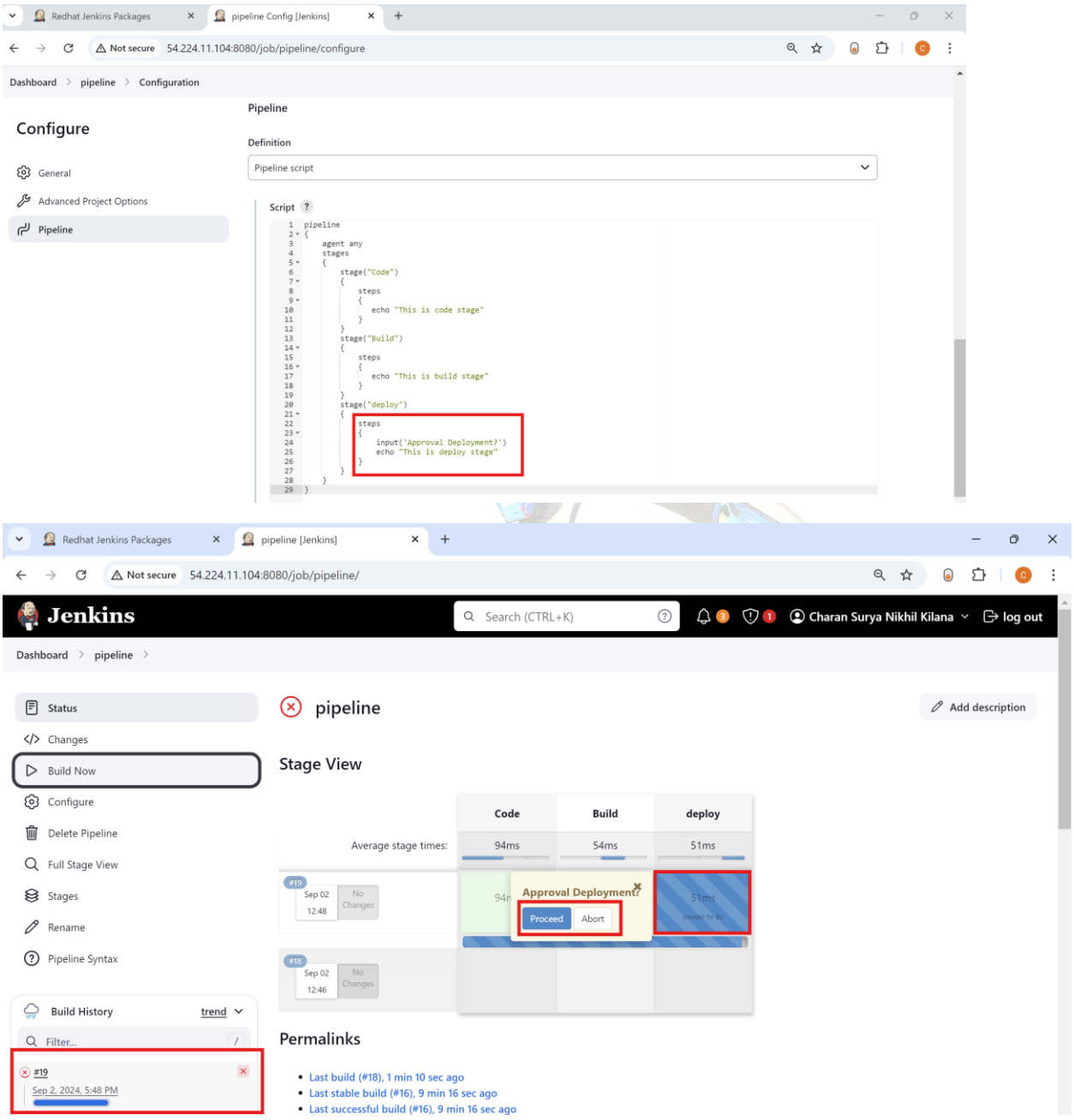
In real-time scenarios, continuous deployment processes typically automate code updates, building, and testing. However, before deploying to production, a manual approval is often required to ensure quality and stability. This approval step can be integrated into the deployment pipeline using an 'input' step that holds the process until it receives the necessary authorization.

The screenshot below is without approval.



Holding Deployment with an Input Message in a Jenkins file:

To control the deployment process and ensure that it only proceeds with explicit approval, you can introduce an input step in the deployment stage of your Jenkins file. This step will prompt users to either proceed with the deployment or abort it based on their decision.



During the deployment stage in Jenkins, an approval prompt appears. To continue, click **Proceed**; this allows the pipeline to complete successfully. If you wish to halt the deployment, click **Abort** to stop the pipeline.

Pipeline with Post-Build Actions: Post-build actions in Jenkins are used to perform tasks after the main build process completes, often to provide status updates or handle cleanup activities.

Note:

- **Tips**: Post-build actions can also help troubleshoot, like identifying a scrum error due to network issues by analyzing various sources across the web.
- To deliberately pause the pipeline for 30-40 seconds, include a sleep command in the steps: sleep 30 or sleep 40. This can be useful for ensuring that all resources are properly released before proceeding or for debugging purposes.

Success

If the pipeline execution succeeds, a success message will be displayed. This provides immediate feedback on the status of the deployment process.

Script ?

```
pipeline
{
    agent any
    stages
    {
        stage ("Code")
        {
            steps
            {
                echo "This is code stage"
            }
        }
    }
    post
    {
        success
        {
            echo "pipeline is successful"
        }
    }
}
```

Here If pipeline successful you get declarative post actions successful.

	Code	Declarative: Post Actions
Average stage times: (Average full run time: ~751ms)	64ms	60ms
#17 Jun 18 06:35 No Changes	65ms	60ms

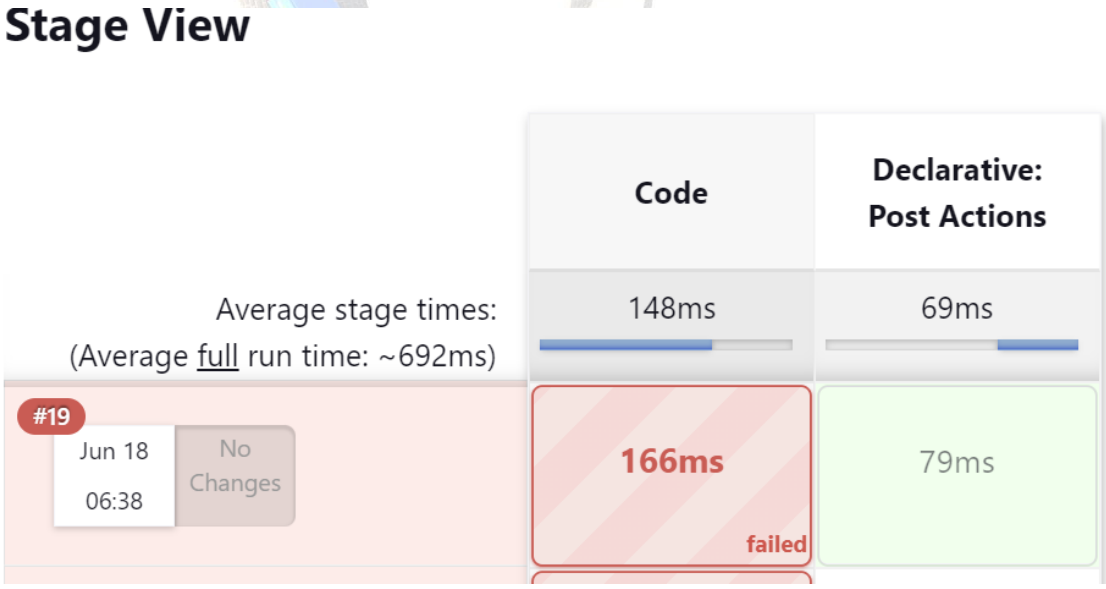
Failure

In the declarative post actions, it is marked as success because the pipeline is configured to trigger a failure message in the post actions section. Also, there's a typo in the 'echo' command.

Script ?

```
pipeline
{
    agent any
    stages
    {
        stage ("Code")
        {
            steps
            {
                eco "This is code stage"
            }
        }
    }
    post
    {
        failure
        {
            echo "pipeline is failed"
        }
    }
}
```

Below is the stage view. Since your code failed you get failure displayed.



Always: Use this to display a message regardless of whether the pipeline fails, succeeds, or is aborted.

Script ?

```
pipeline
{
    agent any
    stages
    {
        stage ("Code")
        {
            steps
            {
                echo "This is code stage"
            }
        }
    }
    post
    {
        failure
        {
            echo "pipeline is failed"
        }
        success
        {
            echo "This will be printed if pipeline success"
        }
        always
        {
            echo "I print always"
        }
    }
}
```

If successful, three messages are printed: 'code stage', 'always', and 'success'. If it fails, two messages are printed: 'failure' and 'always'. The 'always' message is common to both outcomes.

Additional topics:

Changed:

This block runs when the current status is different than the previous one.

post

{

changed

{

echo "This block runs when the current status is different than the previous one."

}

}

fixed:

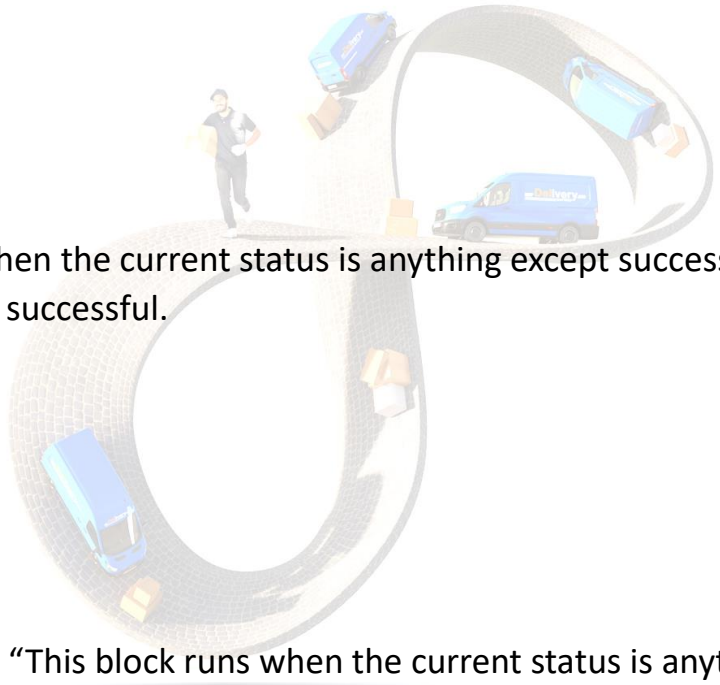
Only changes when failure/aborted to success.

```
post
{
    fixed
    {
        echo “"This block runs when the current status is success, and the previous one was failed or unstable.”
    }
}
```

regression:

This block runs when the current status is anything except success but the previous one was successful.

```
post
{
    regression
    {
        echo “This block runs when the current status is anything except success, but the previous one was successful."
    }
}
```



aborted:

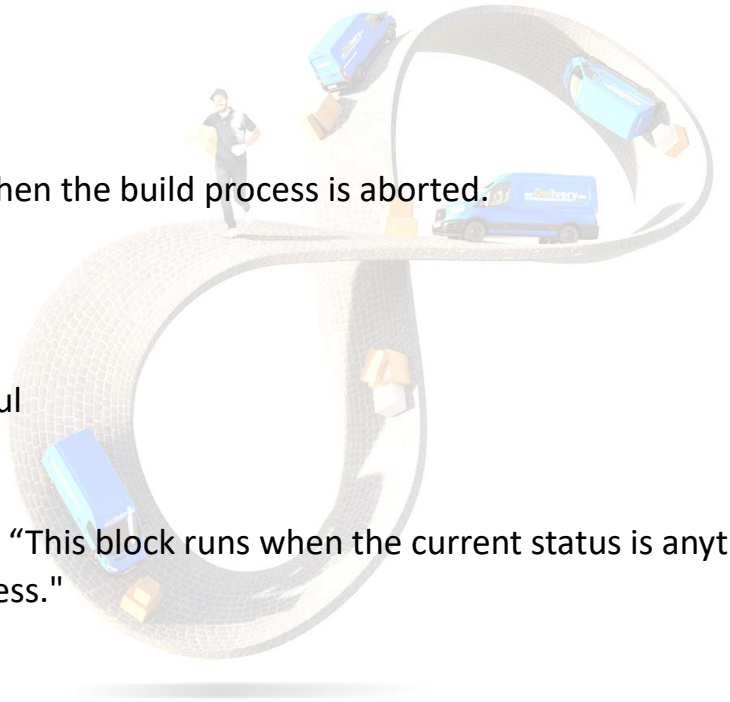
This block runs when the build process is aborted.

```
post
{
    aborted
    {
        echo “This block runs when the build process is aborted."
    }
}
```

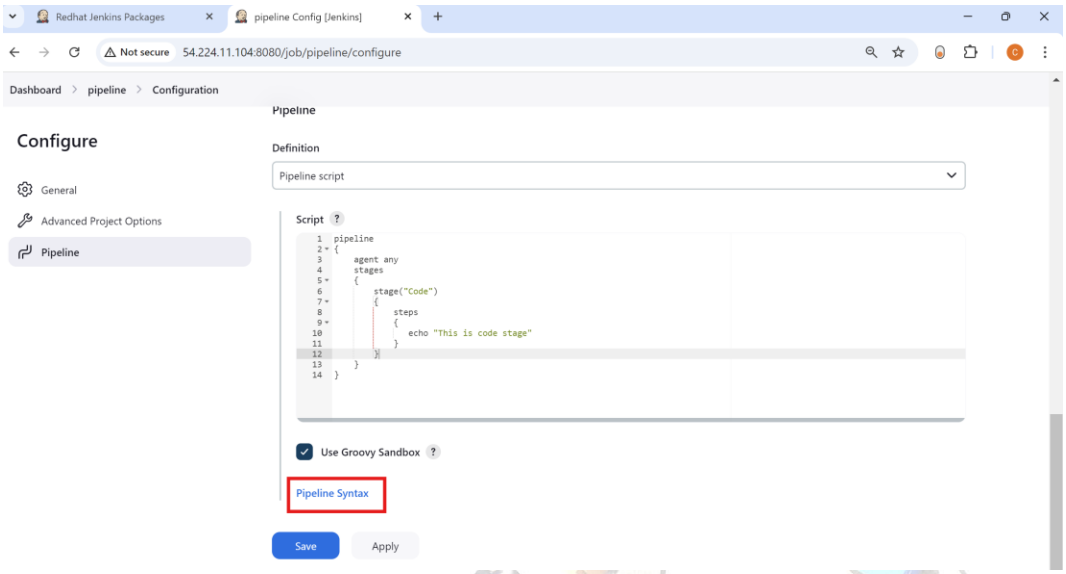
unsuccessful:

This block runs when the build process is aborted.

```
post
{
    unsuccessful
    {
        echo “This block runs when the current status is anything except
        success."
    }
}
```

