



JENKINS

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

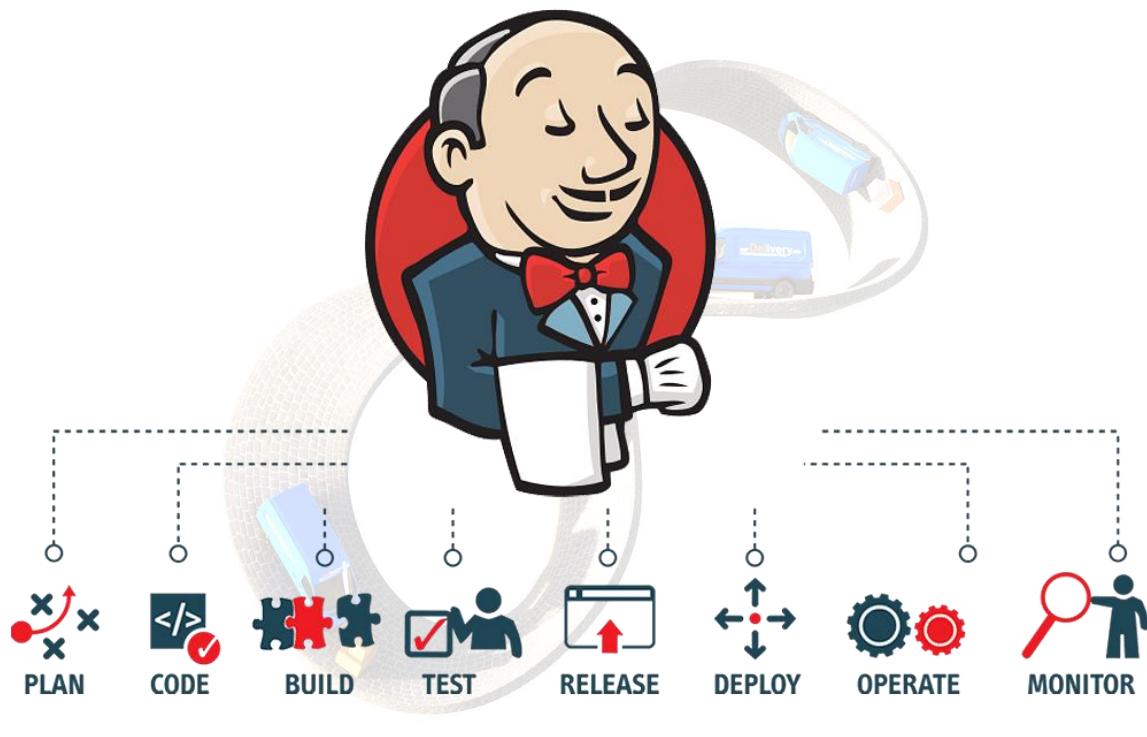
SWIPE LEFT



CHARAN SURYA KILANA

Pipeline:

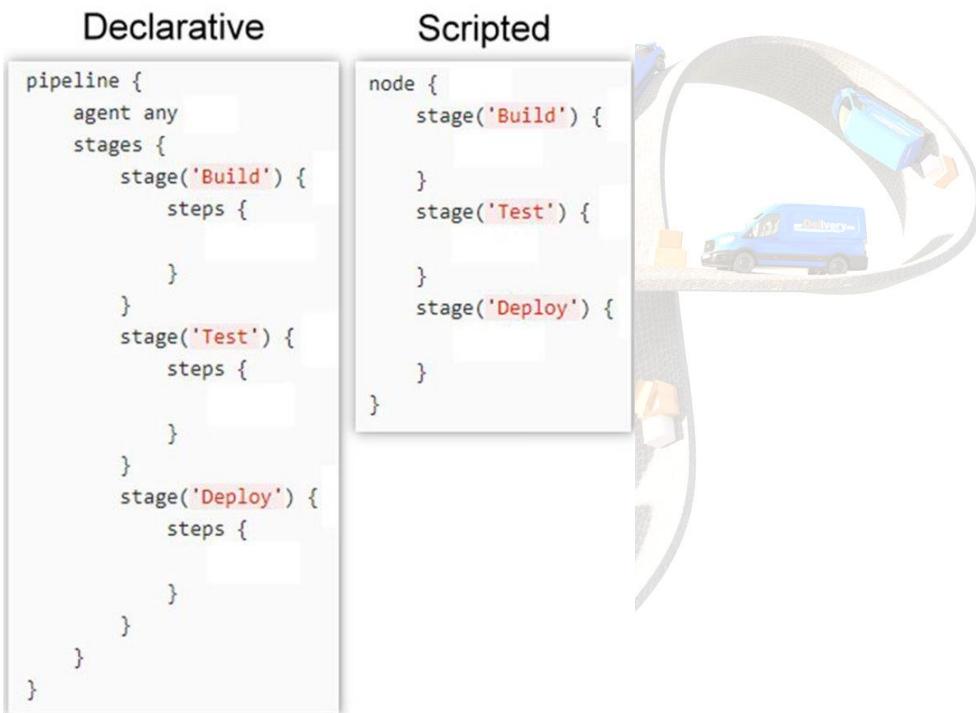
A pipeline refers to a sequence of events or processes that are interconnected, allowing for the systematic execution of tasks or operations. In the context of software development, a pipeline typically automates the steps involved in building, testing, and deploying code, ensuring a streamlined and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



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

{

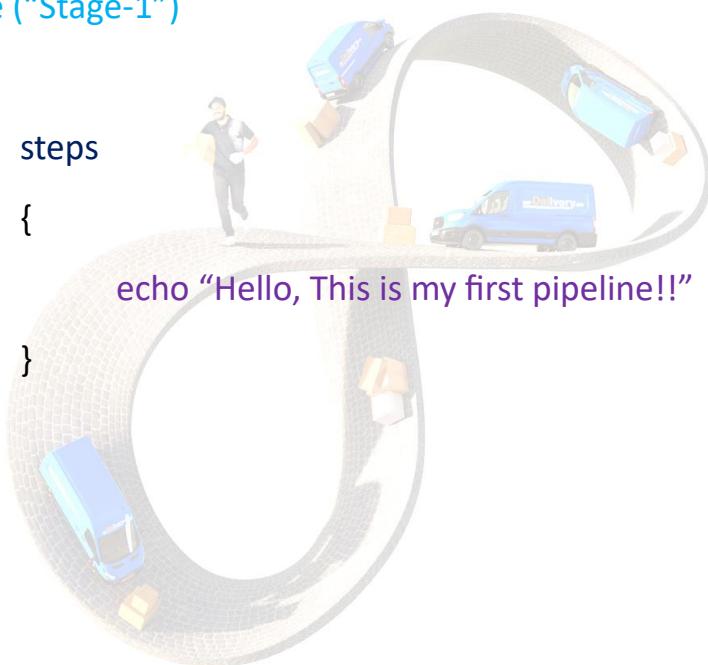
echo "Hello, This is my first pipeline!!"

}

}

}

}



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.

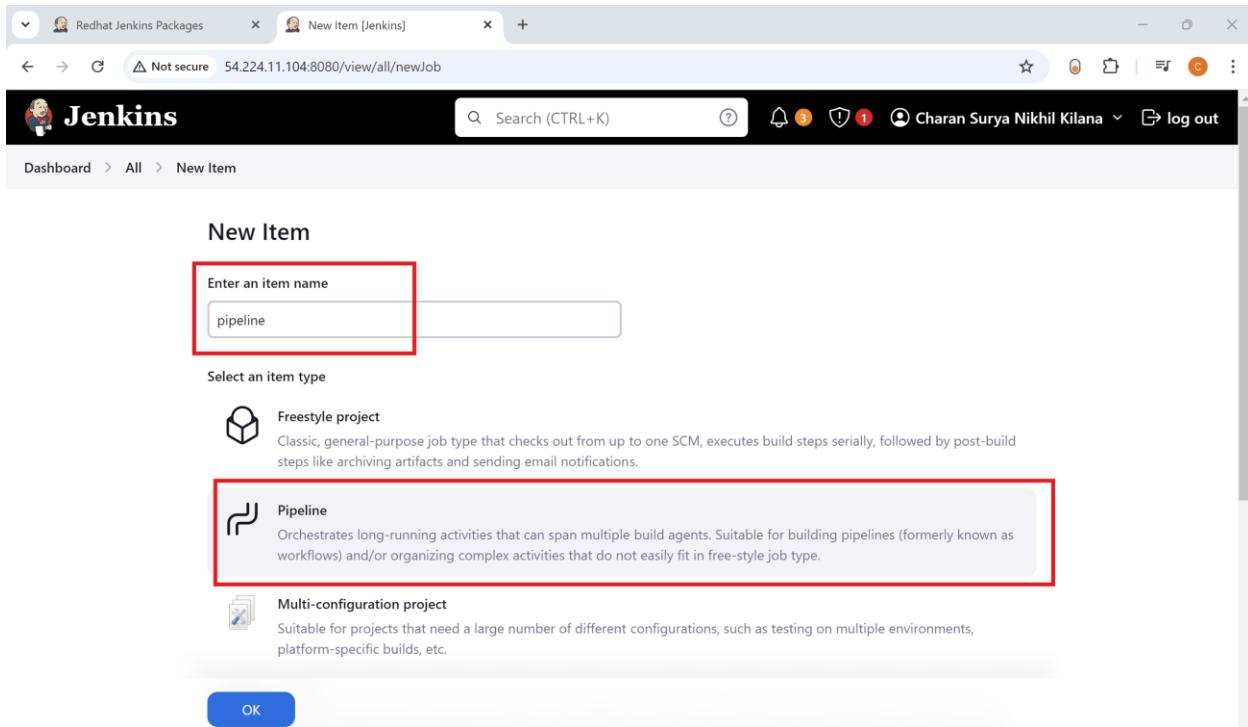
Stage: A stage in a Jenkinsfile represents a logically distinct part of the Pipeline process, such as building, testing, or deploying the application. Each stage contains steps that execute specific tasks required at that point in the Pipeline.

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



The screenshot shows the Jenkins 'New Item' creation interface. In the top-left corner, there are tabs for 'Redhat Jenkins Packages' and 'New Item [Jenkins]'. The main title bar says 'Jenkins' and shows the user 'Charan Surya Nikhil Kilana'. Below the title bar, the breadcrumb navigation shows 'Dashboard > All > New Item'. The main content area is titled 'New Item'. It has a form with a red box around the 'Enter an item name' input field, which contains the value 'pipeline'. Below this, a section titled 'Select an item type' lists three options: 'Freestyle project', 'Pipeline', and 'Multi-configuration project'. The 'Pipeline' option is highlighted with a red box. At the bottom of the form is a blue 'OK' button.

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.

The screenshot shows the Jenkins Pipeline configuration page. The 'Pipeline' tab is selected. In the 'Definition' section, 'Pipeline script' is chosen. The script area contains the following Groovy code:

```
1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage("Code")
7         {
8             steps
9             {
10                echo "Hello, Welcome to pipelines"
11            }
12        }
13    }
14 }
```

A red box highlights the line 'echo "Hello, Welcome to pipelines"' in the script. Below the script, there is a checkbox for 'Use Groovy Sandbox' which is checked. At the bottom, there are 'Save' and 'Apply' buttons.

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.

The screenshot shows the Jenkins Pipeline job page for 'pipeline'. The left sidebar has 'Build Now' highlighted with a red box. The main area shows the 'Stage View' which displays a single stage named 'Code' with a duration of 229ms. Below it is the 'Build History' section, also highlighted with a red box. It lists the following builds:

- Last build (#1), 41 sec ago
- Last stable build (#1), 41 sec ago
- Last successful build (#1), 41 sec ago
- Last completed build (#1), 41 sec ago

At the bottom of the history, there are links for 'Atom feed for all' and 'Atom feed for failures'.

The screenshot shows the Jenkins Pipeline console output for build #1. The output starts with a message from the user 'Charan Surya Nikhil Kilana'. It then shows the pipeline stages: 'Start of Pipeline', 'node', 'stage', 'Code', and 'echo'. The 'echo' stage outputs the message 'Hello, Welcome to pipelines'. Finally, it concludes with 'End of Pipeline' and 'Finished: SUCCESS'. A red box highlights the 'Console Output' section.

```

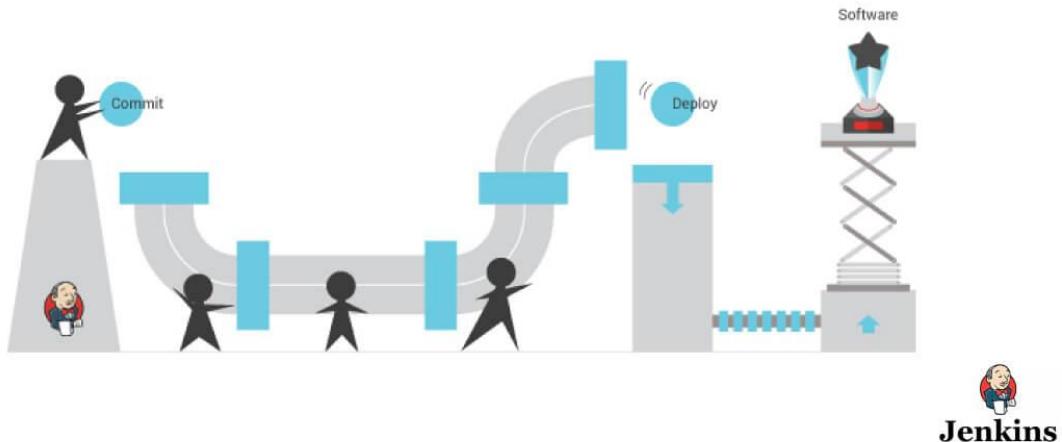
Started by user Charan Surya Nikhil Kilana
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Code)
[Pipeline] echo
Hello, Welcome to pipelines
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

Multi-Stage-Pipeline



Understanding the basics of the Jenkins Pipeline



Script ?

