

Build Automation – Continuous Integration(CI)

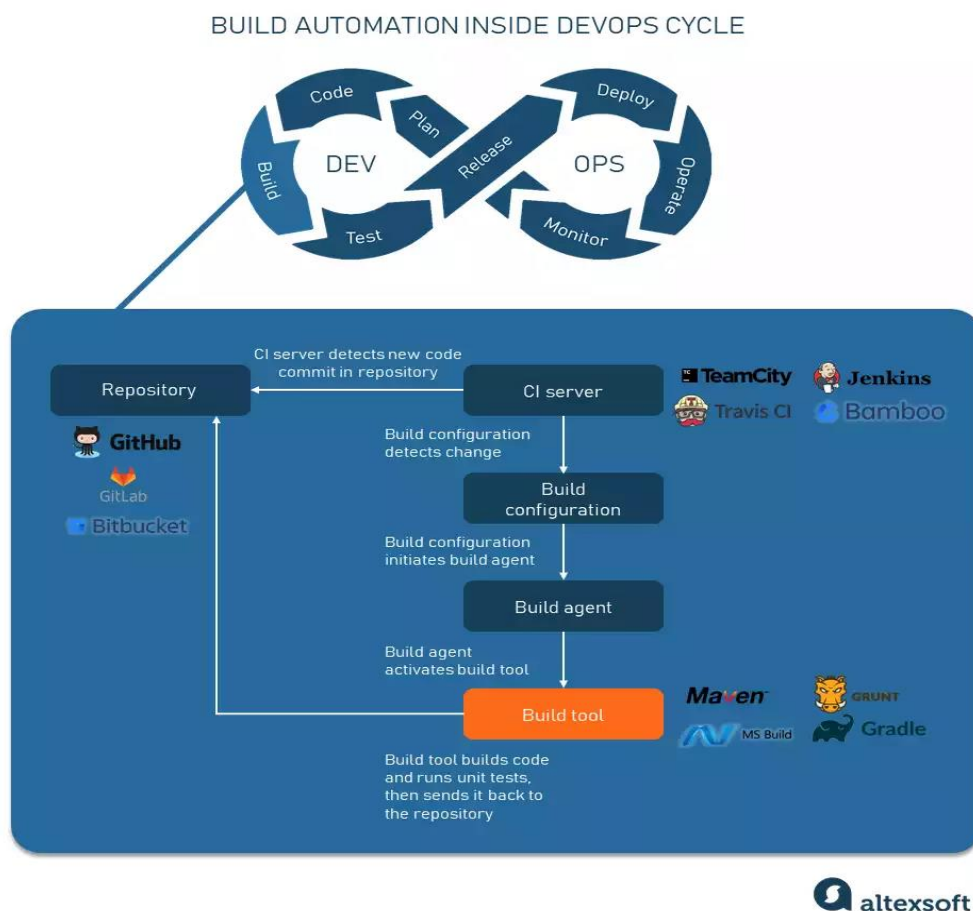
What is build automation?

The build is a process of preparing source code for production so that users can happily interact with it. It may include compilation, file compression, updating database schema, creating an installer or executable file, etc.

Build automation is an approach to handling builds within a CI/CD pipeline that has several steps. A developer commits source code to a repository, a CI server detects the change, runs the build process outside of the developer's IDE (on a dedicated cloud or in-house machine), checks it with unit tests, and either returns for fixes or sends it further down the pipeline.

Build automation process:

Build automation is a part of a larger CI/CD pipeline in DevOps.



1. A developer commits code to a repository. The *commit* term comes from the realm of version control systems (VCS), services that help manage updates to the software. These

may be GitHub, GitLab, Bitbucket, etc. A code commit operation means that an engineer sends new code to a repository residing inside one of those VCSs.

2. CI server detects changes inside the repository. A *CI server* is a system that orchestrates a CI pipeline, allowing teams to commit code multiple times per day, and build automatically outside of their workstations on different target servers. So, a CI server regularly polls a repository to check for new commits and save changes in a separate database.

CI server software allows engineers to tweak *build configuration*, add *build agents* that execute build jobs on dedicated machines, and integrate with *build tools*, which are specific for each programming language. What do these terms mean? Let's keep exploring the process.

3. Build configuration recognizes changes and triggers a build agent. A *build configuration* is a set of rules determining how the build is supposed to happen. For instance, it sets the timing when the build begins, e.g.:

- Once a new code commit arrives
- On an established schedule, say, the team knows that by 5 pm every day they must commit their code, or
- After some external process is finished, like a security scan.

Here you can also define exactly which operations and which tools you need to complete the build, adjust steps, or configure your VCS settings.

Build configuration sees changes in a CI server database and, based on settings, initiates different build agents.

4. Build agent activates build tools. A *build agent* is a program deployed on a build server (normally, outside of the CI server) that takes orders from the CI server and starts a build. It's important that a single commit can trigger several parallel agents on different target machines, say, for different operating systems. Depending on a specific CI server, concurrent build agents' work may be slightly different.

The agent doesn't compile code, compress files, generate installers, or do any kind of low-level dirty work. It just listens to what a server with its configuration has to say and orders *build tools* to do their job in the way a developer configured. There may be a sequence of operations on the code that amounts to a build. The agent would be triggering those operations in a set order.

5. Build tools compile, compress, run tests, etc. Finally, *build tools* do their low-level work. A build tool is a script, framework, or any kind of software created specifically for a given programming language to compile code, run tests on it, or do other operations needed for a build. You'd expect your CI server to support integrations with such tools.

For example, [Maven](#) is popular for [Java](#) compilation and unit testing, you'd use [MSBuild](#) with [C#](#) and C++, to run tasks on [JavaScript](#) code you have [Grunt](#), you get the idea.

At this stage, you'll likely do *unit tests*, the lowest level quality assurance checks that verify that this particular unit or component works as intended. Some build tools already have unit testing support. Sometimes, an agent can initiate several build tools in a sequence to compile code with one tool and test it with another.

6. Build artifacts are sent back to the server. If the code passes all build stages including testing, you'd normally send it back to the CI server, at which point it can go to production, further testing, or other storage, depending on how your CI/CD pipeline works.

What is Continuous integration?

Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Why is Continuous Integration Needed?

Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to *develop software*, but this process has many problems.

- Developer teams have to wait till the complete software is developed for the test results.
- There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
- It slows the software delivery process.
- Continuous feedback pertaining to things like architectural or coding issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
- The whole process was manual which increases the threat of frequent failure.

It is obvious from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction.

So to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.

Continuous integration refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test.

Continuous delivery and continuous deployment follow continuous integration in the DevOps cycle.

[Continuous delivery](#) (CD) picks up where continuous integration ends, automating the delivery of applications to selected infrastructure environments. CD focuses on delivering any validated changes to the code base—updates, bug fixes, even new features—to users as quickly and safely as possible. It ensures the automation of pushing code changes to different environments, such as development, testing and production.

In [Continuous deployment](#), the code changes to an application are released automatically into the production environment. This automation is driven by a series of predefined tests.

Once new updates pass those tests, the system pushes the updates directly to the software's users.

Continuous Integration Benefits

Improve Developer Productivity: Continuous integration helps your team be more productive by freeing developers from manual tasks and encouraging behaviours that help reduce the number of errors and bugs released to customers.

Find and Address Bugs Quicker: With more frequent testing, your team can discover and address bugs earlier before they grow into larger problems later.

Deliver Updates Faster: Continuous integration helps your team deliver updates to their customers faster and more frequently.

CI tools: Jenkins, Bamboo, TeamCity, CircleCI etc.

Jenkins is an open-source CI server for automating the build and testing processes. It is supported on Windows, Mac, and Unix machines and is based in Java. Jenkins is widely customizable through its hundreds of plugins and supports distributed workloads across multiple machines to improve performance and deliver results faster. Hundreds of plugins to integrate with other tools in your pipeline.

What is Jenkins?

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

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

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

For example: If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Advantages and Disadvantages of using Jenkins:

Advantages of Jenkins

- It is an open source tool.
- It is free of cost.
- It does not require additional installations or components. Means it is easy to install.
- Easily configurable.
- It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
- It is built in java and hence it is portable.
- It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- Easy support, since it open source and widely used.
- Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

Disadvantages of Jenkins

- Its interface is out dated and not user friendly compared to current user interface trends.
- Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.

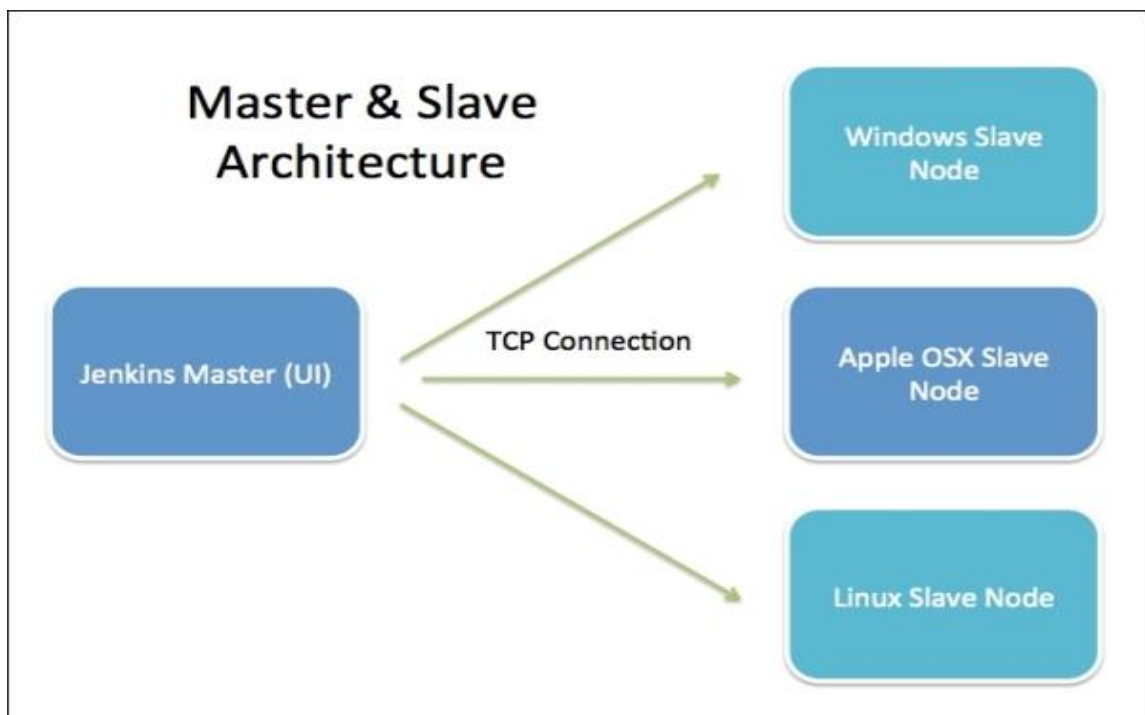
- CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

Jenkins Master-Slave Architecture:

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server



Jenkins Master

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

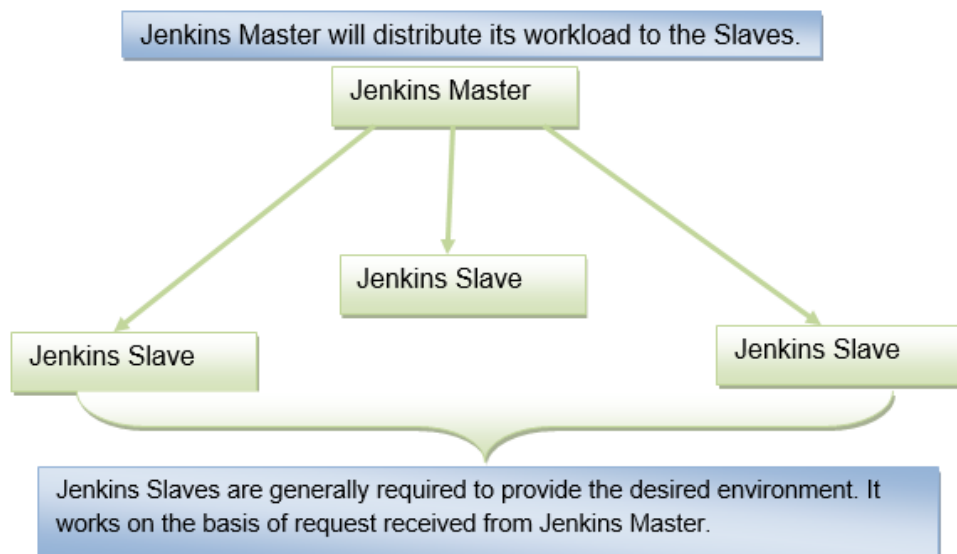
The server's job or master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the nodes/slaves for the actual execution.
- Monitor the nodes/slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master/Server instance of Jenkins can also execute build jobs directly.

Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simply let the Jenkins to pick the next available slave/node.

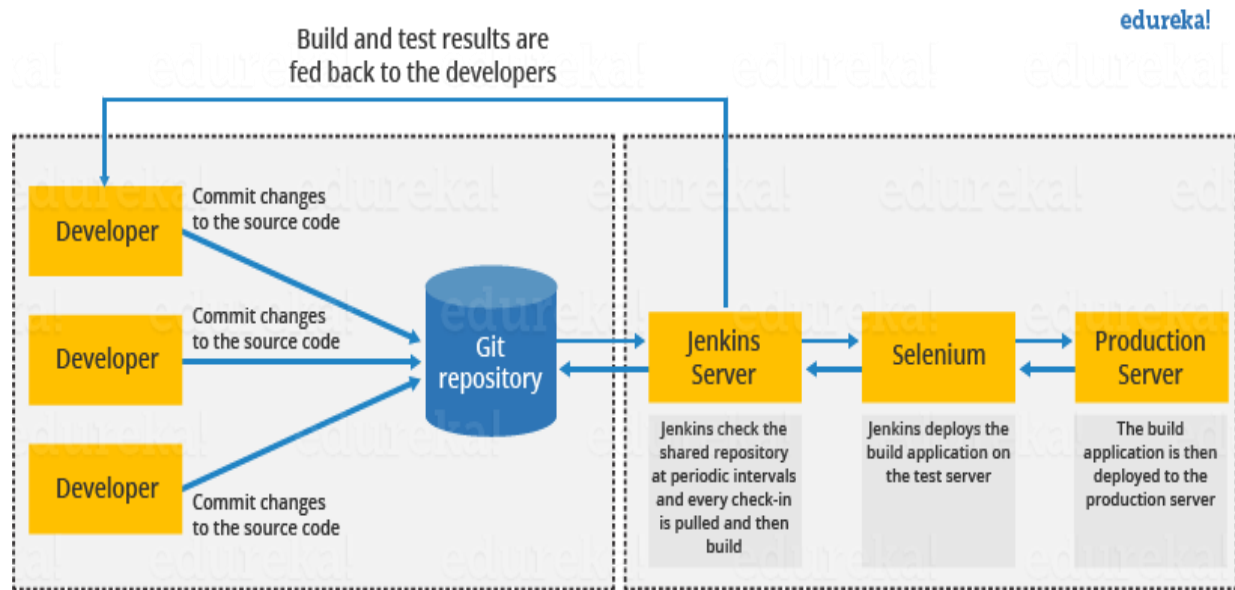
As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.



