**Jenkins: JENKINS INRODUCTION**

* **Following the principles of continuous integration**. **First** the developers will check out the code into a private repository and then they will commit the changes. After they commit the changes those changes will be available to the entire team. Then the **CI server**(**continuous integration server)**, will continue monitoring the repository and as soon as there's a change in the contents of the repository the CI server will check out the changes that are identified in the repository. When the changes are identified, or when there is a change, it triggers an activity and it goes to the continuous integration server to **invoke build**, **run unit**, and **integration test** on the changes.
* **Facilitate continuous integration**. Some of them are **Jenkins**, **TeamCity**, and **Bamboo**. And Jenkins is an improvised version of **Hudson**. These products can help us **plan the build, releases, deployment, and the overall integration processes.**
* **Jenkins** is basically a software which can enable these features and automate all these tasks in order to ensure that we are implementing the principles or practices of DevOps. Jenkins is **open source** so we get the flexibility of tuning it as per our requirements. And Jenkins is **built using Java**, so it supports numerous open-source communities and understands various tool kits, tool sets, or frameworks that were introduced for **Java,** **specifically for testing and other elements**.
* Another important capability of Jenkins is that it is **language neutral**. The fundamental objective of Jenkins is to **provide continuous integration capability to the codes regardless of the languages they were written in**. Jenkins can also be used for continuous deployment and delivery because Jenkins supports various plugins for all the deployment environments, whether it's **Docker, Kubernetes, or even Cloud**. Another important objective that Jenkins fulfills is accelerated software delivery through its integration capabilities. And **Jenkins** is one of the most widely used tools today for continuous integration and deployment.
* **(FEATURES)** With **Jenkins**, we get a simplified administration environment because administrators can be facilitated in Jenkins using GUI. Jenkins can also facilitate the capability of managing a team which is capable of distributed development. Jenkins is secure because it follows the approach of role-based authorization. Apart from these, the architecture of Jenkins ensures that it is highly scalable. Jenkins offers optimum performance due to its scalable nature and it also provides support for various container frameworks, like **Docker**. Finally, Jenkins also supports high availability because it is built on top of **Java**.
* Includes distributed team management, role-based access control, simplified administration, support for lightweight containers, optimized performance, scalability with the use of the right architecture, and advanced security where we can integrate it with LDAP.
* The **first one** is the **master slave architecture**. In this architecture, there will be a master followed by the slaves. The master will take responsibility of distributing the jobs to be executed to various slaves and these slaves will take the responsibility of producing the right report. In the case of testing, it will produce a test report. The **second architecture** is the **master master architecture**, where the masters will take the responsibility of sharing slaves between them. They not only share the slaves, but they also share security. So in case one master fails, we will not have a single point of failure because the other master is also aware of all the other slaves.
* With **Jenkins 2.0**, we now **get built-in pipeline support**, so we no longer need to provision any extra plugins.
* The first tool is **NPM** and this tool is used to build Node.js based applications. Then there's **Git**, which is the common repository. The third one is **Maven**, which is a build tool.
* We will also be installing and provisioning an **Eclipse IDE** using which we'll build applications. We can also use **plugins in Eclipse** so we can automatically submit the jobs to trigger in the Jenkins pipeline.
* **Junit**, **TestNG**, and **Selenium** are some tools that can be used for testing.