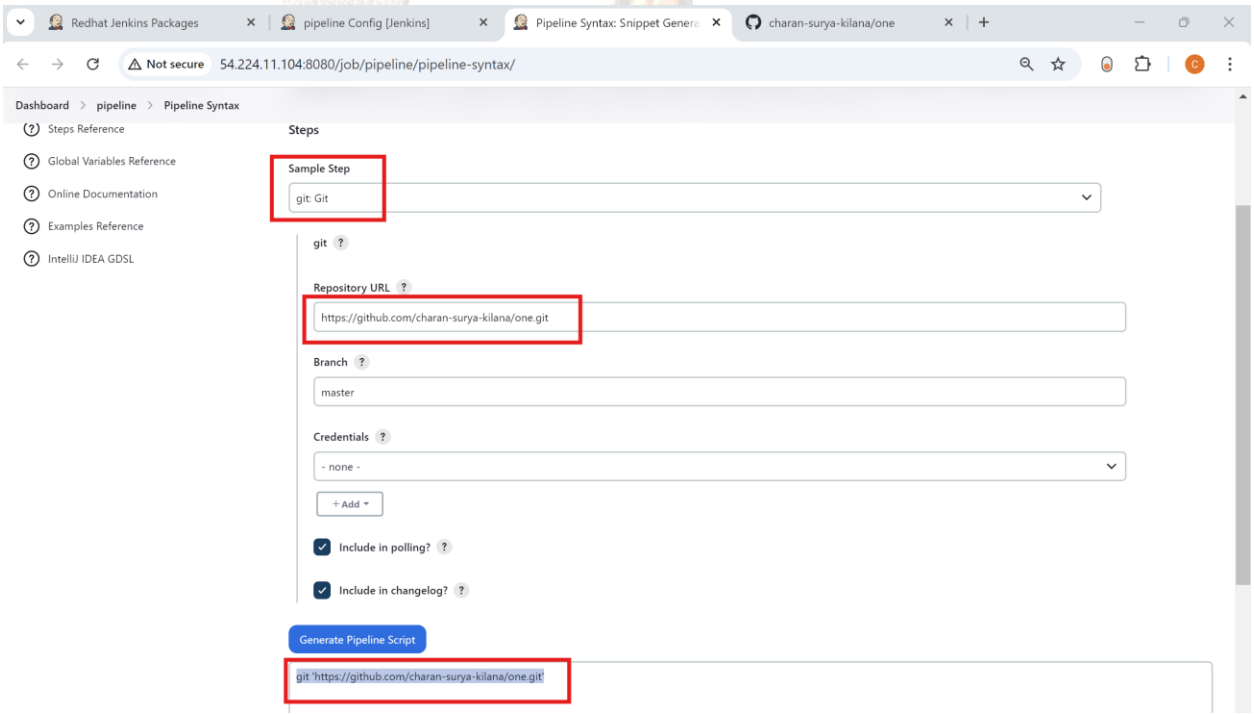


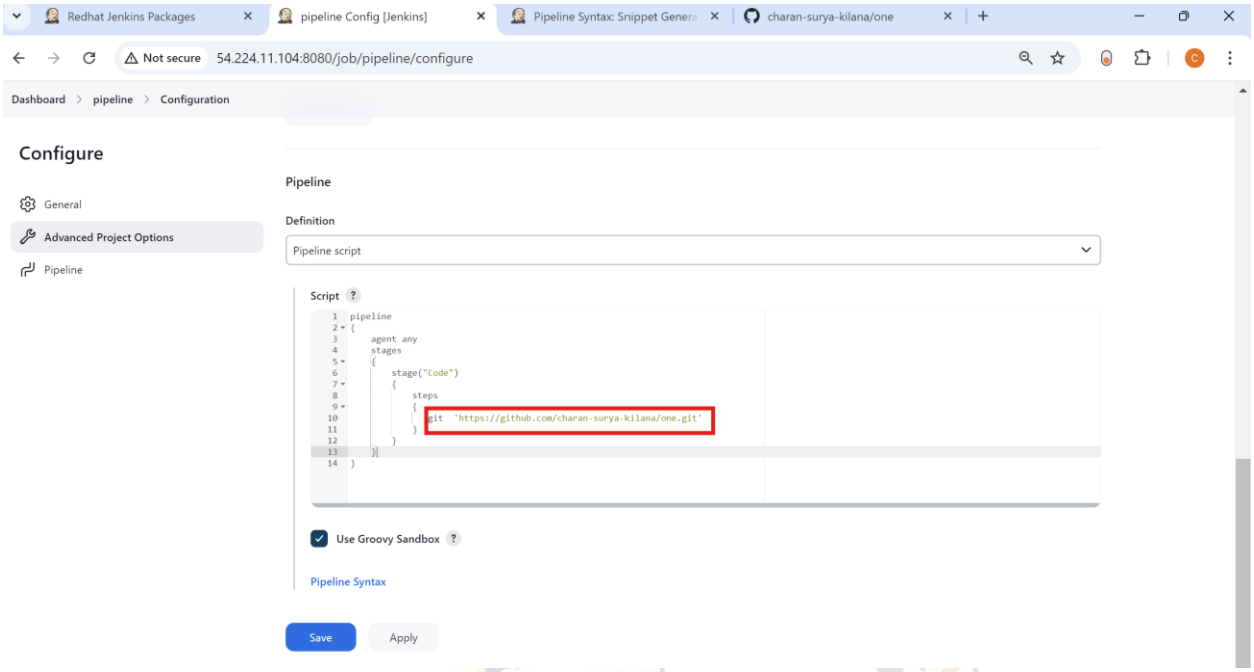
Connecting Git with pipeline:



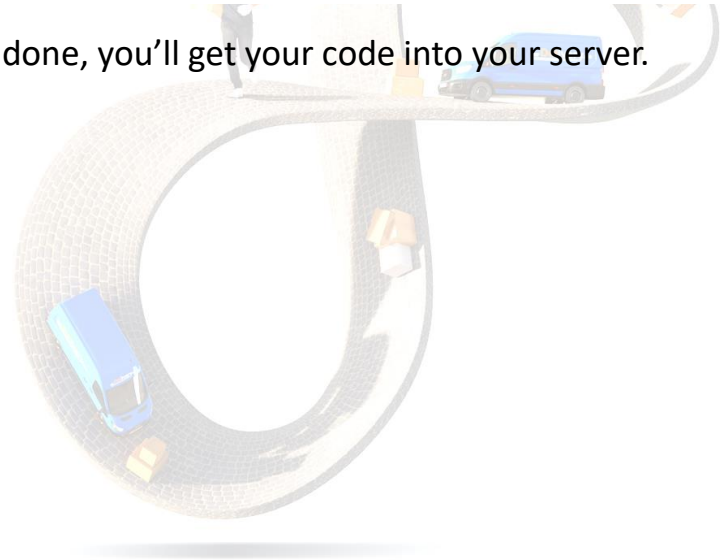
Navigate to Pipeline Syntax, select 'Git' from the options, and paste the repository URL. Click 'Generate Pipeline Script', then paste the generated script into the 'steps' section and save your changes.

If git is not installed in you should install it on your server.





Once the build is done, you'll get your code into your server.

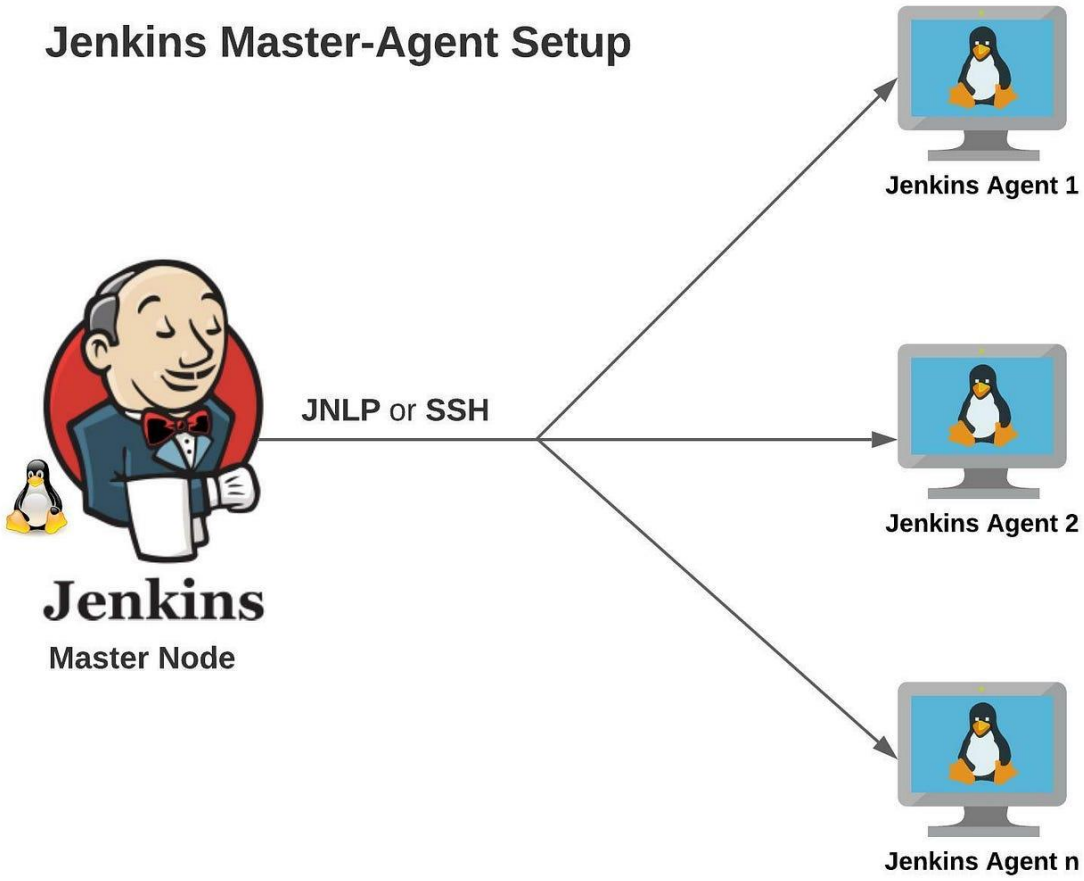


Jenkins Master-Slave Architecture

The Jenkins Master-Slave architecture involves a primary Jenkins server (the "Master") that manages and controls multiple additional servers (the "Slaves").

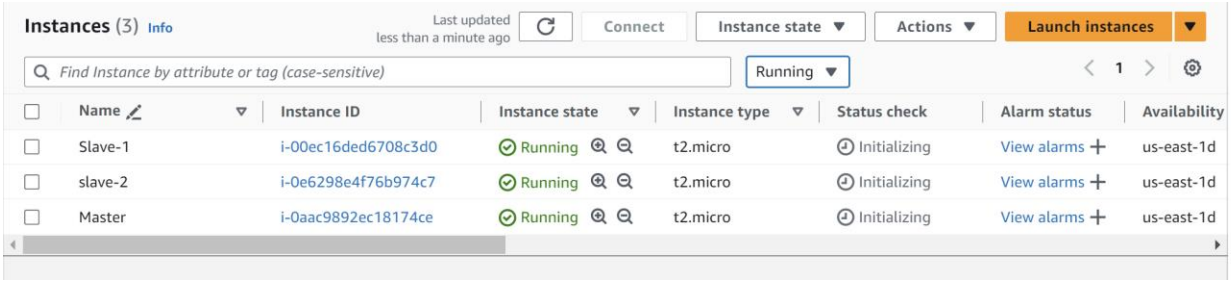
- The Master server coordinates and schedules tasks.
- The Slave servers perform the actual work, such as building, testing, and deploying applications.

This architecture allows you to distribute the workload and run tasks in parallel, enhancing the efficiency and scalability of your Jenkins setup. It's similar to having a manager (Master) who delegates tasks to a team of workers (Slaves) to complete jobs more quickly.



STEPS TO IMPLEMENT JENKINS MASTER-SLAVE:

Step-1: Launch three servers (Instances) with same key-pair(.pem) and same Security group. (**Master, slave-1 and slave-2**)



Step-2: Setup Jenkins setup in Master.

Note: Run the Jenkins setup script in filename.sh

amazon-linux-extras install java-openjdk11 -y

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

yum install jenkins -y

systemctl start jenkins

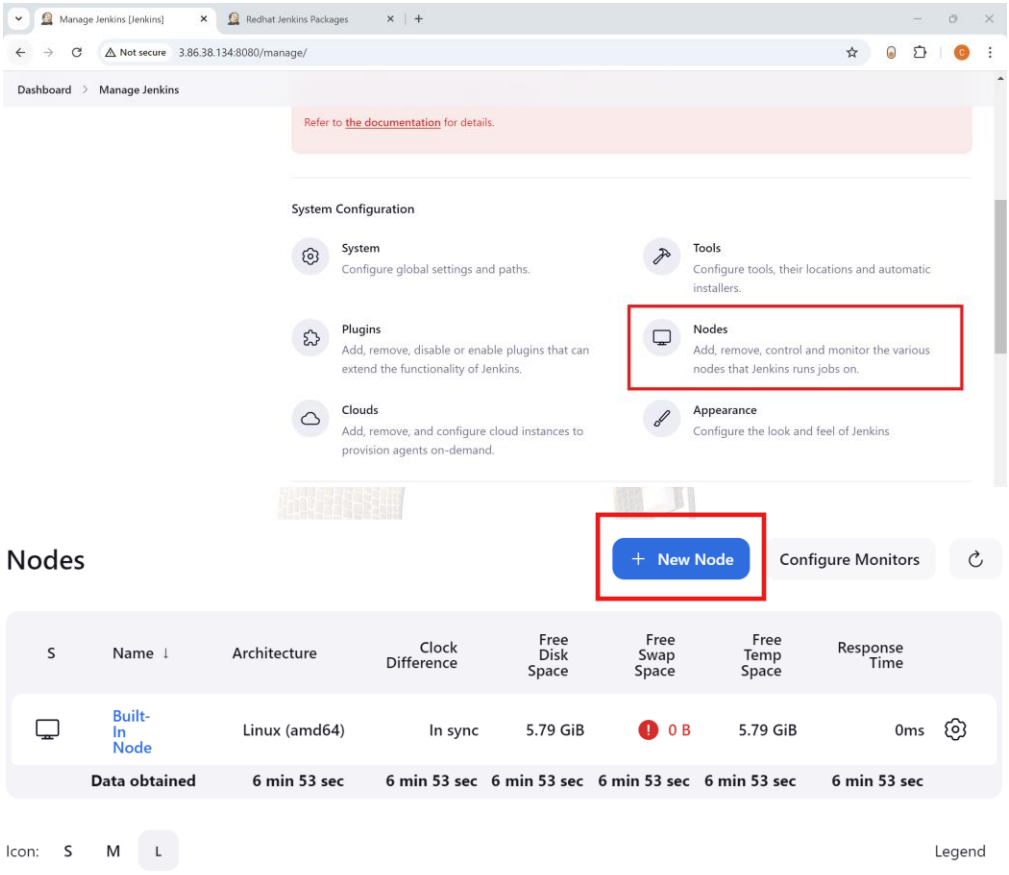
Note: For detailed Jenkins setup instructions, please refer to my guide on LinkedIn: **Jenkins Setup Guide.**

Step-3: Install jdk11 in slave servers.

