Continuous Integration with Jenkins

By
ABHISHK K L
Assistant Professor
Dept Of MCA
RIT, Bengaluru

Continuous Integration with Jenkins

Continuous Integration (CI) is a DevOps software development practice that enables the developers to merge their code changes in the central repository to run automated builds and tests. It refers to the process of automating the integration of code changes coming from several sources.

Features of Continuous Integration

Following are some of the main features or practices for Continuous Integration.

1. **Maintain a single source repository**
2. **Automate the build**
3. **Make your build self-testing**
4. **Every commit should build on an integration machine**
5. **Keep the build fast**
6. **Everyone can see what is happening**
7. **Automate deployment**

Need /Importance of Continuous Integration

1. Reduces Risk
2. Better Communication
3. Higher Product Quality
4. Reduced Waiting Time

What is Jenkins ?

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes and used to build and test software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

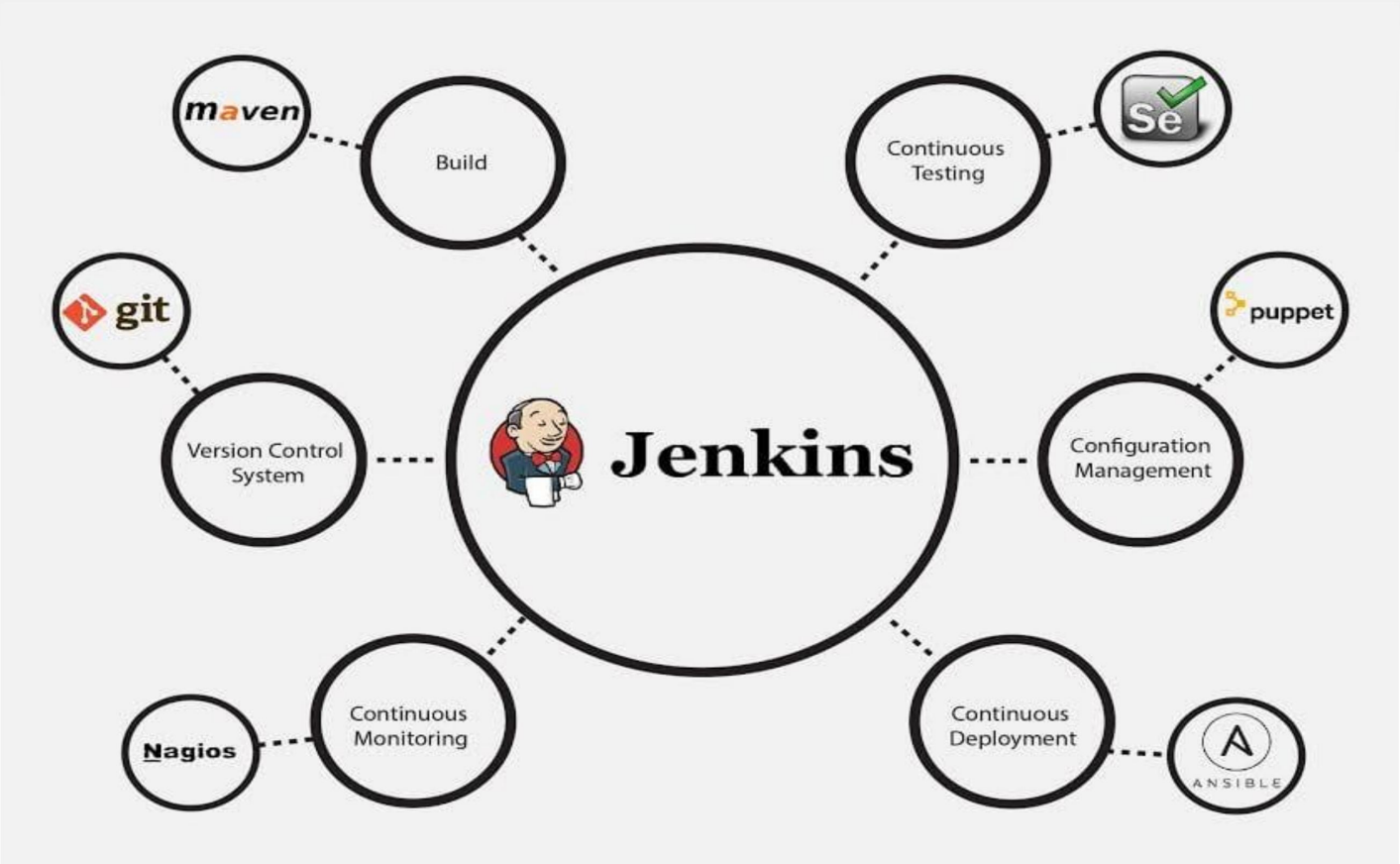
History of Jenkins

- Kohsuke Kawaguchi working at SUN Microsystems, was tired of building the code and fixing errors repetitively
- In 2004, he created an automation server called Hudson that automates build and test task.
- In 2011, Oracle who owned Sun Microsystems had a dispute with Hudson open source community, so they forked Hudson and renamed it as Jenkins.