It consists of a Jenkins Master which is managing three Jenkins Slaves.

Jenkins Architecture

Let us have a look at the Jenkins Architecture below diagram depicts the same.



This single Jenkins server was not enough to meet certain requirements like:

- Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
- If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.

To address the above-stated needs, Jenkins distributed architecture came into the picture.

Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

Jenkins Master

Your main Jenkins server is the Master. The Master's job is to handle:

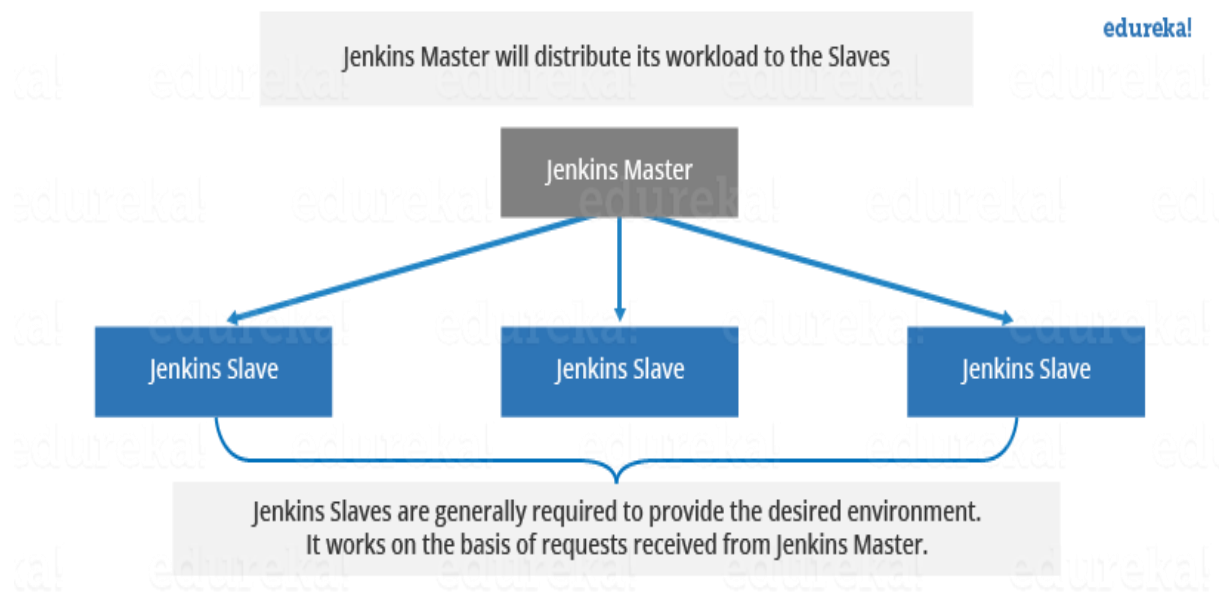
- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

Jenkins Slave

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- You can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

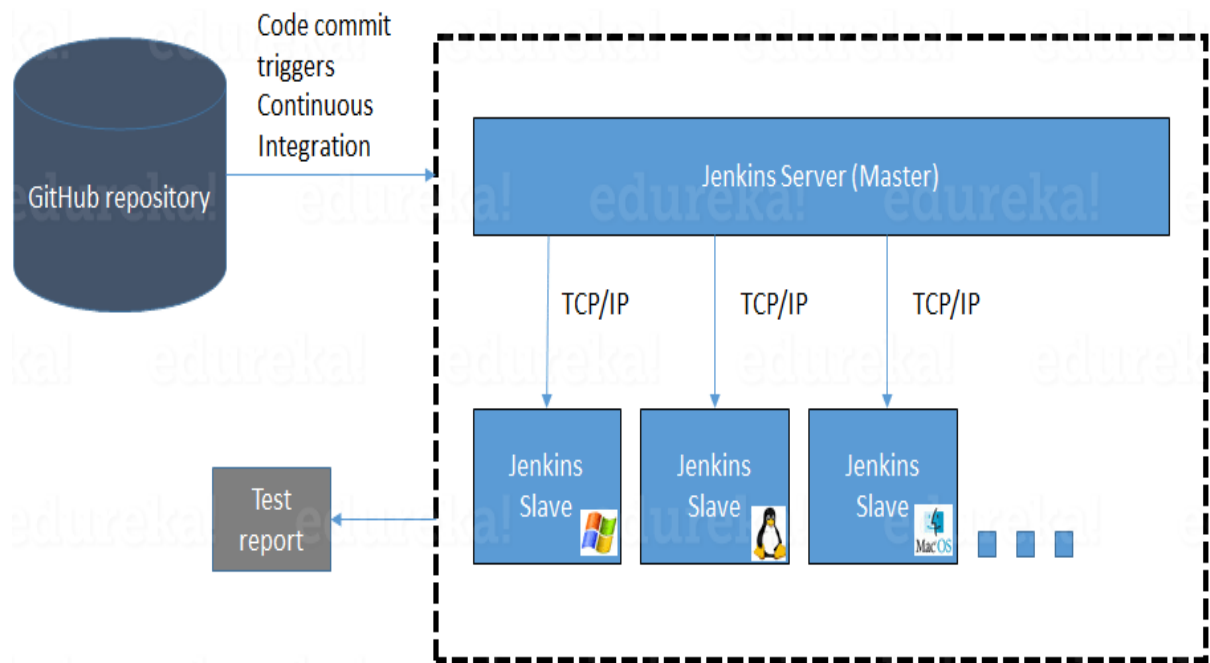
The diagram below is self-explanatory. It consists of a Jenkins Master which is managing three Jenkins Slave.



How Jenkins Master and Slave Architecture works?

Now let us look at an example in which we use Jenkins for testing in different environments like Ubuntu, MAC, Windows, etc.

The diagram below represents the same:



The above image represents the following functions:


- Jenkins checks the Git repository at periodic intervals for any changes made in the source code.
- Each builds requires a different testing environment which is not possible for a single Jenkins server. In order to perform testing in different environments, Jenkins uses various Slaves as shown in the diagram.
- Jenkins Master requests these Slaves to perform testing and to generate test reports.


How to setup Jenkins Master and Slaves?


1. Go to the Manage Jenkins section and scroll down to the section of Manage Nodes.




Next

**Jenkins CLI**
Access/manage Jenkins from your shell, or from your script.


**Script Console**
Executes arbitrary script for administration/trouble-shooting/diagnostics.

**Manage Nodes and Clouds**
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

**About Jenkins**
See the version and license information.



2. Click on New Node

← → ↻ localhost:9191/computer/ ☆ 🔍 🔔 👤 ⋮

**Jenkins** 1 🔍 search 🔔 Arvind Phulare | log out

Jenkins > Nodes > [ENABLE AUTO REFRESH](#)

[Back to Dashboard](#)
[Manage Jenkins](#)
[New Node](#)
[Configure Clouds](#)
[Node Monitoring](#)

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Windows 10 (amd64)	In sync	306.05 GB	7.89 GB	306.05 GB	0ms 
	Data obtained	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec

Refresh status

Build Queue =
No builds in the queue.

Build Executor Status =
1 Idle
2 Idle

3. Give a name for the node, choose the Permanent Agent option and click on Ok.

Jenkins > Nodes

[Back to Dashboard](#)
[Manage Jenkins](#)
New Node
[Configure Clouds](#)
[Node Monitoring](#)

Node name:

☒ 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.

OK

Build Queue: No builds in the queue.

Build Executor Status: 1 Idle, 2 Idle

4. Enter the details of the node slave machine. Here **no. of executors** in nothing but no. of jobs that this slave can run parallelly. Here we have kept it to 2. The **Labels** for which the name is entered as “Slave1” is what can be used to configure jobs to use this slave machine. Select **Usage** to Use this node as much as possible. For **launch method** we select the option of “Launch agent by connecting it to the master”. If this option is not visible then go to **Jenkins home page -> Manage Jenkins -> Configure Global Security**. Here in the Agents section **click on Random** and Save it. Now you will find the required option. Enter **Custom WorkDir path** as the workspace of your slave node. In **Availability** select “Keep this agent online as much as possible”. Click on **Save**.

Configure

[Build History](#)
[Load Statistics](#)
[Script Console](#)
[Log](#)
[System Information](#)
[Disconnect](#)

Build Executor Status: 1 Idle, 2 Idle

of executors:

Remote root directory:

Labels:

Usage:

Launch method:

☐ Disable WorkDir
☐ Fail if workspace is missing

Custom WorkDir path:

Internal data directory:

Advanced...

Availability:

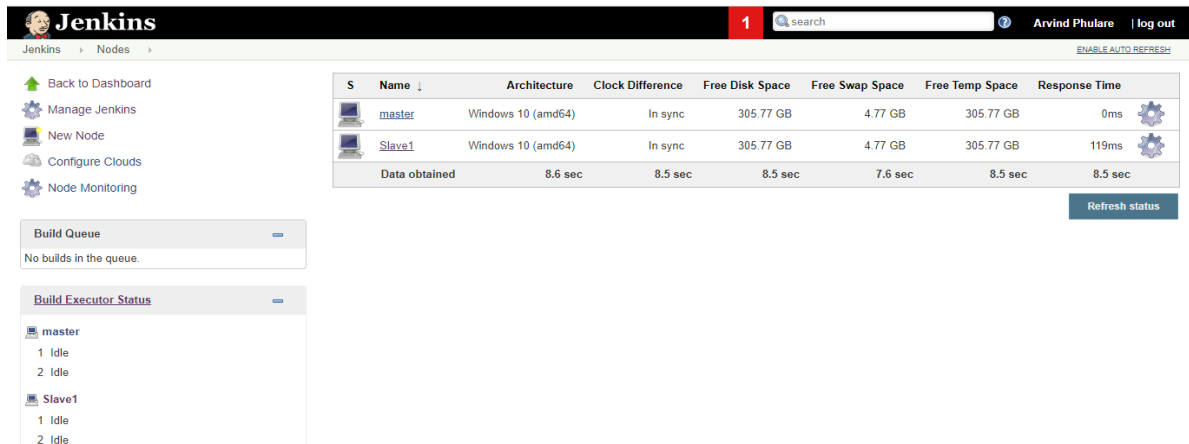
Node Properties

☐ Disable deferred wipeout on this node

Save

Once you complete the above steps, the new node machine will initially be in an offline state but will come online if all the settings in the previous screen were entered correctly.

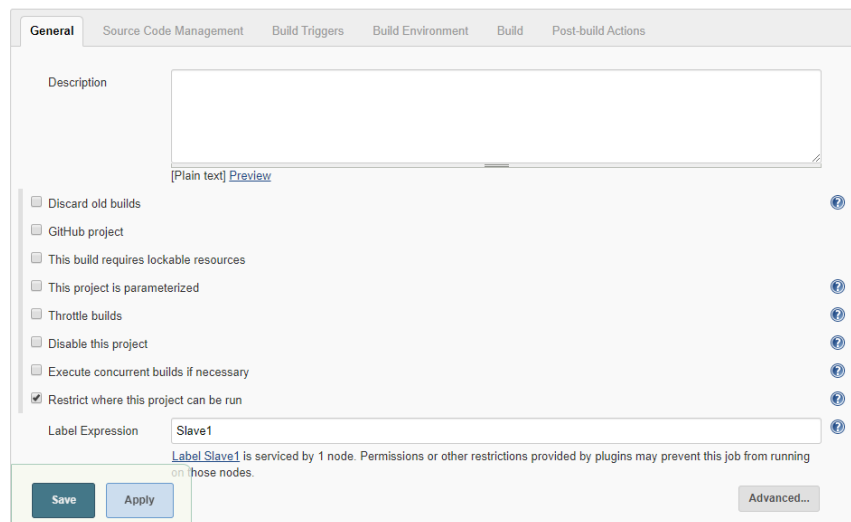
One can at any time make the node slave machine offline if required.



The screenshot shows the Jenkins 'Nodes' page. On the left, there is a sidebar with links: 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. Below these are two expandable sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status'. The 'Build Executor Status' section shows 'master' with 2 idle executors and 'Slave1' with 2 idle executors. The main area displays a table of nodes with columns: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The table lists 'master' and 'Slave1', both on Windows 10 (amd64) and 'In sync'. Below the table is a 'Data obtained' row showing times for various metrics. A 'Refresh status' button is at the bottom right.

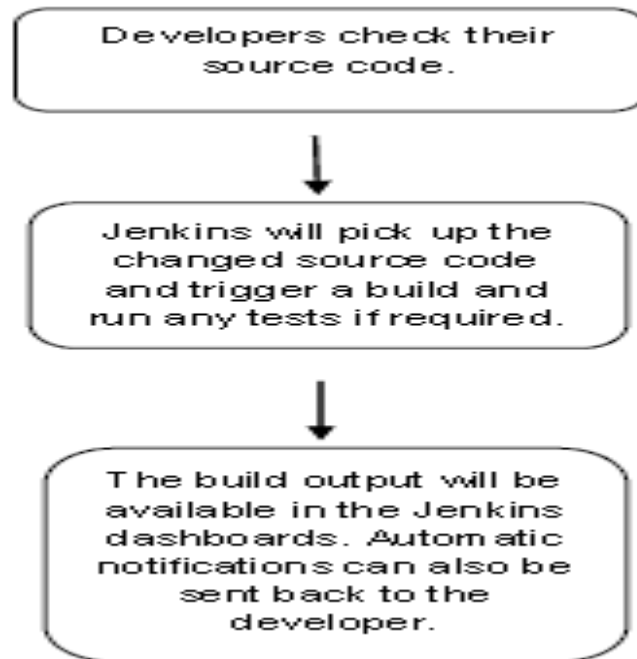
S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Windows 10 (amd64)	In sync	305.77 GB	4.77 GB	305.77 GB	0ms
	Slave1	Windows 10 (amd64)	In sync	305.77 GB	4.77 GB	305.77 GB	119ms
Data obtained		8.6 sec	8.5 sec	8.5 sec	7.6 sec	8.5 sec	8.5 sec

5. Now since your slave is up and running, let's execute a job on slave. For that I already have an existing job and I will run this job on this slave. Open this job and click on configure. Now here in the General section, click on “Restrict where this project can be run”. Here in Label Expression, enter the name of the slave and save it. Now click on Build now and see the output of this job. Everything is correct you will see the output as Success.