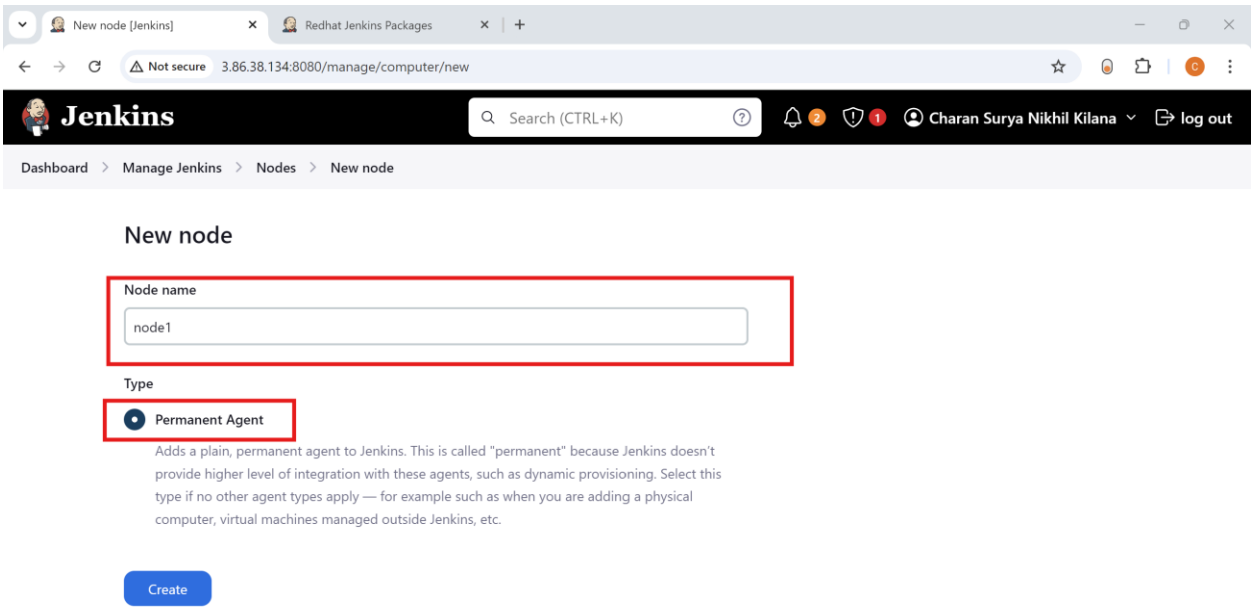
amazon-linux-extras install java-openjdk11 -y

Step-4: Instructions for setting up a new node in Jenkins:

1. Go to Manage Jenkins.
2. Navigate to Manage Nodes.
3. Click on New Node.
4. Enter a name for the node in the Node Name field.
5. Select Permanent Agent.



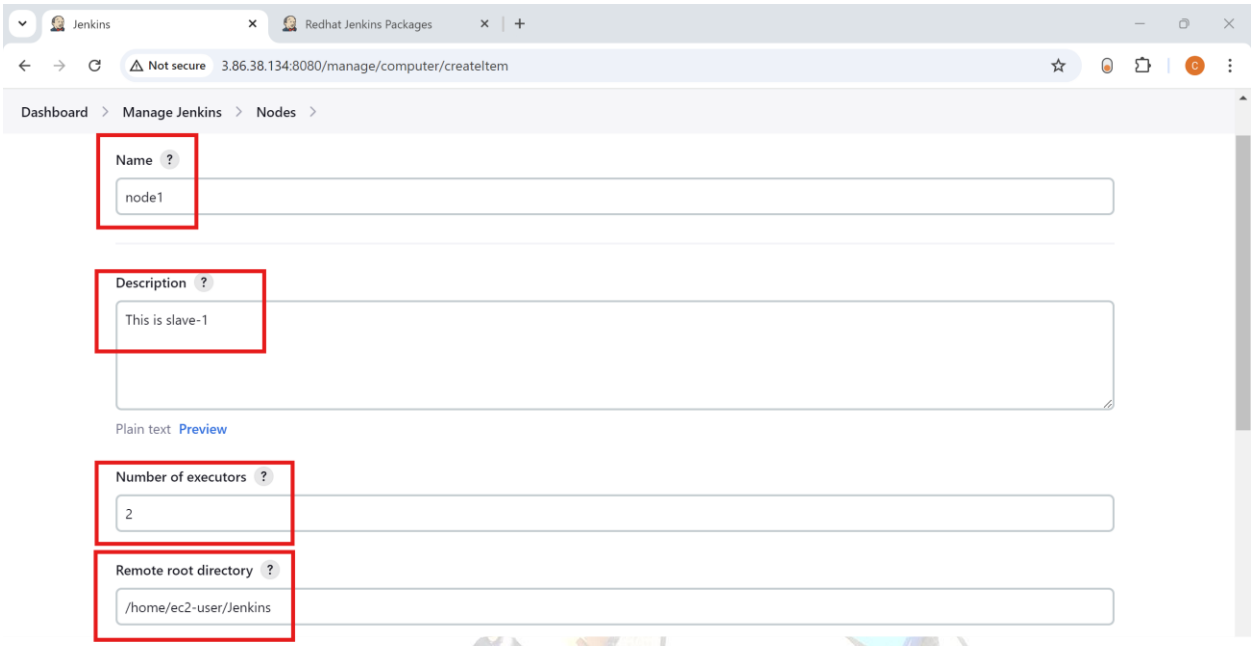
The "**Built-In Node**" in Jenkins refers to the default Jenkins server itself, acting as an agent. This node is responsible for executing jobs directly on the same machine where the Jenkins master is running, unless specified otherwise to run on separate, dedicated agent nodes. This setup allows Jenkins to handle jobs without the need for additional remote machines, simplifying smaller or less complex configurations.



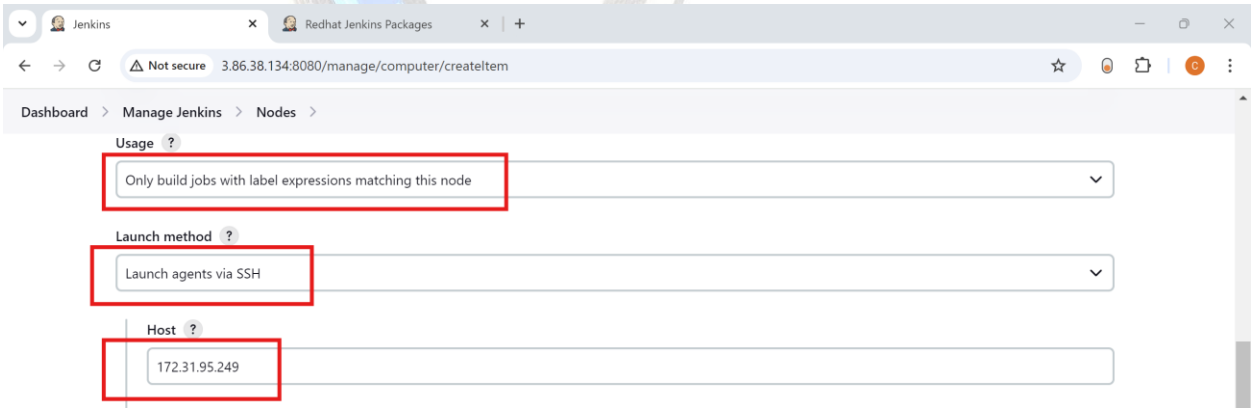
Step-5: Give the configurations.

- In the screenshot below, Executors are components in Jenkins that handle the actual build execution of jobs. Think of them as workers; each executor can handle one job at a time. For instance, if you have six jobs and two executors, two jobs will be built in parallel first, followed by the next two, and finally the last two. The number of executors is configurable based on the workload and concurrency requirements, typically determined by developers or DevOps teams to optimize build throughput and efficiency.

- In the slave server, Jenkins is not installed, so there is no /var/lib/Jenkins directory. Instead, we specify a remote root directory where the Jenkins workspace will be created. The designated remote root directory is: **/home/ec2-user/Jenkins** (with /home/ec2-user being an existing path on which the Jenkins directory is created). If this path is not correctly set, you may encounter a "permission denied" error due to improper directory permissions or access settings.

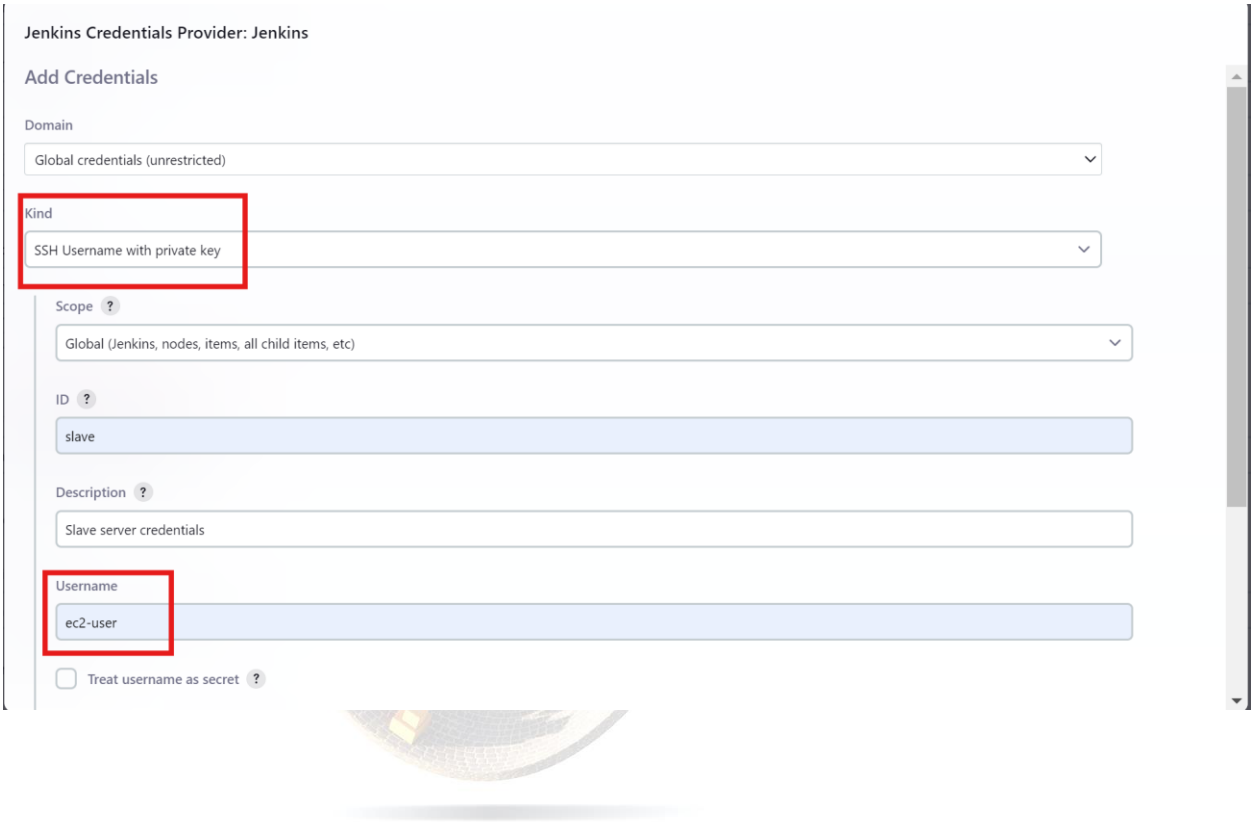


Labels in Jenkins are used to identify one or more slave nodes. They enable you to specify which jobs should run on a particular node by matching the job's label expression with the node. This setup is particularly useful for directing certain tasks to specific nodes optimized for those operations. You can launch agents via SSH, specifying the host as the slave server's IP address. Using a private IP is preferable because private IPv4 addresses do not change, ensuring stable and secure connections between the master and slave servers.



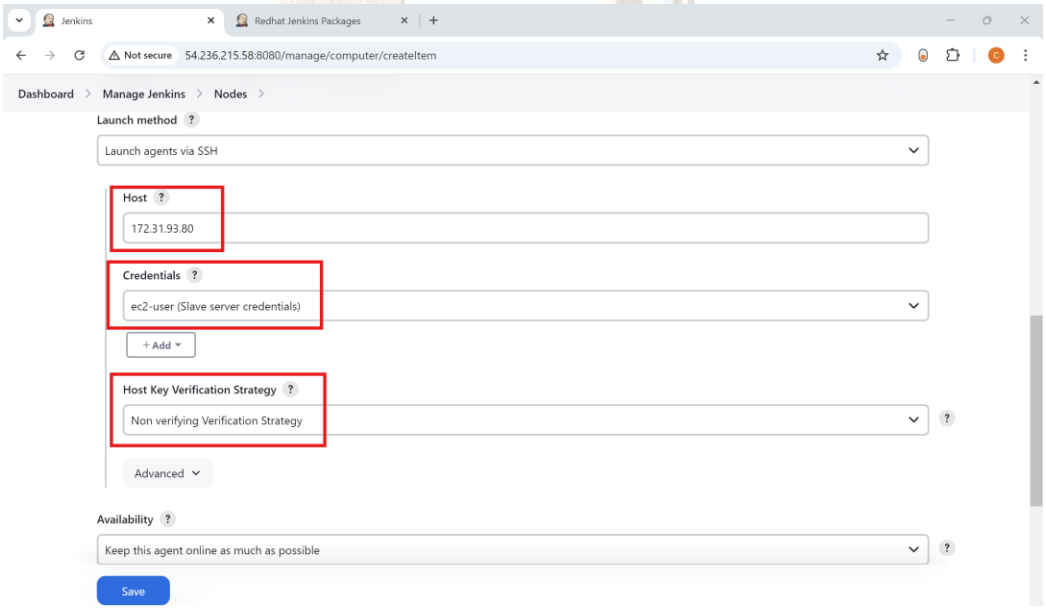
In Jenkins, to set up credentials for SSH access, go through the following steps:

1. Navigate to the "Credentials" section.
2. Choose "Add Credentials".
3. Select "SSH Username with Private Key" from the "Kind" dropdown menu.
4. Enter "ec2-user" as the username, which is typically used for AWS EC2 instances.