```
1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage("Code")
7         {
8             steps
9             {
10                echo "Hello Code"
11            }
12        }
13        stage("Build")
14        {
15            steps
16            {
17                echo "Hello Build"
18            }
19        }
20        stage("Deploy")
21        {
22            steps
23            {
24                echo "Hello Deploy"
25            }
26        }
27    }
28 }
```

The screenshot shows the Jenkins Pipeline configuration page. At the top, there's a code editor window titled 'Script' containing the Groovy pipeline script. The script defines a pipeline with three stages: 'Code', 'Build', and 'Deploy', each containing an 'echo' step. Below the code editor is a decorative illustration of a blue delivery van with a package. The main browser window has a title bar 'Redhat Jenkins Packages pipeline Config [Jenkins]' and a URL '54.224.11.104:8080/job/pipeline/configure'. The navigation bar shows 'Dashboard > pipeline > Configuration'. On the left, there's a sidebar with tabs: 'General', 'Advanced Project Options', and 'Pipeline' (which is highlighted with a red box). At the bottom of the configuration area, there's a checkbox 'Use Groovy Sandbox' with a checked status.

Screenshot of the Jenkins Pipeline status page:

The pipeline has a green status icon and the name "pipeline".

Stage View

Code	Build	Deploy
Average stage times: (Average full run time: ~3s) #2 Sep 02 11:43 No Changes 173ms	107ms	92ms
#1 Sep 02 11:37 No Changes 229ms		

Build History

- #2 | Sep 2, 2024, 4:43 PM (highlighted with a red box)
- #1 | Sep 2, 2024, 4:37 PM

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

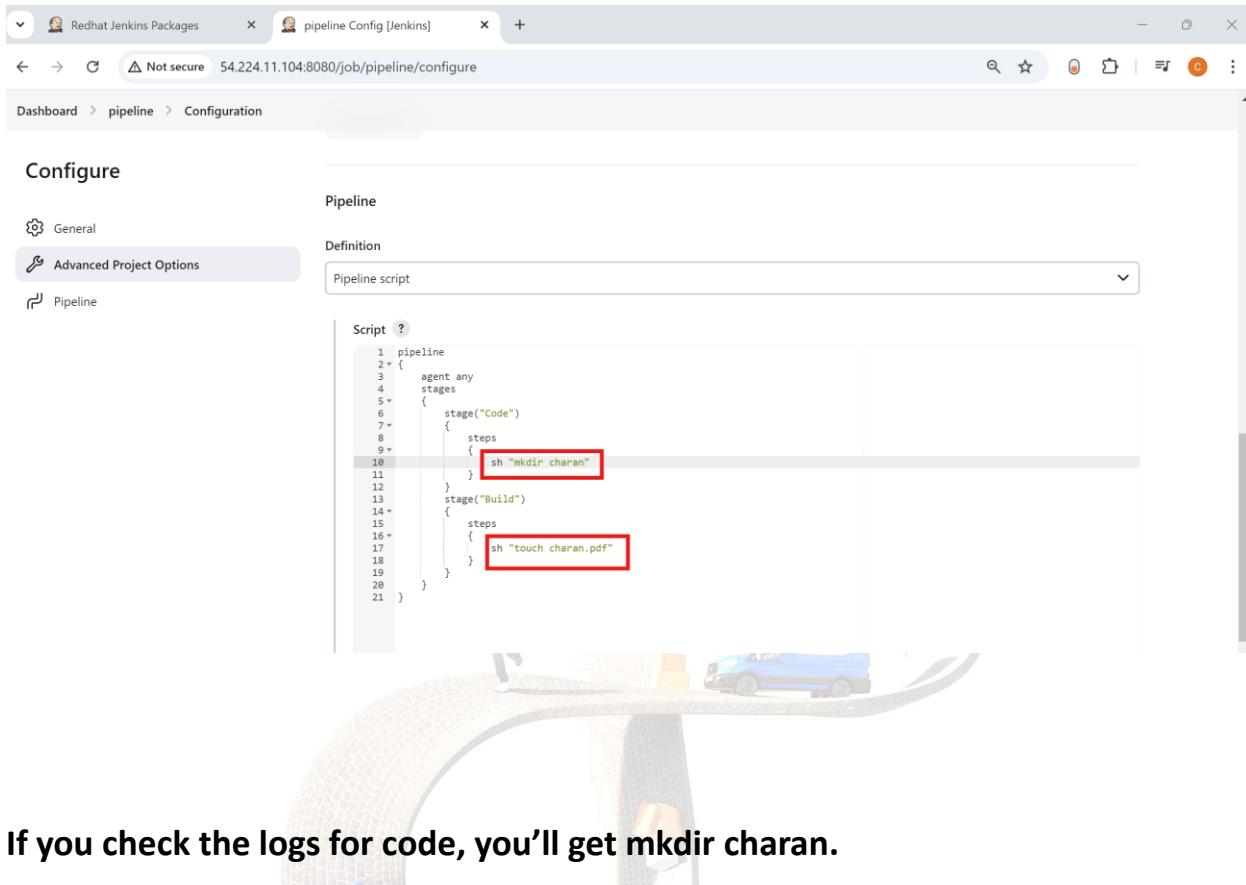
Screenshot of the Jenkins Pipeline #2 Console output page:

The pipeline has a green status icon and the name "pipeline #2".

Console Output

```
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".

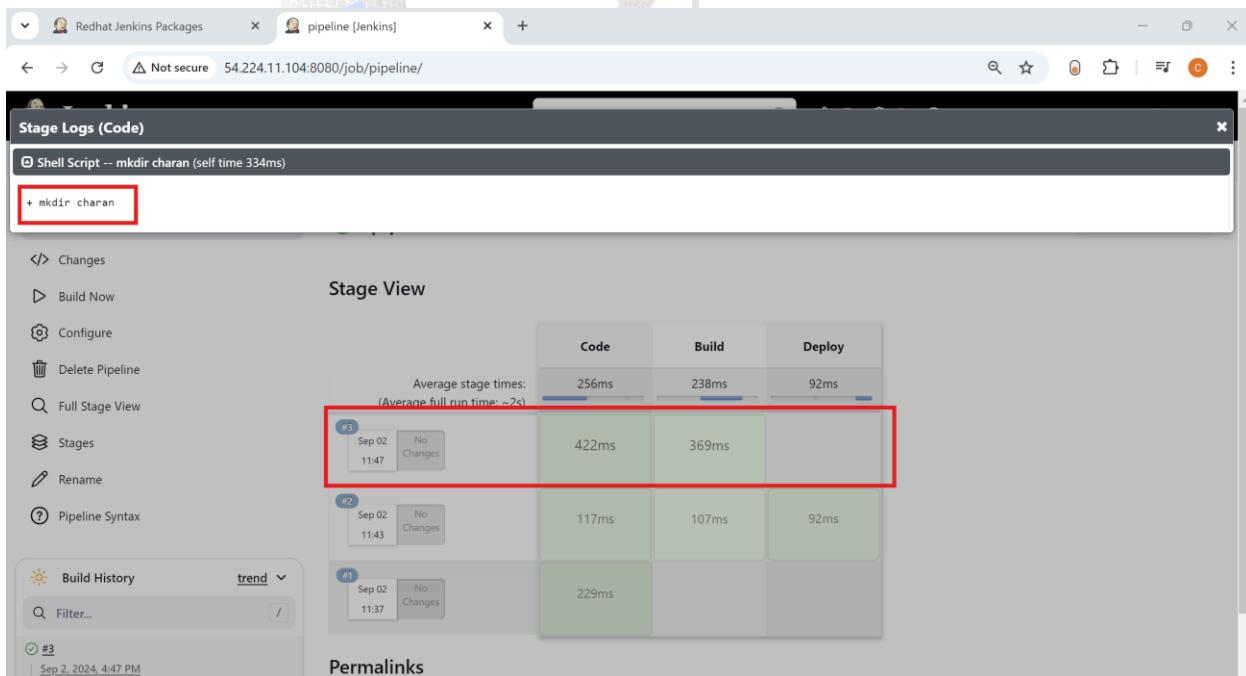


The screenshot shows the Jenkins Pipeline configuration page. Under the 'Definition' section, 'Pipeline script' is selected. The script content is as follows:

```
1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage("Code")
7         {
8             steps
9             {
10                sh "mkdir charan"
11            }
12        }
13        stage("Build")
14        {
15            steps
16            {
17                sh "touch charan.pdf"
18            }
19        }
20    }
21 }
```

Two specific lines of code are highlighted with red boxes: 'sh "mkdir charan"' at line 10 and 'sh "touch charan.pdf"' at line 17.

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



The screenshot shows the Jenkins Stage View for the 'Code' stage. The 'Stage Logs (Code)' panel is open, displaying the log output for the 'Shell Script -- mkdir charan (self time 334ms)' step. The log entry '+ mkdir charan' is highlighted with a red box.

The Stage View table shows three stages with their average times:

Code	Build	Deploy
256ms	238ms	92ms

The first stage (#2) has a duration of 422ms, the second stage (#2) has a duration of 117ms, and the third stage (#1) has a duration of 229ms.

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

```
root@ip-172-31-25-17:/var/lib ~ + | ~
[root@ip-172-31-25-17 ~]# cd /var/lib/jenkins/workspace
[root@ip-172-31-25-17 workspace]# ls
pipeline pipeline@tmp
[root@ip-172-31-25-17 workspace]# cd pipeline
[root@ip-172-31-25-17 pipeline]# ls
charan charan.pdf
[root@ip-172-31-25-17 pipeline]#
```

Multiple commands over a single line in a pipeline stage

```
Script ?  
1 pipeline  
2 {  
3     agent any  
4     stages  
5 {  
6         stage("Code")  
7 {  
8             steps  
9 {  
10                sh "mkdir surya"  
11                sh "touch surya.pdf"  
12                sh "date"  
13                sh "cal"  
14            }  
15        }  
16    }  
17 }
```

The image contains two screenshots of a Jenkins pipeline configuration and its execution.

Screenshot 1: Pipeline Configuration

This screenshot shows the Jenkins pipeline configuration page. The pipeline script is defined as follows:

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

A red box highlights the pipeline script area.

Screenshot 2: Pipeline Execution Stage Logs and View

This screenshot shows the Jenkins pipeline execution stage logs and view. The stage logs for the "Code" stage show the following commands and their execution times:

- Shell Script - mkdir surya (self time 286ms)
- Shell Script - touch surya.pdf (self time 289ms)
- Shell Script -- date (self time 284ms)
- Shell Script -- cal (self time 283ms)

A red box highlights the stage logs area.

The stage view shows the following details:

- Average stage times: 530ms, 238ms, 92ms.
- Code stage duration: 1s.

A red box highlights the stage view area.

Now one more file and folder created in the job.

This screenshot shows a terminal session on a Linux system (root user) where the Jenkins workspace is checked out. The terminal output shows the creation of a directory named "pipeline" and files "charan.pdf" and "surya.pdf".

```
[root@ip-172-31-25-17 ~]# cd /var/lib/jenkins/workspace
[root@ip-172-31-25-17 workspace]# ls
pipeline  pipeline@tmp
[root@ip-172-31-25-17 workspace]# cd pipeline
[root@ip-172-31-25-17 pipeline]# ls
charan  charan.pdf
[root@ip-172-31-25-17 pipeline]# ls
charan  charan.pdf  surya  surya.pdf
[root@ip-172-31-25-17 pipeline]#
```