* **Docker** is a suitable deployment environment.
* **(STACK TOOLS)** We can **provision databases of various types** and they include **MySQL, MariaDB, Hibernate, and Spark SQL**. Our **applications can be provisioned** in **Apache**, **JBoss**, and **Spring** to name a few. There are certain **middleware tools** that we can use like **ActiveMQ** and **Camel** for **integration**, **for security like OpenSSL**. We can use various build tools like **Maven**, **Puppet Stack**, and **Jenkins** to handle the responsibility of continuous integration. And finally, we can use certain **monitoring tools** like **Nagios** to facilitate continuous monitoring. So, when we put all these tools together, it'll **help us implement continuous integration, monitoring, testing, deployment, and delivery of our products.**
* **Eclipse IDE**, will help developers write code and develop applications. We've installed **Maven**, which is the build and we've also installed **Git**, which is the source code management system.
* Now **jobs** can be classified into different categories. Using **Freestyle project**, we can build projects or jobs using triggers from any source code management system and build system. The second classification is **Pipeline**. We can use this project or job type when we need to orchestrate complex build tasks spanning across multiple slaves. **Multi-configuration project** where we're provided with a number of different types of configurations that we can apply on different destinations. For example, we want to test our code across different destinations. In such case, we'll use Multi-configuration project. When we want to build containers to store our nested items, we can use **Folder**. **GitHub project** provides the capability of binding our repository with this particular Jenkins. [**GitHub Organization from the list of projects**.] **Multibranch Pipeline**, we can create pipeline projects according to branches within our **SCM or source code management system**.
* [**Jenkins Build Steps**.] Jenkins follows three-cyclical steps that involve **prebuild**, **build**, and **postbuild actions**. The first step involves all the **prebuild actions**. Here we can use JS minimizers to minimize the code, compress the code, and then push it to build action. The second step involves the **build action**. And, in this step, we typically get the codes from **prebuild**, compile them, and test the codes. The final step is all about **postbuild actions**, where we focus on communication tasks.
* Jenkins will begin by pushing the codes. And, when the codes are being pushed, it will begin pulling to that particular SVN or repository where the codes are being committed. And it will invoke the execution of the job. Now, after the codes have been committed, they will go through the compilation phase. And, after the codes have been compiled, they will be packaged. And then finally, they will be published.
* The code is pushed to GitHub, and the Jenkins continuous integration server polls the GitHub for changes. After it identifies certain changes, it goes and polls Jenkins to see if GitHub job is running, following which it will trigger the pipeline.
* After it triggers the pipeline, Jenkins will start the build job processes. Then the pipeline will take the responsibility of performing all the tasks, like running Jenkins along with various other elements on Jenkins and it will produce the artifacts. After the artifacts have been produced, we'll deploy the image to the production environment or various other environments.
* The role of Jenkins is integration. And so whenever there is a new tool, software, or even frameworks, Jenkins will be constructing plugins for them.
* Now let's explore some of the important plugins that can play critical roles when we want to set up a proper Jenkins architecture. The first one is **LDAP plugin** this plugin is specifically meant for securing the Jenkins environment. The second important plugin is the **Mailer plugin** using this plugin, we'll be able to e-mail reports that we generate from Jenkins. The third important plugin is the **Docker** common plugin this plugin provides the capability of deploying our pipeline to the Docker container. Finally, there's the **PAM authorization plugin** we can use this plugin to facilitate authorization and authentication processes whenever other users need to access Jenkins.