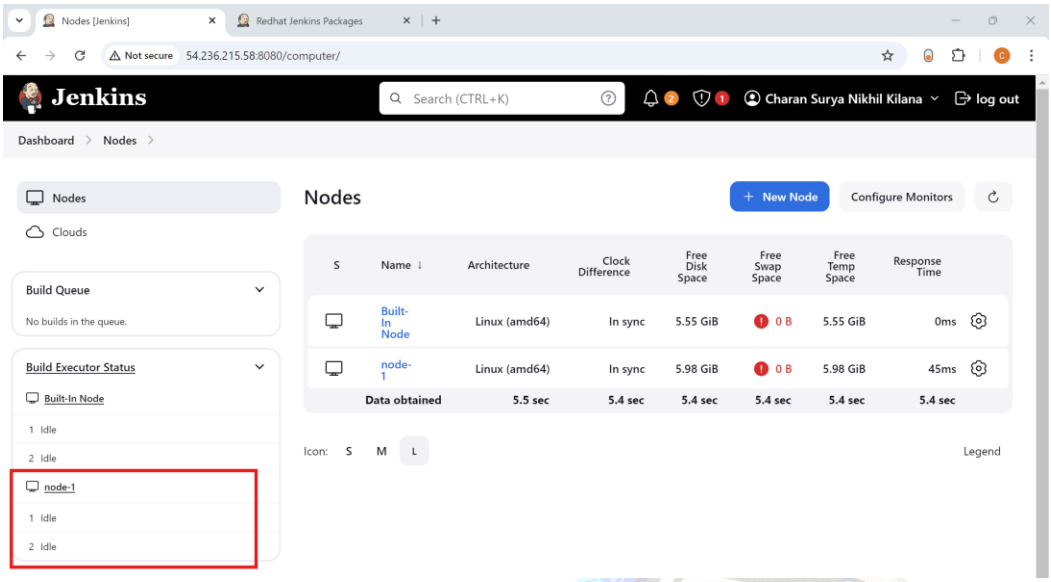


When you are using a **PEM file**, you can directly paste the contents of the file where needed.

In the context of Jenkins, when configuring SSH connections as part of setting up a build job or a slave node, you have the option to specify the **Non-verifying Verification Strategy** is one option available that, as the name suggests, does not verify the SSH host key of the remote server.

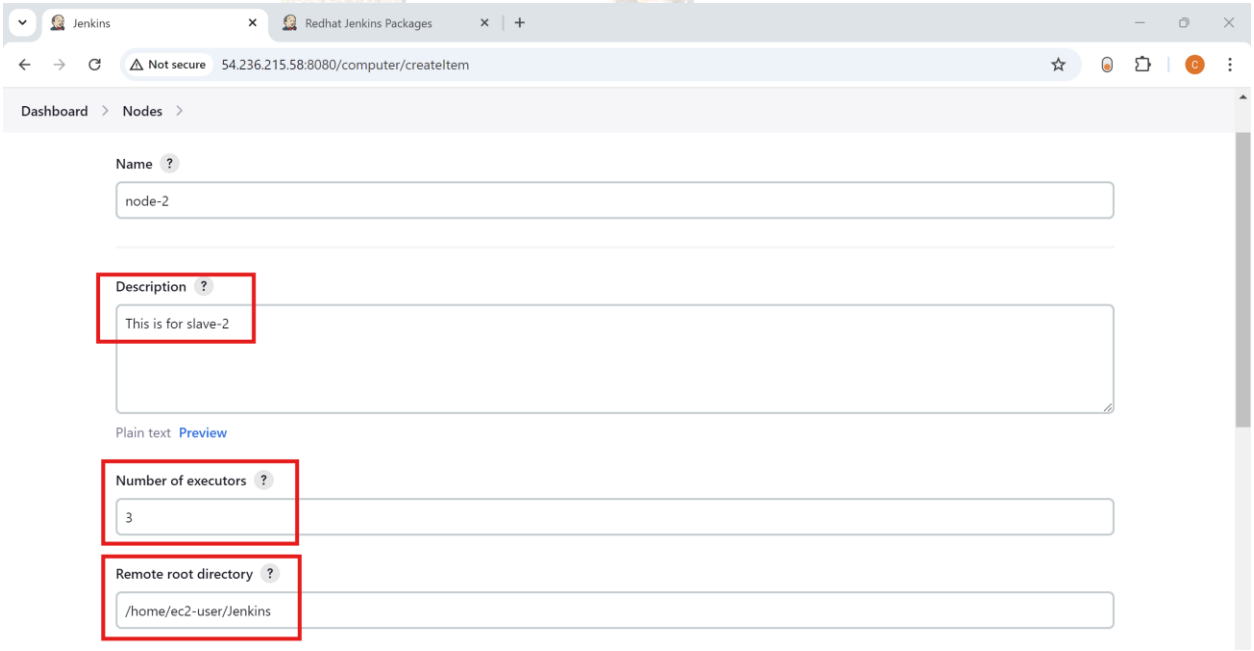


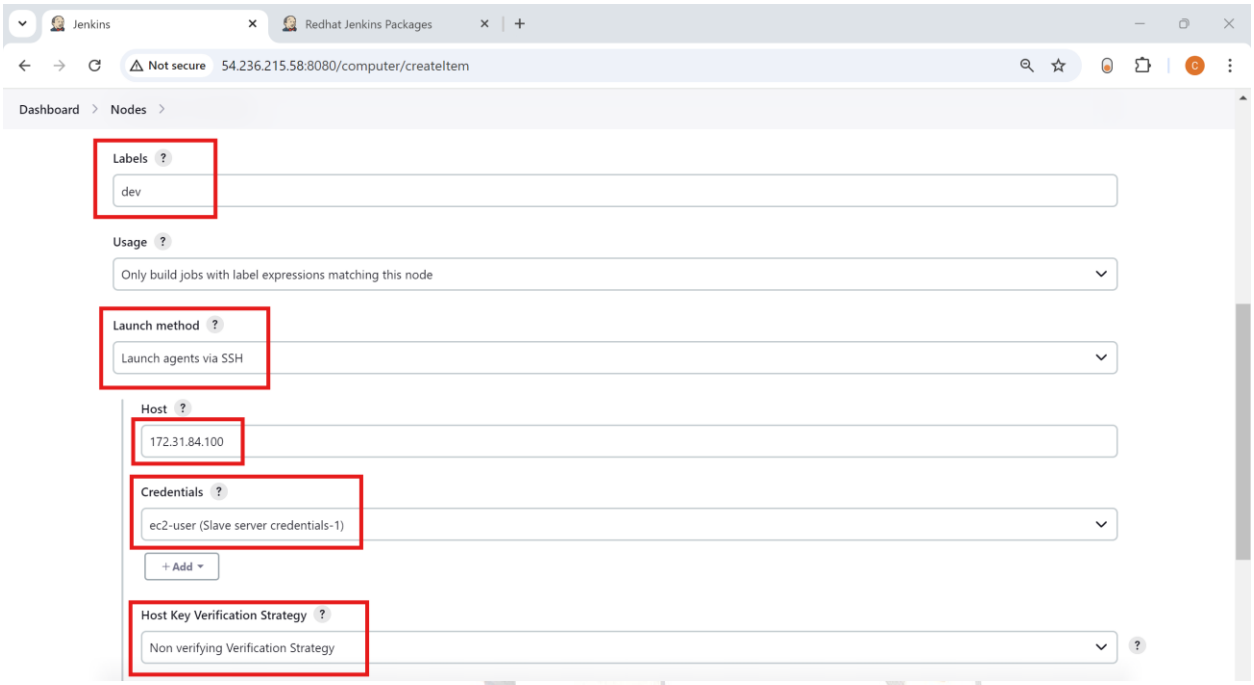
Once the node for slave sever-1 (test) is created you'll get under **Build executor status**, You'll get the node name along with number of executors.



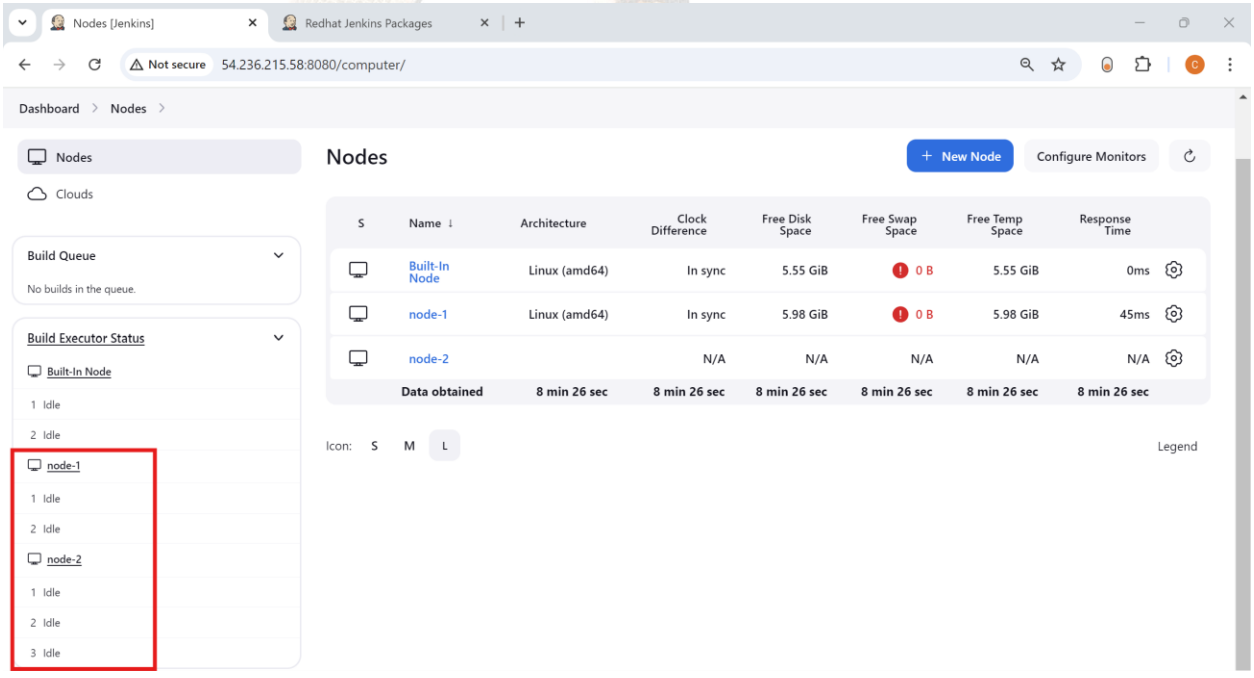
Step-6: Similarly, create additional nodes and assign the appropriate credentials. In this setup, I am creating another node specifically for the development environment.

In this example I gave 3 numbers of executors.

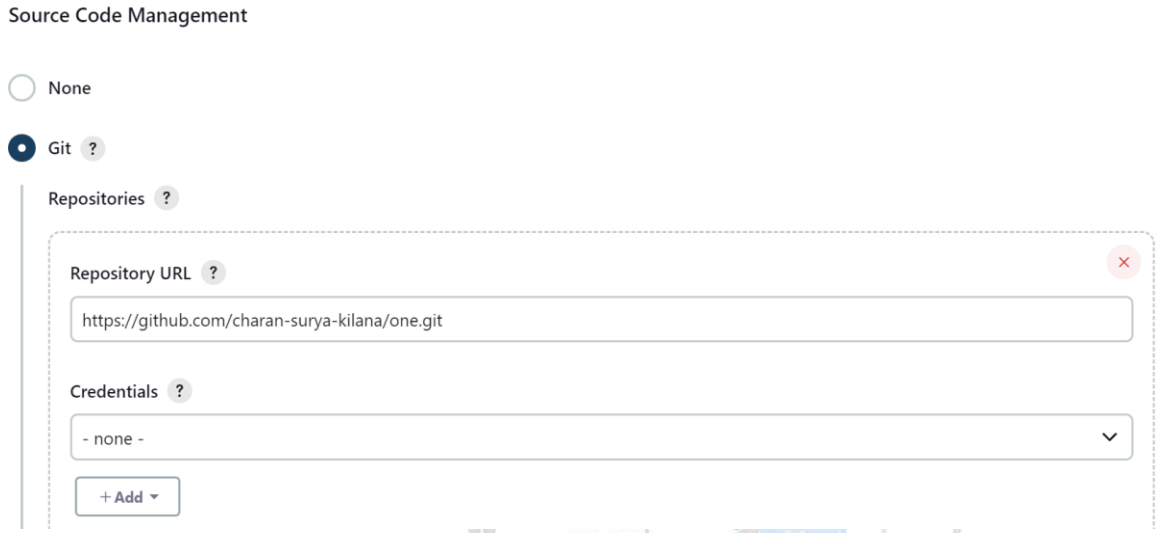




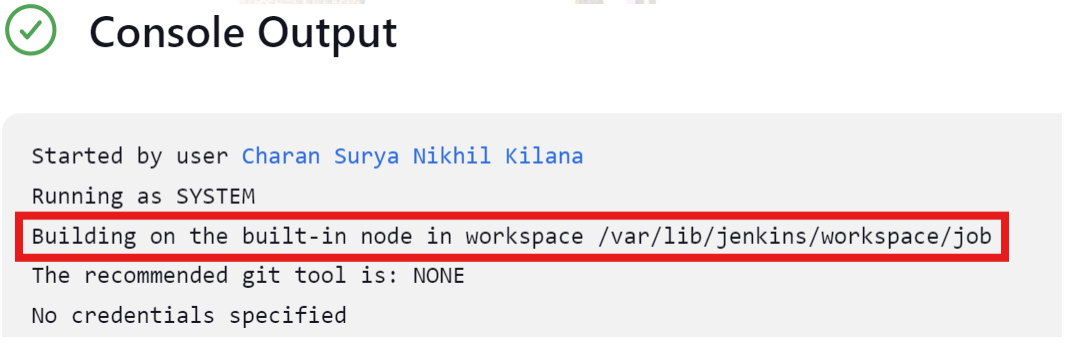
Note: You can use the same credentials if the key pair is identical. However, if you are using a different key pair, you will need to create a separate credential and assign it to node-2.



Step-7: To deploy, first create a new job in Jenkins. Ensure Git is installed on all servers, then add the repository link in the job's configuration to connect to your source code.



Step-9: Before building the job, we need to mention the server where it should run. Or else it will run on the master server.