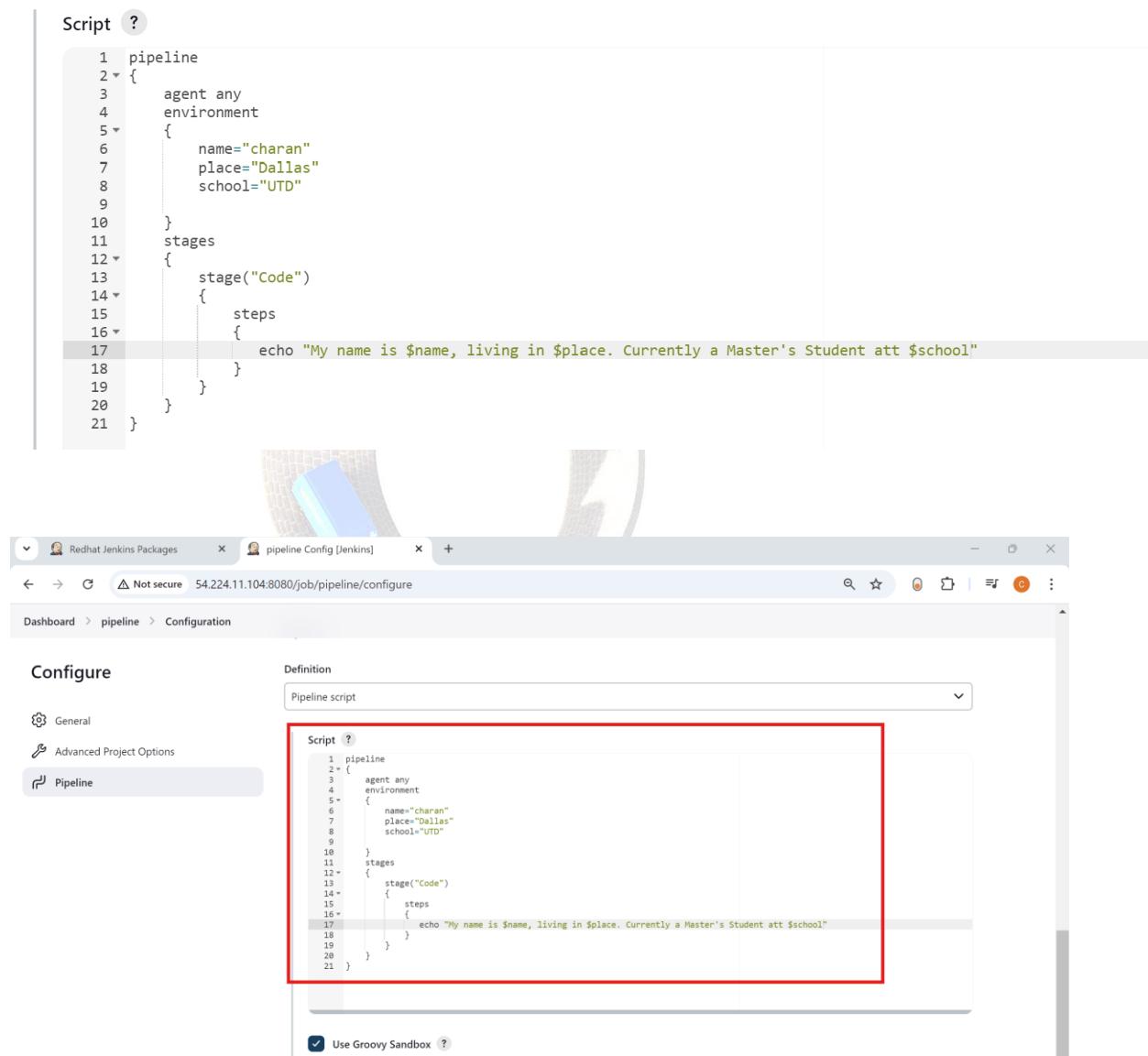
Declaring Variables

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

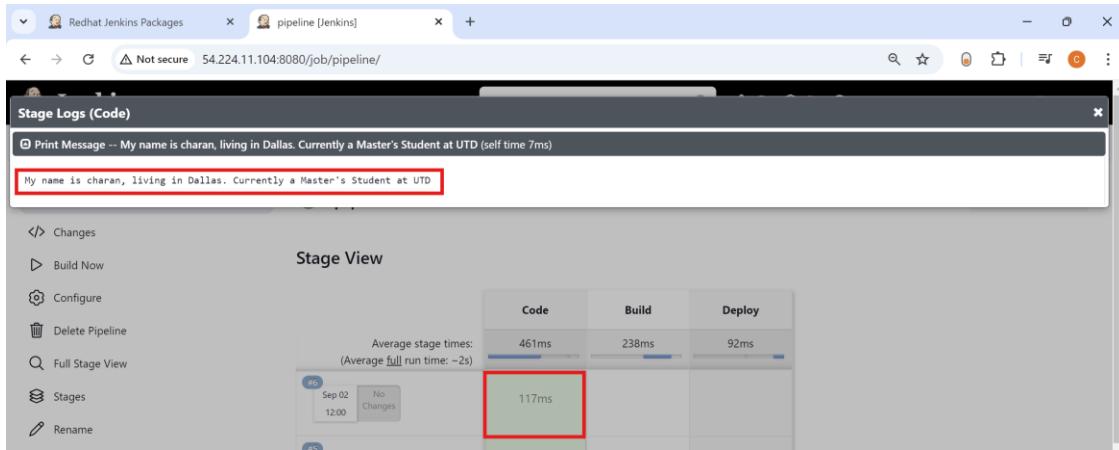
Pipeline script



The screenshot shows the Jenkins Pipeline configuration page. The title bar reads "Redhat Jenkins Packages" and "pipeline Config [Jenkins]". The URL is "Not secure 54.224.11.104:8080/job/pipeline/configure". The navigation bar includes "Dashboard", "pipeline", and "Configuration". On the left, there's a sidebar with "Configure" tabs: "General", "Advanced Project Options", and "Pipeline". The "Pipeline" tab is selected. The main content area is titled "Definition" and contains a "Pipeline script" input field. A red box highlights the Groovy script code within this field. The code is as follows:

```
1 pipeline
2 {
3     agent any
4     environment
5     {
6         name="charan"
7         place="Dallas"
8         school="UTD"
9     }
10 }
11 stages
12 {
13     stage("Code")
14 {
15     steps
16 {
17         echo "My name is $name, living in $place. Currently a Master's Student att $school"
18     }
19 }
20 }
21 }
```

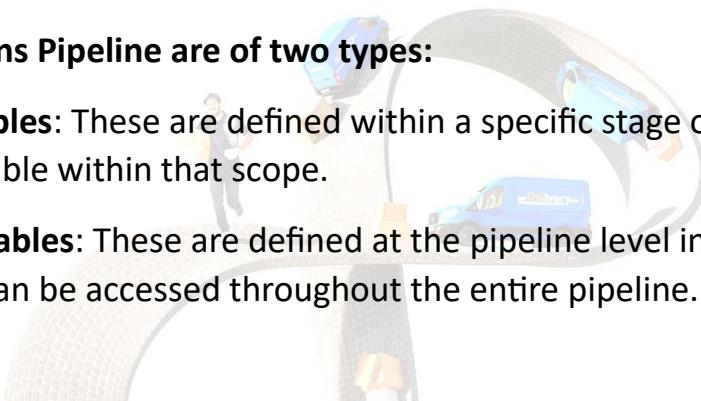
At the bottom of the script editor, there is a checkbox labeled "Use Groovy Sandbox" with a checked status.



The screenshot shows a Jenkins pipeline stage log titled "Stage Logs (Code)". It displays a single log entry: "My name is charan, living in Dallas. Currently a Master's Student at UTD". This log entry is highlighted with a red box. Below the log, there is a "Stage View" section with three tabs: "Code", "Build", and "Deploy". The "Build" tab is selected, showing average stage times: 461ms for Code, 238ms for Build, and 92ms for Deploy. A specific build step is highlighted with a red box, showing a run time of 117ms.

Variables in Jenkins Pipeline are of two types:

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



Script ?

```

1 pipeline
2 {
3     agent any
4     environment
5     {
6         x=100
7     }
8     stages
9     {
10        stage("First Block")
11        {
12            steps
13            {
14                echo "The value of x is $x"
15            }
16        }
17    }
18    stage("Second Block")
19    {
20        environment
21        {
22            y=50
23        }
24        steps
25        {
26            echo "The value of x is $x"
27            echo "The value of y is $y"
28        }
29    }
30 }
31 }
```

The image consists of two screenshots of a Jenkins pipeline interface. The top screenshot shows the 'Stage Logs (First Block)' page, which contains a log entry: 'Print Message -- The value of x is 100 (self time 7ms)' and 'The value of x is 100'. The bottom screenshot shows the 'Stage Logs (Second Block)' page, which contains log entries: 'Print Message -- The value of x is 100 (self time 16ms)' and 'Print Message -- The value of y is 50 (self time 6ms)'. Both screenshots also show a 'Stage View' section with a timeline and duration details.

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.

The screenshot shows the Jenkins Manage Jenkins interface at the URL <http://54.224.11.104:8080/manage/>. The 'Manage Jenkins' page is selected in the breadcrumb navigation. A red box highlights the 'System Configuration' section. Inside this section, the 'System' item is also highlighted with a red box. Other items include 'Tools', 'Plugins', 'Nodes', and 'Clouds'. A message at the top right states: '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.' A 'Dismiss' button is present next to the message.

The screenshot shows the 'System' configuration page at the URL <http://54.224.11.104:8080/manage/configure>. The 'Global properties' section is highlighted with a red box. Under 'Environment variables', there is a 'List of variables' section with an 'Add' button. Other sections include 'Disable deferred wipeout on this node', 'Disk Space Monitoring Thresholds', and 'Tool Locations'. A note at the top states: 'Without a resource root URL, resources will be served from the Jenkins URL with Content-Security-Policy set.'

The screenshot shows the 'System' configuration page at the URL <http://54.224.11.104:8080/manage/configure>. The 'Environment variables' section is highlighted with a red box. Two environment variables are listed: 'course' with value 'Information Technology and Management' and 'school' with value 'University of Texas at Dallas'. Each variable has a red box highlighting its 'Name' and 'Value' fields.

The screenshot shows the Jenkins Pipeline configuration page. The pipeline script is defined as follows:

```
1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage("First Block")
7         {
8             steps
9             {
10                echo "Im studying at $school and my course is $course"
11            }
12        }
13    }
14 }
```

A red box highlights the line `echo "Im studying at $school and my course is $course"`. Below the script, there is a checkbox labeled `Use Groovy Sandbox` which is checked.

The screenshot shows the Jenkins Stage Logs for the 'First Block'. The log output is:

```
Print Message -- Im studying at University of Texas at Dallas and my course is Information Technology and Management (self time 7ms)
Im studying at University of Texas at Dallas and my course is Information Technology and Management
```

A red box highlights the second line of the log output. The Stage View below shows two stages: 'First Block' (86ms) and 'Second Block' (151ms). A tooltip indicates 'Sep 02 12:16 No Changes'.

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.

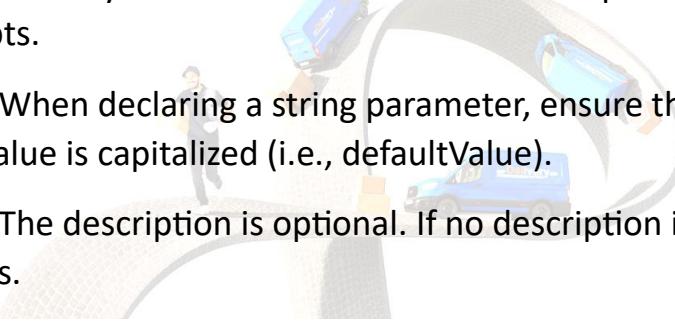
Pipeline with Parameters:

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 ?

```
1 pipeline
2 {
3     agent any
4     parameters
5     {
6         string(name: 'CICD', defaultValue: 'Jenkins', description: '')
7     }
8     stages
9     {
10        stage ("Code")
11        {
12            steps
13            {
14                sh "cat"
15            }
16        }
17    }
18 }
```

Redhat Jenkins Packages pipeline Config [Jenkins] Not secure 54.224.11.104:8080/job/pipeline/configure

Dashboard > pipeline > Configuration

Configure

General

Advanced Project Options

Pipeline

This project is parameterized

Pipeline speed/durability override

Preserve stashes from completed builds

Throttle builds

Build Triggers

Build after other projects are built

Build periodically

GitHub hook trigger for GITScm polling

Poll SCM

Before it was empty, later got filled with parameters.

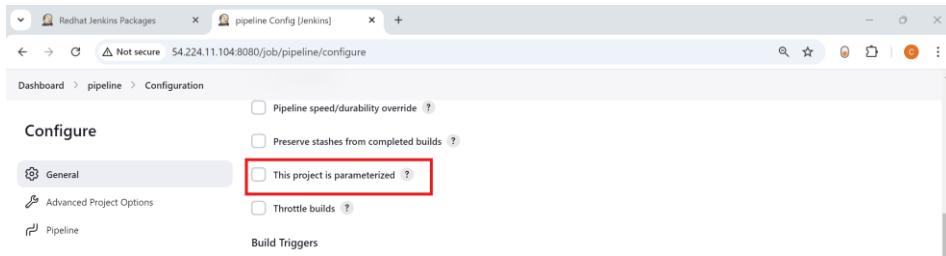
The screenshot shows the Jenkins job configuration interface for a job named 'job-2'. The 'Configure' section is open, and the 'General' tab is selected. A 'String Parameter' is defined with the name 'CICD' and a default value of 'Jenkins'. The 'Advanced Project Options' and 'Pipeline' tabs are also visible.