The screenshot shows the 'General' tab of a Jenkins job configuration. The 'Description' field is empty. Below it, there are several checkboxes: 'Discard old builds', 'GitHub project', 'This build requires lockable resources', 'This project is parameterized', 'Throttle builds', 'Disable this project', 'Execute concurrent builds if necessary', and 'Restrict where this project can be run'. The 'Restrict where this project can be run' checkbox is checked. Below this, the 'Label Expression' field contains 'Slave1'. A message below the field states: 'Label Slave1 is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.' At the bottom, there are 'Save' and 'Apply' buttons, and an 'Advanced...' link.

Jenkins workflow:



- First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
- Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If built is successful, then Jenkins server deploys the built in the test server.
- After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
- It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

Jenkins Pipeline

In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

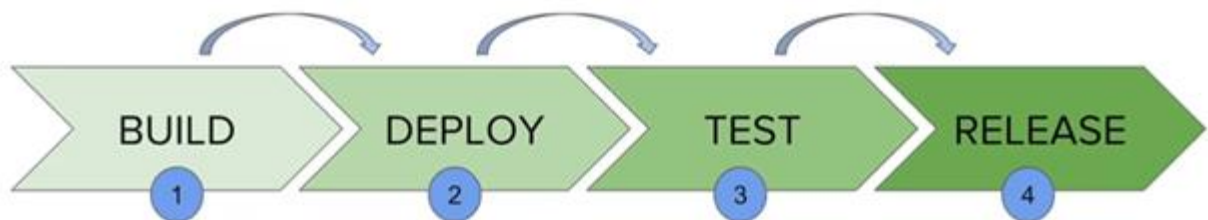
It is a combination of plugins that support the integration and implementation of **continuous delivery pipelines** using Jenkins.

In other words, a Jenkins Pipeline is a collection of jobs or events that brings the software from version control into the hands of the end users by using automation tools. It is used to incorporate continuous delivery in our software development workflow.

A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).

What is Continuous Delivery Pipeline?

In a Jenkins Pipeline, every job has some sort of dependency on at least one or more jobs or events.



The above diagram represents a continuous delivery pipeline in Jenkins. It contains a collection of states such as build, deploy, test and release. These jobs or events are interlinked with each other. Every state has its jobs, which work in a sequence called a continuous delivery pipeline.

A continuous delivery pipeline is an automated expression to show your process for getting software for version control. Thus, every change made in your software goes through a number of complex processes on its manner to being released. It also involves developing the software in a repeatable and reliable manner, and progression of the built software through multiple stages of testing and deployment.

JenkinsFile

Jenkins Pipeline can be defined by a text file called JenkinsFile. You can implement pipeline as code using JenkinsFile, and this can be defined by using a DSL (Domain Specific Language). With the help of JenkinsFile, you can write the steps required for running a Jenkins Pipeline.

The benefits of using JenkinsFile are:

- You can make pipelines automatically for all branches and can execute pull requests with just one JenkinsFile.
- You can review your code on the pipeline.
- You can review your Jenkins pipeline.
- This is the singular source for your pipeline and can be customized by multiple users.

JenkinsFile can be defined by using either Web UI or with a JenkinsFile.

Pipeline syntax

Two types of syntax are used for defining your JenkinsFile.

- Declarative
- Scripted

Declarative:

Declarative pipeline syntax offers a simple way to create pipelines. It consists of a predefined hierarchy to create Jenkins pipelines. It provides you the ability to control all aspects of a pipeline execution in a simple, straightforward manner.

Scripted:

Scripted Jenkins pipeline syntax runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to convert the pipeline into atomic commands.

Both scripted and declarative syntax are different from each other and are defined totally differently.

Why Use Jenkins Pipeline?

Jenkins is a continuous integration server which has the ability to support the automation of software development processes. You can create several automation jobs with the help of use cases, and run them as a Jenkins pipeline

Here are the reasons why you should use Jenkins pipeline:

- Jenkins pipeline is implemented as a code which allows several users to edit and execute the pipeline process.
- Pipelines are robust. So if your server undergoes an unpredicted restart, the pipeline will be automatically resumed.
- You can pause the pipeline process and make it wait to continue until there is an input from the user.
- Jenkins Pipelines support big projects. You can run many jobs, and even use pipelines in a loop.

Jenkins Pipeline Concepts

Pipeline: This is the user-defined block, which contains all the processes such as build, test, deploy, etc. it is a group of all the stages in a JenkinsFile. All the stages and steps are defined in this block. It is used in declarative pipeline syntax.

Node: The node is a machine on which Jenkins runs is called a node. A node block is used in scripted pipeline syntax.

Stage: This block contains a series of steps in a pipeline. i.e., build, test, and deploy processes all come together in a stage. Generally, a stage block visualizes the Jenkins pipeline process.

Step: A step is a single task that executes a specific process at a defined time. A pipeline involves a series of steps defined within a stage block.

What is a Jenkins Freestyle Project?

Jenkins freestyle projects allow users to automate simple jobs, such as running tests, creating and packaging applications, producing reports, or executing commands. Freestyle projects are repeatable and contain both build steps and post-build actions.

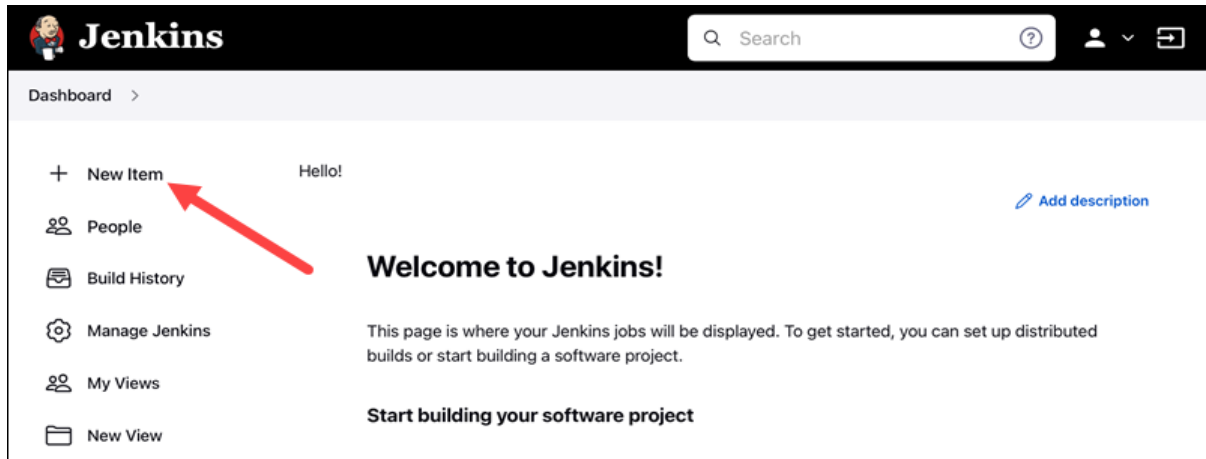
Even though freestyle jobs are highly flexible, they support a limited number of general build and post-build actions. Any specialized or non-typical action a user wants to add to a freestyle project requires additional plugins.

How to Setup a Build Job in Jenkins?

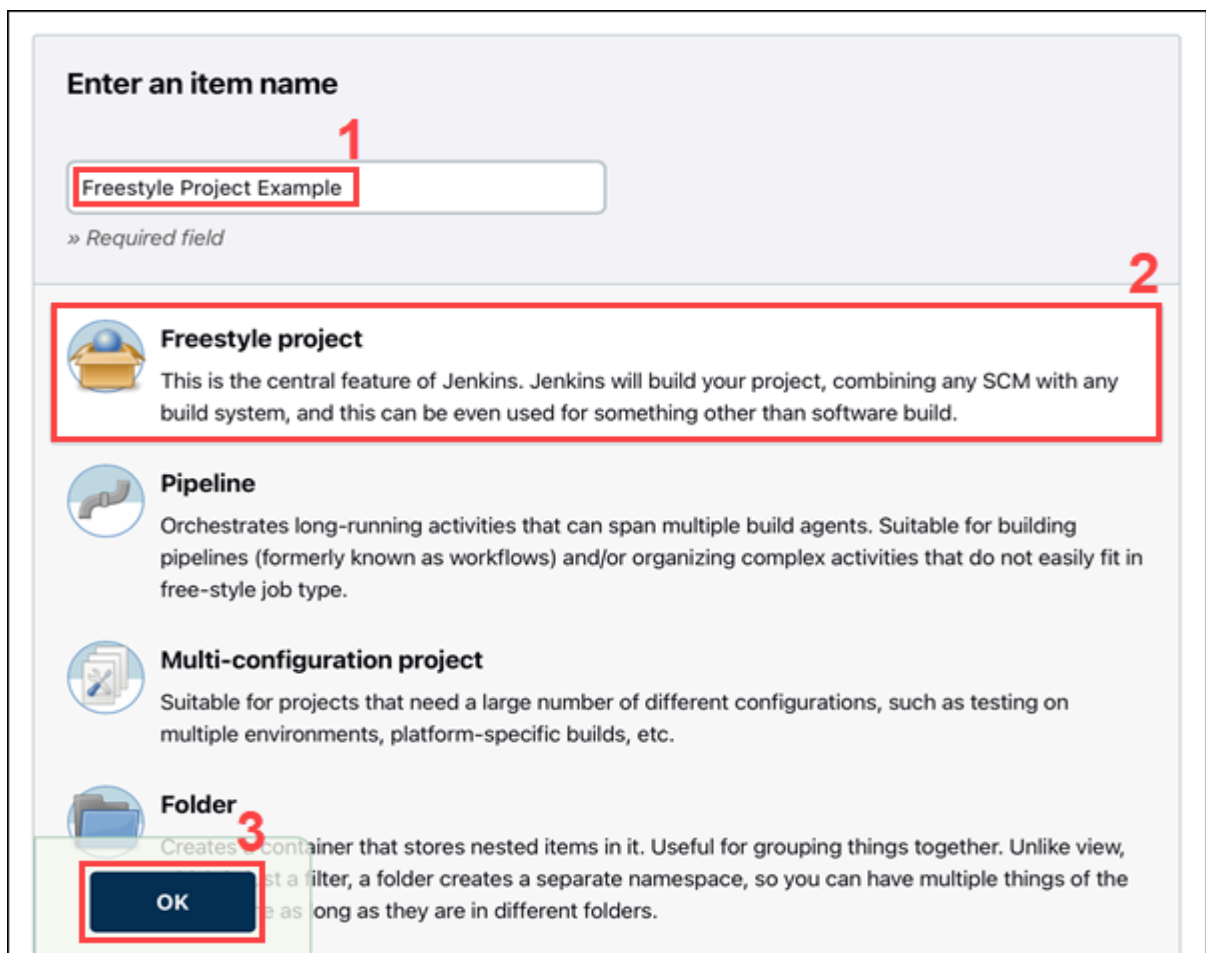
Follow the steps outlined below to set up and run a new Jenkins freestyle project.

Step 1: Create a New Freestyle Project

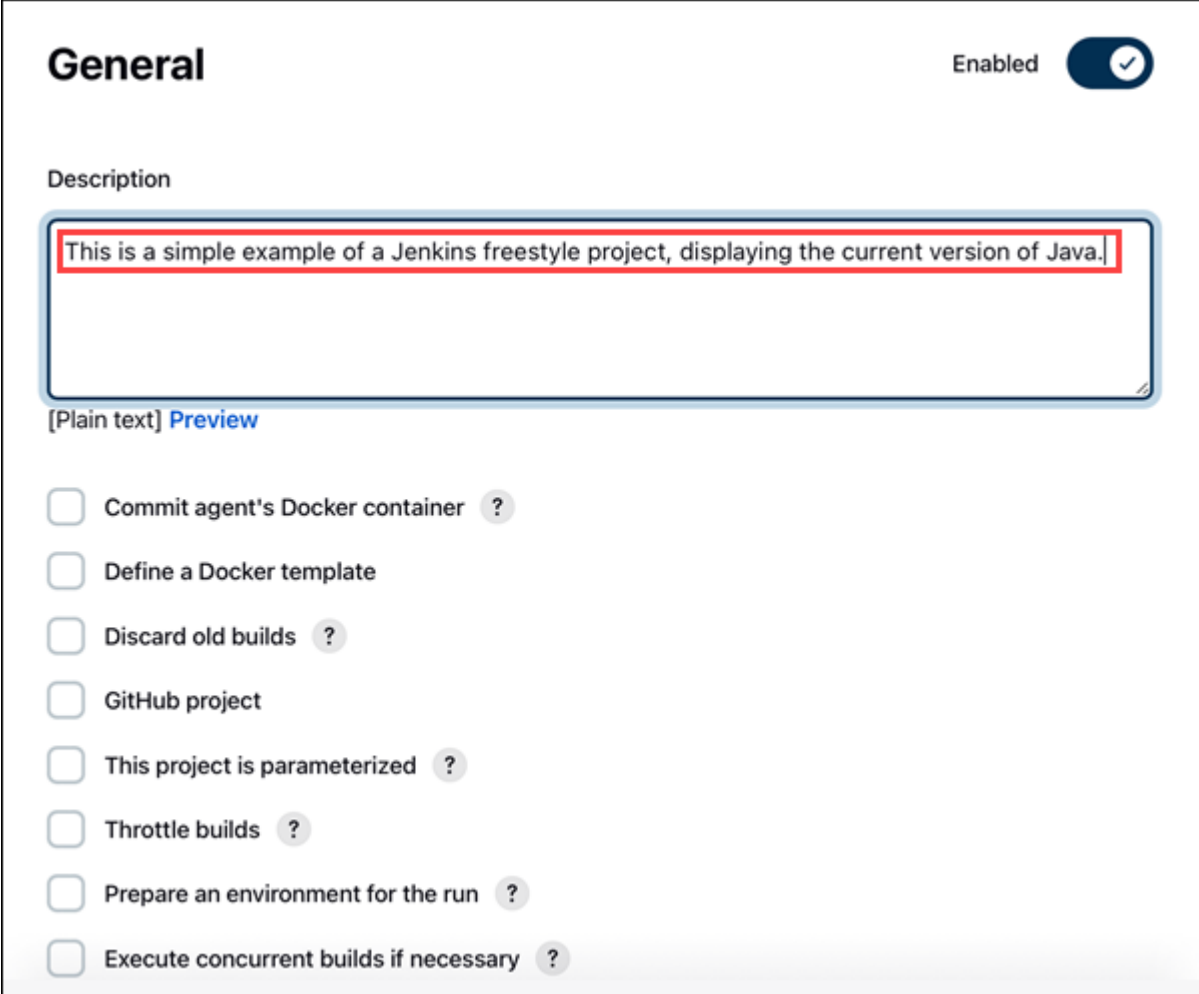
1. Click the **New Item** link on the left-hand side of the Jenkins dashboard.