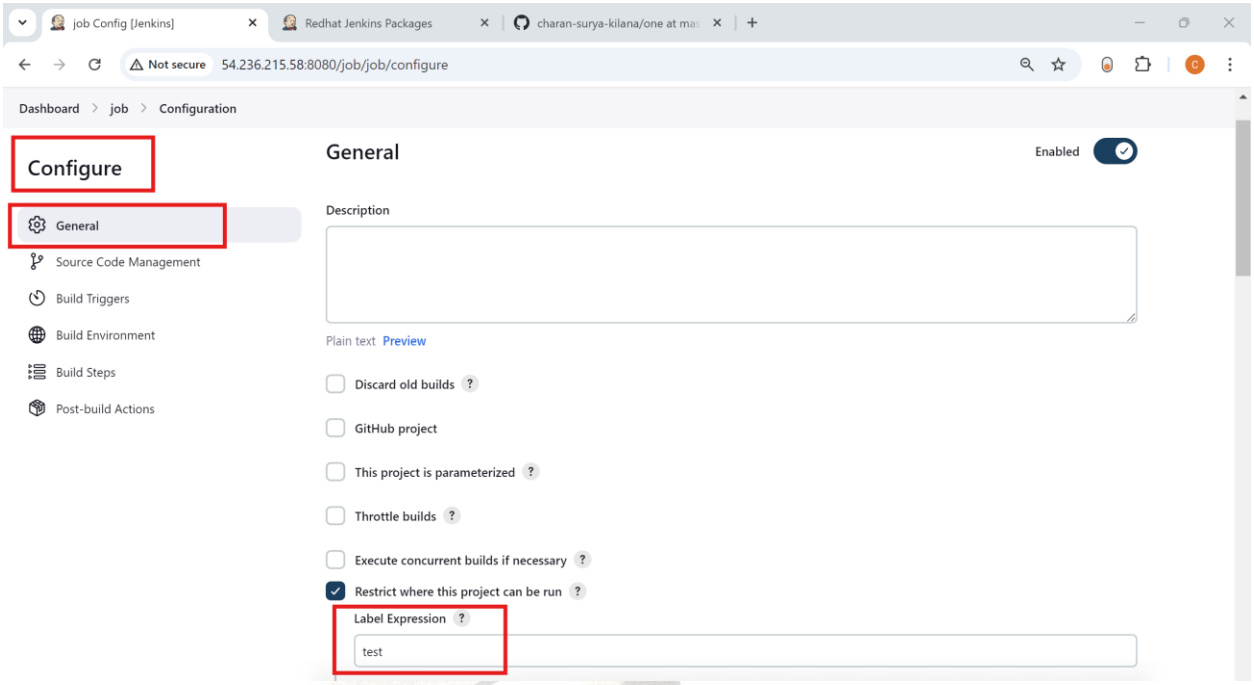
Step-10: Instructions for setting a label in Jenkins to restrict where a project can run:

Setting a Label:

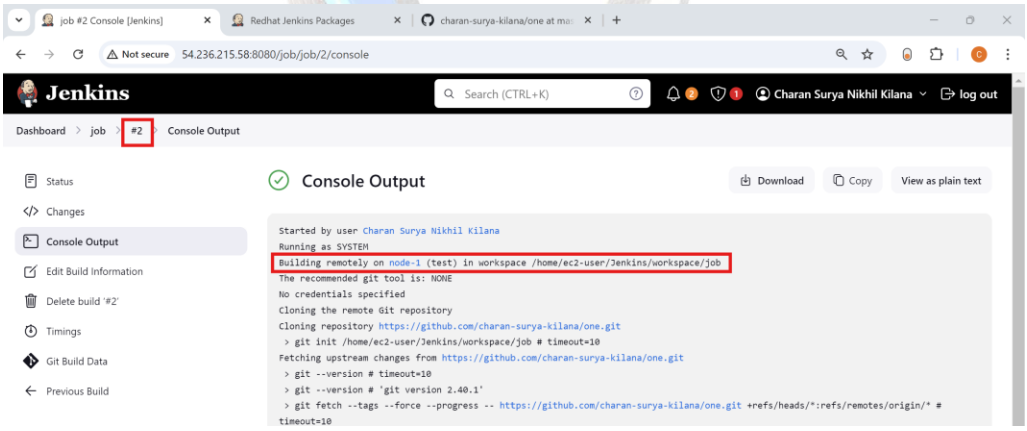
Go to Configure in your job settings.

In the General section, check the box for "Restrict where this project can be run".

Enter the label that corresponds to your intended environment. For this example, use either dev or test depending on where you want the job to execute.

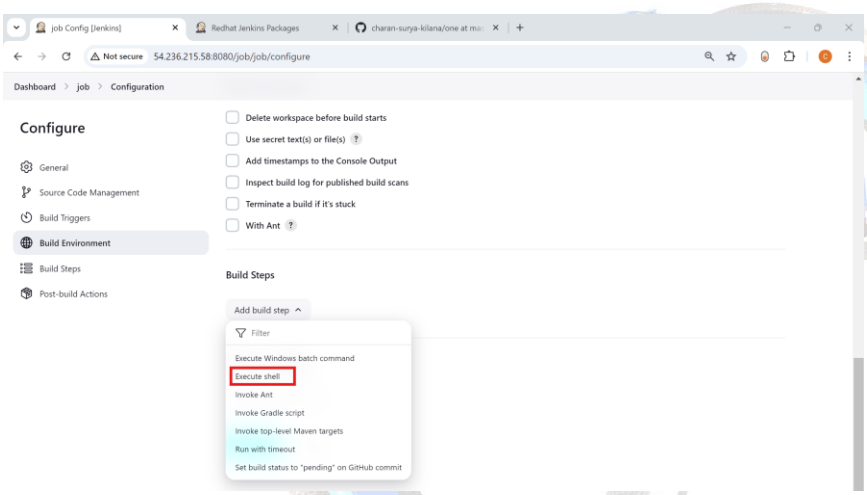


Before building in the master, note that slave-1 (i.e., the test server) does not use the default Jenkins path.

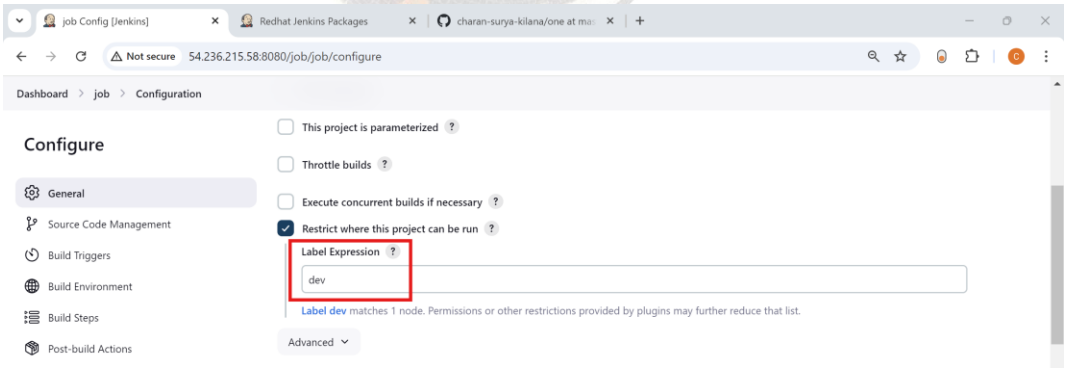


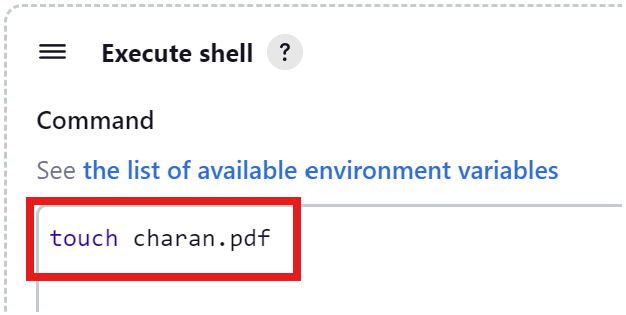
```
root@slave-1:/home/ec2-user   root@slave-2:~
[root@slave-1 ec2-user]# cd /home/ec2-user/Jenkins
[root@slave-1 Jenkins]# ls
remoting  remoting.jar  workspace
[root@slave-1 Jenkins]#
```

Step-11: Create a file on slave-2, which is designated as the development environment.

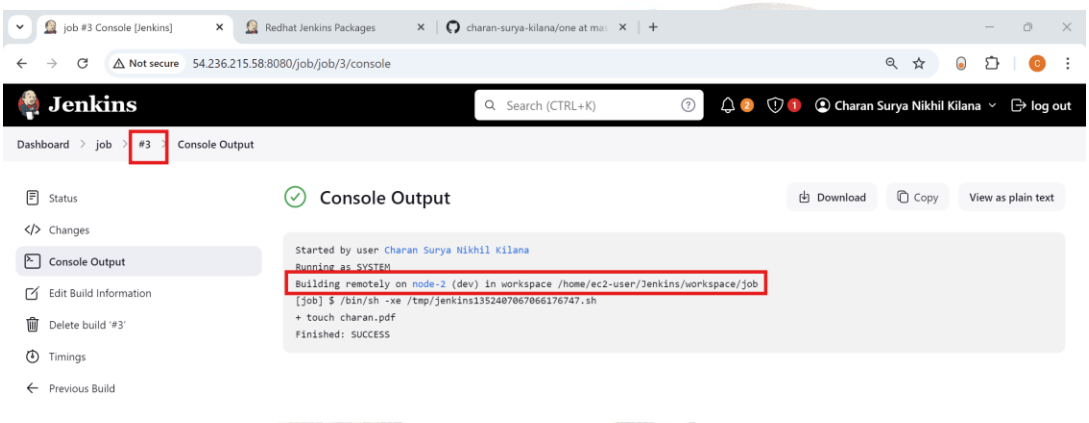


Change the label to dev

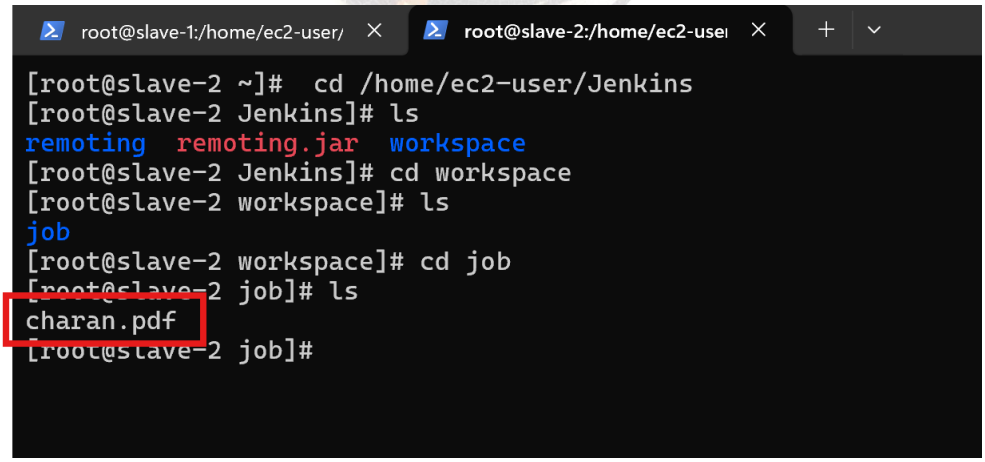




In the screenshot below, you can clearly see that the build is taking place on slave-2, which is designated as the development environment (dev).



The file is created:



Note:

→You cannot give multiple labels at a single time.

Implementing Role-Based Access Control in Jenkins for Enhanced Security and Efficiency

In a real-time environment, it's crucial to restrict users, such as developers, testers, or new team members, from having full permissions to perform actions on the Jenkins server. This is achieved by assigning appropriate permissions to ensure secure and controlled access.

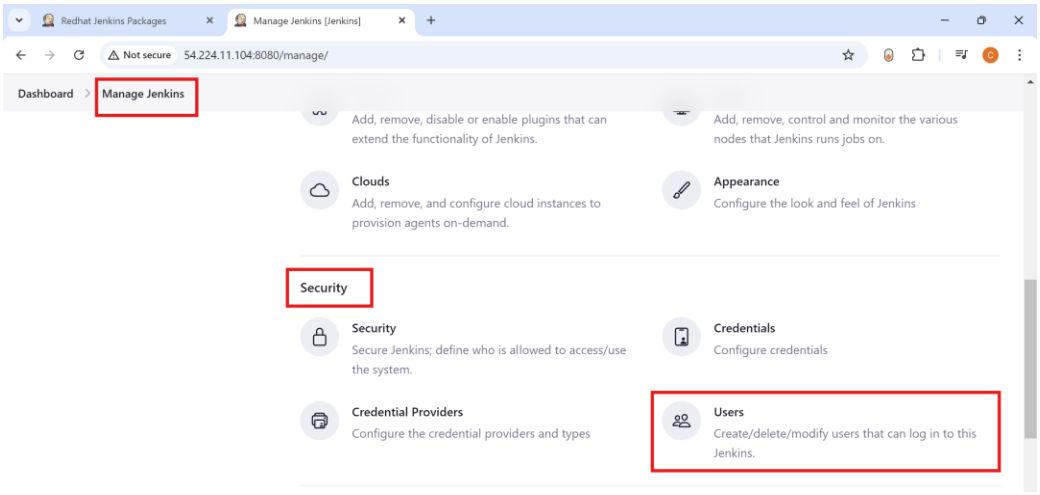
Giving appropriate permissions to respective teams helps maintain operational security and efficiency. For example:

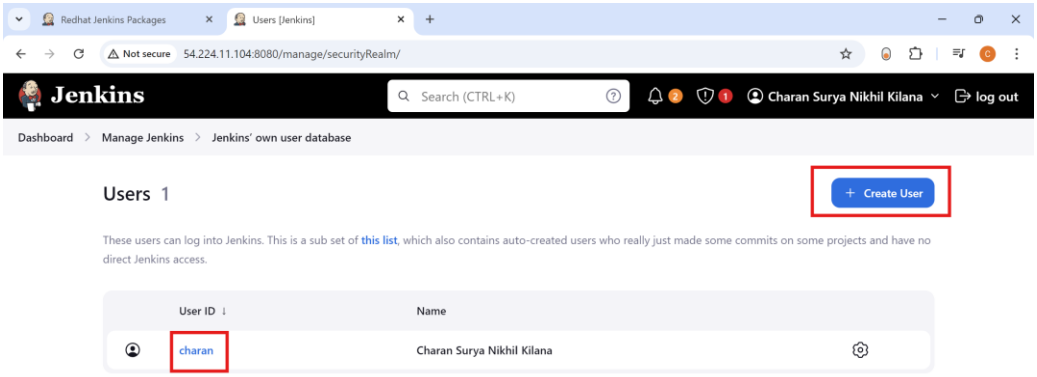
1. Developers need build permissions to execute and manage builds.
2. Testers are granted read-only permissions to view job configurations and results without the ability to make changes.
3. DevOps team members require comprehensive permissions, including build, configure, and administer, to manage Jenkins settings and job configurations effectively.

By setting specific permissions for each group, we prevent unauthorized actions and maintain a streamlined workflow on the Jenkins server.