Boolean Parameter

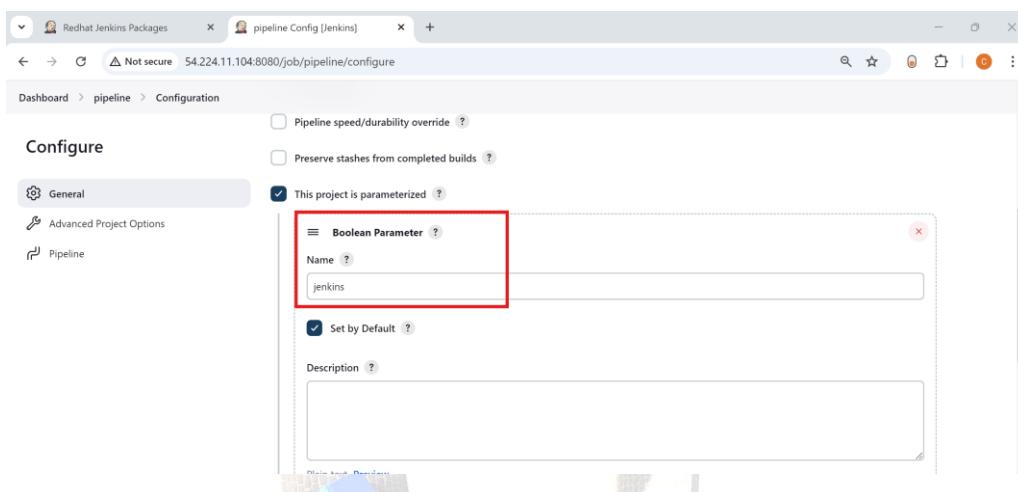
The screenshot shows the Jenkins pipeline configuration interface. The 'Pipeline' tab is selected. In the 'Definition' section, the 'Pipeline script' tab is chosen. The Groovy script defines a boolean parameter named 'Jenkins' with a default value of 'true' and an empty description. The 'Use Groovy Sandbox' checkbox is checked.

```
1 pipeline
2 {
3     agent any
4     parameters
5     {
6         booleanParam(name: 'Jenkins', defaultValue: true, description: '')
7     }
8     stages
9     {
10         stage("First Block")
11         {
12             steps
13             {
14                 echo "I'm studying at $school and my course is $course"
15             }
16         }
17     }
18 }
```

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 ?  
1 pipeline  
2   -> {  
3     agent any  
4     parameters  
5     -> {  
6       choice(name: 'CICD', choices: ['Jenkins', 'bamboo', 'circle-ci', 'aws code pipeline'], description: '')  
7     }  
8     stages  
9     -> {  
10       stage ("Code")  
11       -> {  
12         steps  
13         -> {  
14           sh "cal"  
15         }  
16       }  
17     }  
18 }
```

After Build

This project is parameterized ?

Choice Parameter ?

Name ?

CICD

Choices ?

Jenkins
bamboo
circle-ci
aws code pipeline

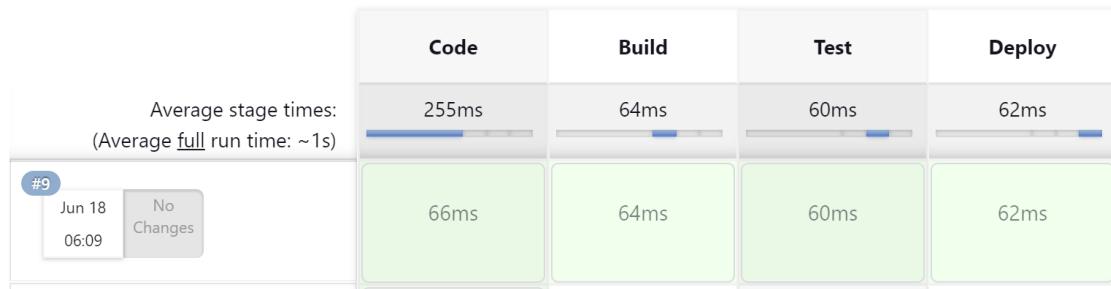
Description ?



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.

Stage View



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.

The screenshots illustrate the Jenkins pipeline configuration and execution process.

Screenshot 1: Pipeline Configuration

```

1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage("Code")
7         {
8             steps
9             {
10                echo "This is code stage"
11            }
12        }
13        stage("Build")
14        {
15            steps
16            {
17                echo "This is build stage"
18            }
19        }
20        stage("deploy")
21        {
22            steps
23            {
24                input('Approval Deployment?')
25                echo "This is deploy stage"
26            }
27        }
28    }
29 }

```

Screenshot 2: Pipeline Execution

The Jenkins sidebar shows the most recent build (#19) is selected:

- Status
- </> Changes
- Build Now** (highlighted)
- Configure
- Delete Pipeline
- Full Stage View
- Stages
- Rename
- Pipeline Syntax

The Stage View shows the pipeline stages and their average times:

Stage	Average Time
Code	94ms
Build	54ms
deploy	51ms

The Deploy stage (#19) is currently executing, with an approval dialog displayed:

#19 Sep 02 12:48 No Changes
94ms Approval Deployment? Proceed Abort 51ms (waited for 0s)

The Jenkins sidebar also shows the most recent build (#19) is selected:

- Status
- </> Changes
- Build Now**
- Configure
- Delete Pipeline
- Full Stage View
- Stages
- Rename
- Pipeline Syntax

Build History (highlighted):

- #19 Sep 2, 2024, 5:48 PM

Permalinks:

- Last build (#18), 1 min 10 sec ago
- Last stable build (#16), 9 min 16 sec ago
- Last successful build (#16), 9 min 16 sec ago

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.

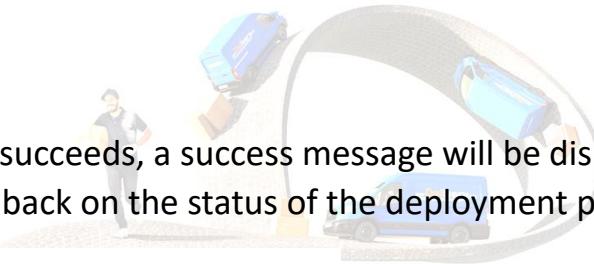
Pipeline with Post-Build Actions: Post-build actions in Jenkins are used to perform tasks after the main build process completes, often to provide status updates or handle cleanup activities.

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 ?  
1 pipeline  
2 {  
3     agent any  
4     stages  
5     {  
6         stage ("Code")  
7         {  
8             steps  
9                 {  
10                    echo "This is code stage"  
11                }  
12            }  
13        }  
14    post  
15    {  
16        success  
17        {  
18            echo "pipeline is successful"  
19        }  
20    }  
21 }
```

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

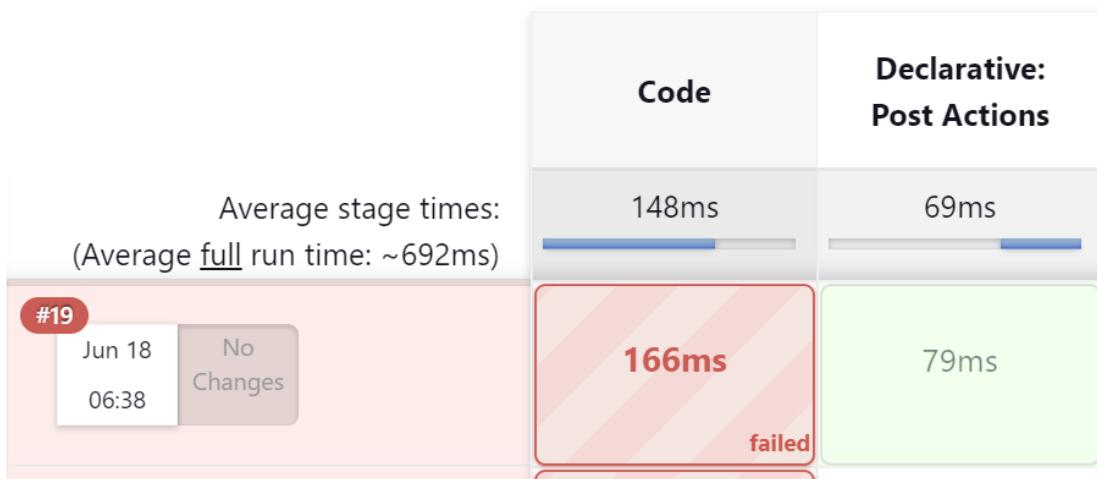
In the declarative post actions, it is marked as success because the pipeline is configured to trigger a failure message in the post actions section. Also, there's a typo in the 'echo' command.

Script ?

```
1 pipeline
2 {
3     agent any
4     stages
5     {
6         stage ("Code")
7         {
8             steps
9             {
10                echo "This is code stage"
11            }
12        }
13    }
14    post
15    {
16        failure
17        {
18            echo "pipeline is failed"
19        }
20    }
21 }
```

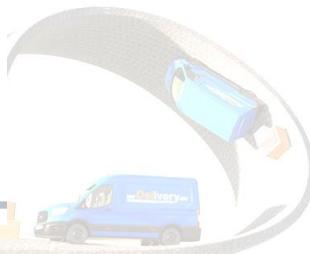
Below is the stage view. Since your code failed you get failure displayed.

Stage View



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

```
Script ?  
1 pipeline  
2 {  
3     agent any  
4     stages  
5     {  
6         stage ("Code")  
7         {  
8             steps  
9             {  
10                echo "This is code stage"  
11            }  
12        }  
13    }  
14    post  
15    {  
16        failure  
17        {  
18            echo "pipeline is failed"  
19        }  
20        success  
21        {  
22            echo "This will be printed if pipeline success"  
23        }  
24        always  
25        {  
26            echo "I print always"  
27        }  
28    }  
29 }
```



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
```

```
{
```

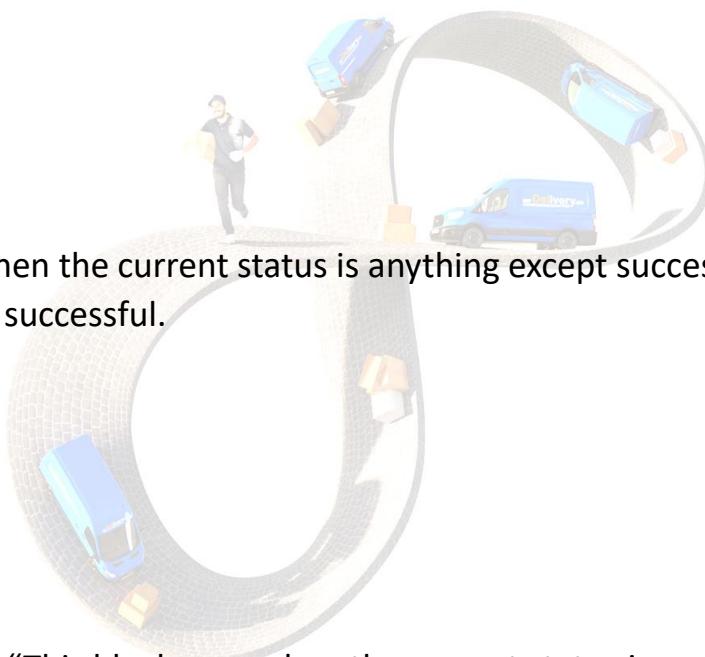
```
    regression
```

```
{
```

```
    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
```

```
{
```

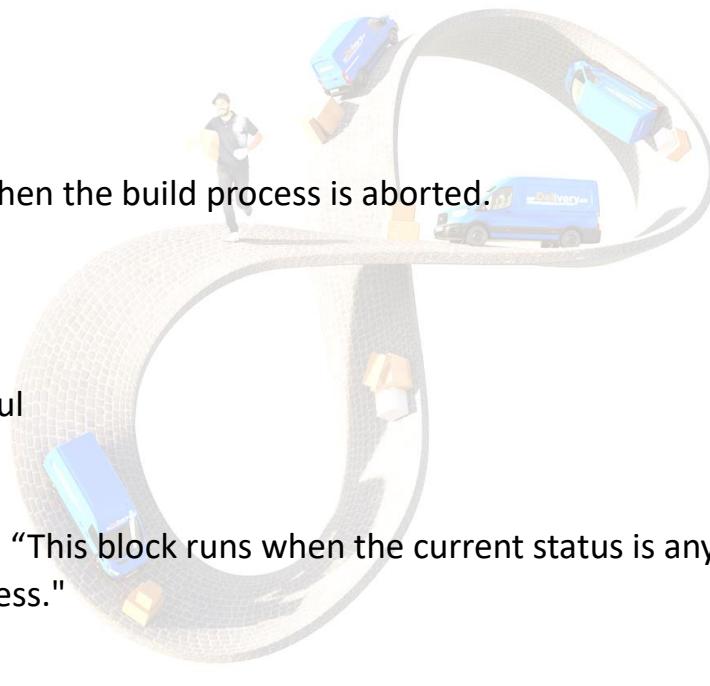
```
    unsuccessful
```

```
{
```

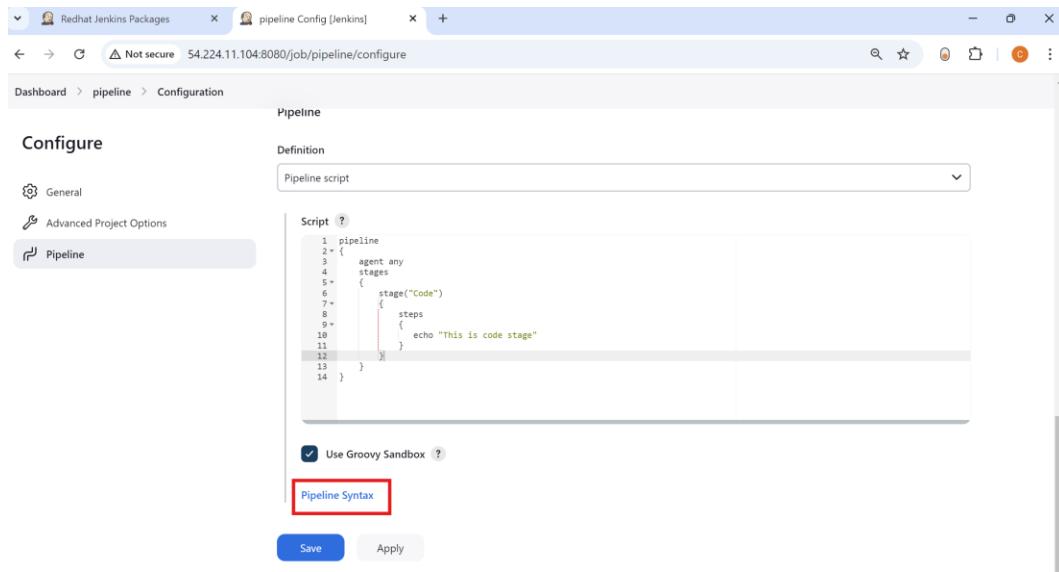
```
        echo "This block runs when the current status is anything except  
        success."
```