2. Enter the new project's name in the **Enter an item name** field and select the **Freestyle project** type. Click **OK** to continue.



3. Under the *General* tab, add a project description in the **Description** field.



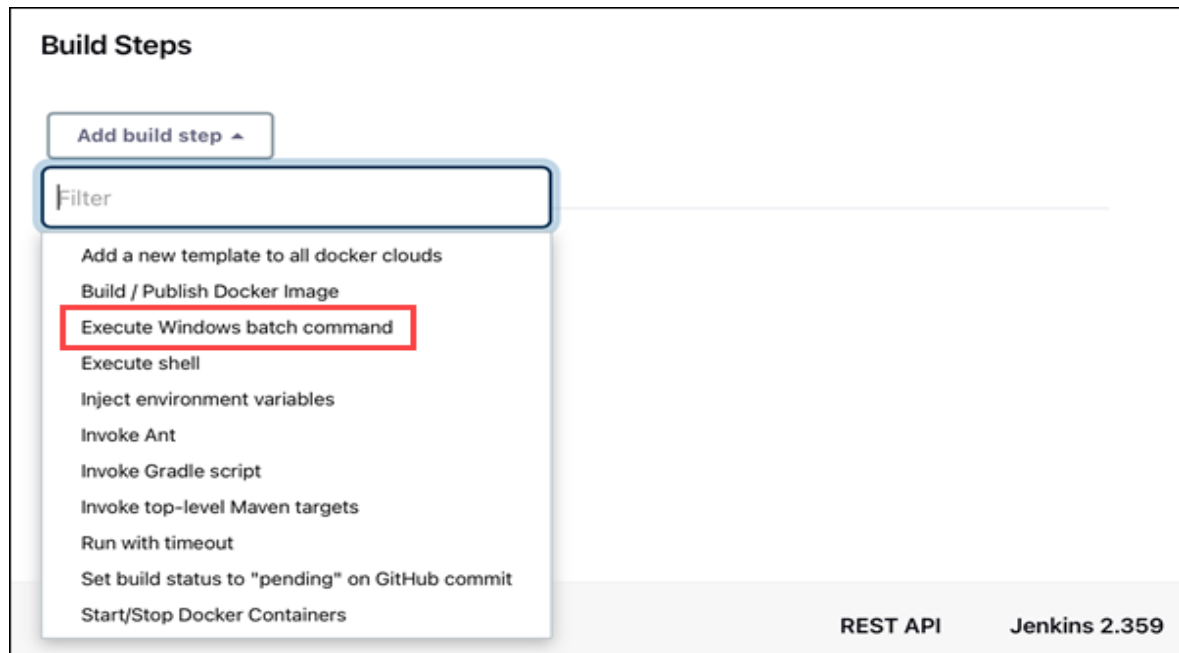
The screenshot shows the 'General' tab of a Jenkins project configuration. At the top right, there is a toggle switch labeled 'Enabled' which is turned on. Below this, the 'Description' section contains a text area with the text 'This is a simple example of a Jenkins freestyle project, displaying the current version of Java.' The text area is highlighted with a red border. Below the text area, there is a link that says '[Plain text] Preview'. At the bottom of the configuration page, there is a list of checkboxes with labels and help icons (question marks):

- ☐ Commit agent's Docker container ?
- ☐ Define a Docker template
- ☐ Discard old builds ?
- ☐ GitHub project
- ☐ This project is parameterized ?
- ☐ Throttle builds ?
- ☐ Prepare an environment for the run ?
- ☐ Execute concurrent builds if necessary ?

Step 2: Add a Build Step

1. Scroll down to the *Build* section.

2. Open the **Add build step** drop-down menu and select **Execute Windows batch command**.



3. Enter the commands you want to execute in the **Command** field. For this tutorial, we are using a simple set of commands that display the current version of Java and Jenkins working directory:

```
java -version  
dir
```

4. Click the **Save** button to save changes to the project.

Build Steps

≡ Execute Windows batch command ?

✕

Command

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

```
java -version
dir
```

Advanced...

Add build step ▾

Save

Apply

Step 3: Build the Project

1. Click the **Build Now** link on the left-hand side of the new project page.

↑ Back to Dashboard

Status

</> Changes

Workspace

▶ Build Now

⚙️ Configure

🗑️ Delete Project

✎ Rename

Project Freestyle Project Example

This is a simple example of a Jenkins freestyle project, displaying the current version of Java.

[Edit description](#)

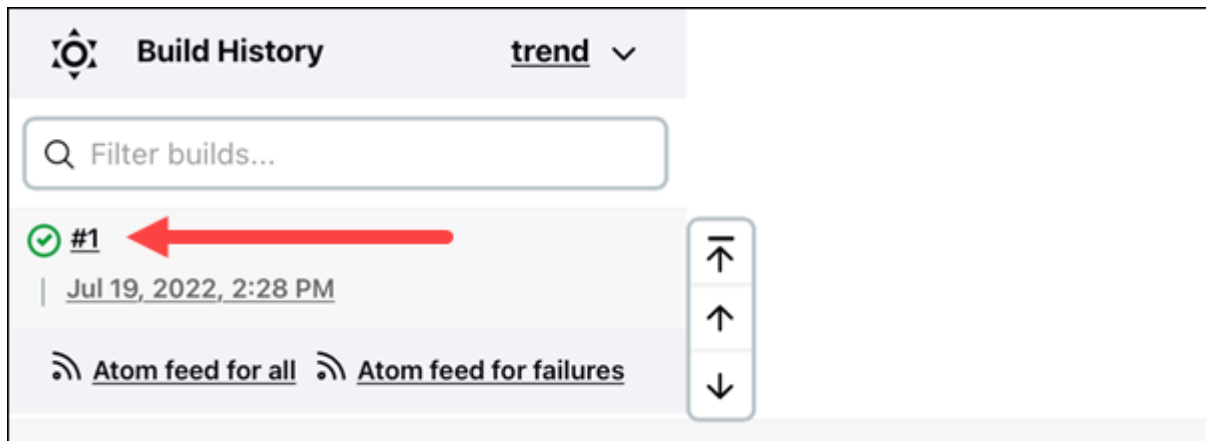
Disable Project

Workspace

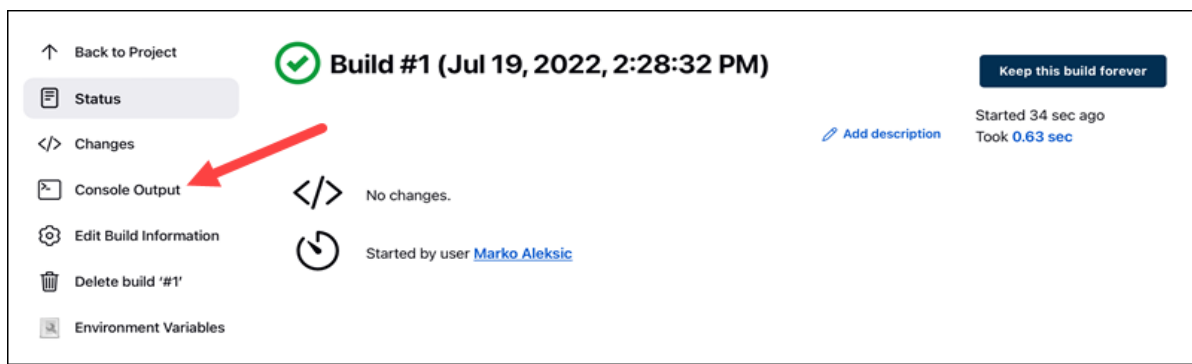
Recent Changes

Permalinks

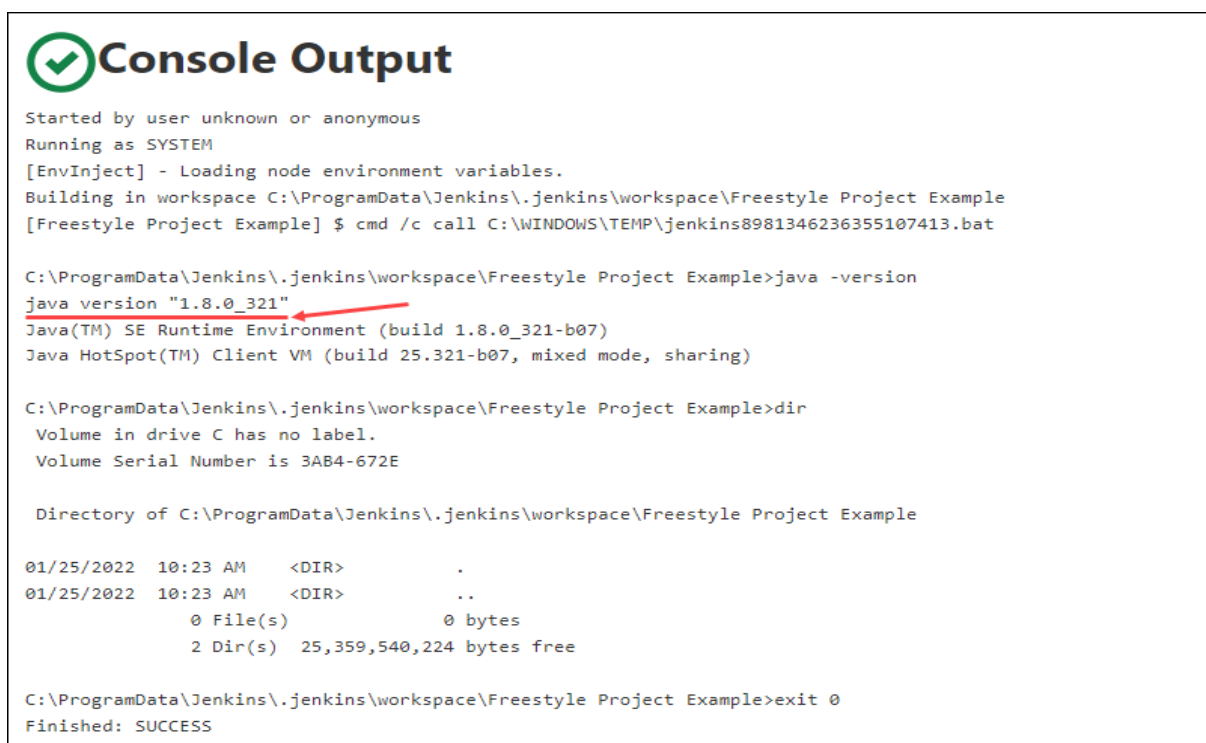
2. Click the link to the latest project build in the *Build History* section.



3. Click the **Console Output** link on the left-hand side to display the output for the commands you entered.



4. The console output indicates that Jenkins is successfully executing the commands, displaying the current version of Java and Jenkins working directory.



.What is Jenkins Pipeline?

Jenkins Pipeline is a combination of plugins that supports integration and implementation of continuous delivery pipelines. It has an extensible automation server to create simple and complex delivery pipelines as code via pipeline DSL. A Pipeline is a group of events interlinked with each other in a sequence.

What is a JenkinsFile?

Jenkins pipelines can be defined using a text file called **JenkinsFile**. You can implement pipeline as code using JenkinsFile, and this can be defined by using a domain specific language (DSL). With JenkinsFile, you can write the steps needed for running a Jenkins pipeline.

The benefits of using **JenkinsFile** are:

- You can create pipelines automatically for all branches and execute pull requests with just one **JenkinsFile**.
- You can review your Jenkins code on the pipeline
- You can audit your Jenkins pipeline
- This is the singular source for your pipeline and can be modified by multiple users.

JenkinsFile can be defined by either Web UI or with a Jenkins File.

Declarative versus Scripted pipeline syntax:

There are two types of Jenkins pipeline syntax used for defining your JenkinsFile.

1. Declarative
2. Scripted

Declarative:

Declarative pipeline syntax offers an easy way to create pipelines. It contains a predefined hierarchy to create Jenkins pipelines. It gives you the ability to control all aspects of a pipeline execution in a simple, straight-forward manner.

Scripted:

Scripted Jenkins pipeline runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to translate the pipeline into atomic commands. Both declarative and scripted syntax are different from each other and are defined totally differently.

Why Use Jenkin's Pipeline?

Jenkins is an open continuous integration server which has the ability to support the automation of software development processes. You can create multiple automation jobs with the help of use cases, and run them as a Jenkins pipeline.

Here are the reasons why you use should use Jenkins pipeline:

- Jenkins pipeline is implemented as a code which allows multiple users to edit and execute the pipeline process.
- Pipelines are robust. So if your server undergoes an unforeseen restart, the pipeline will be automatically resumed.
- You can pause the pipeline process and make it wait to resume until there is an input from the user.
- Jenkins Pipelines support big projects. You can run multiple jobs, and even use pipelines in a loop.

Jenkins Pipeline Concepts

Term	Description
------	-------------

	The pipeline is a set of instructions given in the form of code for continuous delivery Pipeline and consists of instructions needed for the entire build process. With pipeline, you can build, test, and deliver the application.
--	---

Node	The machine on which Jenkins runs is called a node. A node block is mainly used in scripted pipeline syntax.
------	--

Stage	A stage block contains a series of steps in a pipeline. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process.
-------	--

Step	A step is nothing but a single task that executes a specific process at a defined time. A pipeline involves a series of steps.
------	--

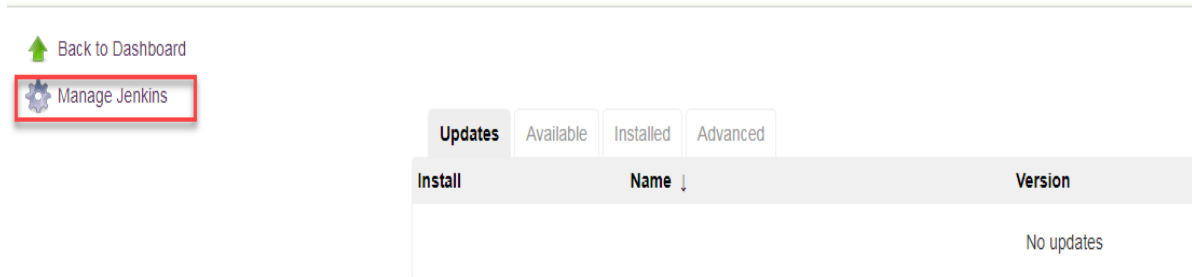
Install Build Pipeline Plugin in Jenkins

With the **build pipeline** plugin, you can create a pipeline view of incoming and outgoing jobs, and create triggers which require manual intervention.