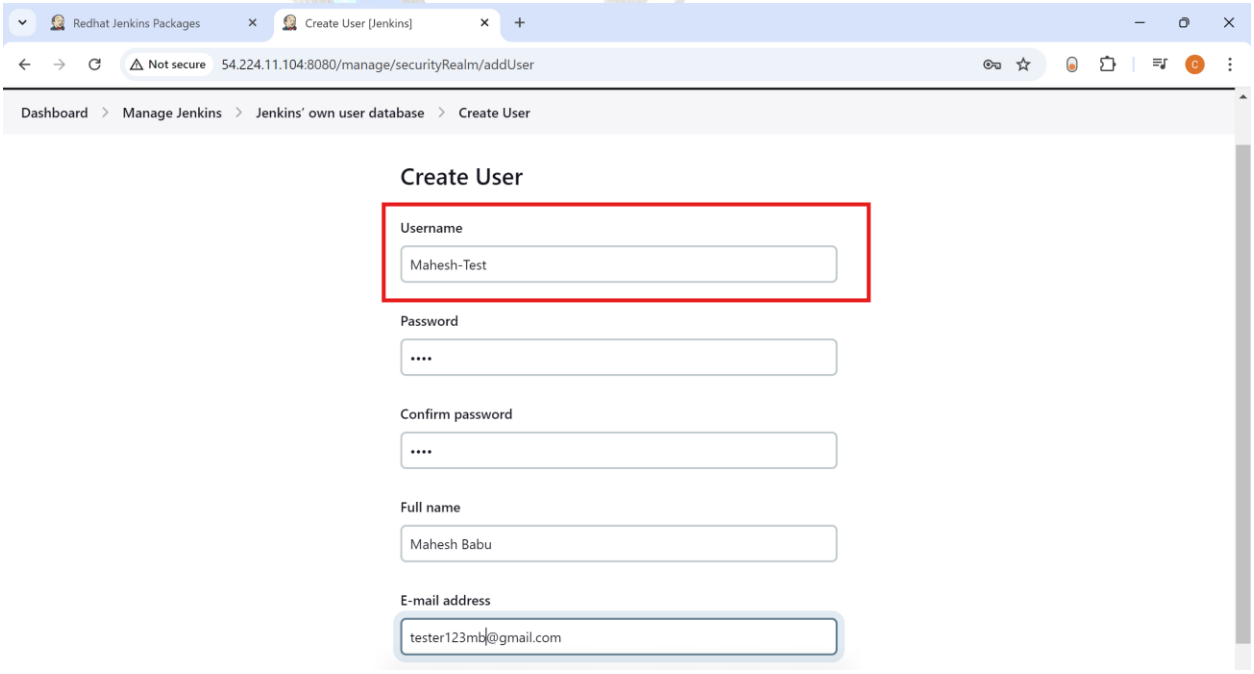
Step-1: Creating Users in Jenkins

1. Navigate to the **Jenkins Dashboard**.
2. Go to **Manage Jenkins**.
3. Under the **Security** section, find and click on **Users** (you will see the default user, typically the admin or master).
4. Click on **Create Users**.
5. Enter the required details: username, password, and email address.

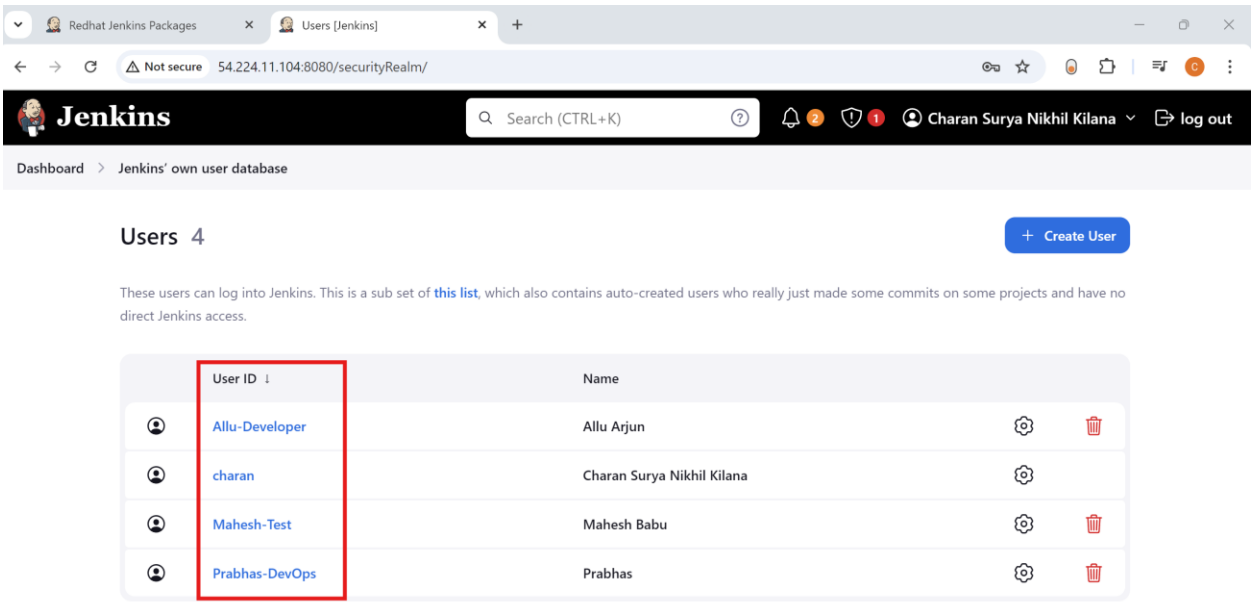




By default, the user "charan" is set as the administrator in the Jenkins system.

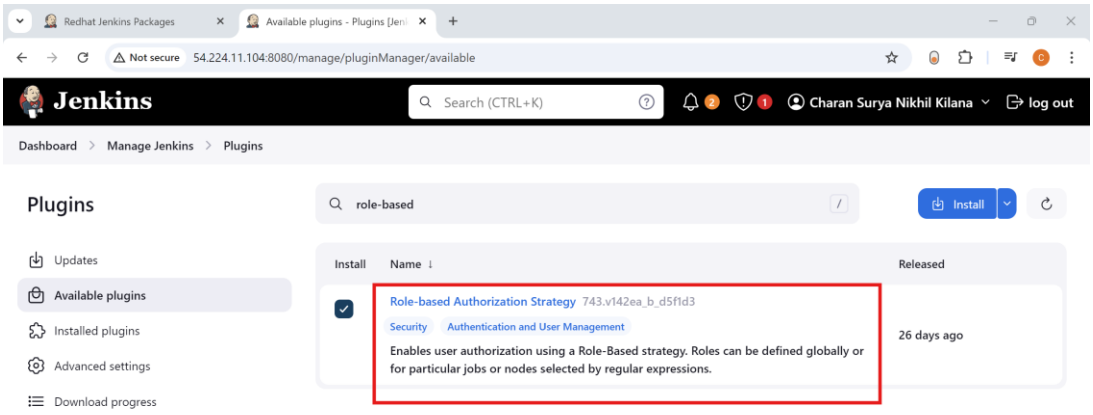


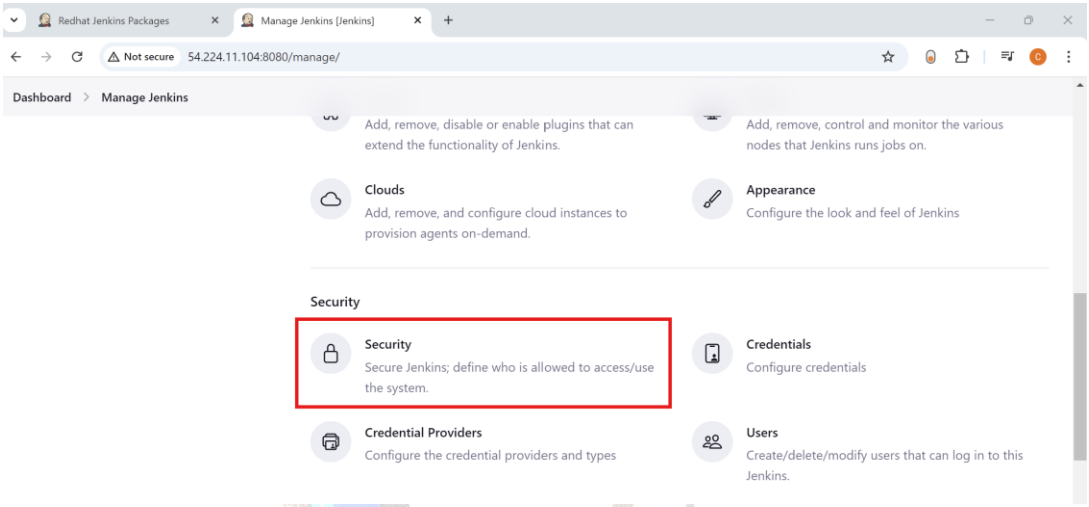
Similarly, you'll create two more users for DEV and DevOps.

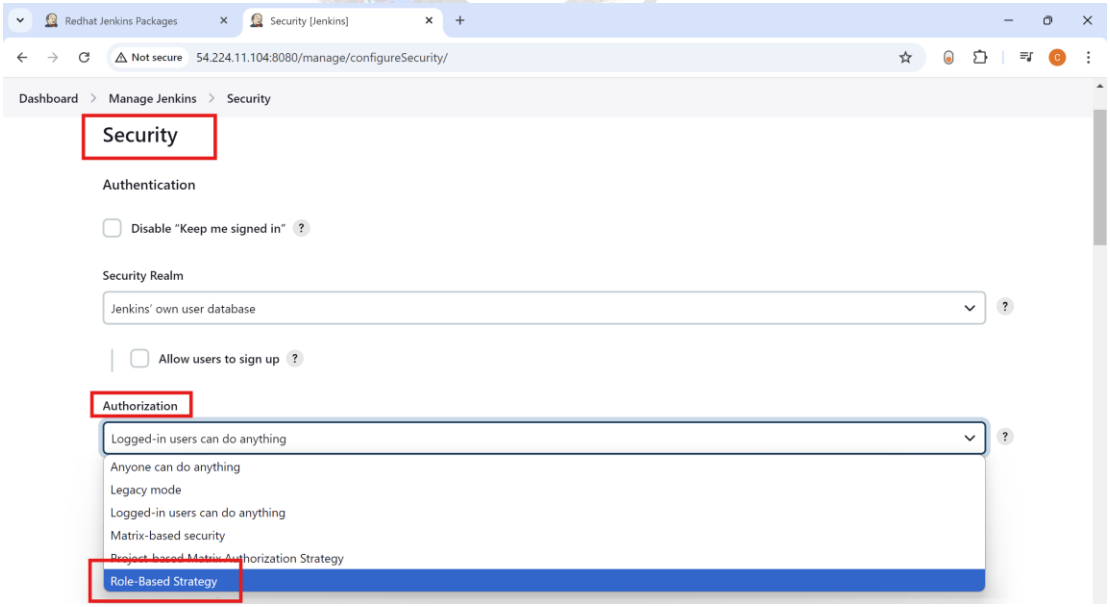


Step-2: Creating Roles in Jenkins

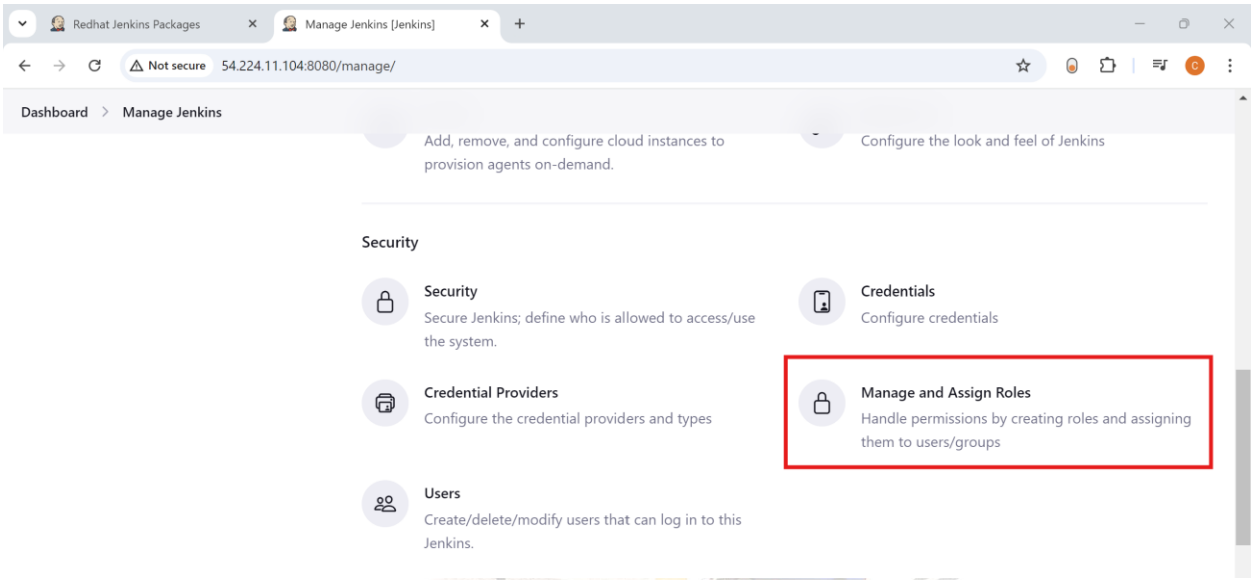
1. First, ensure the **Role-based Authorization Strategy** plugin is installed:
    - Go to **Manage Jenkins** on the Jenkins Dashboard.
    - Select **Manage Plugins**.
    - Search for the **Role-based Authorization Strategy** plugin in the Available tab and install it.
2. Once the plugin is installed, configure role-based security:
    - Return to the **Manage Jenkins** page.
    - Click on **Configure Global Security** under the Security section.
    - In the **Authorization** section, select **Role-Based Strategy** to enable role-based access control.
3. After setting the authorization strategy to role-based, you can start creating and assigning roles according to your organization's needs.


Redhat Jenkins Packages
Available plugins - Plugins [Jen
Not secure 54.224.11.104:8080/manage/pluginManager/available
Jenkins
Search (CTRL+K)
Charan Surya Nikhil Kilana
log out
Dashboard > Manage Jenkins > Plugins
Plugins
role-based
Install
Updates
Available plugins
Installed plugins
Advanced settings
Download progress
Install
Name
Released
Role-based Authorization Strategy 743.v142ea_b_d5f1d3
Security
Authentication and User Management
Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.
26 days ago


Redhat Jenkins Packages
Manage Jenkins [Jenkins]
Not secure 54.224.11.104:8080/manage/
Dashboard > Manage Jenkins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Clouds
Add, remove, and configure cloud instances to provision agents on-demand.
Appearance
Configure the look and feel of Jenkins
Security
Security
Secure Jenkins; define who is allowed to access/use the system.
Credentials
Configure credentials
Credential Providers
Configure the credential providers and types
Users
Create/delete/modify users that can log in to this Jenkins.


Redhat Jenkins Packages
Security [Jenkins]
Not secure 54.224.11.104:8080/manage/configureSecurity/
Dashboard > Manage Jenkins > Security
Security
Authentication
Disable "Keep me signed in"
Security Realm
Jenkins' own user database
Allow users to sign up
Authorization
Logged-in users can do anything
Anyone can do anything
Legacy mode
Logged-in users can do anything
Matrix-based security
Project-based Matrix Authorization Strategy
Role-Based Strategy

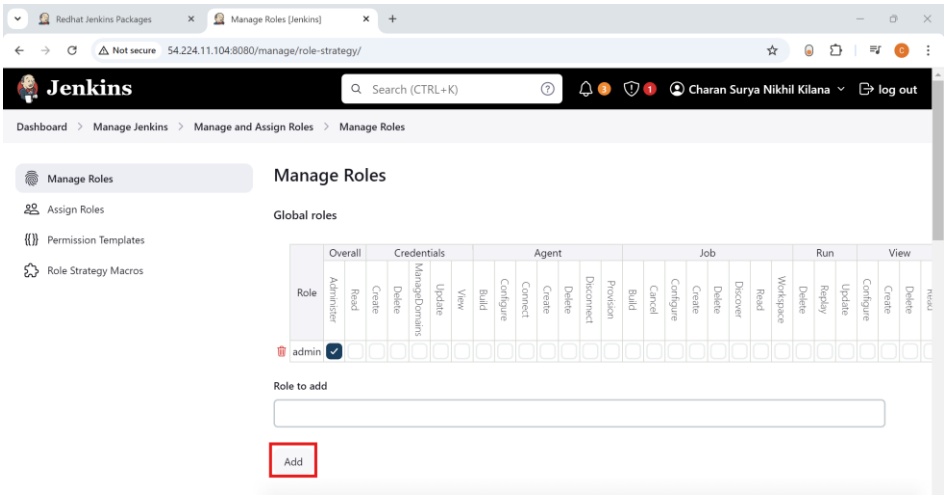
After selecting the Role-Based Strategy, a new option will appear under the Security section titled 'Manage and Assign Roles'. This allows you to define and distribute roles to manage permissions effectively within your Jenkins environment.