```
}
```

```
}
```

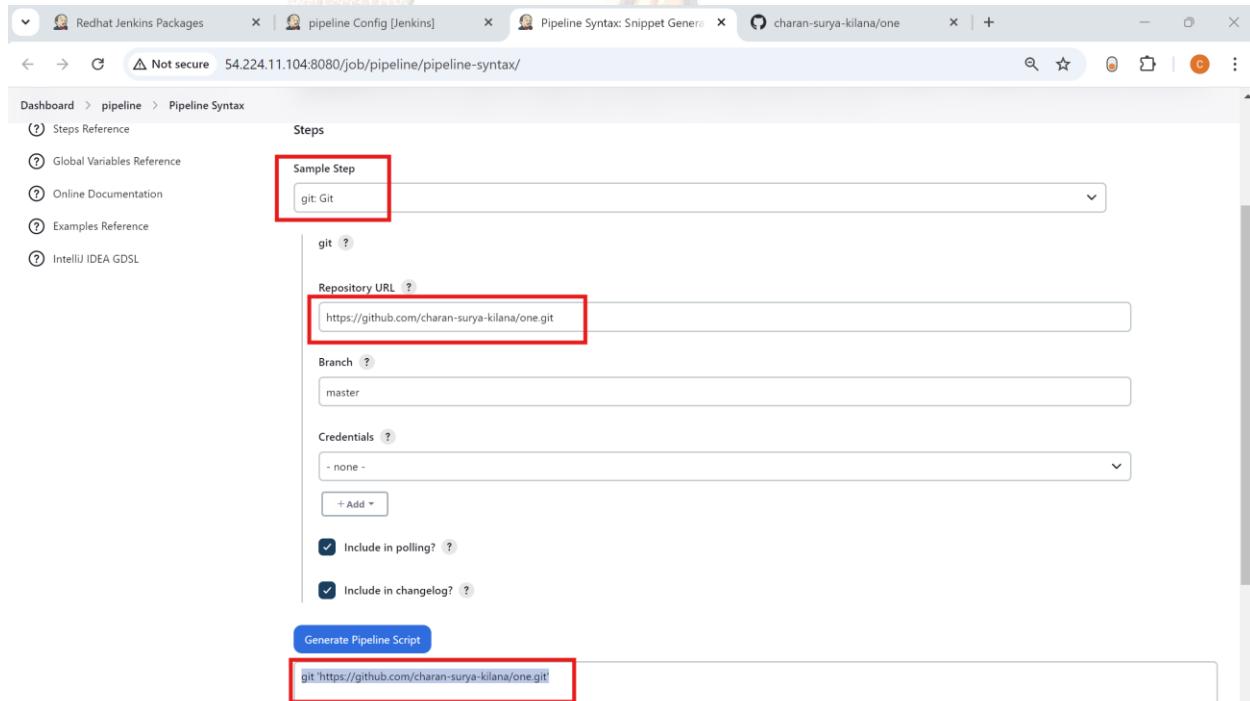


Connecting Git with pipeline:



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.



Redhat Jenkins Packages pipeline Config [Jenkins] Pipeline Syntax: Snippet Generator charan-surya-kilana/one

Not secure 54.224.11.104:8080/job/pipeline/configure

Dashboard > pipeline > Configuration

Configure

Pipeline

General Advanced Project Options Pipeline

Definition: Pipeline script

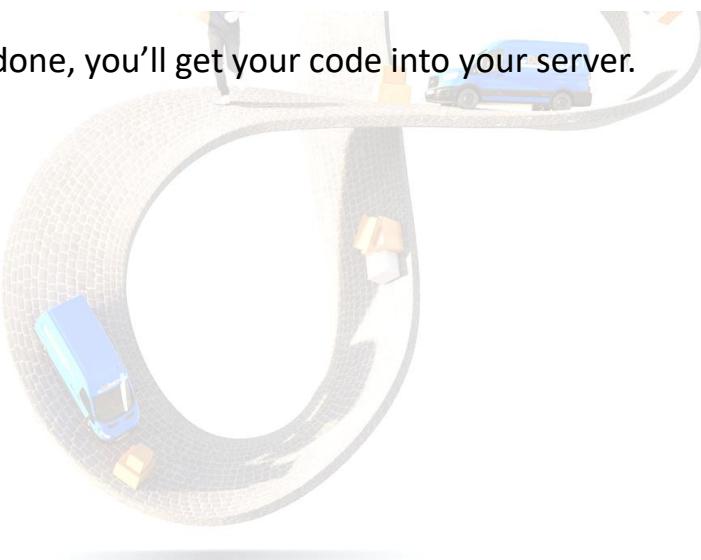
```
Script ?  
1 pipeline  
2 {  
3   agent any  
4   stages  
5   {  
6     stage("Code")  
7     {  
8       steps  
9       {  
10         git 'https://github.com/charan-surya-kilana/one.git'  
11       }  
12     }  
13   }  
14 }
```

Use Groovy Sandbox ?

Pipeline Syntax

Save Apply

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

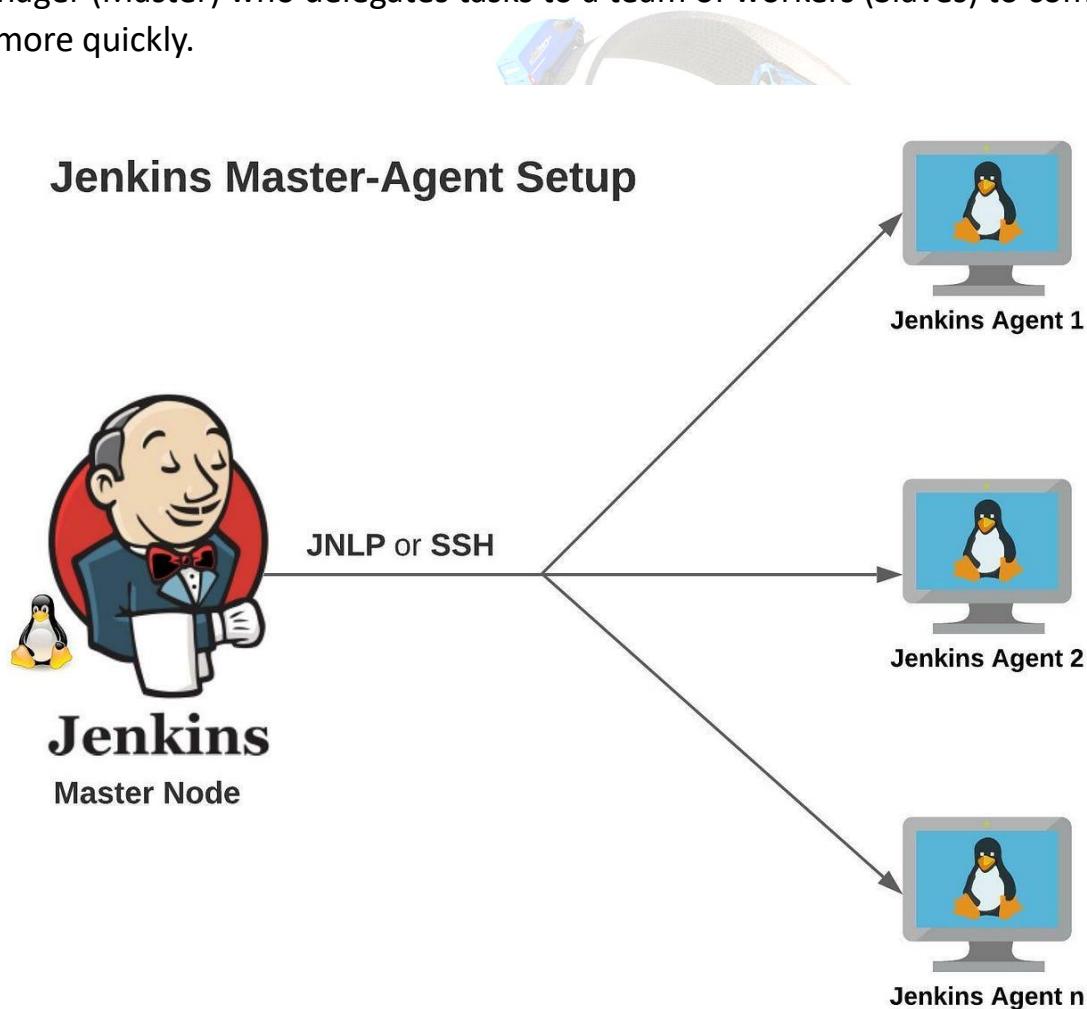


Jenkins Master-Slave Architecture

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**)

Instances (3) Info		Last updated less than a minute ago	Connect	Instance state ▾	Actions ▾	Launch instances ▾	
				Running ▾			< 1 >
<input type="checkbox"/>	Name ▾	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
<input type="checkbox"/>	Slave-1	i-00ec16ded6708c3d0	Running	t2.micro	Initializing	View alarms	us-east-1d
<input type="checkbox"/>	slave-2	i-0e6298e4f76b974c7	Running	t2.micro	Initializing	View alarms	us-east-1d
<input type="checkbox"/>	Master	i-0aac9892ec18174ce	Running	t2.micro	Initializing	View alarms	us-east-1d

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

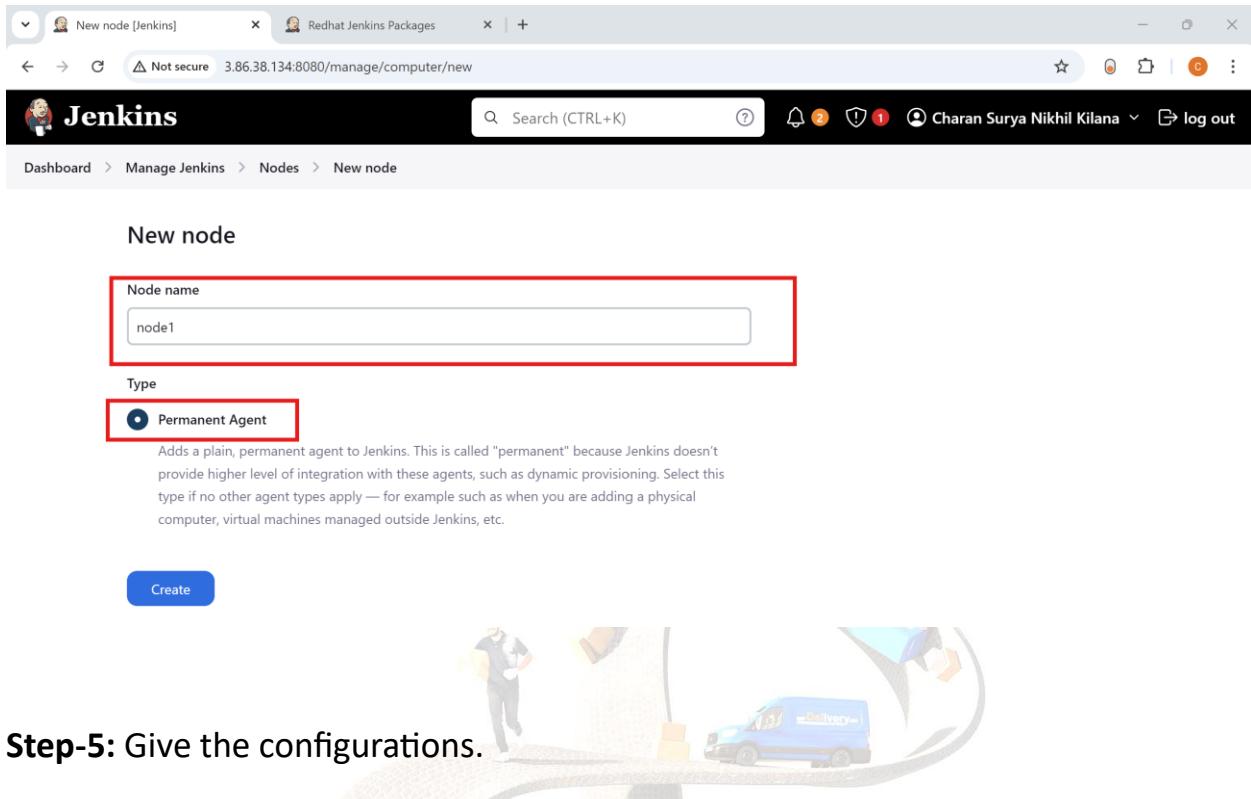
Step-4: Instructions for setting up a new node in Jenkins:

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 screenshot shows the Jenkins 'Manage Jenkins' page. In the top navigation bar, the URL is 3.86.38.134:8080/manage/. Below the navigation, there's a 'Refer to the documentation for details.' link. The main content area is titled 'System Configuration' and contains several sections: '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, highlighted with a red box), 'Clouds' (Add, remove, and configure cloud instances to provision agents on-demand), and 'Appearance' (Configure the look and feel of Jenkins). Below this is a 'Monitors' section with a 'New Monitor' button. The main part of the page is titled 'Nodes' and displays a table of existing nodes. The table has columns: S (Status), Name (Built-In Node), Architecture (Linux (amd64)), Clock Difference (In sync), Free Disk Space (5.79 GiB), Free Swap Space (0 B), Free Temp Space (5.79 GiB), and Response Time (0ms). A 'Configure Monitors' button is also present. At the bottom, there are icons for S, M, and L, and a 'Legend' link.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.79 GiB	0 B	5.79 GiB	0ms
Data obtained 6 min 53 sec							

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.



New node

Node name

node1

Type

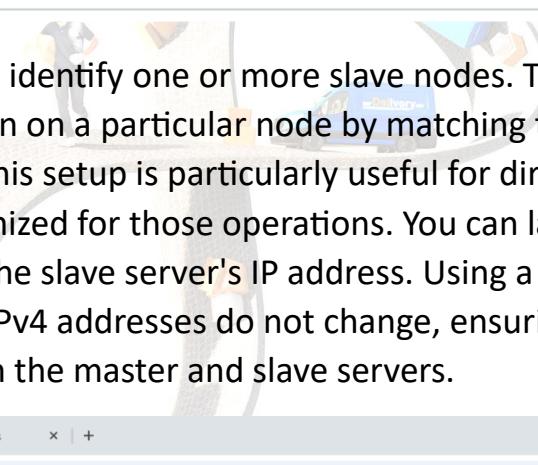
Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Create

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.

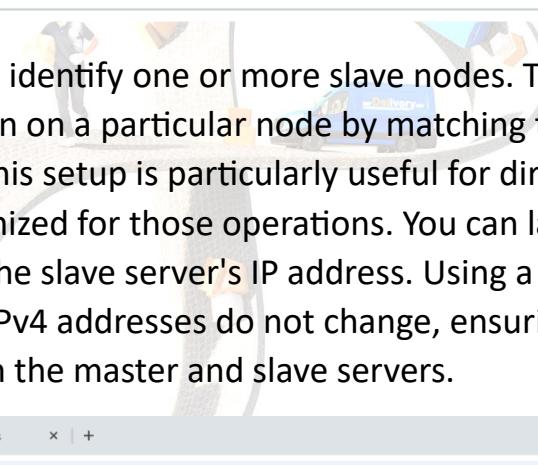


Screenshot of the Jenkins 'Create New Node' configuration page. The page shows fields for basic node setup:

- Name**: node1
- Description**: This is slave-1
- Number of executors**: 2
- Remote root directory**: /home/ec2-user/Jenkins

The 'Name' and 'Description' fields are highlighted with red boxes.

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.



Screenshot of the Jenkins 'Create New Node' configuration page, showing the 'Usage' section:

- Usage**: Only build jobs with label expressions matching this node
- Launch method**: Launch agents via SSH
- Host**: 172.31.95.249

The 'Usage' and 'Host' fields are highlighted with red boxes.

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.

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted)

Kind

SSH Username with private key

Scope ?

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

ID ?

slave

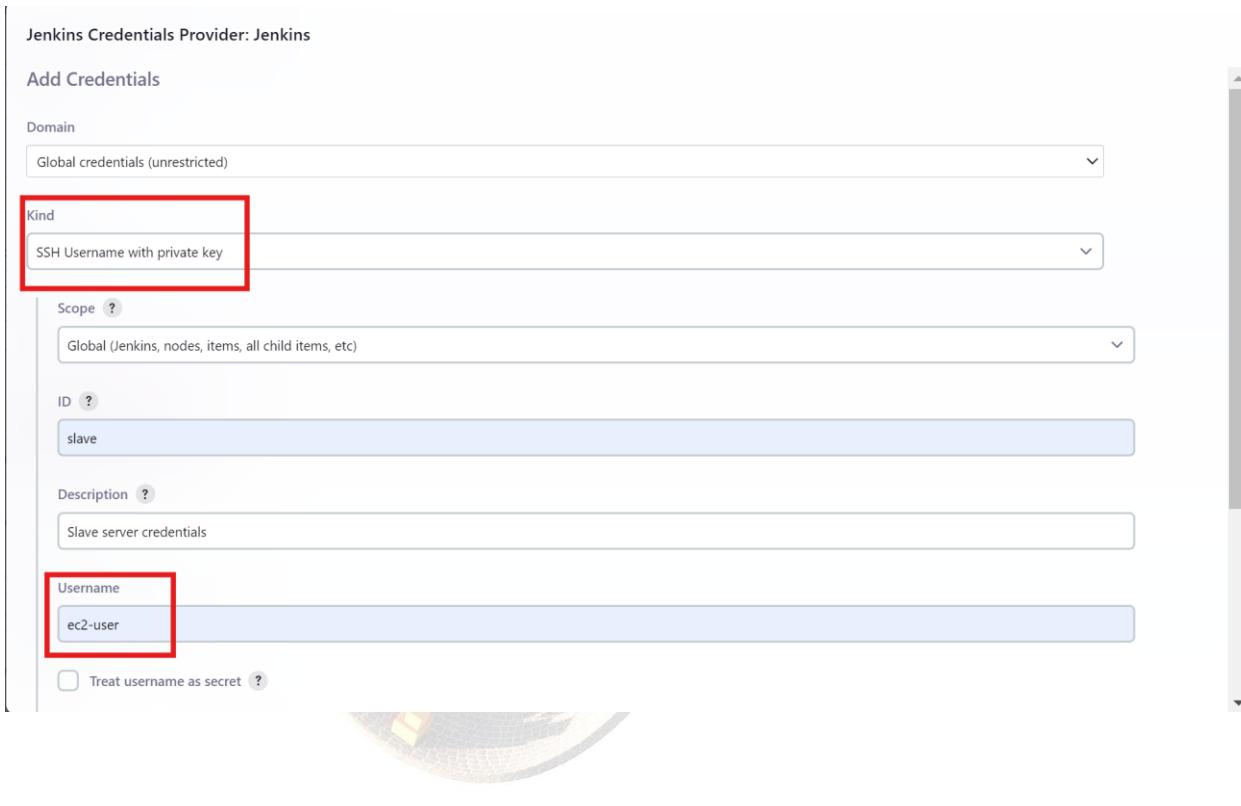
Description ?

Slave server credentials

Username

ec2-user

Treat username as secret ?



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

Private Key

Enter directly

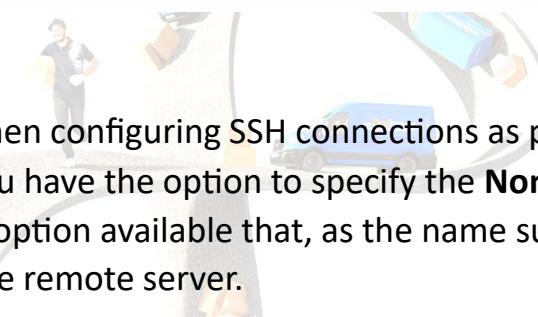
Key