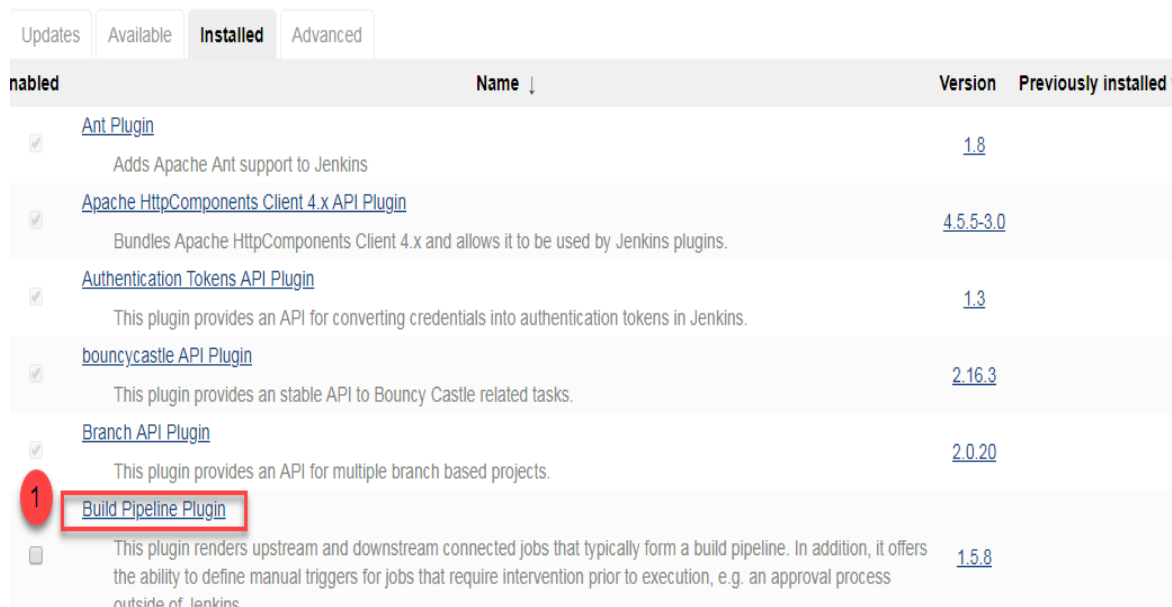
Here is how you can install the **build pipeline** plugin in your Jenkins:

Step 1) The settings for the plugin can be found under,

Manage Jenkins > Manage Plugins.



If you have already installed the plugin, it is shown under the installed tab.



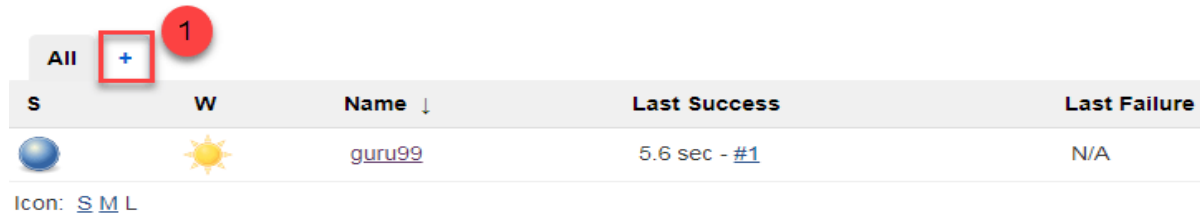
Step 2) If you do not have the plugin previously installed, it shows up under the **Available** tab.

Once you have successfully installed the **build pipeline** plugin in your Jenkins, follow these steps to create your Jenkins pipeline:

How to Create Jenkins Pipeline

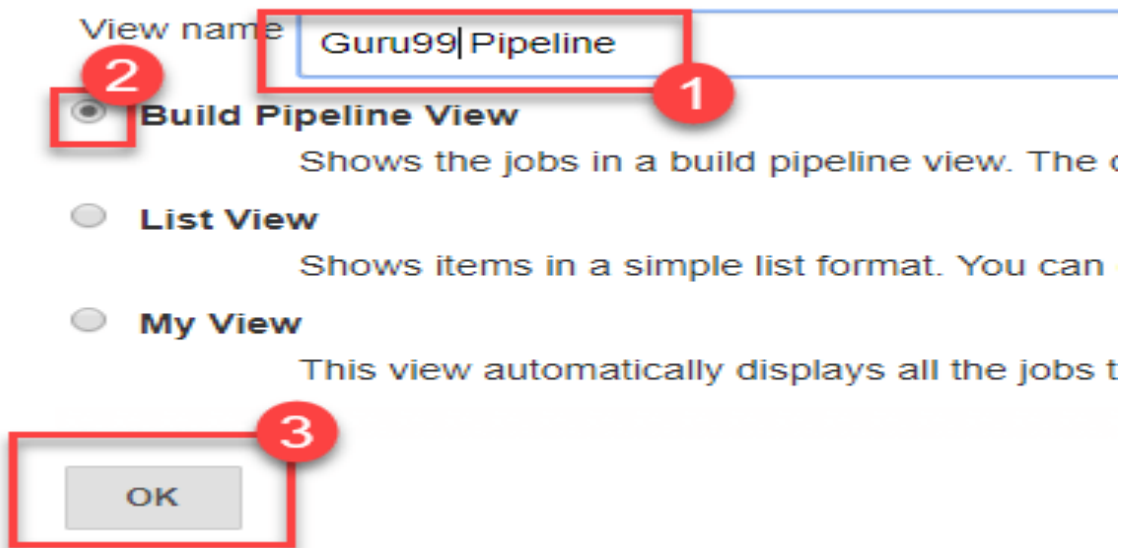
Once you are logged in to your Jenkins dashboard:

Step 1) Click on the “+” button on the left-hand side of your Jenkins dashboard to create a pipeline.



Step 2)

1. You will be asked to give a name to the pipeline view. We shall call it “**Guru99 Pipeline**” for the duration of this demo.
2. Select **Build a pipeline view** under **options**
3. Click **ok**



Step 3) In the next page, you will be asked for some more details to configure your Jenkins pipeline. Just accept the default settings, and make sure you choose the first job under the settings.

Click on **Apply** and then **OK**.

Name	guru99
Description	<div></div>
	[Plain text] Preview
Filter build queue	<input type="checkbox"/>
Filter build executors	<input type="checkbox"/>
Build Pipeline View Title	<div></div>
Pipeline Flow	
Layout	Based on upstream/downstream relationship
	<small>This layout mode derives the pipeline structure based on the upstream/downstream trigger relation supported layout mode, but is open for extension.</small>
	Upstream / downstream config
Select Initial Job	guru99
Trigger Options	
Build Cards	Standard build card
	<small>Use the default build cards</small>
Restrict triggers to most recent successful builds	<input type="radio"/> Yes <input checked="" type="radio"/> No
OK	Apply

This will show you the sample pipeline view of your item, as given below:

Build Pipeline

Run

History

Configure

Add Step

Delete

Manage

Pipeline

#16

#16 Hello world

Jun 18, 2018 8:28:28 AM

4.9 sec

admin

Running a Pipeline build

Step 1) For running a pipeline build, you need to chain your jobs first. For this, go to your first job and click on configure.

The screenshot shows the Jenkins dashboard with a sidebar on the left containing links like 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Credentials', and 'New View'. The main area displays the 'Guru99 pipeline' view with a table of builds. A context menu is open for the 'guru99' project, showing options like 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'Rename'. The 'Configure' option is highlighted with a red box, and a red circle with the number '1' is next to it.

S	W	Name ↓	Last Success
		guru99	16 min - #1
		Hello world	N/A

Icon: [S](#) [M](#) [L](#)

Step 2) Now, under **Build Triggers**, check the **Build after other projects are built** option.

The screenshot shows the 'Build Triggers' configuration page. The 'Build after other projects are built' checkbox is checked and highlighted with a red box, with a red circle containing the number '1' next to it. Below this, the 'Projects to watch' field contains the text 'guru99'. A dropdown menu is open, showing a list of projects: 'Guru99' (highlighted with a blue bar and a red box) and 'Guru99 Project 1'. A red circle containing the number '2' is next to the 'Guru99' project in the dropdown.

Thus, a chain for all your jobs has been created.

Step 3) Install the **Build Pipeline view** plugin if you don't have it installed already.

Step 4) Go to your Jenkins dashboard and create a view by clicking on the “+” button. Select the **Build Pipeline View** option and click **OK**.

Name	<input type="text" value="guru99"/>
Description	<div></div>
	[Plain text] Preview
Filter build queue	<input type="checkbox"/>
Filter build executors	<input type="checkbox"/>
Build Pipeline View Title	<input type="text"/>
Pipeline Flow	
Layout	Based on upstream/downstream relationship
	<small>This layout mode derives the pipeline structure based on the upstream/downstream supported layout mode, but is open for extension.</small>
	Upstream / downstream config
Select Initial Job	<input type="text" value="guru99"/>
Trigger Options	
Build Cards	Standard build card
	<small>Use the default build cards</small>
Restrict triggers to most recent successful builds <input type="radio"/> Yes <input checked="" type="radio"/> No	
<input type="button" value="OK"/>	<input type="button" value="Apply"/>

Step 5) Under **Pipeline view configuration**, locate **Pipeline Flow**.

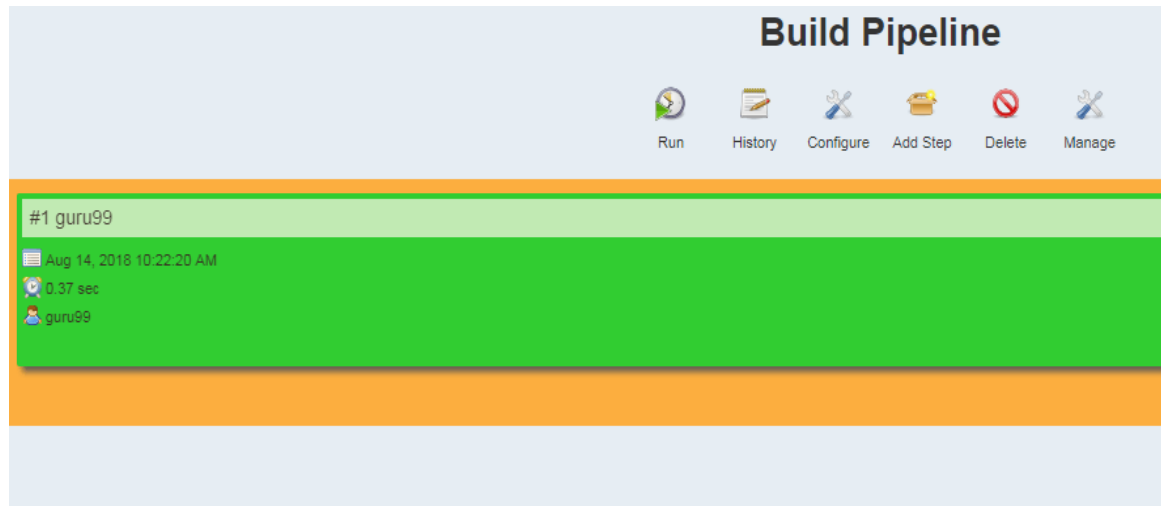
Under **Pipeline flow**, select the initial job to run. Now choose the job which has chains to other jobs, as shown in **Step 1** and **Step 2**.

Pipeline Flow	1
Layout	Based on upstream/downstream relationship
	<small>This layout mode derives the pipeline structure based on the upstream/downstream supported layout mode, but is open for extension.</small>
	Upstream / downstream config
2	<div> <div>Select Initial Job</div> <div> demo2 demo2 Guru99 Guru99 Project 1 </div> </div>
Trigger Options	

Here we have selected **Guru99 Project 1** as the initial job, chained to other jobs. So, one by one, the jobs will run in the pipeline.

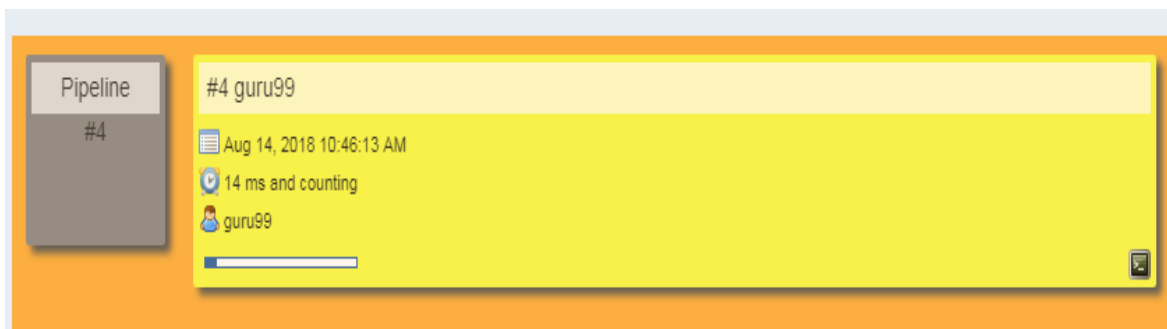
When the Jenkins pipeline is running, you can check its status with the help of Red and Green status symbols. Red means the pipeline has failed, while green indicates success.

In this Jenkins pipeline example, we see that the button is green. Hence, the pipeline is successful.



Running Jenkins pipeline

Click on **Run** to run the Jenkins pipeline. It will look something like this:



In the Jenkins pipeline script example above, we are demonstrating a simple “helloworld.java” program. But in real time projects, you will be responsible for creating and building complex pipelines in Jenkins. See below for a sample pipeline view.

		build	test: integration-&-quality	test: functional	test: load-&-security	approval	deploy: prod
	Average stage times: (Average full run time: ~5s)	836ms	20min 43s	9ms	7ms	89ms	5ms
#17	Sep 22 15:05 No Changes Retry Download	538ms	10s	10ms	8ms	72ms (paused for 7s)	4ms
#16	Sep 22 15:04 No Changes Retry Download	479ms	6s	9ms	9ms	74ms (paused for 6s)	5ms
#15	Sep 22 15:03 No Changes Retry Download	922ms	6s	10ms	9ms failed		
#14	Sep 22 15:03 No Changes Retry Download	1s	8s	12ms	9ms	80ms (paused for 5s)	5ms
#13	Sep 22 15:02 No Changes Download	942ms	9s	13ms failed			
#12	Sep 22 15:02 No Changes Retry Download	1s	6s	13ms	11ms	111ms (paused for 5s) aborted	

Create and Manage Users in Jenkins (User Management in Jenkins)

Generally, in a large organization, there are multiple separate teams to run and manage jobs in Jenkins. But managing this crowd of users and assigning roles to them is too difficult.

