

# Jenkins

## 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.

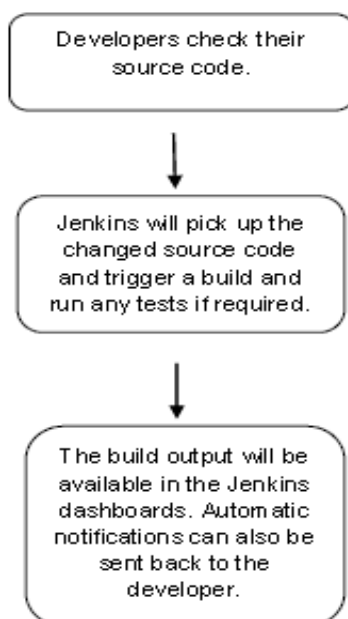
Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

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.

Possible steps executed by Jenkins are for example:

- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script
- Archive a build result
- Running software tests

## Work Flow:



## What is Continuous Integration?

Continuous Integration (CI) is a development practice in which the developers are needed to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

## Continuous Integration with Jenkins

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.

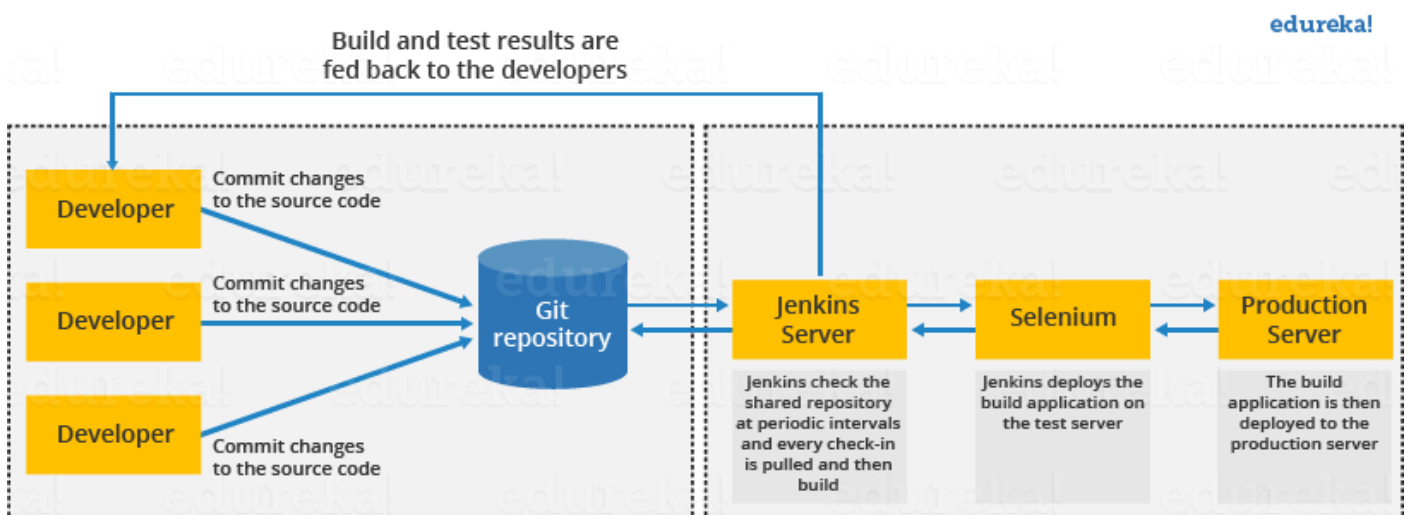
- Developer teams have to wait till the complete software is developed for the test results.
- There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
- It slows the 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.

This is what Continuous Integration (CI) is all about. Jenkins is the most mature Continuous Integration tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



## Let's see how Jenkins works.

The above diagram is representing the following functions:

- 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.

## Benefits of Using Jenkins

- **Faster delivery cycles:** Automated builds and deployments drastically reduce manual intervention and streamline the delivery process.
- **Improved software quality:** Automated testing helps identify bugs early in the development cycle, leading to a more robust application.
- **Enhanced collaboration:** Jenkins provides a central platform for developers and testers to track builds and collaborate effectively.
- **Reduced errors:** Automation minimizes human error and ensures consistency in the delivery process.
- **Cost-effectiveness:** Being open-source, Jenkins eliminates licensing costs, making it an attractive option for businesses of all sizes

## Jenkins Architecture

Certain needs could not be met with one Jenkins server. Firstly, we may need multiple distinct environments in which to test our builds. A single Jenkins server will not be able to do this. Secondly, if larger and heavier projects are being produced regularly, a single Jenkins server will be overwhelmed.

Jenkins distributed architecture was created to meet the above requirements.

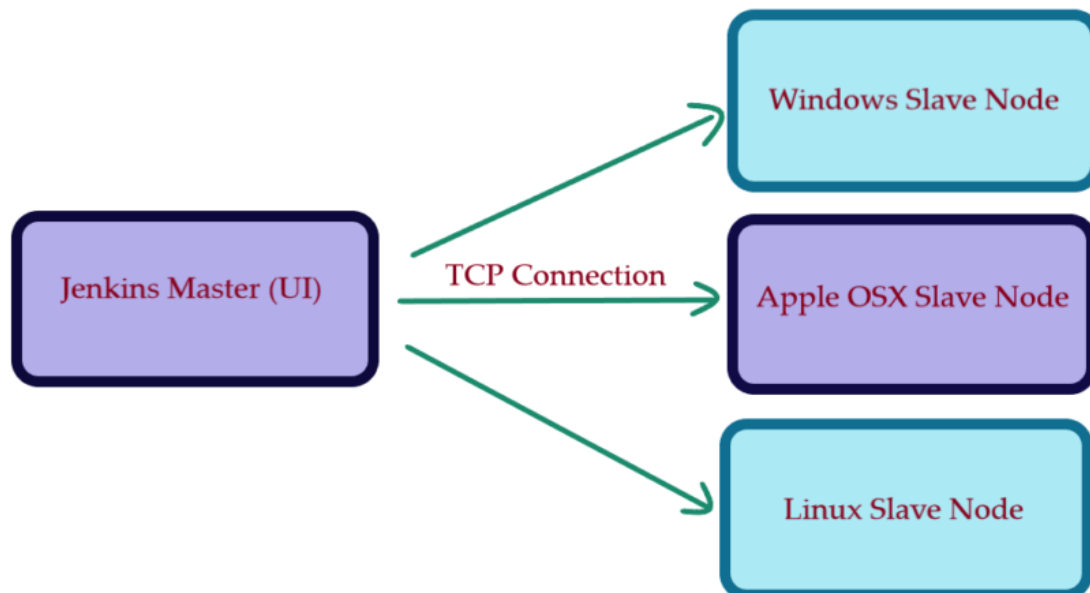
Furthermore, Jenkins manages distributed builds using a Master-Slave architecture. The TCP/IP protocol is used to communicate between the Master and the Slave in this design.