Jenkins Features

1. Easy Installation
2. Easy Configuration
3. Available Plugins
4. Extensible
5. Easy Distribution
6. Free Open Source

Why we use Jenkins?

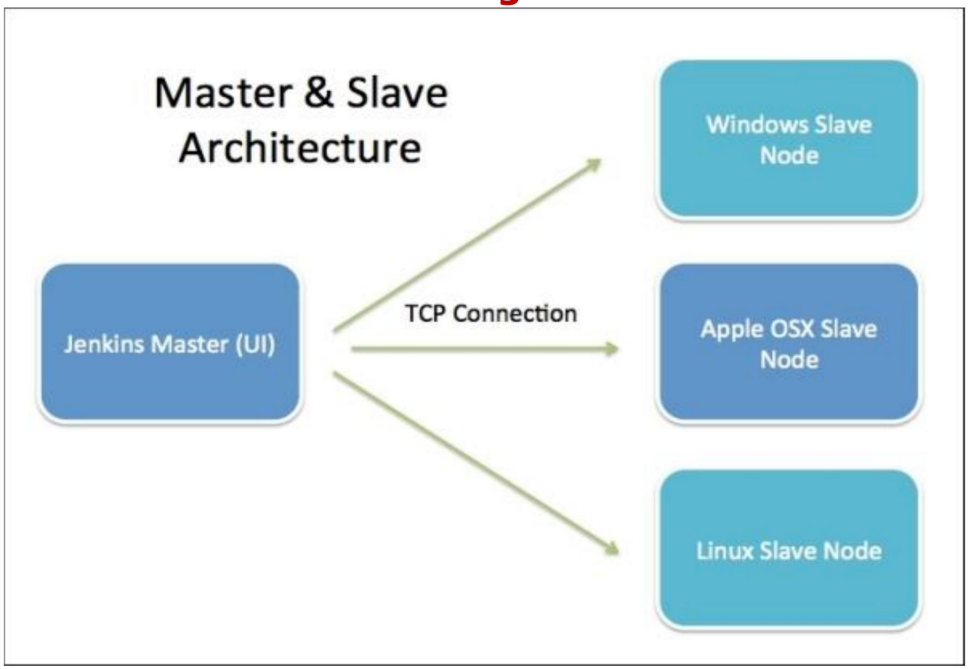


Jenkins Architecture

Jenkins architecture has two components:

1. **Jenkins Master/Server**
2. **Jenkins Slave/Node/Build Server**

In the below image, the Jenkins master is in charge of the UI and the slave nodes are of different OS types.