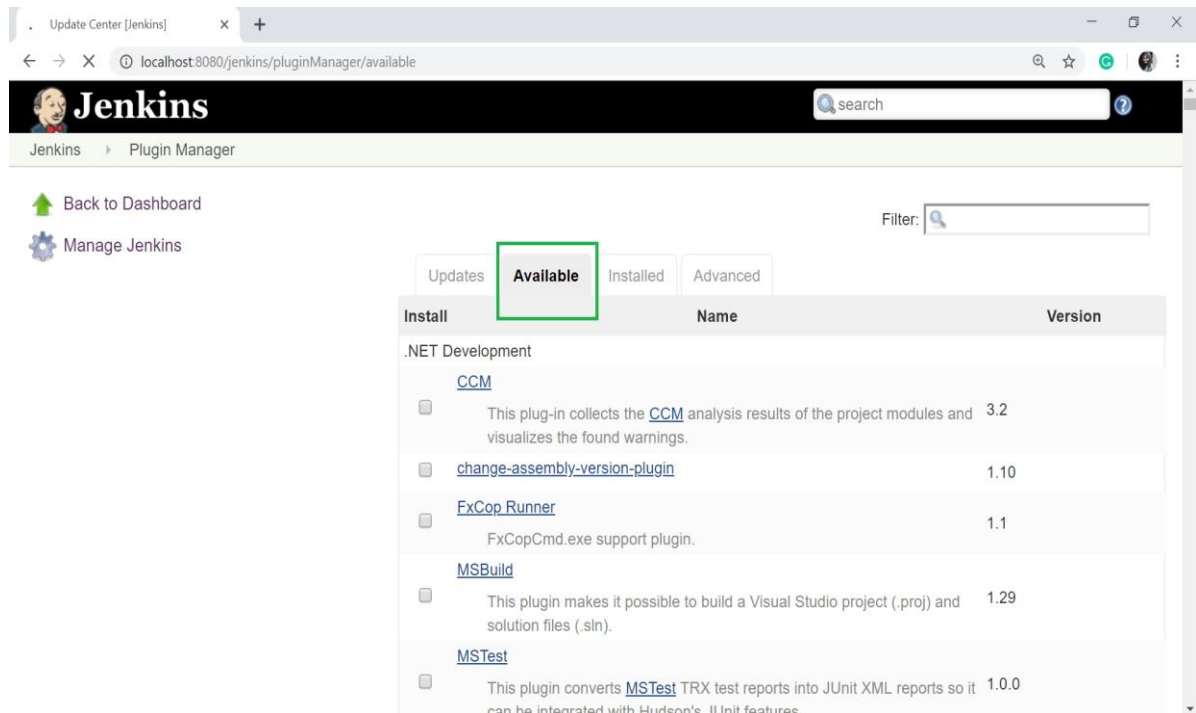
By default, when you create a user in Jenkins, it can access almost everything. In this, you can create multiple users but can only assign the same global roles and privileges to them. This is not ideal, especially for large organizations.

To enable you to assign different roles and privileges to different users in Jenkins, you should have to install the **Role Strategy Plugin**.

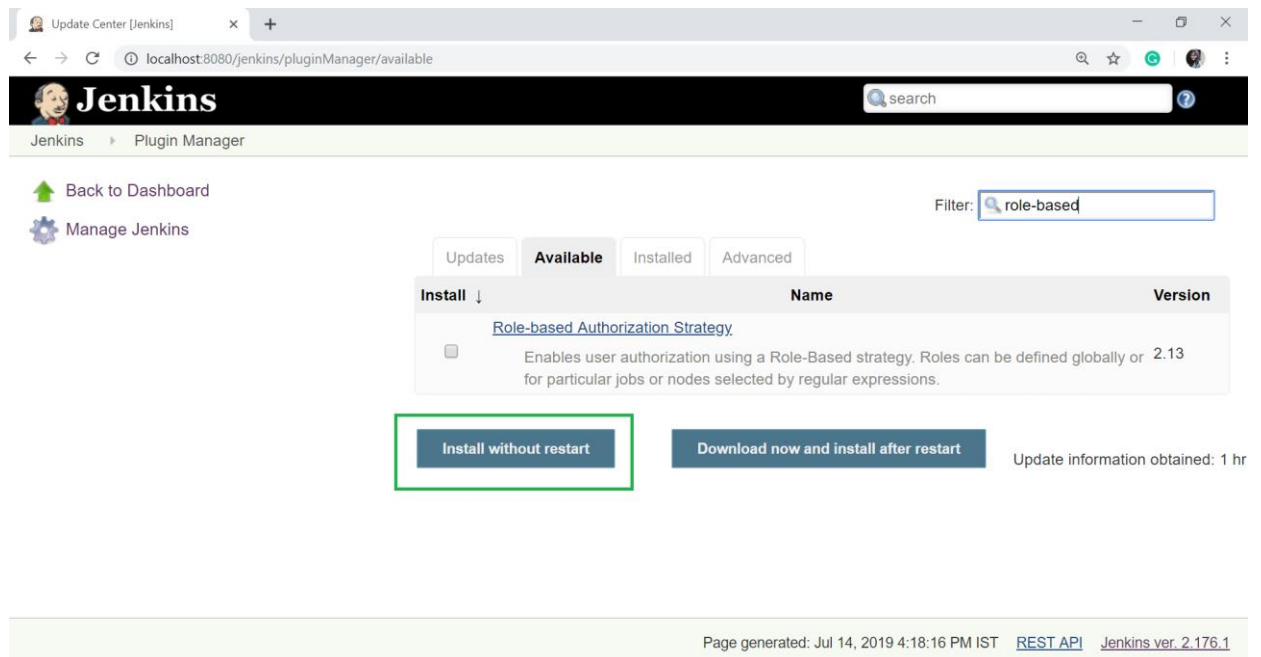
Install Role-based Authorization Strategy Plugin

Step 1: Open your Jenkins dashboard by visiting <http://localhost:8080/jenkins>

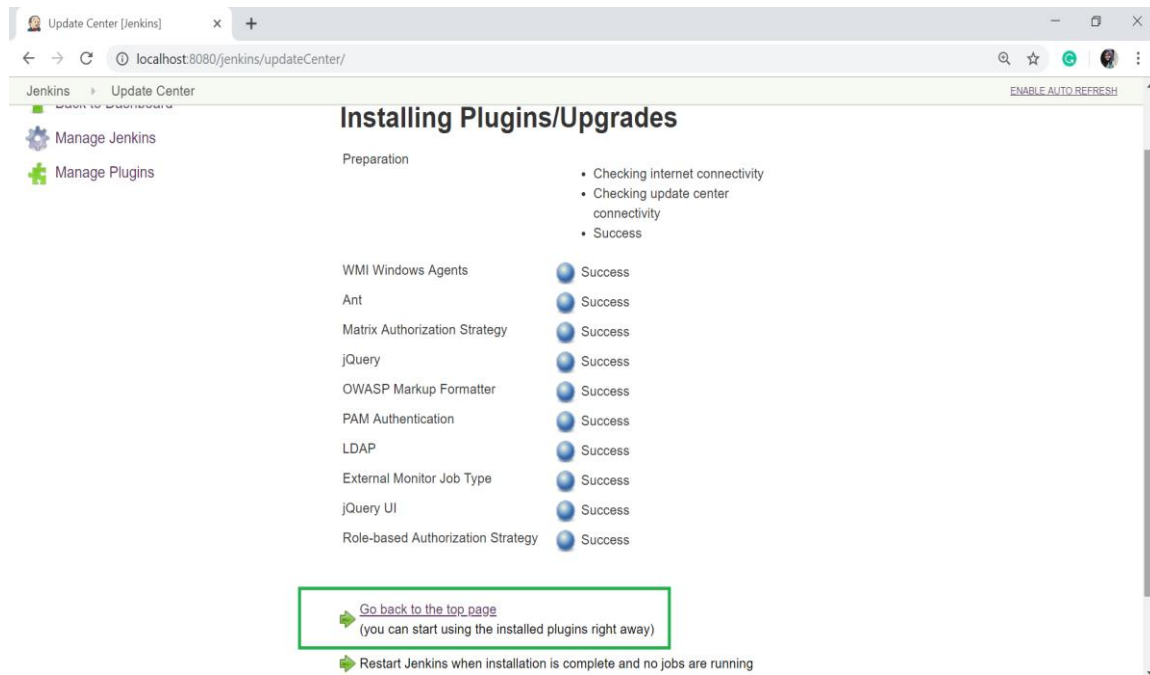
Steps 2: Click on 'Manage Jenkins' and select the 'Available' tab.



Step 3: On the filter option, type "role-based" and press Enter.



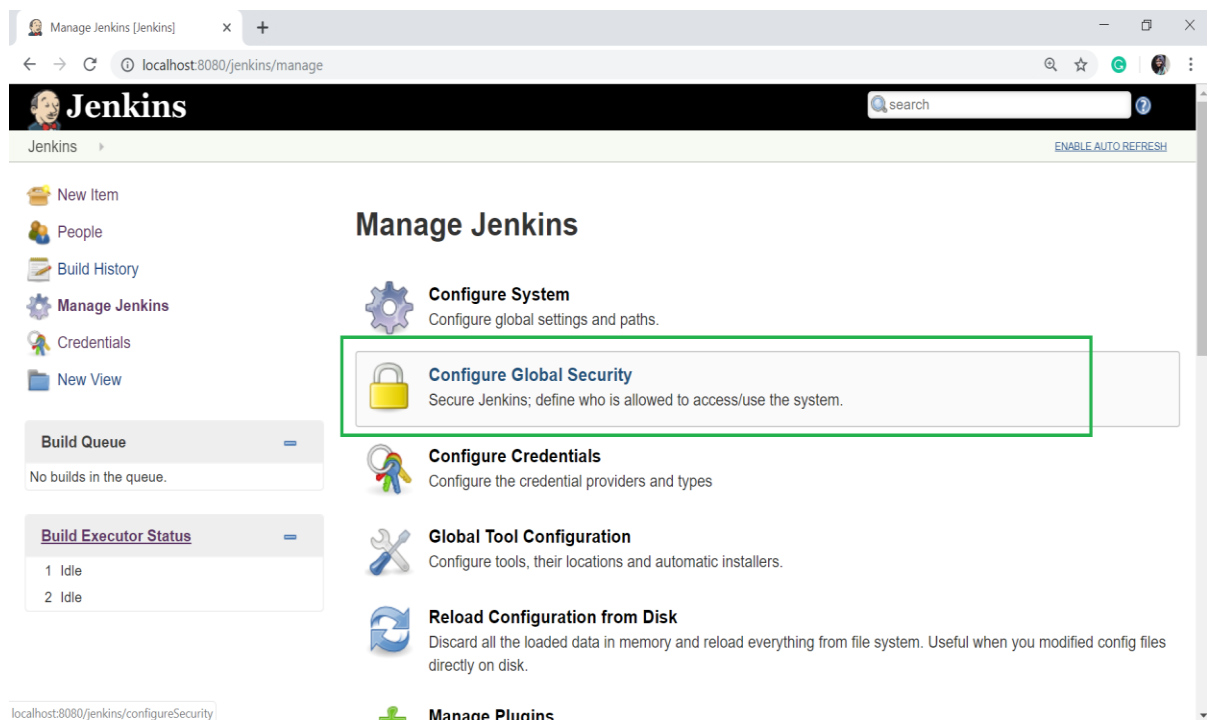
Step 4: Now, select the plugin and click on 'Install without restart' button.



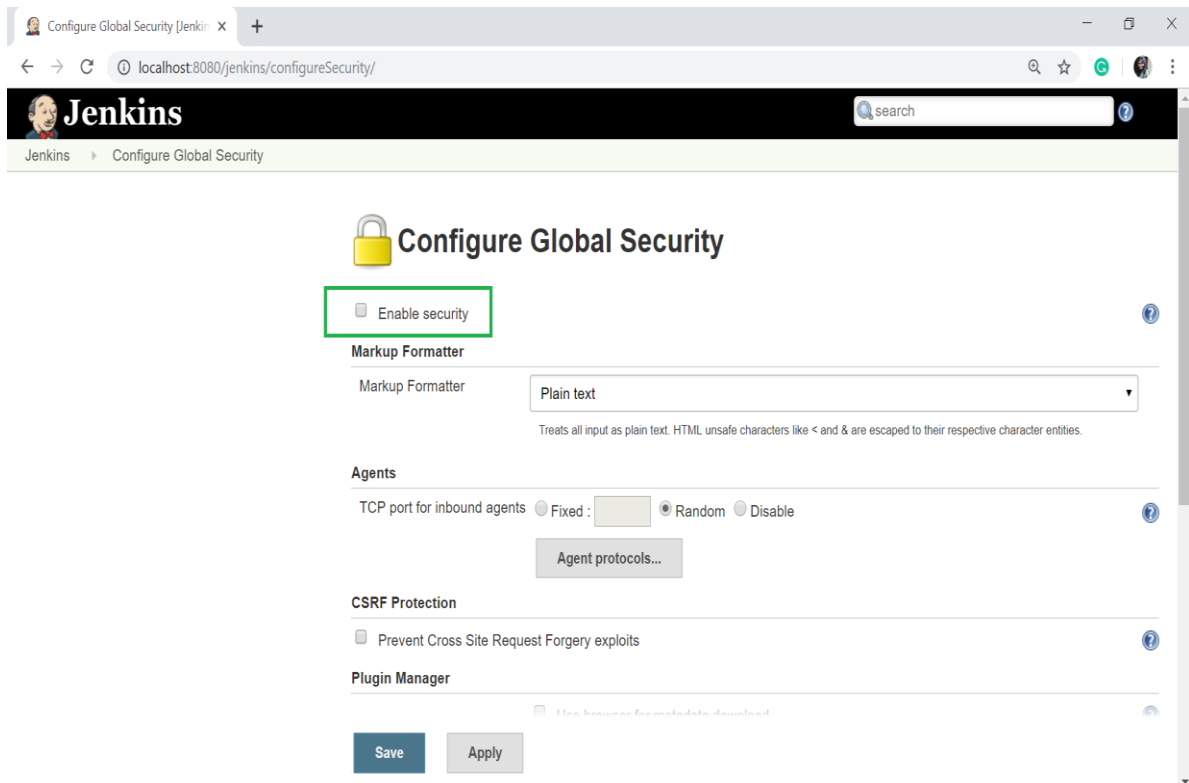
Step 5: Click on 'Go back to the top page'.

Enable Role-Based Strategy on Jenkins

Step 1: After Plugin installation, go to the 'Manage Jenkins' and then click on 'Configure Global Security'.