### Jenkins Master

The Jenkins master is in charge of scheduling the jobs, assigning slaves, and sending builds to slaves to execute the jobs. It'll also keep track of the slave state (offline or online) and retrieve the build result responses from slaves and display them on the console output.

### Jenkins Slave

It runs on the remote server. The Jenkins server follows the requests of the Jenkins master and is compatible with all operating systems. Building jobs dispatched by the master are executed by the slave. Also, the project can be configured to choose a specific slave machine.



## How to install jenkins

- First create an Instance with Linux AMI in aws
- Connect to Instance and go to root use: `sudo -i`
- then check java version and install use: `(sudo yum install java-17-amazon-corretto-headless)`
- We need to Download Jenkins repo for Linux use this url: `(sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo )`
- Import Key from 'jenkins.io' `(sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key )`
- Now Install Jenkins use : `yum install jenkins`
- After Installing jenkins Enable jenkins use : `systemctl enable jenkins`
- Start the Jenkins Server : `systemctl start jenkins`
- See the status of jenkins server : `systemctl status jenkins`
- Now go to the instance's security settings select security groups then edit [inbound rules with] port 8080 and save it
- Copy the public Ip of instance and paste it into new tab followed by port number 8080 (public Ip 19496296265:8080)
- Copy the path and paste it into instance with cat and see the secret key `(/var/lib/jenkins/secrets/initialAdminPassword)`
- Cat path and copy the secret key and paste it into the password box in the new tab
- Enter all the details to sign up into Jenkins
- Now the jenkins is ready to use

# Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins

Jenkins 2.440.3

## What are Jenkins plugins?

Jenkins plugins are the components of the Jenkins environment. Plugins are small, independent programs that can improve and extend the functionality of Jenkins CI/CD. There are over 1,900+ Jenkins plugins available right now and it keeps on increasing month by month. So, there is a plugin to suit almost every need in software development. We can install the Jenkins Plugins from the Jenkins Update Centre. Jenkins Update Centre is a centralised repository of Jenkins plugins which is being maintained by the Jenkins community.

## Benefits of Using Jenkins Plugins

- **Helps to Automate Task:** Jenkins Plugins are very useful to automate tasks to reduce manual efforts in the software development process.
- **Simplifies Testing:** Jenkins Plugins can improve the quality of software by automating testing and analysis.
- **Reduces Human Error:** Automating any task using the Plugins can reduce human errors.
- **Increased Agility:** Jenkins Plugins will increase the agility of software development teams by making it easier to release new software changes and resolve bugs more frequently.

## Popular Jenkins plugins

Sr. No	Jenkins plugins	Purpose
1	Git Plugin	Used to transfer GitHub repository data and helps to schedule your project builds and automatically triggers after commit changes.

2	Kubernetes plugins	The Kubernetes Plugin for Jenkins enables dynamic provisioning of build agents as Kubernetes pods, optimizing resource utilization and scalability in CI/CD pipelines.
3	Docker plugin	Helps to manage Docker containers and services -creating, building, pushing, running, and managing images.
4	Jira plugin	Helps to integrates Jenkins with Atlassian Jira Software to automatically transfer the development and build-related data.
5	Maven integration plugin	Supports Spring boot applications and Apache Maven projects.

## How to install jenkins plugins?

- Open your Jenkins Dashboard > go to manage jenkins.
- click on the manage plugins. Then search for the plugins you want to install.
- After your jenkins plugin selection > click the install button to confirm the installation.
- After you install the Jenkins plugins, you need to restart Jenkins for the changes to take effect.
- Select > Restart Jenkins when installation is complete and no jobs are running

## Download progress

### Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

### Jersey 2 API

✓ Success

### Jira

✓ Success

### Loading plugin extensions

✓ Success

→ [Go back to the top page](#)

(you can start using the installed plugins right away)

→ ☐ Restart Jenkins when installation is complete and no jobs are running

- Now you can see your installed plugin in the manage plugins > installed plugins.

### Plugins

Updates

1

Available plugins

Installed plugins

Advanced settings

Q ji

Name ↓

Enabled

Jira plugin 3.13

This plugin integrates Jenkins to Atlassian Jira.  
[Report an issue with this plugin](#)



## How to create Freestyle job

1. From the Jenkins dashboard, click new item from the sidebar menu.

Search (CTRL+K)

?

faizan khan

log out

Dashboard >

+ New Item

People

Build History

Manage Jenkins

My Views

All

+

S	W	Name ↓	Last Success	Last Failure	Last Duration
		sample	N/A	N/A	N/A

Build Queue

▼

No builds in the queue.

Build Executor Status

▼

1 Idle

2 Idle

Icon: S M L

Icon legend

Atom feed for all

Atom feed for failures

Atom feed for just latest builds

Add description

Description

this is first job

Plain text

Preview

Jira site

▼

☐ Discard old builds
 

?

☐ GitHub project

☐ This project is parameterized
 

?

☐ Throttle builds
 

?

☐ Execute concurrent builds if necessary
 

?

Advanced

▼

- Now select New Item --→ Enter name
- Click OK to start configuring your new Freestyle job.
- write a short description of what the job does. This helps others understand its purpose.