Jenkins Architecture----------Jenkins Master

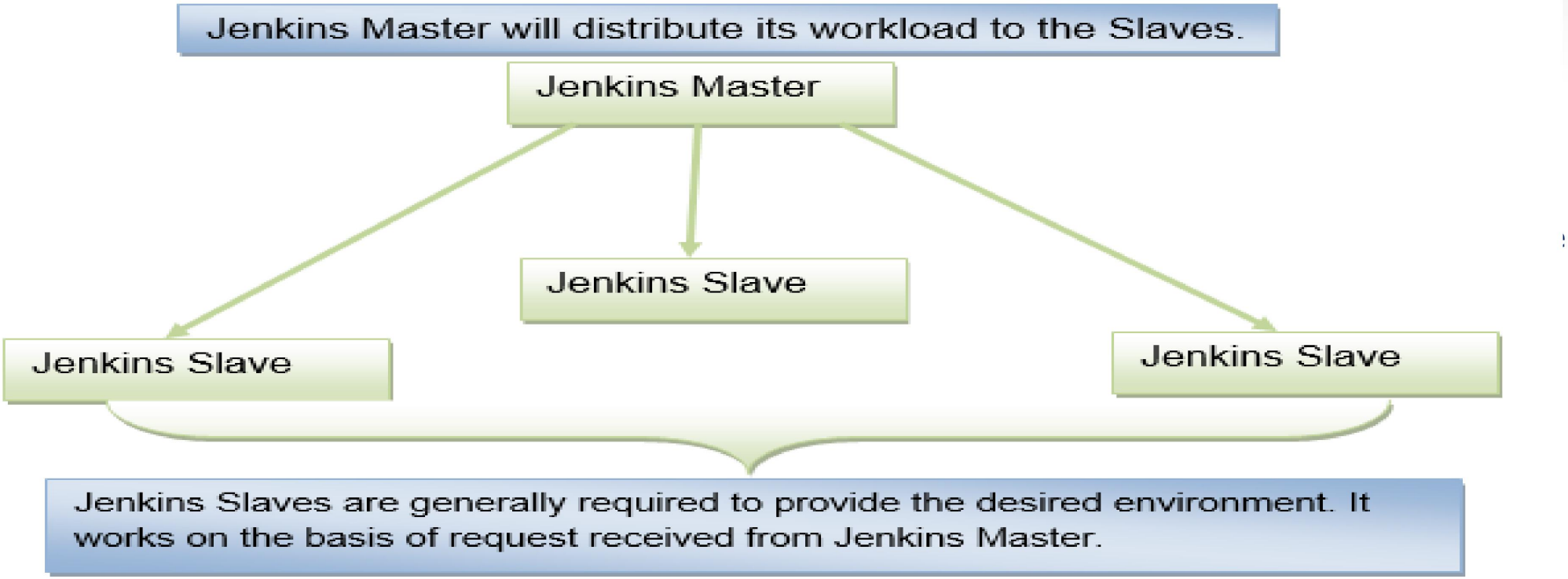
- The main server of Jenkins is the Jenkins Master.
- It is a web dashboard(UI) which is nothing but powered from a war file. By default it runs on 8080 port.
- With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave.
- By default one node (slave) is configured 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:

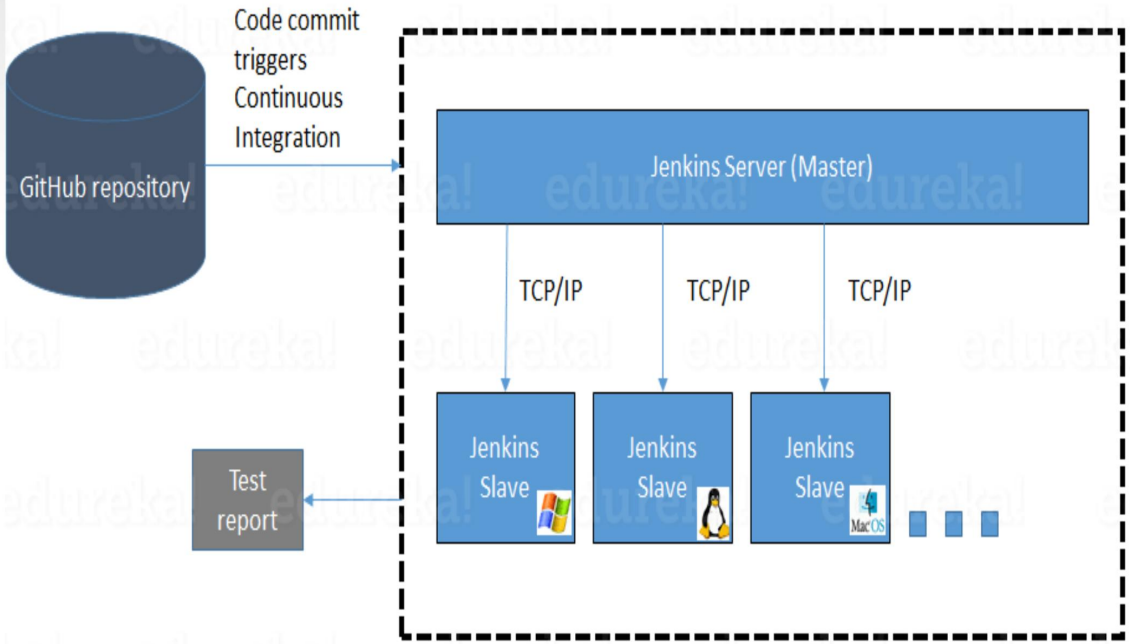
1. Scheduling build jobs.
2. Dispatching builds to the nodes/slaves for the actual execution.
3. Monitor the nodes/slaves (possibly taking them online and offline as required).
4. Recording and presenting the build results.
5. A Master/Server instance of Jenkins can also execute build jobs directly.

Jenkins Architecture---------Jenkins Slave

A slave is just a device that is configured to act as an executor on behalf of the master.



Jenkins is used for testing in different environments



The following functions are performed in the above image:

- 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.

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.