**PLUGINS CLASSIFICATION**

* Exploring four important categories of plugins. The first category is the **source code management plugins**. Within this category, there are various Jenkins supported plugins like **Bitbucket, CVS, GitHub, Git, Clearcase**. **Build** is an important task in Jenkins. Some of the build tools include **Ant, Maven, Gradle, Node JS, and MSBuild**.
* Some of the **distributed build tools** include **Matrix Project, SSH slaves, and Windows slaves** we can use the plugins that support these tools to ensure that we're able to install slaves on different boxes. And finally, there's another important tool set that we can use for reporting and collaboration. And we term this category as **build analysis and reporting**. Some of these tools include **Checkstyle, Cobertura, Junit, and xunit**. [**Htmlpublisher and Warnings are the other Build analysis and reporting tools**.] We can use these tools to help us improvise the way reports are generated and sent to the developers for further improvisation.
* The first category of plugins are used by **.NET developers**. For example, they'll be using this **MSBuild plugin** to ensure the codes are built in a common repository. Then the **MSTest plugin** generates various test reports for the applications that are written using .NET. Another category of plugins is the **Agent Launchers and Controllers** these plugins are important when we want to manage infrastructures. For example, if we want to copy certain jobs to a slave, we'll be using this **Copy To Slave plugin**. And, if we want to execute the job dynamically in Amazon, Instances, or Containers, we'll be using this **Amazon EC2 Container Service plugin**.
* **JClouds plugins** will help us build cloud services. Now, if we want to communicate, execute, and build our jobs on OpenStack Notes, we can use the **Openstack Cloud plugins**. There's a whole host of plugins for mobile application development listed under the category **Android Development**. Some of the prominent plugins are **Appetize.io and Android Lint**. Android Lint can help us create content, parse, and then display the results for analysis on the Jenkins dashboard.
* Under the **tab Installed**. The first plugin is the **Ant Plugin** which provides Apache Ant support to Jenkins. To facilitate better build management, there are certain plugins like the Branch **API Plugin** which helps us manage multiple branches like development, testing, and some others. Then there's **Build Timeout** which takes the responsibility of automatically terminating the build in case it takes longer than what we've specified for it.
* if we want to execute our plugin on Docker containers, we can use the **Docker Pipeline**. And, if we want to send e-mails from Jenkins to specify the current status of job execution, we can use the **Email Extension Plugin**. Also there's a plugin for **Git** which is basically used for source code management and they're typically used with **Maven**. **Gradle PLUGIN** is an enhanced version of **Maven**. With Gradle, all the scripting capabilities are minimized as compared to Maven. **Jackson 2 API Plugin** is used in order to help us parse, read, or construct JSON APIs. So, if we happen to be working heavily with JSON data, we can use this plugin.
* To enable and configure security in Jenkins, there are certain plugins like the **LDAP Plugin**, which can help us implement LDAP to manage authentication and authorizations. Then there's **PAM Authentication plugin** used to implement pluggable authentication modules. In order to secure our application, we can use **OWASP Markup Formatter Plugin**.
* For example, if we want to build and execute pipelines that are capable of understanding scripts written in Groovy, we can use the **Pipeline: Groovy plugin**. If we want some sort of human intervention to kickstart or control the execution of the scripts, we can use the **Pipeline: Input Step plugin**.
* We need **build tools** and we need certain **plugins** that can integrate these build tools with Jenkins. We also need **destination plugins** that help us execute jobs across different slaves running on different types of containers including the Cloud.

# Jenkins: Creating Application Builds

**Different Job types:**

1. **Freestyle**

* General Purpose
* Maximum Flexibility
* Can use any stack of SCM or software

1. **Maven Project**

* Builds maven
* POM centric

1. **External Job**

* Record execution of processes outside Jenkins
* Preferentially used as dashboard for automation

1. **Multiconfiguration Job**

* Same job in multiple configurations
* Can use different databases/machines

**Jobs can be executed in:**

1. Sequence

* where the jobs will be executed in sequence. It means that we will need to complete one job before we can start another job.

1. Parallel

* Then we have parallel job execution where we have multiple jobs running simultaneously. We can plan and decide the type of job, execution environment, or the type of execution that we want to adopt in Jenkins accordingly.

The Jenkins job execution begins with certain pre-built tasks. And the job execution steps are sequential in nature. After the pre-built tasks are completed, the main job will be invoked. And after the main job has been executed, Jenkins will execute the post-build steps or tasks. When the job goes through this cycle of execution steps, information is enriched while the job goes through each steps. So we are building a job chain with these sequential execution steps. And such, job chain is termed as pipeline.

Important triggers to initiate jobs

1. Manual Bind
2. Scheduled build
3. Code Change

* means that whenever there is a change in the code base, Jenkins will kickstart and initialize the job.

1. Polling SCM

* source code management tool, which will poll our SCM, and as soon as there is a conflict of code, it will initialize and execute the build job.

1. Trigger on other build cycles

* whenever we initiate a new build, which depends on the build cycle of other builds, we can set the trigger based on those build cycles as well.

**Job Parameters**

1. Project type

* Freestyle
* Maven
* Flow
* Pipeline
* External Job

1. Concurrent

* Enable/disable concurrent execution

1. Node

* Location where job will run

1. Retry Count

* Jenkins retry attempts in case of failure

1. Quiet Period

* Number of seconds to wait between consecutive runs

1. Auth Token

* Authentication token that allows new build

1. Logrotate

* Manage history