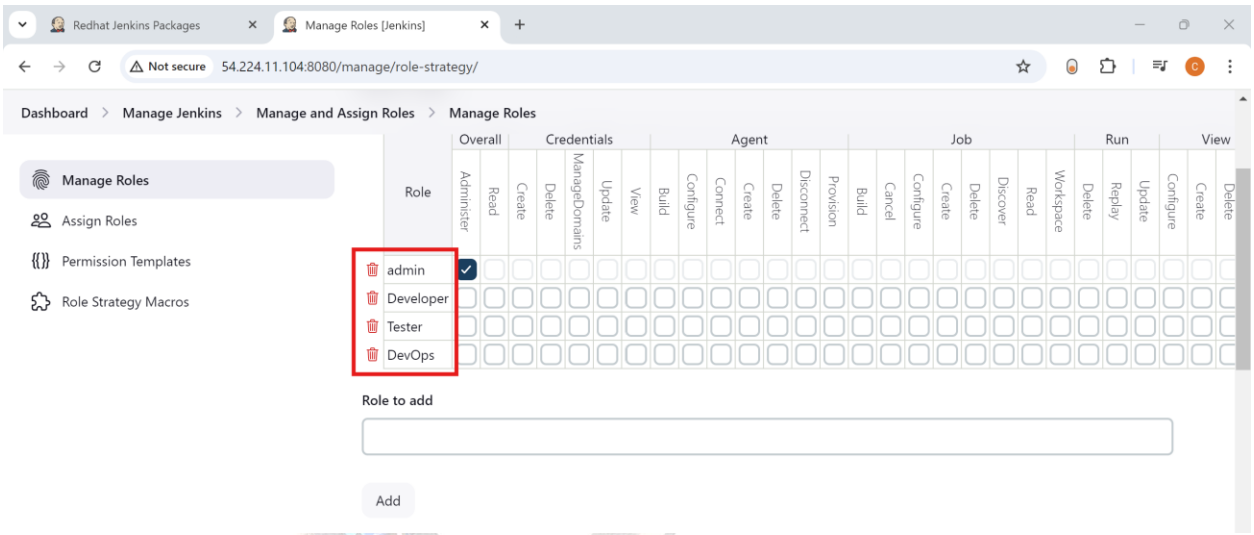


Step-3: Assigning Roles and Permissions

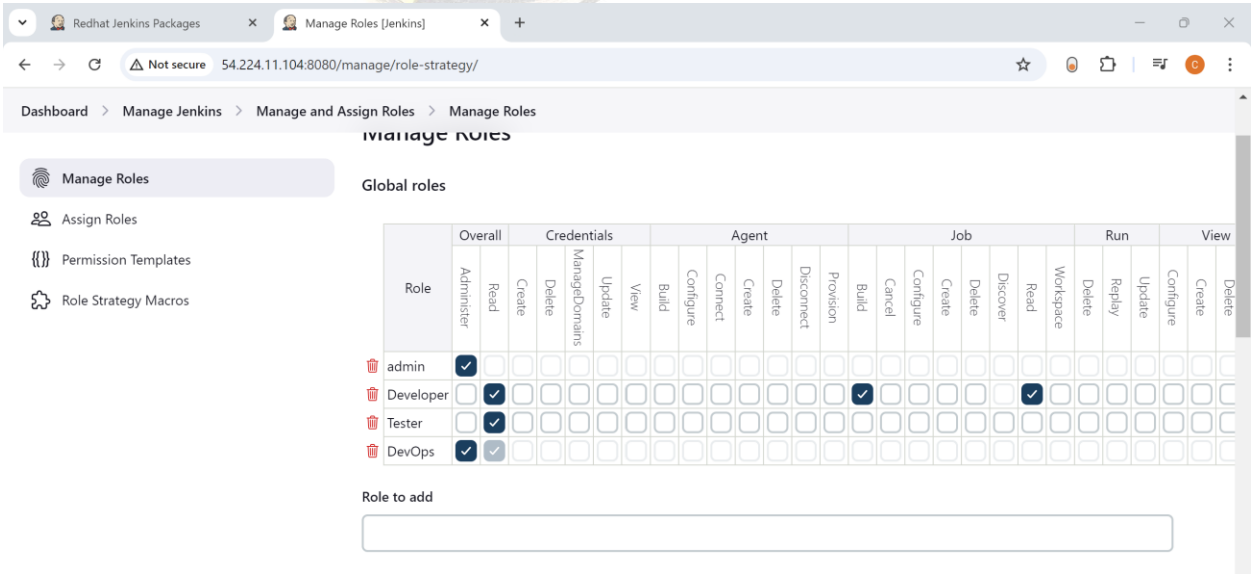
1. Navigate to the **Jenkins Dashboard**.
2. Click on **Manage Jenkins**.
3. Under the **Security** section, select **Manage and Assign Roles**.
4. Click on **Manage Roles** to define new roles or modify existing ones.
    - Add roles such as *Developer*, *Tester*, and *DevOps*.
    - Assign **Overall Read** permissions to all roles to ensure basic access.
    - For the **Developer** role, grant **Build** and **Read** permissions under **Job**.
    - Assign **Administrator** permissions to the **DevOps** role for full control.

This setup ensures that each team member has the appropriate level of access based on their role, enhancing both security and workflow efficiency in your Jenkins environment.





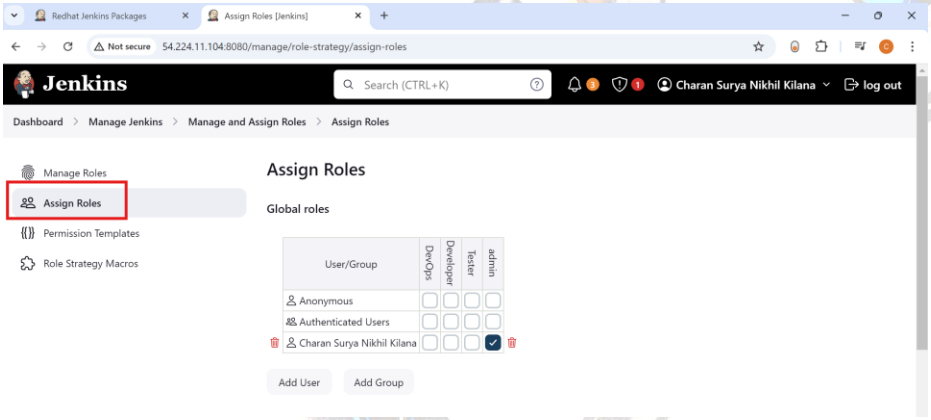
After giving permission:



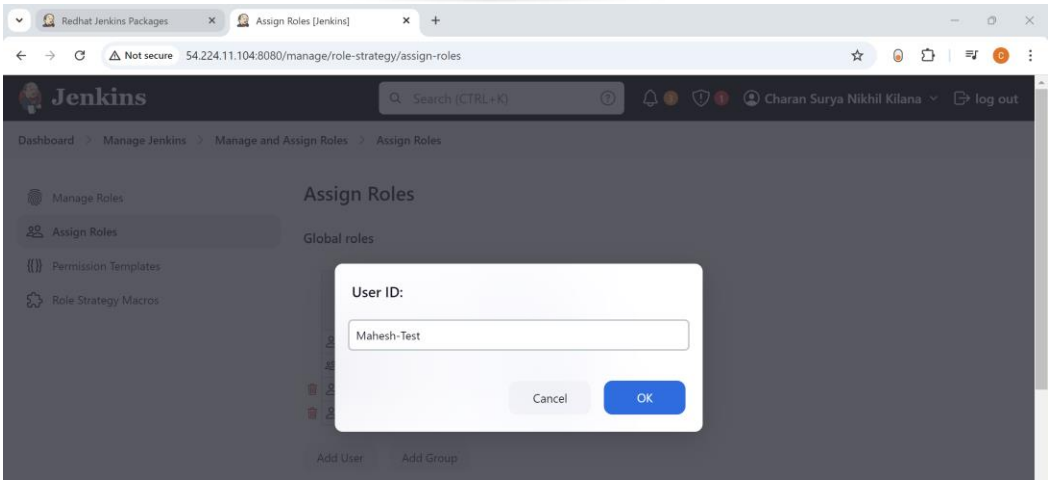
Step 4: Assigning Global Roles to Users

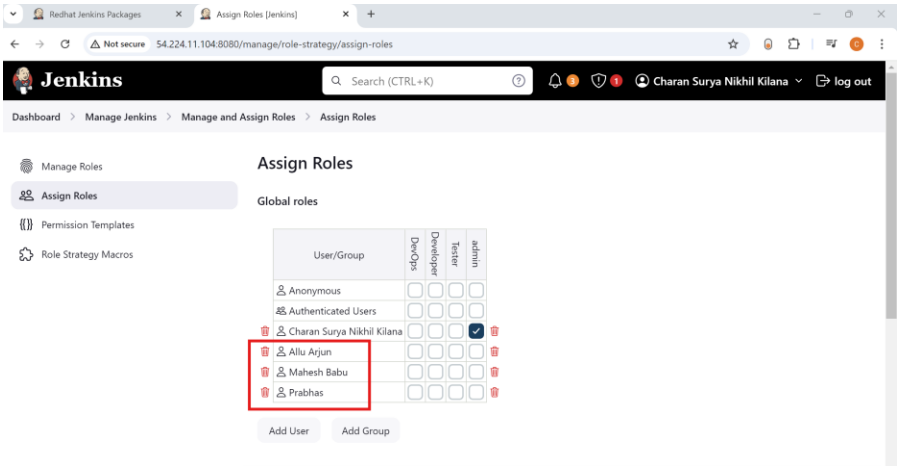
1. Navigate to the **Jenkins Dashboard**.
2. Select **Manage Jenkins** from the options.
3. Under the **Security** section, click on **Manage and Assign Roles**.
4. Choose **Assign Roles** from the left sidebar.
5. Under **Global roles**, add users and assign them to the appropriate global roles you've created earlier.

This process ensures that users are properly assigned to roles that dictate their permissions across the Jenkins environment, aligning access rights with their responsibilities and needs.



Add User:





Step 5: Verifying User Permissions

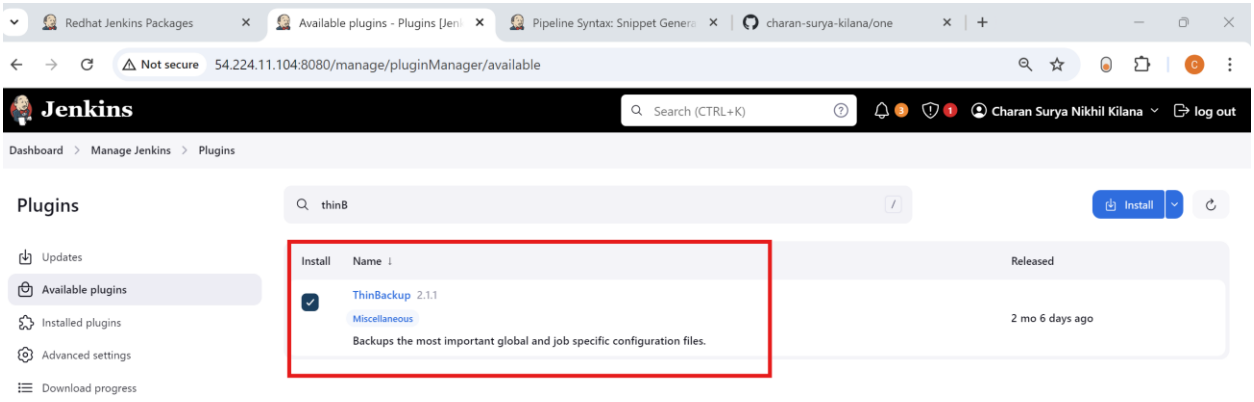
1. Log out of the current administrative account on the Jenkins Dashboard.
2. Log in using the credentials of a specific user role, such as a tester.
3. Verify that the permissions associated with their role are correctly enforced:
    - For example, a tester should not have build permissions. Check to ensure they cannot initiate builds or modify job configurations.

Note: One limitation of the role-based strategy is that it allows testers or newcomers to see every job, which may not be desirable in all scenarios. To address this, consider switching from a role-based strategy to a project-based matrix in the security settings. This change enables job-level permissions, offering finer control over who can see and interact with specific jobs.

Build Backup:

Backups are crucial for maintaining a history of build details, typically stored at /var/lib/Jenkins/jobs/(Job Name)/builds/. In the event that build history is inadvertently deleted, it can be restored from a backup. To facilitate this, first install the Thin Backup plugin:

1. Go to **Manage Jenkins**.
2. Under **Tools and Actions**, select **ThinBackup**.



Step 1: Create a folder on the server in the /opt/ directory from the root.