```
-----END RSA PRIVATE KEY-----
```

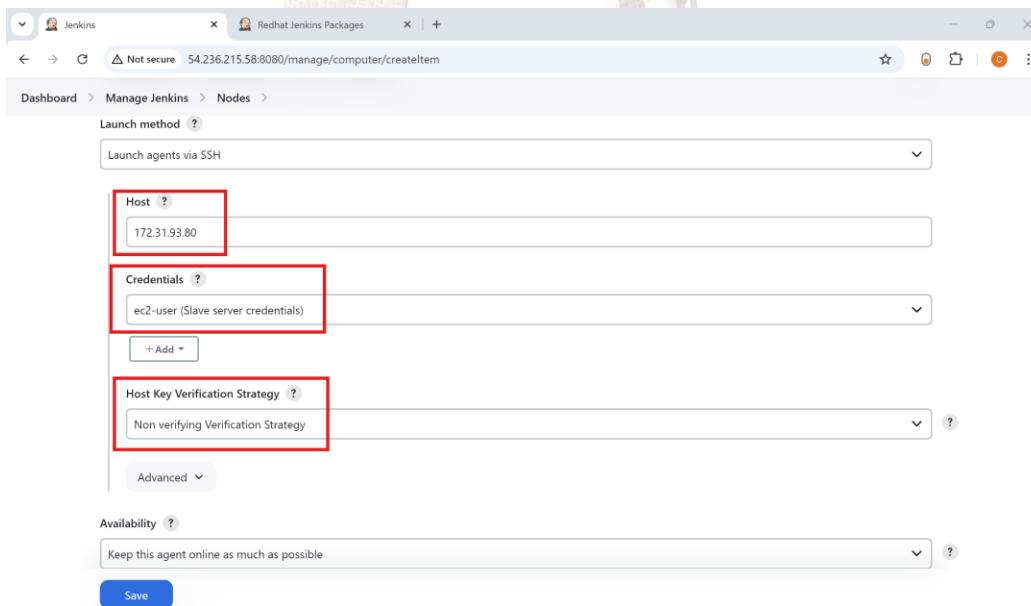
Enter New Secret Below

```
-----BEGIN RSA PRIVATE KEY-----
```

[A large block of RSA private key data is displayed here.]



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.



The screenshot shows the Jenkins configuration interface for creating a new node. The 'Host' field is set to '172.31.93.80'. The 'Credentials' dropdown is set to 'ec2-user (Slave server credentials)'. The 'Host Key Verification Strategy' dropdown is set to 'Non verifying Verification Strategy'. The 'Save' button is visible at the bottom.

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.

The screenshot shows the Jenkins interface for managing nodes. On the left, there's a sidebar with sections for 'Build Queue' (empty), 'Build Executor Status' (showing 'Built-In Node' with 1 idle and 2 idle executors), and a 'Clouds' section. The main area is titled 'Nodes' and lists two entries:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	5.55 GiB	0 B	5.55 GiB	0ms
	node-1	Linux (amd64)	In sync	5.98 GiB	0 B	5.98 GiB	45ms

A red box highlights the 'node-1' entry.

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.

The screenshot shows the Jenkins 'createItem' page for creating a new node. The form fields are as follows:

- Name:** node-2
- Description:** This is for slave-2
- Number of executors:** 3
- Remote root directory:** /home/ec2-user/Jenkins

Red boxes highlight the 'Description', 'Number of executors', and 'Remote root directory' fields.

Jenkins

Redhat Jenkins Packages

Not secure 54.236.215.58:8080/computer/createItem

Dashboard > Nodes >

Labels ?
dev

Usage ?
Only build jobs with label expressions matching this node

Launch method ?
Launch agents via SSH

Host ?
172.31.84.100

Credentials ?
ec2-user (Slave server credentials-1)

+ Add ▾

Host Key Verification Strategy ?
Non verifying Verification Strategy

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.

Nodes [Jenkins]

Redhat Jenkins Packages

Not secure 54.236.215.58:8080/computer/

Dashboard > Nodes >

Nodes

Build Queue
No builds in the queue.

Build Executor Status

- Built-In Node
 - 1 Idle
 - 2 Idle
- node-1
 - 1 Idle
 - 2 Idle
- node-2
 - 1 Idle
 - 2 Idle
 - 3 Idle

Nodes

+ New Node Configure Monitors

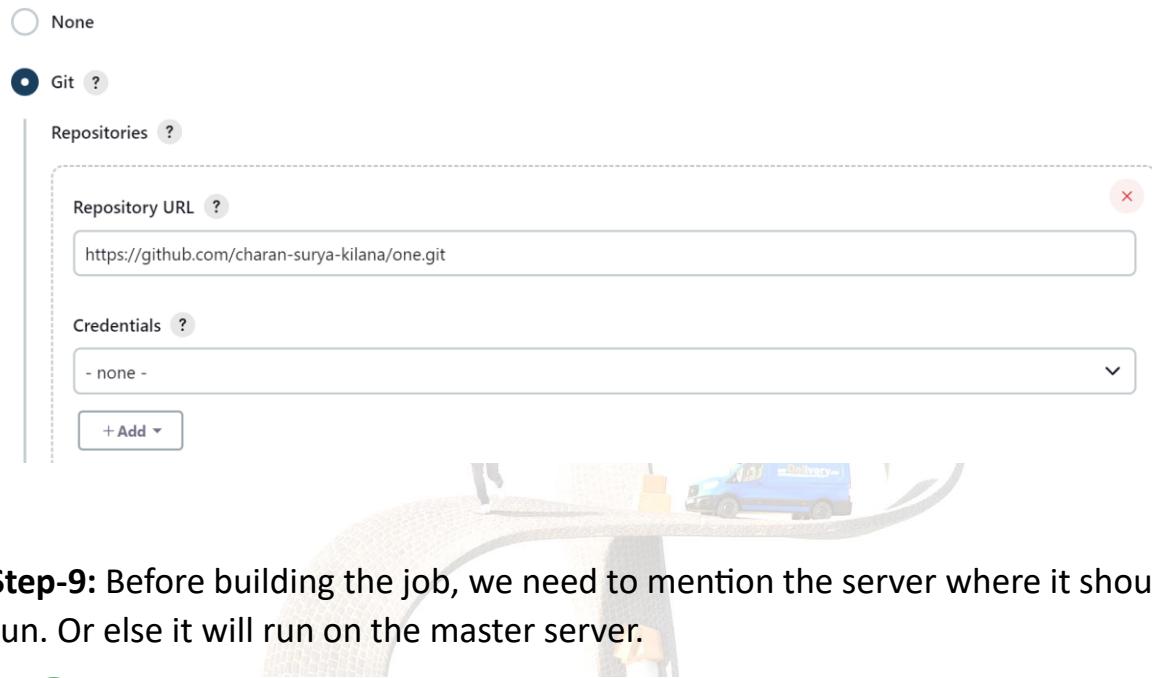
S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
Built-In Node	Linux (amd64)	In sync	5.55 GiB	! 0 B	5.55 GiB	0ms	⚙️
node-1	Linux (amd64)	In sync	5.98 GiB	! 0 B	5.98 GiB	45ms	⚙️
node-2		N/A	N/A	N/A	N/A	N/A	⚙️
	Data obtained	8 min 26 sec	8 min 26 sec	8 min 26 sec	8 min 26 sec	8 min 26 sec	

Icon: S M L

Legend

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.

Source Code Management



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.

✓ Console Output