Options	Description
Discard old builds	If we want to discard old builds while starting the execution of the new build then we select this option.
GitHub Project	This option specifies that we are running our build with GitHub. We specify the URL of the GitHub project.
This project is parameterized	If we want to run our build with different parameters that would be passed during run time then we will use this option. Every parameter has some Name as well as Value.
Throttle builds	This option enforces a minimum time between builds based on the desired maximum rate.
Disable this project	If this option will be checked then no new build of this project will be executed.

Execute concurrent builds if necessary

If this option will be checked then we can execute multiple builds of this project in parallel.

## 5. Source Code Management: Configure the source code repository from which Jenkins will pull code for building. Options may include Git, Subversion, Mercurial, etc.

### Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

 Please enter Git repository.

Credentials ?

- none -

+ Add

Advanced

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

## 6. Build Triggers: Specify conditions that trigger a build, such as a code commit to the repository, a time-based schedule, or triggering builds from other projects.



---

## Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts) ?
  - ☐ Build after other projects are built ?
  - ☐ Build periodically ?
  - ☐ GitHub hook trigger for GITScm polling ?
  - ☐ Poll SCM ?
- 

Options	Description
Trigger builds remotely	This option is used when we want to trigger new builds by accessing a special predefined URL.
Build after other projects are built	A new build will be triggered for this project just after other builds are finished.
Build periodically	In the build periodically option, we need to give a proper format of time during which we need to build our job.
GitHub Pull requests	Trigger that integrates with GitHub Pull Requests and Issues activities and launches runs in response.
GitHub hook trigger for GITScm polling	If this option is checked, it means the build will be executed with the help of GitHub webhooks.
Poll SCM	Poll SCM option is almost similar to the build periodically option. Here also, we can give the timer but the difference is that build will only be executed when any code changes will be detected during that time duration.

7. Build Environment: Set up the build environment, including defining environment variables, configuring build tools, and specifying pre-build or post-build actions.

## Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Add timestamps to the Console Output
- ☐ Generate Release Notes
- ☐ Inspect build log for published build scans
- ☐ Terminate a build if it's stuck
- ☐ With Ant ?

8. Build: Define the build steps that Jenkins should execute when building the project. This can include shell commands, Windows batch commands, or executing scripts.

Add build step ^

Filter

Execute Windows batch command

Execute shell

Invoke Ant

Invoke Gradle script

Invoke top-level Maven targets

Options	Description
Execute Windows batch command	This option runs a windows batch script for building the project. The script runs with the workspace as the current directory.
Execute shell	This option runs a shell script for building the project. The script runs with the workspace as the current directory.
Invoke Ant	This option specifies a list of Ant targets to be invoked (separated by spaces ), or leave it empty to invoke the default Ant target specified in the build script.
Invoke Gradle Script	This option is for those projects that use Gradle as the build system. Here, Jenkins invokes Gradle with the given switches and tasks.
Invoke toplevel Maven targets	This is for those projects that use Maven as the build system. This leads Jenkins to invoke Maven with the given goals and options. Jenkins passes various environment variables to Maven, which you can access from Maven as "\${env. VARIABLE NAME} ".

Run with timeout	If a build does not complete by the specified amount of time, then the build will be terminated automatically and marked as aborted. Default time would be at least 3minutes.
------------------	---

9. Post-build Actions: After the build finishes, you can set up what should happen next. This could include things like saving important files, starting other projects, or sending out notifications.

Execute shell ?

Command

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

```
java -version
dir
```

Advanced ▾

Add build step ▾

10. Save: Once you've set everything up, click "Save" to save your project configuration.
11. Build: After saving, click "Build Now" to start your first build.

**Jenkins**

1
 faizan khan ▾
 log out

Dashboard > First job > #1 > Console Output

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#1'

Next Build

✓ Console Output

```

Started by user faizan khan
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/First job
[First job] $ /bin/sh -xe /tmp/jenkins4338757347580225198.sh
+ java -version
openjdk version "17.0.11" 2024-04-16 LTS
OpenJDK Runtime Environment Corretto-17.0.11.9.1 (build 17.0.11+9-LTS)
OpenJDK 64-Bit Server VM Corretto-17.0.11.9.1 (build 17.0.11+9-LTS, mixed mode, sharing)
+ dir
Finished: SUCCESS

```

## Jenkins Build Triggers:

1. Trigger builds remotely:
  - Use when you want to initiate a build from anywhere at any time.
  - Requires an authorization token for security.
  - Example URL format: `JENKINS_URL/job/JobName/build?token=TOKEN_NAME`
  - Example: `http://example.com/job/myproject/build?token=12345`
2. Build after other projects are built:
  - Useful when your project depends on the completion of another project.
  - Specify project names to watch and trigger conditions (e.g., stable build, unstable build, or build failure).
  - Example: Trigger this project after Project A and Project B complete successfully.
3. Build periodically:

- Schedule builds at specific time intervals.
  - Follows cron syntax (`MINUTE HOUR DOM MONTH DOW`).
  - Example: Build every 15 minutes (`H/15 \* \* \* \*`).
4. GitHub webhook trigger for GITScm polling:
    - Set up webhooks in GitHub to trigger builds upon commits.
    - Jenkins listens for HTTP POST requests from GitHub.
    - Example: Configure GitHub webhook to notify Jenkins on every push event.
  5. Poll SCM:
    - Periodically checks the SCM for changes and triggers a build if new commits are detected.
    - Schedule polling duration using cron syntax.
    - Example: Poll SCM every 5 minutes (`H/5 \* \* \* \*`).

---

By understanding and effectively utilizing these build triggers, you can automate your Jenkins jobs based on various events and schedules, improving your CI/CD pipeline's efficiency and responsiveness.

### **Post-build Actions in Jenkins:**

Post-build actions are tasks performed after a build completes. They are crucial for automating processes like artifact archiving, notification sending, and deployment. Here are common post-build actions:

1. Archive the artifacts:
  - Store files generated during the build for future reference or deployment.
  - Example: Archive `.jar` files after a successful Maven build.
2. Send email notifications:
  - Notify relevant parties about build results, successes, failures, or other conditions.
  - Example: Notify the development team upon a failed build.
3. Publish JUnit test result reports:
  - Display test results in a standardized format for analysis and tracking.
  - Example: Publish unit test results to Jenkins for visibility.
4. Deploy artifacts:
  - Automatically deploy built artifacts to staging or production environments.
  - Example: Deploy Docker images to a Kubernetes cluster after a successful build.
5. Trigger other projects:
  - Start downstream projects or pipelines based on build outcomes.
  - Example: Initiate integration tests on a separate Jenkins job upon successful artifact generation.
6. Execute shell scripts:
  - Perform custom actions using shell scripts after the build completes.
  - Example: Run a script to update a database schema after deploying a new version.
7. Publish HTML reports:
  - Display custom HTML reports generated during the build process.
  - Example: Publish code coverage reports generated by Jacoco or Cobertura.