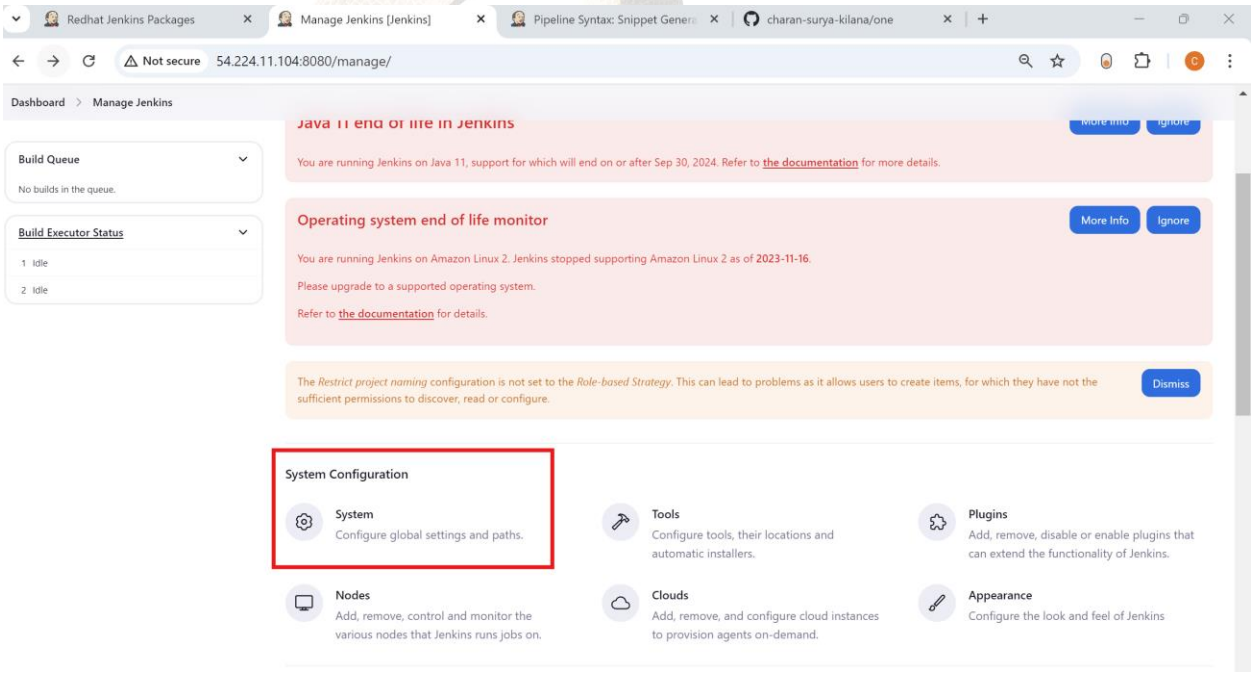
```
[root@jenkins ~]# cd /opt/
[root@jenkins opt]# ll
total 0
drwxr-xr-x 4 root root 33 Jun 10 23:35 aws
drwxr-xr-x 2 root root  6 Aug 16  2018 rh
[root@jenkins opt]# mkdir mybackup
[root@jenkins opt]# ll
total 0
drwxr-xr-x 4 root root 33 Jun 10 23:35 aws
drwxr-xr-x 2 root root  6 Jun 19 08:11 mybackup
drwxr-xr-x 2 root root  6 Aug 16  2018 rh
[root@jenkins opt]#
```

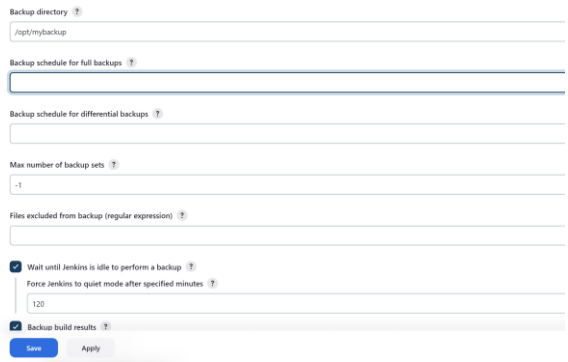
Step 2: By default, the owner of any Jenkins job is jenkins:jenkins. Change the ownership of the new folder (mybackup) to jenkins:jenkins to ensure Jenkins has the necessary permissions.

```
[root@jenkins ~]# cd /opt/
[root@jenkins opt]# ll
total 0
drwxr-xr-x 4 root root 33 Jun 10 23:35 aws
drwxr-xr-x 2 root root  6 Aug 16  2018 rh
[root@jenkins opt]# mkdir mybackup
[root@jenkins opt]# ll
total 0
drwxr-xr-x 4 root root 33 Jun 10 23:35 aws
drwxr-xr-x 2 root root  6 Jun 19 08:11 mybackup
drwxr-xr-x 2 root root  6 Aug 16  2018 rh
[root@jenkins opt]# chown -R jenkins:jenkins mybackup
[root@jenkins opt]# ll
total 0
drwxr-xr-x 4 root    root    33 Jun 10 23:35 aws
drwxr-xr-x 2 jenkins jenkins  6 Jun 19 08:11 mybackup
drwxr-xr-x 2 root    root     6 Aug 16  2018 rh
[root@jenkins opt]#
```

Step 3: Copy the path of the mybackup folder and configure it in ThinBackup:

- Go to **Manage Jenkins**.
- Under **System Configuration**, select **System**.
- Enter the mybackup path in the ThinBackup configuration settings.





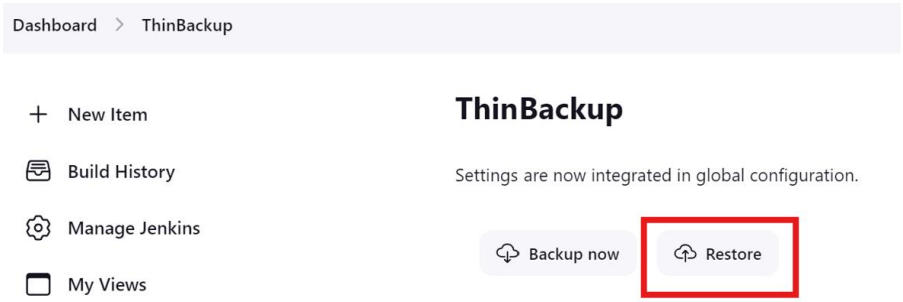
Step 4: Once everything is set up, navigate to the general job path /var/lib/jenkins/jobs and delete the jobs.

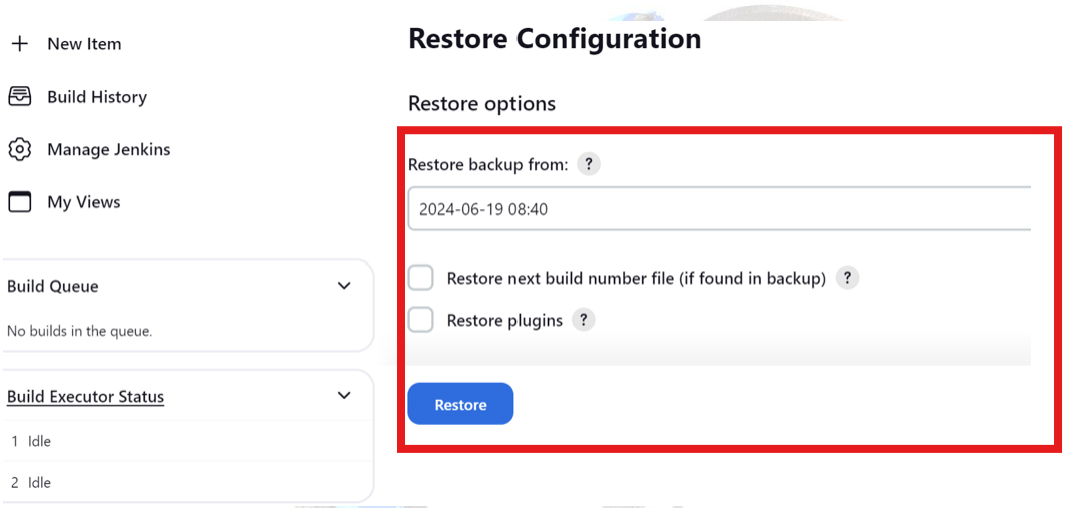
```
drwxr-xr-x  3 jenkins jenkins   19 Jun 19 08:08 jobs
drwxr-xr-x  3 jenkins jenkins   45 Jun 19 07:49 logs
-rw-r--r--  1 jenkins jenkins 1037 Jun 19 07:46 nodeMonitors.xml
-rw-r--r--  1 jenkins jenkins  169 Jun 19 08:40 org.jenkinsci.plugins
l
-rw-r--r--  1 jenkins jenkins  290 Jun 19 08:40 org.jenkinsci.plugins
-rw-r--r--  1 jenkins jenkins  170 Jun 19 08:40 org.jenkinsci.plugins
-rw-r--r--  1 jenkins jenkins  230 Jun 19 08:40 org.jenkinsci.plugins
-rw-r--r--  1 jenkins jenkins  248 Jun 19 08:40 org.jenkinsci.plugins
-rw-r--r--  1 jenkins jenkins 1079 Jun 19 08:40 org.jvnet.hudson.plug
drwxr-xr-x 92 jenkins jenkins 8192 Jun 19 08:15 plugins
-rw-r--r--  1 jenkins jenkins  258 Jun 19 08:20 queue.xml
-rw-r--r--  1 jenkins jenkins   64 Jun 19 07:46 secret.key
-rw-r--r--  1 jenkins jenkins    0 Jun 19 07:46 secret.key.not-so-sec
drwx------  2 jenkins jenkins  237 Jun 19 08:19 secrets
drwxr-xr-x  2 jenkins jenkins  149 Jun 19 07:48 updates
drwxr-xr-x  2 jenkins jenkins   24 Jun 19 07:46 userContent
drwxr-xr-x  3 jenkins jenkins   57 Jun 19 07:48 users
drwxr-xr-x  3 jenkins jenkins   19 Jun 19 08:19 workspace
[root@jenkins jenkins]# rm -rf jobs
```

After deletion of jobs folder

```
-rw-r--r--  1 jenkins jenkins  171 Jun 19 07:46 jenkins.telemetry.Correla
drwxr-xr-x  3 jenkins jenkins   45 Jun 19 07:49 logs
-rw-r--r--  1 jenkins jenkins 1037 Jun 19 07:46 nodeMonitors.xml
-rw-r--r--  1 jenkins jenkins  169 Jun 19 08:40 org.jenkinsci.plugins.dis
l
-rw-r--r--  1 jenkins jenkins  290 Jun 19 08:40 org.jenkinsci.plugins.git
-rw-r--r--  1 jenkins jenkins  170 Jun 19 08:40 org.jenkinsci.plugins.wor
-rw-r--r--  1 jenkins jenkins  230 Jun 19 08:40 org.jenkinsci.plugins.wor
-rw-r--r--  1 jenkins jenkins  248 Jun 19 08:40 org.jenkinsci.plugins.wor
-rw-r--r--  1 jenkins jenkins 1079 Jun 19 08:40 org.jvnet.hudson.plugins.
drwxr-xr-x 92 jenkins jenkins 8192 Jun 19 08:15 plugins
-rw-r--r--  1 jenkins jenkins  258 Jun 19 08:20 queue.xml
-rw-r--r--  1 jenkins jenkins   64 Jun 19 07:46 secret.key
-rw-r--r--  1 jenkins jenkins    0 Jun 19 07:46 secret.key.not-so-secret
drwx------  2 jenkins jenkins  237 Jun 19 08:19 secrets
drwxr-xr-x  2 jenkins jenkins  149 Jun 19 07:48 updates
drwxr-xr-x  2 jenkins jenkins   24 Jun 19 07:46 userContent
drwxr-xr-x  3 jenkins jenkins   57 Jun 19 07:48 users
drwxr-xr-x  3 jenkins jenkins   19 Jun 19 08:19 workspace
[root@jenkins jenkins]#
```

Step 6: Go to ThinBackup in manage jenkins and select restore. You should see the job restored successfully.





Step-7: After the backup is completed, your job folder is securely backed up.

```
-rw-r--r--  1 jenkins jenkins  179 Jun 19 08:40 jenkins.tasks.filter
-rw-r--r--  1 jenkins jenkins  171 Jun 19 07:46 jenkins.telemetry.Co
drwxr-xr-x  3 jenkins jenkins   19 Jun 19 08:40 jobs
drwxr-xr-x  3 jenkins jenkins   45 Jun 19 07:49 logs
-rw-r--r--  1 jenkins jenkins 1037 Jun 19 07:46 nodeMonitors.xml
-rw-r--r--  1 jenkins jenkins  169 Jun 19 08:40 org.jenkinsci.plugin
l
-rw-r--r--  1 jenkins jenkins  290 Jun 19 08:40 org.jenkinsci.plugin
-rw-r--r--  1 jenkins jenkins  170 Jun 19 08:40 org.jenkinsci.plugin
-rw-r--r--  1 jenkins jenkins  230 Jun 19 08:40 org.jenkinsci.plugin
-rw-r--r--  1 jenkins jenkins  248 Jun 19 08:40 org.jenkinsci.plugin
-rw-r--r--  1 jenkins jenkins 1079 Jun 19 08:40 org.jvnet.hudson.plu
drwxr-xr-x 92 jenkins jenkins 8192 Jun 19 08:15 plugins
-rw-r--r--  1 jenkins jenkins  258 Jun 19 08:20 queue.xml
-rw-r--r--  1 jenkins jenkins   64 Jun 19 07:46 secret.key
-rw-r--r--  1 jenkins jenkins    0 Jun 19 07:46 secret.key.not-so-se
drwx------  2 jenkins jenkins  237 Jun 19 08:19 secrets
drwxr-xr-x  2 jenkins jenkins  149 Jun 19 07:48 updates
drwxr-xr-x  2 jenkins jenkins   24 Jun 19 07:46 userContent
drwxr-xr-x  3 jenkins jenkins   57 Jun 19 07:48 users
drwxr-xr-x  3 jenkins jenkins   19 Jun 19 08:19 workspace
[root@jenkins jenkins]#
```

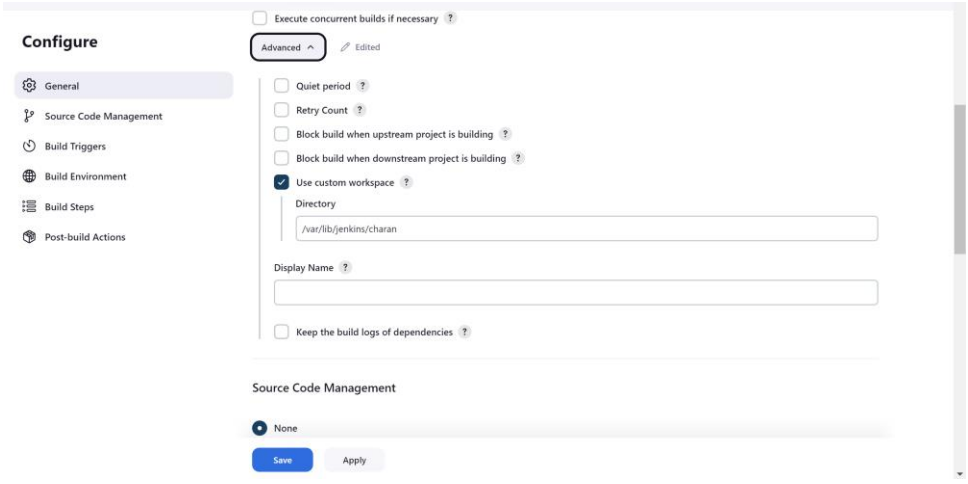
Note: Not all files are included in the default backup. To ensure every file is backed up, specify them under "Backup additional files" in the ThinBackup configuration settings.

Custom Workspace:

By default, job files are saved in **/var/lib/jenkins/workspace**. If you want to set a custom workspace directory, such as **/var/lib/jenkins/(folder_name),** follow these steps:

1. Go to the **Configure** section of the job.
2. Scroll down to the **General** section and click on **Advanced**.
3. Enter the desired directory path in the **Custom workspace** field.
4. Save the configuration.

```
[root@jenkins charan]# pwd
/var/lib/jenkins/charan
[root@jenkins charan]#
```



Note: If setting the custom workspace fails due to permission issues, provide full permissions to the directory by using the command chmod 777 <directory_name>.

Reminder: The custom workspace path should always start with /var/lib/jenkins/ to ensure consistency with Jenkins's default directory structure.

```
[root@jenkins jenkins]# chmod 777 charan
[root@jenkins jenkins]# cd charan
[root@jenkins charan]# ll
total 0
-rw-r--r-- 1 jenkins jenkins 0 Jun 19 08:59 charan.pdf
[root@jenkins charan]#
```