```
Started by user Charan Surya Nikhil Kilana
Running as SYSTEM
Building on the built-in node in workspace /var/lib/jenkins/workspace/job
The recommended git tool is: NONE
No credentials specified
```

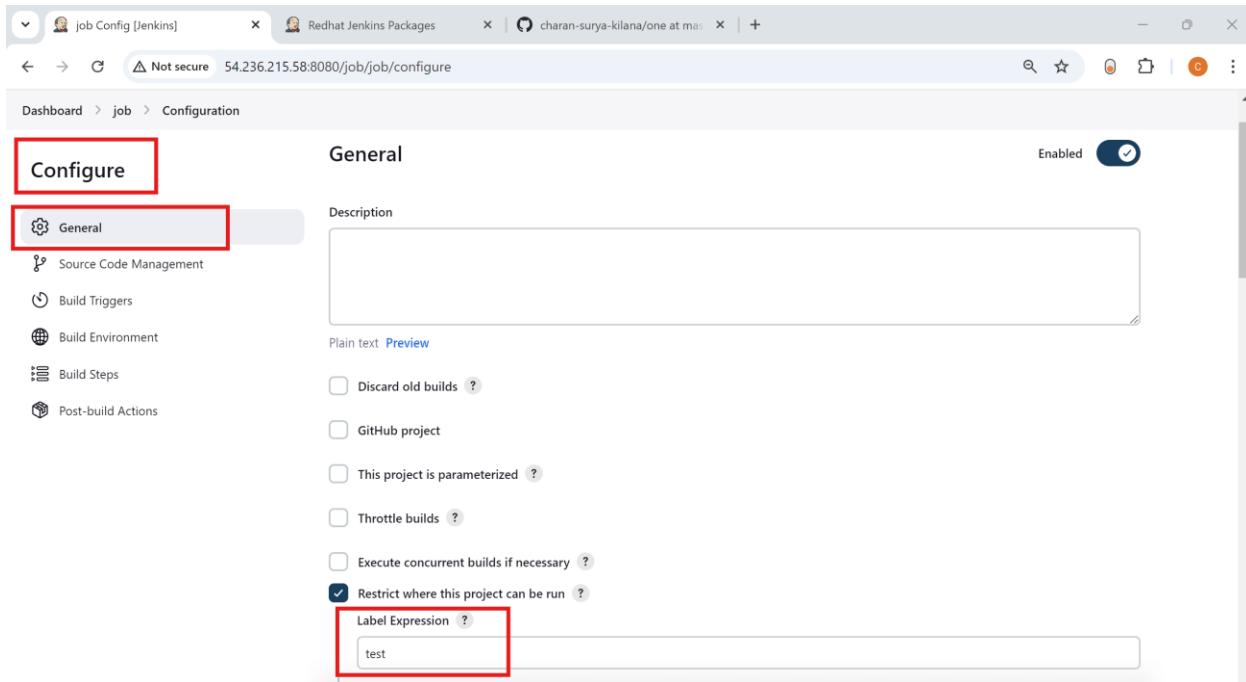
Step-10: Instructions for setting a label in Jenkins to restrict where a project can run:

Setting a Label:

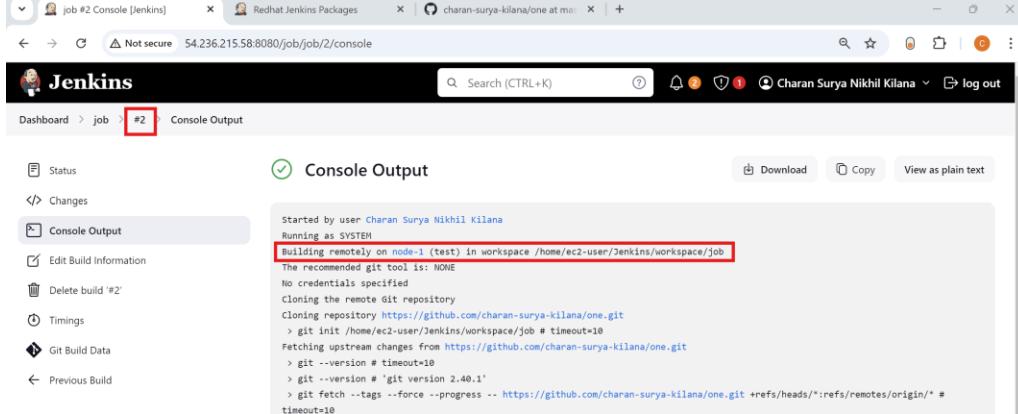
Go to Configure in your job settings.

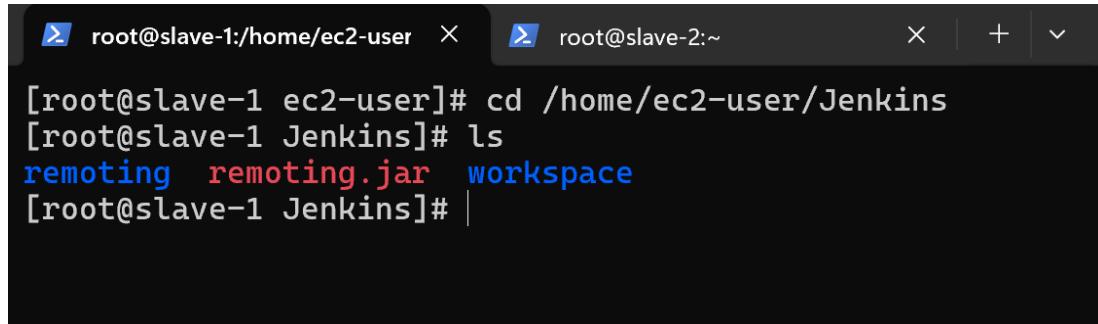
In the General section, check the box for "Restrict where this project can be run".

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.

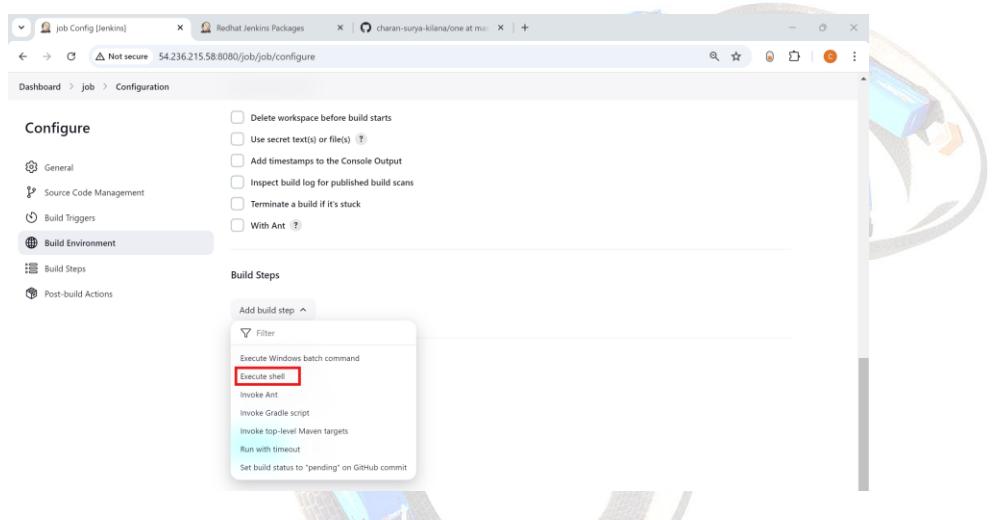




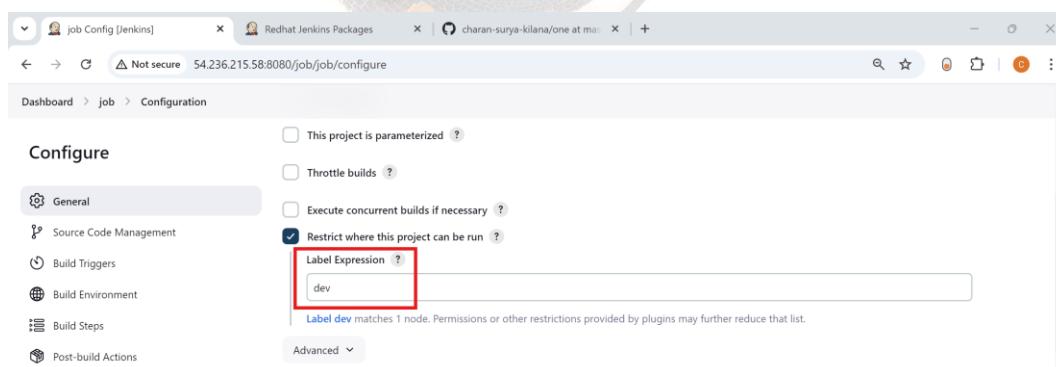
A terminal window showing two sessions. The left session is on slave-1 with command: [root@slave-1 ec2-user]# cd /home/ec2-user/Jenkins. The right session is on slave-2 with command: [root@slave-2:~]. The Jenkins directory contains remoting, remoting.jar, and workspace.

```
[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



☰ Execute shell ?

Command

See [the list of available environment variables](#)

touch charan.pdf

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 screenshot shows the Jenkins interface with a job named 'job #3' selected. The 'Console Output' tab is active, displaying the build logs. The logs show the command 'touch charan.pdf' being executed successfully.

```
Started by user Charan Surya Nikhil Kilana
Running as SYSTEM
Building remotely on node-2 (dev) in workspace /home/ec2-user/Jenkins/workspace/job
[job] $ /bin/sh -xe /tmp/jenkins1352407067066176747.sh
+ touch charan.pdf
Finished: SUCCESS
```

The file is created:

The screenshot shows a terminal window with two tabs. The current tab is 'root@slave-2:/home/ec2-user/' and it displays the following command-line session:

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

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.

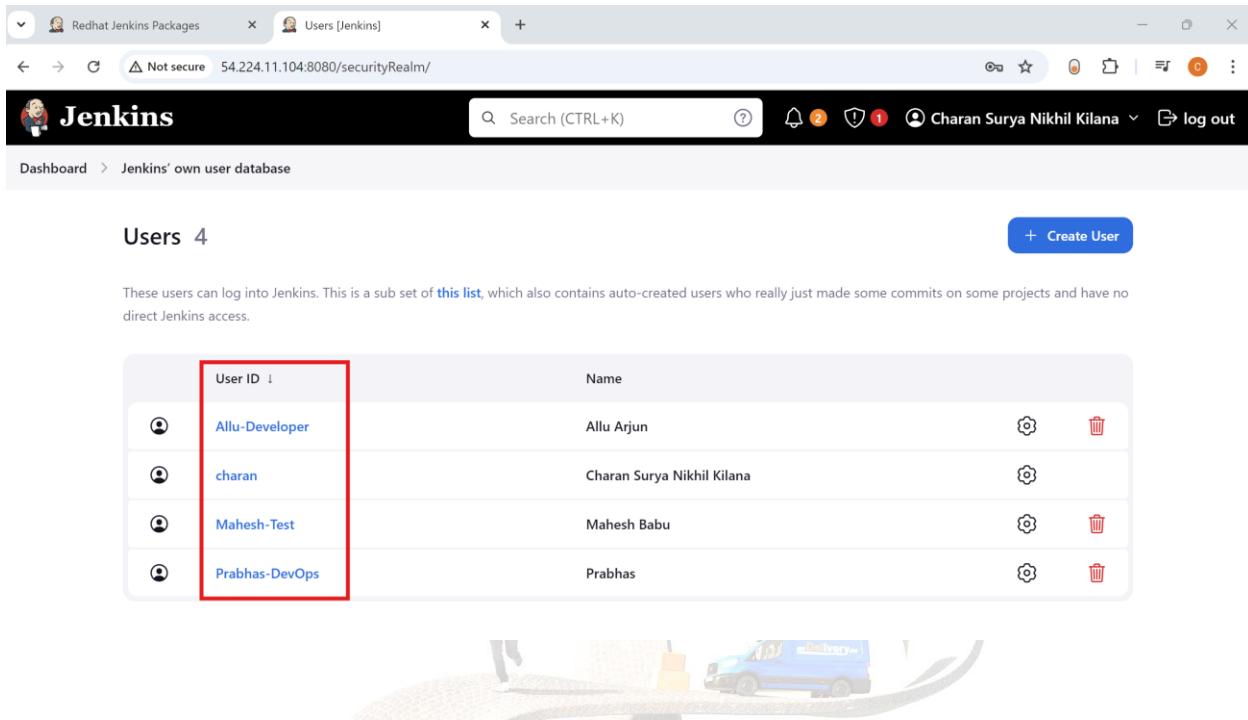
The screenshot shows the Jenkins Manage Jenkins interface. The 'Manage Jenkins' tab is selected. In the 'Security' section, the 'Users' item is highlighted with a red box. The 'Users' section contains a table with one row, where the 'User ID' column value 'charan' is also highlighted with a red box.