## 8. Collect build metrics:

- Gather and display build-related metrics for monitoring and analysis.
- Example: Track build duration trends over time using Jenkins metrics.

## 9. Publish Cobertura coverage reports:

- Publish code coverage reports generated by the Cobertura plugin for Java projects.
- Example: Display code coverage metrics for unit tests in Jenkins.

## 10. Archive build logs:

- Store build logs for future troubleshooting or analysis.
- Example: Archive console output and error logs after each build.

## 11. Trigger parameterized builds:

- Initiate builds with parameters passed from the current build.
- Example: Trigger a deployment job with environment-specific parameters (e.g., target server URL, credentials).

## 12. Generate HTML artifacts:

- Create HTML-based artifacts for documentation or visualization.
- Example: Generate API documentation using tools like Swagger or Javadoc and publish it as an artifact.

## What is Log files in jenkins?

The log file in Jenkins is a detailed record of events and messages generated during the execution of a build job or task. It includes console output, error messages, timestamps, and build status, providing valuable information for troubleshooting and analysis.

### How to see log file in jenkins:

1. Go to the root (sudo -i)
2. To find the location of your log files use (cd /var/log)
3. List all (ls -lrt)

```
[root@ip-172-31-45-14 ~]# cd /var/log
[root@ip-172-31-45-14 log]# ls -lrt
total 1540
-rw----- 1 root root 0 May 3 23:44 tallylog
-rw-r--r-- 1 root root 193 May 3 23:44 grubby_prune_debug
-rw----- 1 root root 0 May 3 23:45 spooler
drwxr-sr-x+ 3 root systemd-journal 46 May 9 09:25 journal
drwx----- 2 root root 23 May 9 09:25 audit
-rw-r--r-- 1 root root 29355 May 9 09:25 dmesg
drwxr-xr-x 3 root root 17 May 9 09:25 amazon
-rw----- 1 root root 210 May 9 09:25 maillog
-rw-r----- 1 root root 2906 May 9 09:25 cloud-init-output.log
-rw-r--r-- 1 root root 101808 May 9 09:25 cloud-init.log
-rw----- 1 root root 12087 May 10 03:24 boot.log-20240510
drwxr-xr-x 2 root root 68 May 11 00:00 sa
-rw----- 1 root utmp 282624 May 11 02:40 btmp
-rw----- 1 root root 4400 May 11 03:39 boot.log-20240511
drwxr-x--- 2 chrony chrony 276 May 11 03:39 chrony
-rw-rw-r-- 1 root utmp 3456 May 11 14:17 wtmp
-rw-r--r-- 1 root root 292292 May 11 14:23 lastlog
```

## What is 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).

1. In jenkins dashboard, Click new item.
2. Enter the name of the project and select the pipeline and then click ok
3. Write a shot description about your project.
4. Scroll down to the "Pipeline" section.
5. click pipeline script and write pipeline script and save it.
6. Run the project