When you click the Configure Global Security option then you will see the following page:



Configure Global Security (Jenkins) x +

localhost:8080/jenkins/configureSecurity/

Jenkins

Jenkins > Configure Global Security

Configure Global Security

☐ Enable security

Markup Formatter

Markup Formatter: Plain text

Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Agents

TCP port for inbound agents: ☐ Fixed: ☒ Random ☐ Disable

Agent protocols...

CSRF Protection

☐ Prevent Cross Site Request Forgery exploits

Plugin Manager

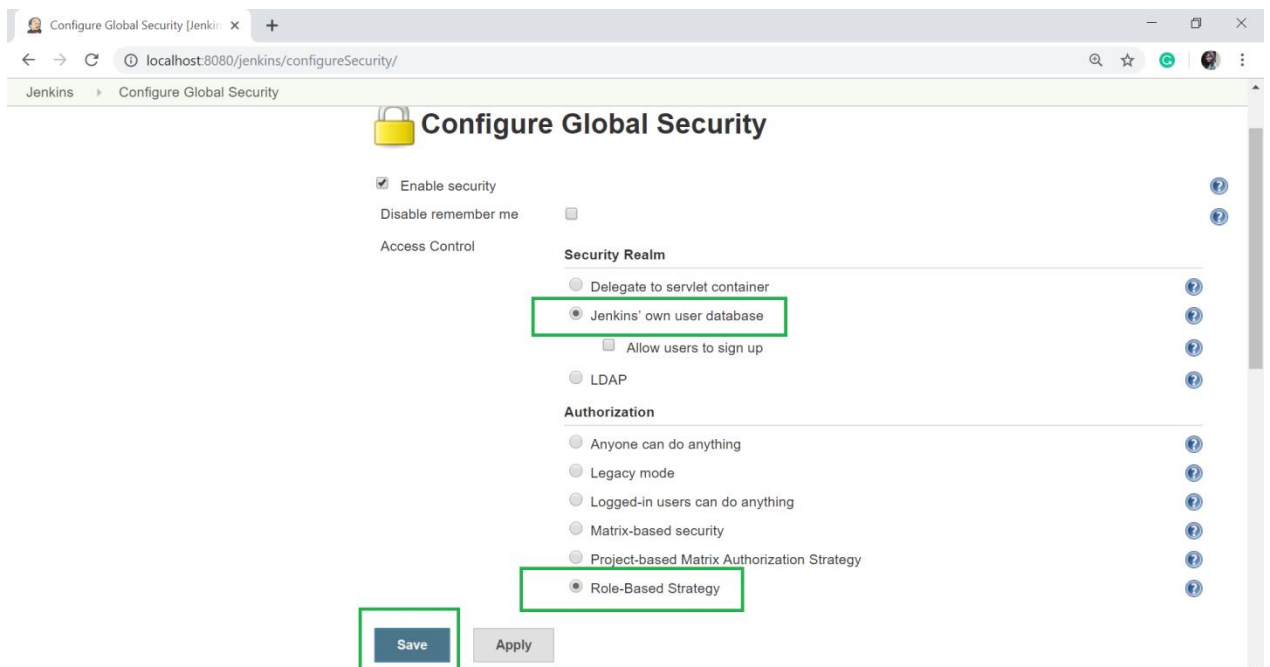
☐ Use hudson for metadata download

Save Apply

Check on **Enable security** option.

Step 2: On the Security Realm section, select '**jenkins' own user database**'.

- On Authorization section, select '**Role-Based Strategy**'.



Configure Global Security (Jenkins) x +

localhost:8080/jenkins/configureSecurity/

Jenkins

Jenkins > Configure Global Security

Configure Global Security

☒ Enable security

Disable remember me ☐

Access Control

Security Realm

☐ Delegate to servlet container

☒ Jenkins' own user database

☐ Allow users to sign up

☐ LDAP

Authorization

☐ Anyone can do anything

☐ Legacy mode

☐ Logged-in users can do anything

☐ Matrix-based security

☐ Project-based Matrix Authorization Strategy

☒ Role-Based Strategy

Save Apply

Click on **Save**.

Step 3: You will be prompted to add your first user. Here, we are setting up an admin user for the system.

Create First Admin User [Jenkins] x +

localhost:8080/jenkins/securityRealm/firstUser

Jenkins 2 search log in

Jenkins > Jenkins' own user database

Back to Dashboard
Manage Jenkins
Create User

Create First Admin User

Username: admin
Password:
Confirm password:
Full name: Administrator
E-mail address: javatpoint@gmail.com

Create First Admin User

Page generated: Jul 14, 2019 4:51:49 PM IST [Jenkins ver. 2.176.1](#)

Click on **Create First Admin User**.

Jenkins x +

localhost:8080/jenkins/securityRealm/createFirstAccount

Jenkins 2 search Administrator | log out

Jenkins > Jenkins' own user database

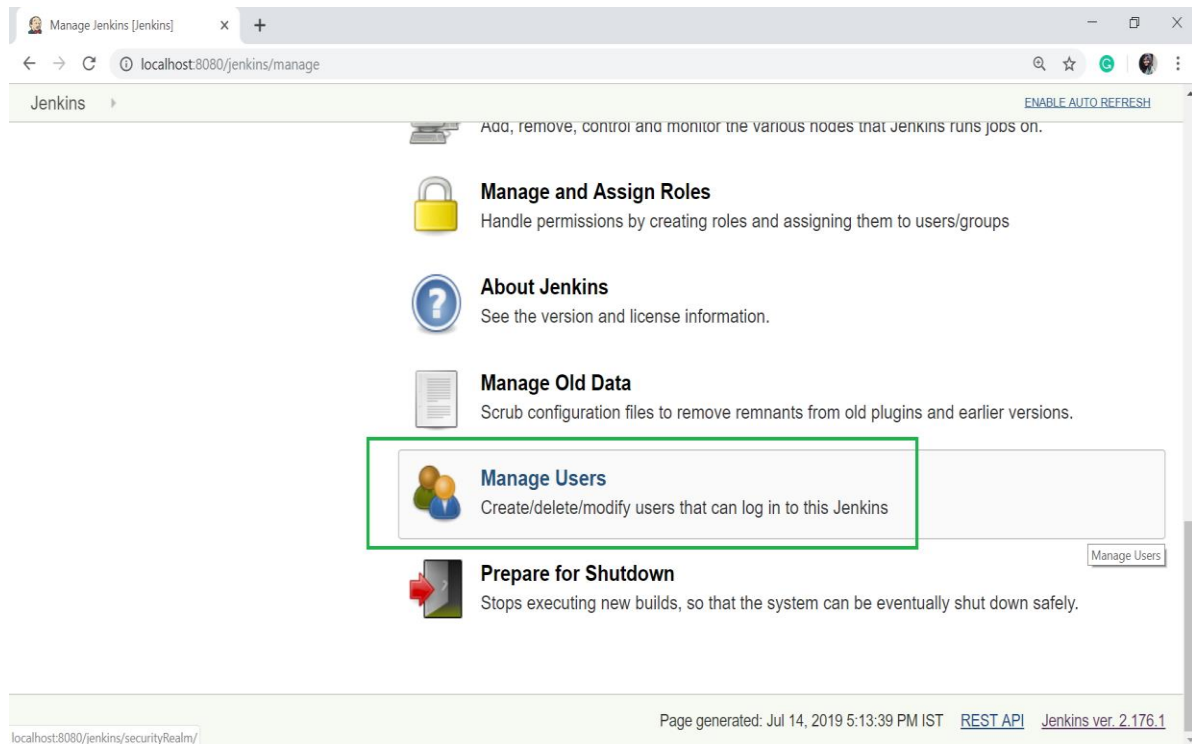
New Item
People
Build History
Manage Jenkins

Success

You are now logged in. Go back to [the top page](#).

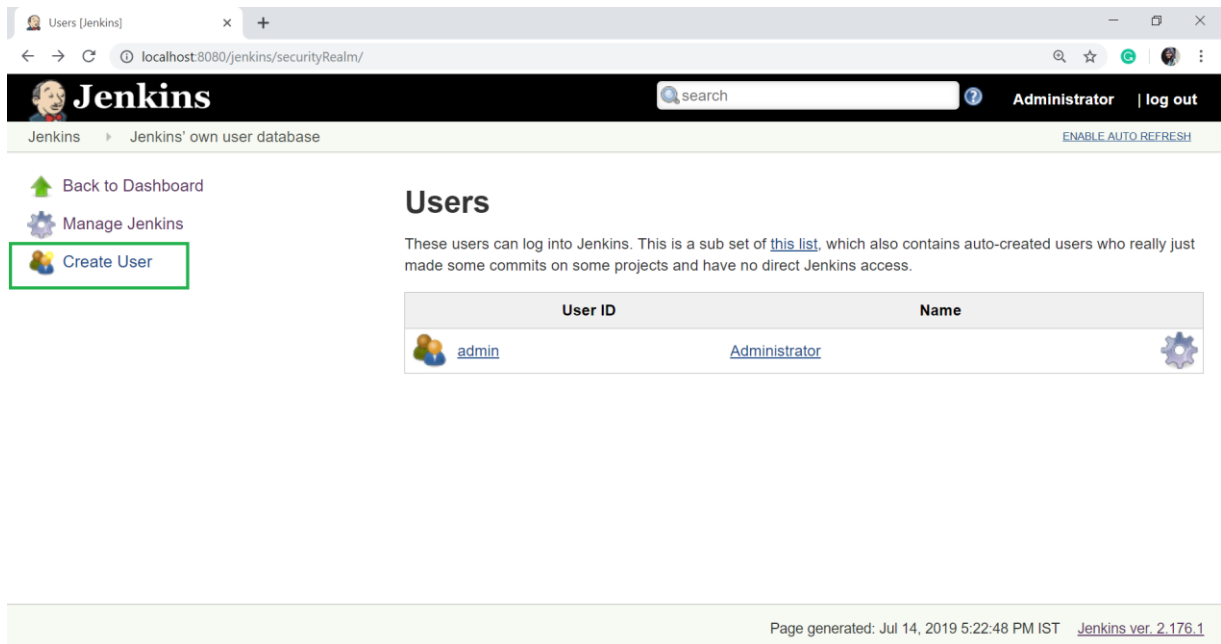
Creating User on Jenkins

Step 1: Now it's time to setup your users in the system. When you go to manage Jenkins and scroll down, you will see the '**Manage Users**' option. Click on this option.





Step 2: Just Like, you defined the admin user, and start creating other users for the system. Let's create another user:

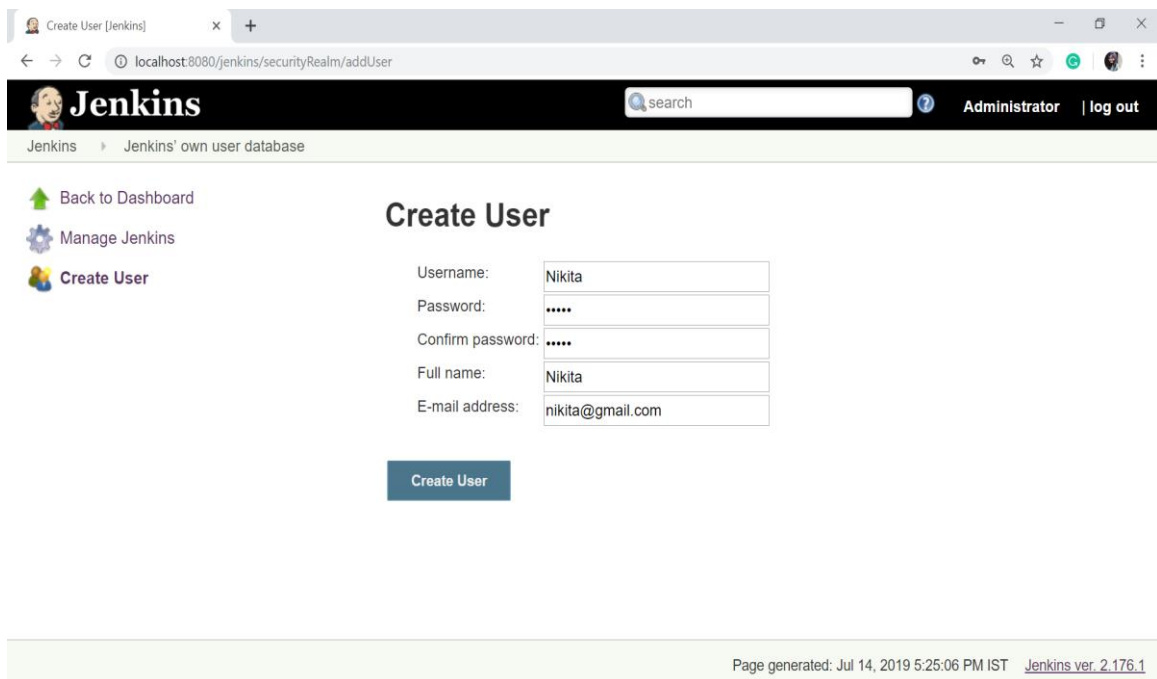
- To create another user click on 'Create User' option on the left hand side of the Manage Users page.



The screenshot shows the Jenkins 'Users' page. The browser address bar is 'localhost:8080/jenkins/securityRealm/'. The Jenkins header includes a search bar, the user 'Administrator', and a 'log out' link. The left sidebar has links for 'Back to Dashboard', 'Manage Jenkins', and 'Create User' (which is highlighted with a green box). The main content area is titled 'Users' and contains a table of users.

User ID	Name
 admin	Administrator 

Page generated: Jul 14, 2019 5:22:48 PM IST [Jenkins ver. 2.176.1](#)



The screenshot shows the Jenkins 'Create User' page. The browser address bar is 'localhost:8080/jenkins/securityRealm/addUser'. The left sidebar has links for 'Back to Dashboard', 'Manage Jenkins', and 'Create User' (which is highlighted). The main content area is titled 'Create User' and contains a form with the following fields:

- Username: Nikita
- Password:
- Confirm password:
- Full name: Nikita
- E-mail address: nikita@gmail.com

Below the form is a blue 'Create User' button.