The screenshot shows the Jenkins Users page. The 'Users' table has one entry: 'charan' (User ID) with name 'Charan Surya Nikhil Kilana'. A blue 'Create User' button is visible at the top right of the table area.

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

The screenshot shows the Jenkins Create User page. The 'Username' field contains 'Mahesh-Test', which is highlighted with a red box. Other fields include 'Password' (redacted), 'Confirm password' (redacted), 'Full name' ('Mahesh Babu'), and 'E-mail address' ('tester123mb@gmail.com').

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



The screenshot shows the Jenkins 'Users' page with four users listed:

User ID	Name	Action
Allu-Developer	Allu Arjun	⚙️ 🗑️
charan	Charan Surya Nikhil Kilana	⚙️ 🗑️
Mahesh-Test	Mahesh Babu	⚙️ 🗑️
Prabhas-DevOps	Prabhas	⚙️ 🗑️

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.

The screenshot shows the Jenkins Plugins page. A search bar at the top right contains the text "role-based". Below it, a button labeled "Install" is visible. A list of plugins is shown, with one item highlighted by a red box: "Role-based Authorization Strategy 743.v142ea_b_d5fid3". The description for this plugin states: "Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions." To the right of the plugin details, the word "Released" and the date "26 days ago" are displayed.

The screenshot shows the Jenkins Manage Jenkins page. On the left, there's a sidebar with icons for "Plugins", "Clouds", "Appearance", and "Security". The "Security" icon is highlighted with a red box. The main content area has a heading "Security" and several sections: "Security" (which is also highlighted with a red box), "Credential Providers", "Credentials", and "Users".

The screenshot shows the Jenkins Security configuration page. At the top, a red box highlights the "Security" tab. Below it, under "Authentication", there's a checkbox for "Disable 'Keep me signed in'" which is unchecked. Under "Security Realm", a dropdown menu shows "Jenkins' own user database". Under "Authorization", a red box highlights the "Role-Based Strategy" option in a dropdown menu, which is currently selected. Other options in the dropdown include "Logged-in users can do anything", "Anyone can do anything", "Legacy mode", "Logged-in users can do anything", "Matrix-based security", and "Project-based Matrix Authorization 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.

The screenshot shows the Jenkins 'Manage Jenkins' page under the 'Security' section. It includes options for 'Security', 'Credentials', 'Credential Providers', and 'Users'. The 'Manage and Assign Roles' option, which is described as handling permissions by creating roles and assigning them to users/groups, is highlighted with a red box.

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:

The screenshot shows the Jenkins Manage Roles interface. The left sidebar has links for Assign Roles, Permission Templates, and Role Strategy Macros. The main area is titled "Global roles" and lists four roles: admin, Developer, Tester, and DevOps. Each role has checkboxes for various Jenkins features like Overall, Credentials, Agent, Job, Run, View, and several Jenkins-specific features like Workspace, Configure, Delete, Update, Provision, Discover, Create, Disconnect, Configure, Cancel, Build, Delete, Create, Connect, Configure, Build, View, Update, Create, Read, Delete, and ManageDomains.

Role	Overall	Credentials	Agent	Job	Run	View
Administrator	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Developer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Tester	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DevOps	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Role to add

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.

The screenshot shows the Jenkins Assign Roles interface. On the left sidebar, the 'Assign Roles' option is highlighted with a red box. The main area is titled 'Assign Roles' and contains a 'Global roles' section. This section includes a table with columns for 'User/Group' and several Jenkins roles: Anonymous, Authenticated Users, and Charan Surya Nikhil Kilana. The 'Charan Surya Nikhil Kilana' row has a checked checkbox in the 'Admin' column. Below the table are 'Add User' and 'Add Group' buttons.

Add User:

The screenshot shows the Jenkins Assign Roles interface with a modal dialog box overlaid. The dialog is titled 'User ID:' and contains a single input field with the value 'Mahesh-Test'. At the bottom of the dialog are 'Cancel' and 'OK' buttons. The background of the Jenkins interface is dimmed.

The screenshot shows the Jenkins 'Assign Roles' configuration page. On the left, there's a sidebar with links: 'Manage Roles', 'Assign Roles' (which is selected and highlighted in grey), 'Permission Templates', and 'Role Strategy Macros'. The main area is titled 'Assign Roles' and has a sub-section 'Global roles'. It displays a grid where users or groups are mapped to specific roles. The columns represent roles: 'Anonymous', 'Authenticated Users', 'Charan Surya Nikhil Kilana' (selected by a red box), 'Allu Arjun', 'Mahesh Babu', and 'Prabhas'. The rows represent User/Groups: 'Anonymous', 'Authenticated Users', 'Charan Surya Nikhil Kilana' (selected by a red box), 'Allu Arjun', 'Mahesh Babu', and 'Prabhas'. In the grid, under the 'Charan Surya Nikhil Kilana' row, the 'Tester' column has a checked checkbox, while other columns like 'Developer' and 'admin' have unchecked checkboxes. At the bottom of the grid, there are 'Add User' and 'Add Group' buttons.

Step 5: Verifying User Permissions

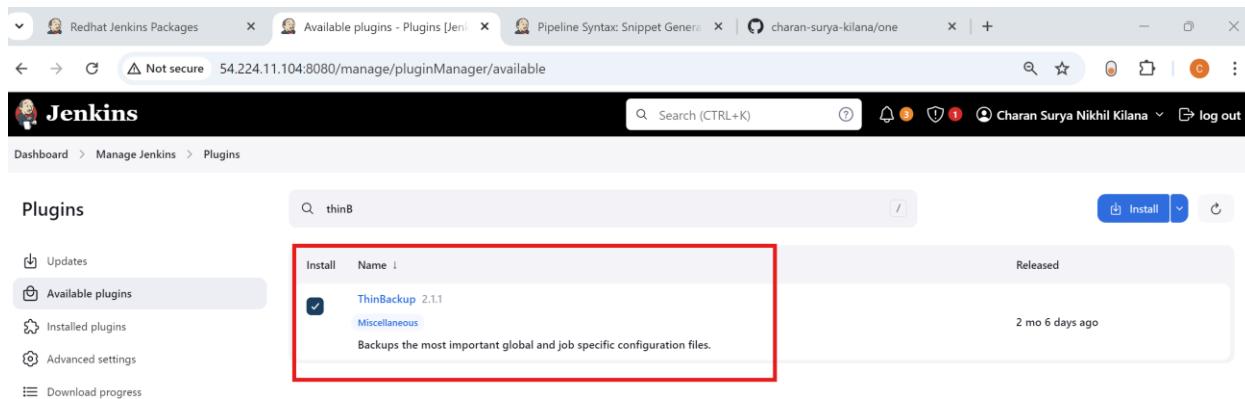
1. Log out of the current administrative account on the Jenkins Dashboard.
2. Log in using the credentials of a specific user role, such as a tester.
3. Verify that the permissions associated with their role are correctly enforced:
 - o For example, a tester should not have build permissions. Check to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



The screenshot shows the Jenkins Manage Jenkins dashboard. At the top, there are several status indicators and notices:

- Java 11 end of life in Jenkins**: You are running Jenkins on Java 11, support for which will end on or after Sep 30, 2024. [More Info](#) | [Ignore](#)
- Operating system end of life monitor**: You are running Jenkins on Amazon Linux 2. Jenkins stopped supporting Amazon Linux 2 as of 2023-11-16. Please upgrade to a supported operating system. [More Info](#) | [Ignore](#)
- A notice 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. [Dismiss](#)

In the main content area, there is a section titled **System Configuration** with a red border around it. It contains the following options:

- System**: Configure global settings and paths. (Selected)
- Nodes**: Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- Tools**: Configure tools, their locations and automatic installers.
- Clouds**: Add, remove, and configure cloud instances to provision agents on-demand.
- Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- Appearance**: Configure the look and feel of Jenkins.

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 171 Jun 19 08:00 jobs
drwxr-xr-x 3 jenkins jenkins 45 Jun 19 07:49 logs
-rw-r--r-- 1 jenkins jenkins 1037 Jun 19 07:40 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.plugin
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]# rm -rf jobs|
```

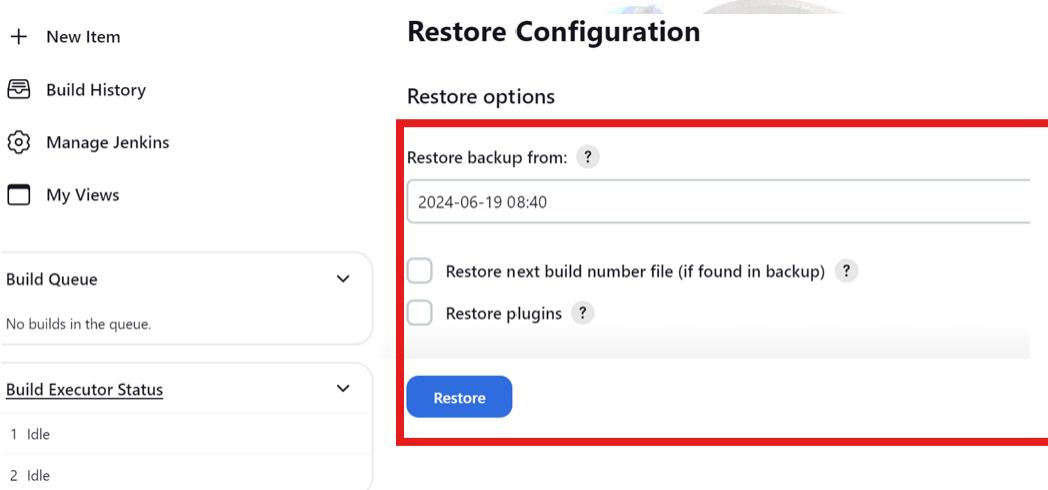
After deletion of jobs folder

```
-rw-r--r-- 1 jenkins jenkins 171 Jun 19 07:46 jenkins.telemetry.Correla
logs
drwxr-xr-x 3 jenkins jenkins 45 Jun 19 07:49 nodeMonitors.xml
-rw-r--r-- 1 jenkins jenkins 1037 Jun 19 07:40 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.



The screenshot shows the Jenkins ThinBackup interface. On the left, there's a sidebar with options: '+ New Item', 'Build History', 'Manage Jenkins', and 'My Views'. In the center, the title 'ThinBackup' is displayed above a message: 'Settings are now integrated in global configuration.' Below this are two buttons: 'Backup now' and 'Restore'. The 'Restore' button is highlighted with a red box.



The screenshot shows the 'Restore Configuration' page. It has a sidebar with 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). The main area is titled 'Restore options' and contains a section for selecting a backup to restore from, with a dropdown set to '2024-06-19 08:40'. There are also checkboxes for 'Restore next build number file (if found in backup)' and 'Restore plugins'. A large red box highlights this entire section. At the bottom is a blue 'Restore' button.

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]#
```

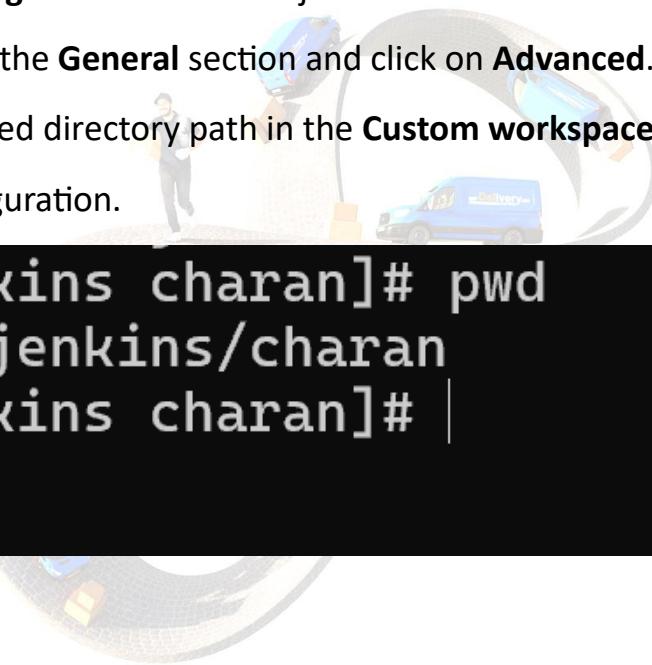
Note: Not all files are included in the default backup. To ensure every file is backed up, specify them under "Backup additional files" in the ThinBackup configuration settings.

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]# |
```



The screenshot shows the Jenkins job configuration interface. The left sidebar lists configuration sections: General, Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The General section is currently selected and expanded. In the General settings, there is a checkbox for "Execute concurrent builds if necessary". Below it, the "Advanced" button is highlighted, indicating the current view. Under the Advanced tab, several options are listed: Quiet period, Retry Count, Block build when upstream project is building, Block build when downstream project is building, and a checked checkbox for "Use custom workspace". The "Use custom workspace" option is expanded, showing a "Directory" input field containing the path "/var/lib/jenkins/charan". There is also a "Display Name" input field and a checkbox for "Keep the build logs of dependencies". At the bottom of the General section, there is a "Source Code Management" section with a "None" radio button selected. At the very bottom of the configuration page are "Save" and "Apply" buttons.

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]#
```