The screenshot shows the Jenkins web interface. On the left is a sidebar with navigation links: Status, Changes, Console Output (selected), View as plain text, Edit Build Information, Delete build '#2', Restart from Stage, Replay, Pipeline Steps, Workspaces, Previous Build, and Next Build. The main area is titled 'Console Output' with a green checkmark icon. It displays the following log:

```
Started by user faizan khan
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (CIDI test pipeline)
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

## What is Master and slave?

A Jenkins master comes with the basic installation of Jenkins, and in this configuration, the master handles all the tasks for your build system.

If you are working on multiple projects, you may run multiple jobs on each project. Some projects need to run on some nodes, and in this process, we need to configure slaves. Jenkins slaves connect to the Jenkins master using the Java Network Launch Protocol.

### Jenkins Master

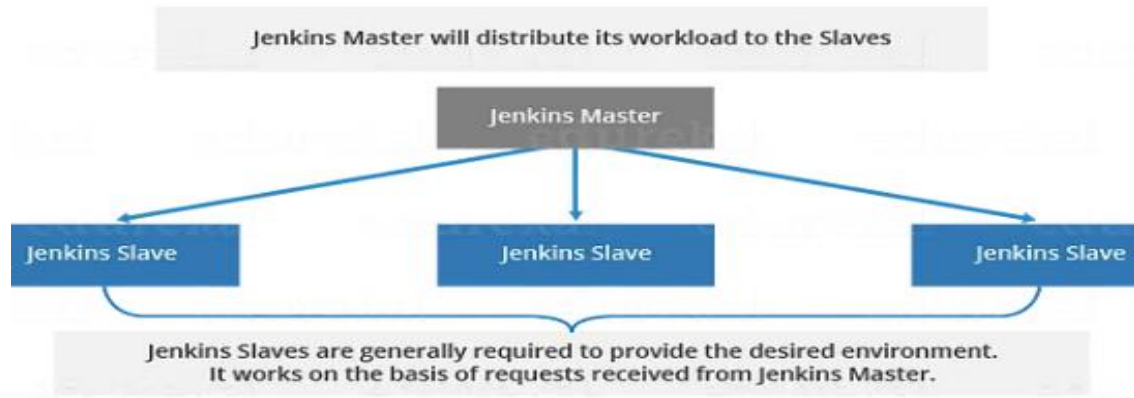
Your main Jenkins server is the Master. The Master's job is to handle:

- 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.



## Steps to Configure Jenkins Master and Slave

1. First create two instance one for Master and one for slave
2. You need to install jdk+Jenkins in master and only jdk in slave
3. Now lets start with the master instance first, go to root: `sudo -i`
4. If you have already installed Jenkins then check its status use: `systemctl status Jenkins` (or) `service Jenkins status`
5. Jenkins status
6. Now give command: → `vim /etc/passwd` ( replace false to bash in the last line )
7. Set password use the command → `passwd Jenkins` ( same password will be given in slave)
8. Now in password authentication change no to yes use the command: `vim /etc/ssh/sshd_config`
9. Now restart sshd use the command: `service sshd restart` (or) `systemctl restart sshd`
10. Check the status of sshd (active-running) : `systemctl status sshd` (or) `service sshd status`
11. Now come to slave isntace, got to root: `sudo -i`
12. Create a user called Jenkins: `useradd Jenkins`
13. Set the same password for slave also: `passwd Jenkins`
14. Give → `visudo` (Now scroll down to root & below that add: `Jenkins ALL=(ALL) NOPASSWD: ALL`)
15. Now in password authentication change no to yes use: `vim /etc/ssh/sshd_config`
16. Now restart sshd: `systemctl restart sshd` (or) `service sshd restart`
17. check the status of sshd (active-running) : `service sshd status` (or) `systemctl status sshd`
18. Login into user use the command: `su - Jenkins`
19. Now install java into slave: `sudo yum install java-17-amazon-corretto-headless`
20. Now go to master instance give: `visudo` (come to root & below root add `Jenkins ALL=(ALL) NOPASSWD: ALL`)
21. `NOPASSWD: ALL`)
22. Now login into user: `su - Jenkins`
23. Give : `ssh-keygen`
24. Now give: `ssh-copy-id jenkins@localhost`
25. Now copy the above command: `ssh 'jenkins@localhost'`
26. Now exit: `exit`
27. Use: `ssh-copy-id jenkins@private ip of slave` ( It will ask for password, enter the password )
28. Now copy the above command: `ssh 'jenkins@172.311.91'`
29. Create a directory: `mkdir workspace`
30. Go to that workspace: `cd workspace`
31. Check the present working directory use: `pwd` (copy the path)
32. Come to Jenkins dashboard
33. Go to manage Jenkins > Nodes
34. Now click on New Node
35. Give some name ex: slave, then click on permanent agent & then create
  - Add some description ex: this is slave, > Select number of executor 2 > in remote root directory paste the path of instance(which was copied) ex: `/home/Jenkins/workspace` > label slave\_node > use this node as much as possible > in launch method select Launch agent via SSH > Host (copy &

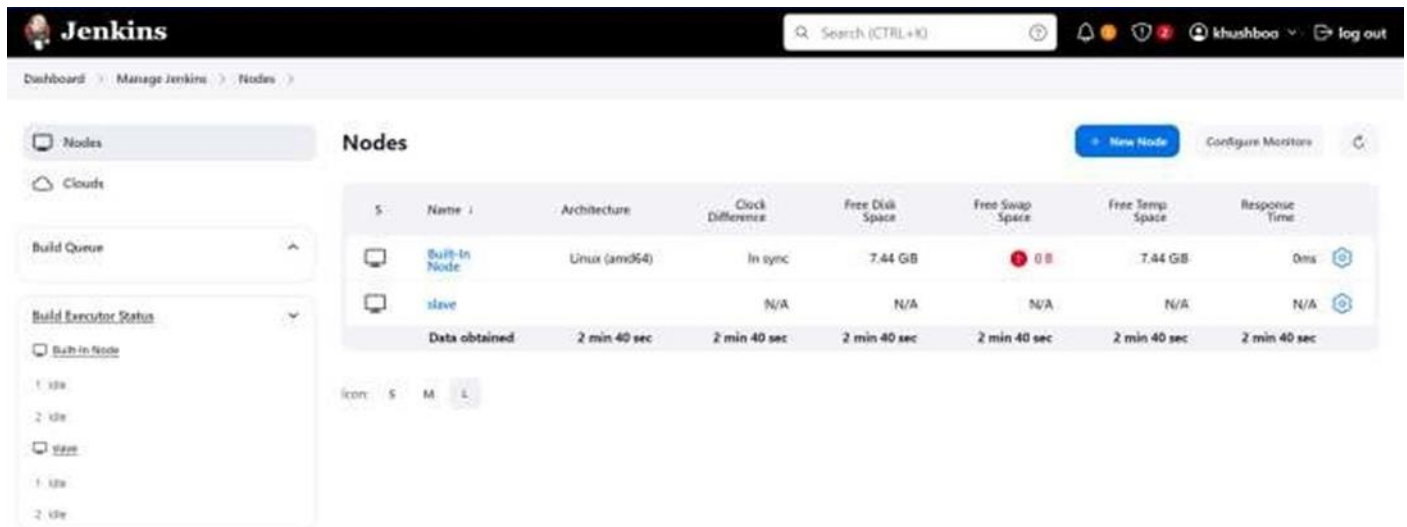


paste the private ip of slave(from instance) > in credentials add Jenkins as username > password same.

36. Now select the added credentials

37. Host key verification select > manually trusted key verification strategy & save it

38. Now you can verify slave on node as well as on the dashboard.



The screenshot shows the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search bar, and user information. The main content area is titled 'Nodes' and features a table with columns: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The table lists two nodes: 'Built-In Node' and 'slave'. The 'Built-In Node' has a status of 'Online' and a response time of '0ms'. The 'slave' node has a status of 'Offline' and a response time of 'N/A'. Below the table, there are buttons for 'New Node', 'Configure Monitors', and 'Refresh'. On the left sidebar, there are links for 'Nodes', 'Clouds', 'Build Queue', and 'Build Executor Status'.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	7.44 GB	0 B	7.44 GB	0ms
	slave		N/A	N/A	N/A	N/A	N/A
Data obtained			2 min 40 sec	2 min 40 sec	2 min 40 sec	2 min 40 sec	2 min 40 sec

## what is maven?

Maven is a powerful build management tool for Java projects to help execute a build life cycle framework. The basis of the Maven is the concept of POM (Project Object Model) in which all configurations can be done with the help of a pom.xml file. It is a file that includes all the project and configuration-related information such as source directory, dependencies with its version, plugins, and builds information, test source directory, etc. A few of the key features of Maven are:

- Maven describes how the project is built. It builds a project using the Project Object Model.
- Maven automatically downloads, update as well as validate the compatibility between dependencies.
- In Maven, dependencies are retrieved from the dependency repository, whereas plugins are retrieved from plugin repositories, so maven keeps proper isolation between project dependencies and plugins.
- Maven can also generate documentation from the source code, compiling and then packaging compiled code into JAR files or ZIP files.

## Steps to Configure Jenkins maven

- First install maven plugin in Jenkins
- Now install the tar file in ec2 : `wget https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.tar.gz`
- Unzip the tar file using `tar -xvzf apache-maven-3.9.6-bin.tar.gz`
- Rename it : `mv apache-maven-3.9.6 maven-3.9`
- Now get into the directory: `cd maven-3.9`



- Now check the path and copy it : pwd
- List it and open bin directory: ls and cd bin
- Now open the file : vi ~/.bash\_profile
- Add these lines in the file :  