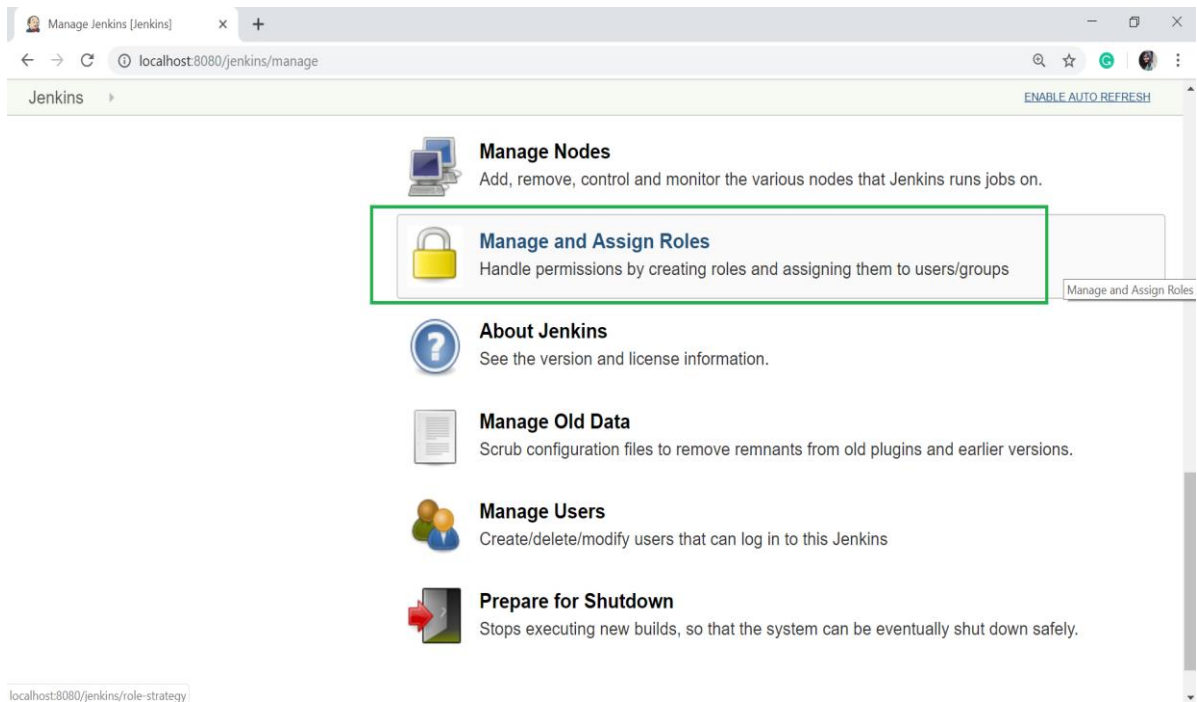
Page generated: Jul 14, 2019 5:25:06 PM IST [Jenkins ver. 2.176.1](#)

- Click on **Create User** button.

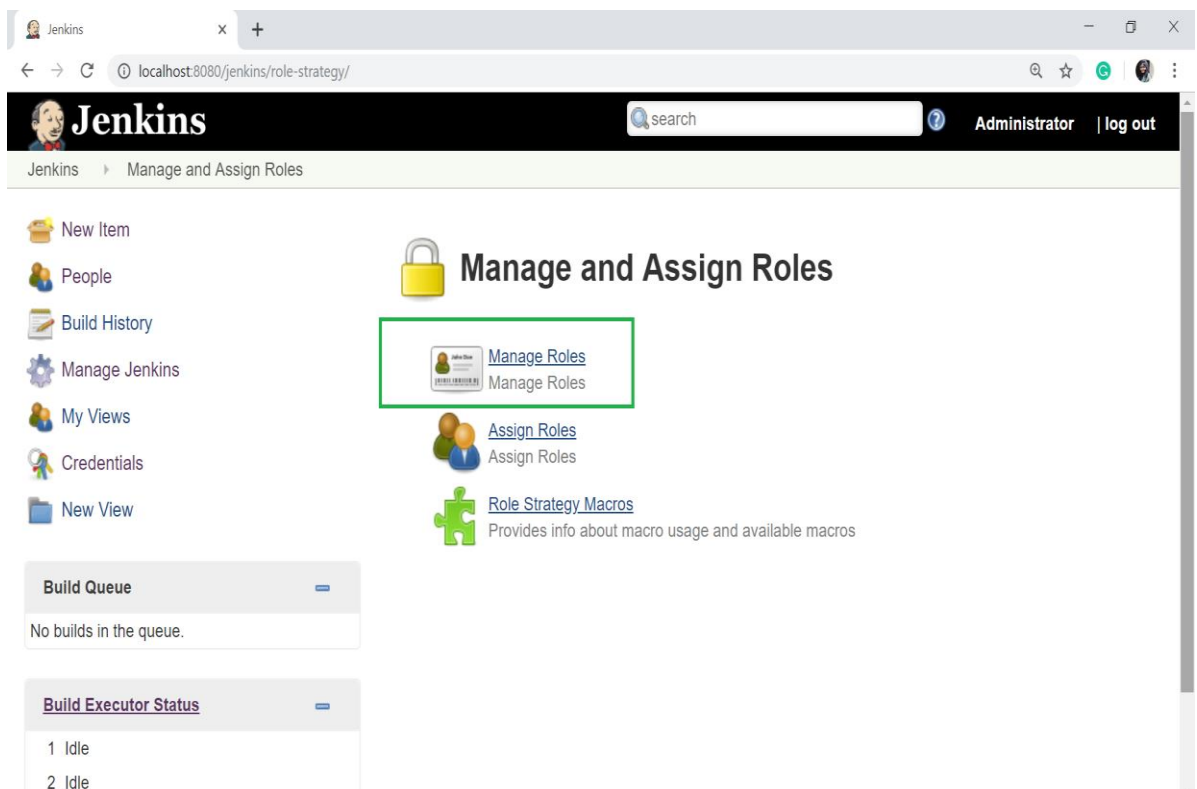
Managing User Roles on Jenkins

Step 1:

- Click on **Manage Jenkins**.
- Select **Manage and Assign Roles**. Note that, Manage and Assign Role will only be visible if you have installed the Role strategy plugin.



Step 2: Click on **Manage Roles** option to add new roles.



Step 2: To create a new role called "developer".

- Type "developer" in the **Role to add** option.

- Click on **Add** to create a new role.
- Now, select the appropriate permissions that you want to assign to the developer role.
- Then Click on **Save** button.

The screenshot shows the Jenkins 'Manage Roles' interface. At the top, there's a search bar and a user profile 'Administrator' with a 'log out' link. Below this is a breadcrumb 'Jenkins > Manage and Assign Roles'. The main section is titled 'Manage Roles'. Under 'Global roles', there's a table with columns: Role, Overall, Credentials (Administer, Read, Create, Delete, ManageDomains, Update, View, Build, Configure, Connect), and Agent (Create, Delete, Disconnect, Provision, Build, Cancel, Configure, Create). The 'developer' role is highlighted with a green border. Below the table, there's a 'Role to add' input field and an 'Add' button, both highlighted with green borders. The 'Add' button is labeled 'Add' and the input field contains the text 'developer'.

Role	Overall	Credentials							Agent										
		Administer	Read	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
developer	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Role to add:

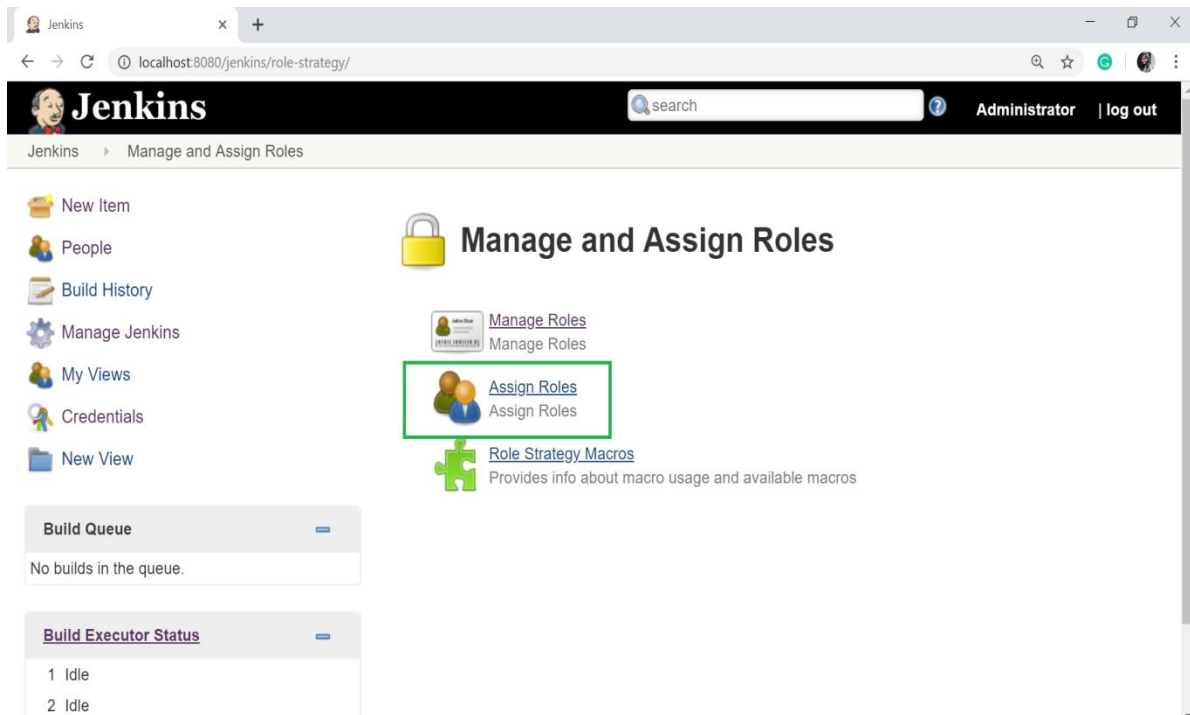
Project roles

Role	Pattern	Credentials							Job							Run		SCM	
		Create	Delete	ManageDomains	Update	View	Build	Cancel	Configure	Create	Delete	Disconnect	Move	Read	Webhooks	Delete	Deploy	Update	Test

Assigning User Roles on Jenkins

Step 1: Now, you have created roles, let's assign them to specific users.

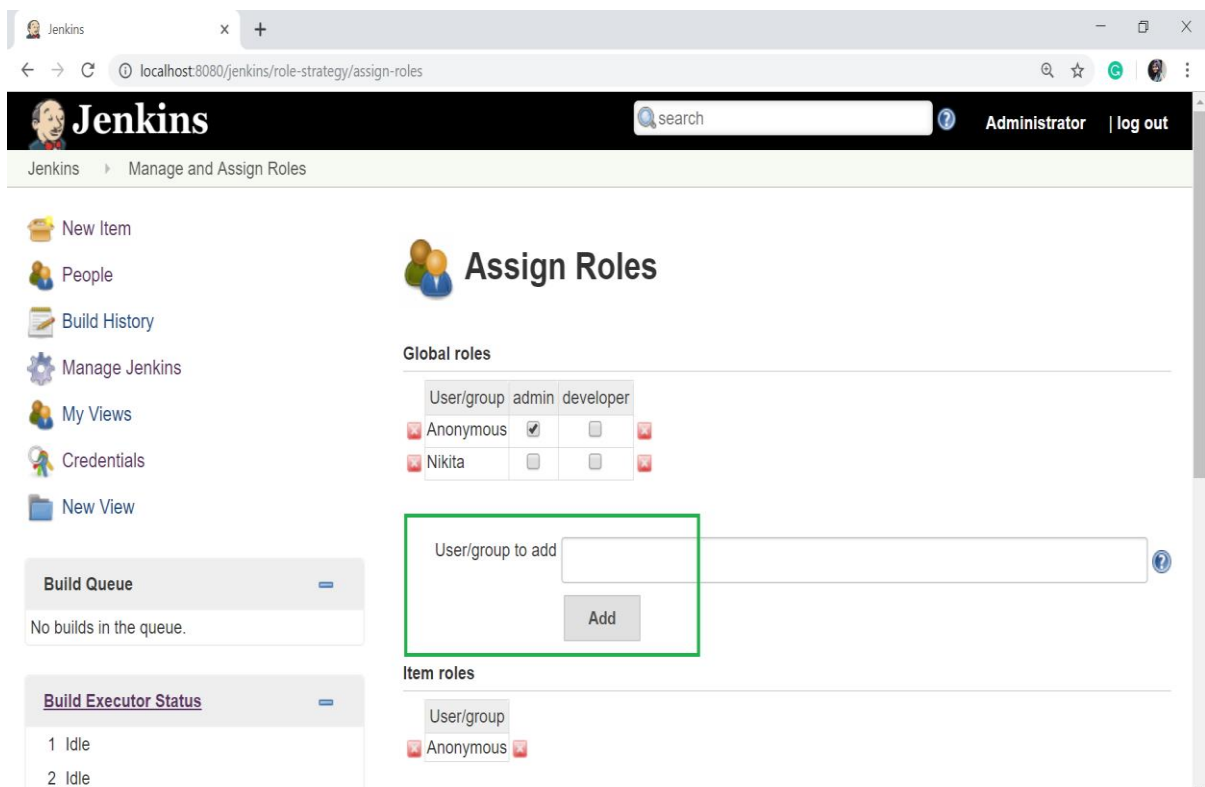
- Click on **Manage Jenkins**.
- Then select **Manage and Assign Roles**.
- Click on **Assign Roles**.



The screenshot shows the Jenkins web interface at the URL `localhost:8080/jenkins/role-strategy/`. The page title is "Manage and Assign Roles". On the left sidebar, there are links for "New Item", "People", "Build History", "Manage Jenkins", "My Views", "Credentials", and "New View". Below these are sections for "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing 1 Idle and 2 Idle executors). The main content area has a yellow padlock icon and the title "Manage and Assign Roles". It contains three links: "Manage Roles" (with a sub-link "Manage Roles"), "Assign Roles" (with a sub-link "Assign Roles" and highlighted by a green box), and "Role Strategy Macros" (with a sub-link "Provides info about macro usage and available macros").

Step 2: Let's add new role "developer" to user.

- Add the User name on **User/group to add** option.
- Click on **Add** button.



The screenshot shows the Jenkins web interface at the URL `localhost:8080/jenkins/role-strategy/assign-roles`. The page title is "Assign Roles". On the left sidebar, there are links for "New Item", "People", "Build History", "Manage Jenkins", "My Views", "Credentials", and "New View". Below these are sections for "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing 1 Idle and 2 Idle executors). The main content area has a group of three people icon and the title "Assign Roles". It contains a section for "Global roles" with a table:

User/group	admin	developer
<input checked="" type="checkbox"/> Anonymous	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Nikita	<input type="checkbox"/>	<input type="checkbox"/>

Below the table is a form with a text input field labeled "User/group to add" (highlighted by a green box) and an "Add" button. At the bottom, there is a section for "Item roles" with a table:

User/group
<input checked="" type="checkbox"/> Anonymous

- Select "developer" role check box.

Jenkins

localhost:8080/jenkins/role-strategy/assign-roles

Administrator | log out

Jenkins Manage and Assign Roles

New Item

People

Build History

Manage Jenkins

My Views

Credentials

Assign Roles

Global roles

User/group	admin	developer
Anonymous	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Nikita	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

- You can assign any role to any user as per your requirement.
- Then click on **Save**.