```
# .bash_profile
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
# User specific environment and startup programs
M2_HOME=pwd
M2=pwd/bin
PATH=$PATH:$HOME/bin:$M2_HOME:$M2
export PATH
```
- 10. Check maven version: mvn -version
- Now check java path: find / -name java-17\*
- Example of jvm path in ec2 : /usr/lib/jvm/java-17-amazon-corretto.x86\_64
- Go to Jenkins dashboard and open tools
- Now you can select maven project in your New Item for creating maven job
- In build section just add pom.xml and below that add clean package as well as in source code management section select git and add its repository and save it.

## What is GitHub ?

GitHub is a developer platform that allows developers to create, store, manage and share their code. It uses Git software, providing the distributed version control of Git plus access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project.

GitHub essentials are:

- Repositories
- Branches
- Commits
- Pull Requests

**Repositories:** A repository (or "repo") is a central location where code files and related resources for a project are stored. It contains the entire history of the project, including all commits and branches.

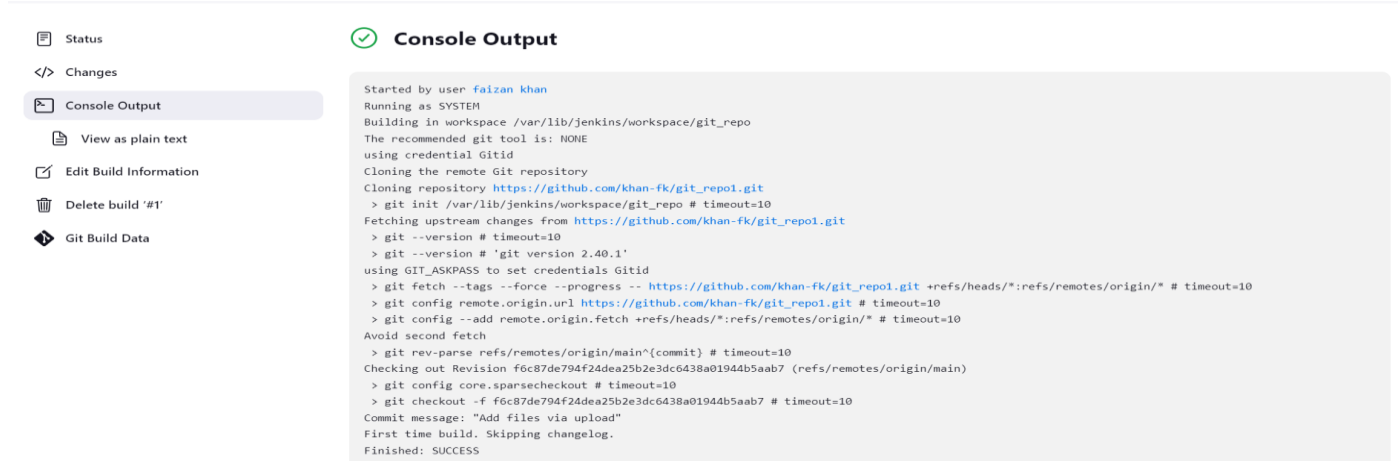
**Branches:** A branch is a parallel version of a repository's codebase. It allows developers to work on separate features or changes without affecting the main codebase. Branches can be created, merged, and deleted as needed.

**Commits:** A commit is a snapshot of changes made to the codebase at a specific point in time. Each commit represents a set of changes (such as additions, deletions, or modifications) made to one or more files. Commits include a message describing the changes made.

**Pull Requests:** A pull request (PR) is a request to merge changes from one branch (often a feature branch) into another (often the main branch, such as "master" or "main"). Pull requests facilitate code review, discussion, and collaboration among team members before changes are merged into the main codebase.

## Integration Of GitHub in Jenkins:

- Create a GitHub account
- Create a git repository
- Add sum files into git repository
- Run command “yum install git -y” in EC2 instance
- Open jenkins
- open manage jenkins -> tool -> git -> name (your option) path(git)
- save and apply
- create a free style job in jenkins
- description
- source code mangement -> select git enter in git reposi path and set the credentials
- in build trrigers select fourth option "github hook trriger for GIT SCM polling"
- go to github location
- open settings -> webhooock -> enter the jenkins path ex:- <http://13.4.5.6:8080/github-webhooock/>->save
- go to jenkins and click buld now
- ofter update delete anything your git hub code it will be automatically updated.



```
Status
</> Changes
Console Output
View as plain text
Edit Build Information
Delete build '#1'
Git Build Data

Console Output
Started by user faizan khan
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/git_repo
The recommended git tool is: NONE
using credential GitId
Cloning the remote Git repository
Cloning repository https://github.com/khan-fk/git_repo1.git
> git init /var/lib/jenkins/workspace/git_repo # timeout=10
Fetching upstream changes from https://github.com/khan-fk/git_repo1.git
> git --version # timeout=10
> git --version # 'git version 2.40.1'
using GIT_ASKPASS to set credentials GitId
> git fetch --tags --force --progress -- https://github.com/khan-fk/git_repo1.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/khan-fk/git_repo1.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision f6c87de794f24dea25b2e3dc6438a01944b5aab7 (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f f6c87de794f24dea25b2e3dc6438a01944b5aab7 # timeout=10
Commit message: "Add files via upload"
First time build. Skipping changelog.
Finished: SUCCESS
```

## Advantages and Disadvantages of 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.

## **Interview Questions**

### **1. What is Jenkins?**

Jenkins is an open-source automation server used for continuous integration and continuous delivery (CI/CD) of software projects.

### **2. What is Continuous Integration (CI)?**

Continuous Integration is a development practice where developers frequently integrate code changes into a shared repository, triggering automated builds and tests to detect and fix integration errors early.

### **3. What is Continuous Delivery (CD)?**

Continuous Delivery is a development practice where software is automatically built, tested, and deployed to production environments in a consistent and repeatable manner, enabling rapid and reliable software delivery.

### **4. What is a Jenkins pipeline?**

A Jenkins pipeline is a scriptable and extensible way to define the entire software delivery process as code, including stages, steps, conditions, and integrations with external tools and services.

### **5. What is the difference between a Jenkins freestyle project and a pipeline?**

A Jenkins freestyle project allows users to configure build jobs using a web-based interface, while a Jenkins pipeline allows users to define build workflows as code using a Jenkinsfile.

### **6. What is a Jenkinsfile?**

A Jenkinsfile is a text file that defines the entire pipeline workflow as code. It contains the script that describes the stages, steps, and conditions of the pipeline.

## **7. How do you trigger a Jenkins job?**

Jenkins jobs can be triggered manually by users or automatically based on events such as code commits, timer triggers, webhook notifications, or other build dependencies.

## **8. What is a Jenkins agent?**

A Jenkins agent, also known as a node or slave, is a worker machine that executes build jobs as instructed by the Jenkins master.

## **9. How do you configure a Jenkins agent?**

Jenkins agents can be configured using the Jenkins web interface by adding them as nodes in the Jenkins configuration, specifying connection details such as hostname, credentials, and launch method.

## **10. What is a Jenkins master?**

A Jenkins master is the central server that manages the entire Jenkins instance. It schedules and coordinates build jobs, monitors agents, and provides a web-based user interface for configuring and monitoring builds.

## **11. How do you install Jenkins plugins?**

Jenkins plugins can be installed using the Jenkins Plugin Manager, accessible from the Jenkins web interface. Users can browse, search, install, update, and manage plugins directly from the Plugin Manager.

## **12. What is Jenkins Job DSL?**

Jenkins Job DSL (Domain Specific Language) is a Groovy-based scripting language that allows users to define Jenkins jobs programmatically. It provides a way to generate and manage Jenkins job configurations dynamically.

## **13. What is the Jenkins Script Console?**

The Jenkins Script Console is a built-in feature that allows users to execute arbitrary Groovy scripts directly on the Jenkins master or agents. It provides a way to perform administrative tasks, troubleshoot issues, or interact with the Jenkins API.

## **14. How do you secure Jenkins?**

Jenkins can be secured by enabling authentication, authorization, and other security features such as role-based access control (RBAC), SSL encryption, and network segmentation. Users can configure security settings in the Jenkins global configuration.

## **15. What is Jenkinsfile Declarative Pipeline?**

Jenkinsfile Declarative Pipeline is a more structured and opinionated way to define pipelines in Jenkins. It provides a higher-level syntax for defining pipeline stages, steps, and post-actions, making pipelines easier to read, write, and maintain.

### **16. How do you define a Jenkins pipeline stage?**

Jenkins pipeline stages are defined using the stage directive in a Jenkinsfile. Each stage represents a logical step in the pipeline workflow, such as build, test, deploy, or notify.

### **17. What is the purpose of the Jenkins Pipeline Syntax?**

The Jenkins Pipeline Syntax provides a set of predefined steps and directives that users can use to define pipeline workflows in a Jenkinsfile. It simplifies the process of writing pipeline scripts by offering reusable building blocks for common tasks.

### **18. How do you archive artifacts in a Jenkins pipeline?**

Artifacts can be archived in a Jenkins pipeline using the archiveArtifacts step. This step allows users to specify files or directories to be archived as build artifacts, making them available for later use or inspection.

### **19. What is the purpose of the Jenkins pipeline input step?**

The Jenkins pipeline input step allows users to pause the execution of a pipeline and wait for manual input from a user before proceeding. It provides a way to interactively control and validate pipeline execution.

### **20. How do you trigger downstream jobs in a Jenkins pipeline?**

Downstream jobs can be triggered in a Jenkins pipeline using the build step or the build keyword. This allows users to start other jobs as part of the pipeline execution, chaining multiple jobs together.

### **21. What is Jenkins Blue Ocean?**

Jenkins Blue Ocean is a user interface designed to simplify the creation, visualization, and management of Jenkins pipelines. It provides a modern and intuitive interface for building, monitoring, and visualizing CI/CD workflows.

### **22. How do you define environment variables in a Jenkins pipeline?**

Environment variables can be defined in a Jenkins pipeline using the environment directive in the pipeline script. This allows users to specify key-value pairs that are available to all steps within the pipeline.

### **23. What is Jenkins Shared Libraries?**

Jenkins Shared Libraries allow users to define reusable code and functions that can be shared across multiple Jenkins pipelines. They provide a way to encapsulate common logic, reduce duplication, and maintain consistency in pipeline scripts.

#### **24. How do you trigger a Jenkins pipeline from a webhook?**

Jenkins pipelines can be triggered from webhooks using the Generic Webhook Trigger plugin or by configuring Jenkins to listen for incoming HTTP requests and trigger pipeline builds based on predefined conditions or criteria.

#### **25. What is Jenkins Matrix Project?**

Jenkins Matrix Project is a project type that allows users to define and execute multiple configurations of a build job in parallel, using different combinations of axes such as operating systems, JDK versions, or build parameters.

#### **26. How do you configure Jenkins for distributed builds?**

Jenkins can be configured for distributed builds by setting up multiple agents (nodes) to handle build jobs. Agents can be configured to run on different machines or environments, allowing Jenkins to distribute workload and scale builds horizontally.

#### **27. What is Jenkinsfile Scripted Pipeline?**

Jenkinsfile Scripted Pipeline is a traditional way to define pipelines in Jenkins using imperative Groovy scripts. It provides more flexibility and control over the pipeline execution but may be more complex and verbose compared to Declarative Pipeline syntax.

#### **28. How do you parallelize stages in a Jenkins pipeline?**

Stages can be parallelized in a Jenkins pipeline using the parallel directive. This allows users to execute multiple stages concurrently, improving pipeline execution time and resource utilization.

#### **29. What is Jenkins Pipeline Syntax Generator?**

Jenkins Pipeline Syntax Generator is a built-in feature that allows users to interactively generate pipeline syntax snippets for common pipeline steps and directives. It provides a visual interface for exploring and experimenting with pipeline syntax.

#### **30. How do you trigger downstream pipelines from a Jenkins pipeline?**

Downstream pipelines can be triggered from a Jenkins pipeline using the build step or the build keyword, similar to triggering downstream jobs. This allows users to start other pipelines as part of the pipeline execution.

#### **31. What is Jenkins Job DSL Plugin?**

Jenkins Job DSL Plugin allows users to define Jenkins jobs programmatically using Groovy-based scripts. It provides a way to generate and manage job configurations dynamically, making it easier to maintain and scale Jenkins configurations.

### **32. How do you define a Jenkins pipeline agent?**

Jenkins pipeline agent can be defined using the agent directive in a Jenkinsfile. This directive specifies where the pipeline will execute, such as on the Jenkins master or on a specific agent node.

### **33. What is Jenkins Blue Ocean Pipeline Editor?**

Jenkins Blue Ocean Pipeline Editor is a visual editor that allows users to create, edit, and visualize Jenkins pipelines in a graphical interface. It provides a drag-and-drop interface for building and configuring pipeline stages and steps.

### **34. How do you publish artifacts in a Jenkins pipeline?**

Artifacts can be published in a Jenkins pipeline using the archiveArtifacts step or the stash step. These steps allow users to specify files or directories to be archived or stored as build artifacts, making them available for later use.

### **35. What is Jenkins Pipeline Unit Testing?**

Jenkins Pipeline Unit Testing is a practice of testing Jenkins pipeline scripts (Jenkinsfiles) using automated unit tests. It allows users to validate pipeline logic, syntax, and behavior in isolation, improving pipeline reliability and maintainability.

### **36. How do you handle errors and exceptions in a Jenkins pipeline?**

Errors and exceptions can be handled in a Jenkins pipeline using try-catch blocks or the catchError step. This allows users to gracefully handle errors, log exceptions, and perform cleanup tasks in case of failures.

### **37. What is Jenkins Pipeline Script Approval?**

Jenkins Pipeline Script Approval is a security feature that controls the execution of unsafe or untrusted Groovy scripts in Jenkins pipelines. It requires administrators to review and approve scripts before they can be executed.

### **38. How do you define custom stages in a Jenkins pipeline?**

Custom stages can be defined in a Jenkins pipeline using the stage directive. This allows users to create named stages and organize pipeline steps into logical units, making the pipeline script more readable and maintainable.

### **39. What is Jenkins Pipeline Multibranch?**

Jenkins Pipeline Multibranch is a project type that automatically creates a separate Jenkins pipeline for each branch in a source code repository. It allows users to define and execute separate pipelines for feature branches, pull requests, or branches with different configurations.

#### **40. How do you trigger a Jenkins pipeline from a Git webhook?**

Jenkins pipeline can be triggered from a Git webhook by configuring Jenkins to listen for incoming webhook notifications from the Git repository. This allows Jenkins to automatically start pipeline builds in response to code changes.

#### **41. What is Jenkins Pipeline Shared Libraries?**

Jenkins Pipeline Shared Libraries are reusable code and functions that can be shared across multiple Jenkins pipelines. They provide a way to encapsulate common logic, reduce duplication, and maintain consistency in pipeline scripts.

#### **42. How do you define post-actions in a Jenkins pipeline?**

Post-actions in a Jenkins pipeline can be defined using the post directive. This allows users to specify actions to be executed after the pipeline stages have completed, such as sending notifications, publishing reports, or cleaning up resources.

#### **43. What is Jenkins Pipeline Syntax Generator?**

Jenkins Pipeline Syntax Generator is a built-in feature that allows users to interactively generate pipeline syntax snippets for common pipeline steps and directives. It provides a visual interface for exploring and experimenting with pipeline syntax.

#### **44. How do you trigger downstream pipelines from a Jenkins pipeline?**

Downstream pipelines can be triggered from a Jenkins pipeline using the build step or the build keyword, similar to triggering downstream jobs. This allows users to start other pipelines as part of the pipeline execution.

#### **45. What is Jenkins Job DSL Plugin?**

Jenkins Job DSL Plugin allows users to define Jenkins jobs programmatically using Groovy-based scripts. It provides a way to generate and manage job configurations dynamically, making it easier to maintain and scale Jenkins configurations.

#### **46. How do you define a Jenkins pipeline agent?**

Jenkins pipeline agent can be defined using the agent directive in a Jenkinsfile. This directive specifies where the pipeline will execute, such as on the Jenkins master or on a specific agent node.

#### **47. What is Jenkins Blue Ocean Pipeline Editor?**



Jenkins Blue Ocean Pipeline Editor is a visual editor that allows users to create, edit, and visualize Jenkins pipelines in a graphical interface. It provides a drag-and-drop interface for building and configuring pipeline stages and steps.

#### **48. How do you publish artifacts in a Jenkins pipeline?**

Artifacts can be published in a Jenkins pipeline using the `archiveArtifacts` step or the `stash` step. These steps allow users to specify files or directories to be archived or stored as build artifacts, making them available for later use.

#### **49. What is Jenkins Pipeline Unit Testing?**

Jenkins Pipeline Unit Testing is a practice of testing Jenkins pipeline scripts (Jenkinsfiles) using automated unit tests. It allows users to validate pipeline logic, syntax, and behavior in isolation, improving pipeline reliability and maintainability.

#### **50. How do you handle errors and exceptions in a Jenkins pipeline?**

Errors and exceptions can be handled in a Jenkins pipeline using try-catch blocks or the `catchError` step. This allows users to gracefully handle errors, log exceptions, and perform cleanup tasks in case of failures.