OR

Jenkins pipeline is a set of modules or plugins which enable the implementation and integration of Continuous Delivery pipelines within 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 software development workflow.**

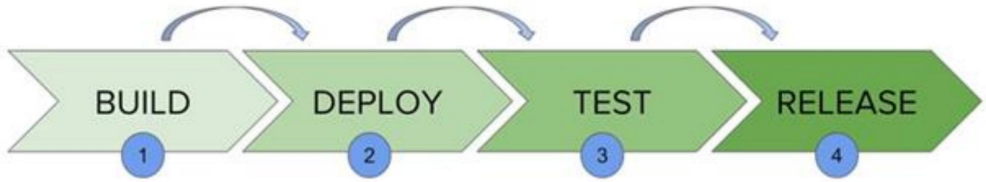
- A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).

What is Continuous Delivery Pipeline?

Continuous delivery is the capability to release software at all times. It is a practice which ensures that the software is always in a production-ready state.

It means that every time a change is made to the code or the infrastructure, the software team must work in such a way that these changes are built quickly and tested using various automation tools after which the build is subjected to production.

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

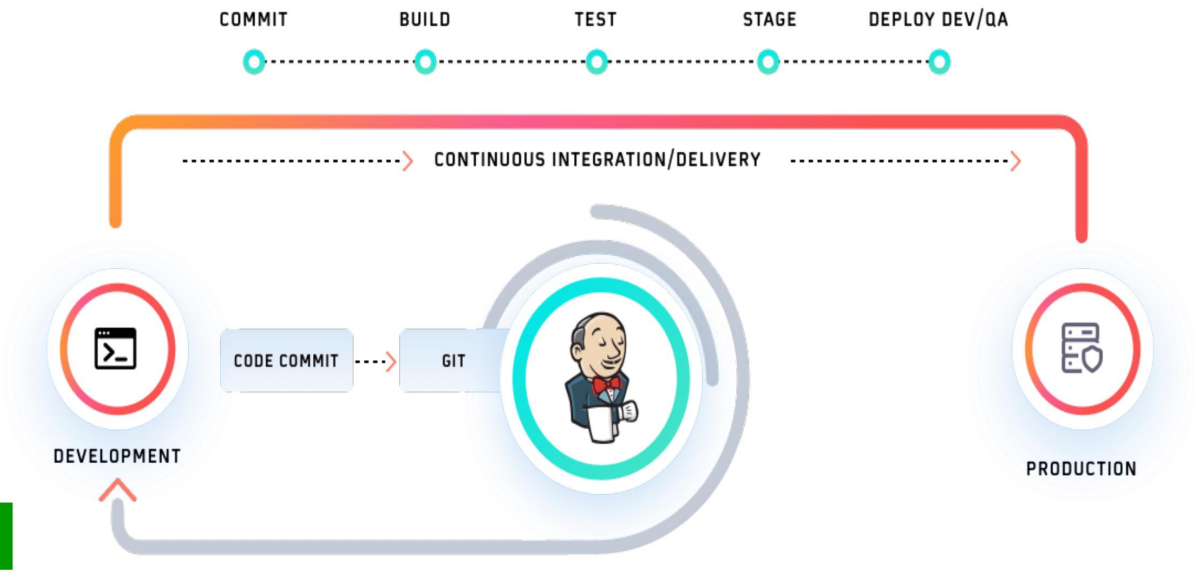


- The above diagram represents a continuous delivery pipeline in Jenkins. It contains a collection of states such as build, deploy, test and release. These jobs or events are interlinked with each other. Every state has its jobs, which work in a sequence called a continuous delivery pipeline.

Continuous Integration with Jenkins

- By speeding up the delivery process, the development team will get more time to implement any required feedback. This process, of getting the software from the build to the production state at a faster rate is carried out by implementing continuous integration and continuous delivery.

- Continuous delivery ensures that the software is built, tested, and released more frequently. It reduces the cost, time, and risk of incremental software releases. To carry out continuous delivery, Jenkins introduced a new feature called Jenkins pipeline



RAMAIAH Institute of Technology

Advantages of Jenkins Pipeline

1. By using Groovy DSL (Domain Specific Language), it models easy to complex pipelines as code.
2. Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
3. It supports complex pipelines by incorporating conditional loops, fork or join operations and allowing tasks to be performed in parallel
4. It is durable in terms of unplanned restart of the Jenkins master
5. The code is stored in a text file called the Jenkinsfile which can be checked into a SCM (Source Code Management)
6. It can integrate with several other plugins

JenkinsFile

A Jenkinsfile is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system. It can be reviewed in a Source Code Management (SCM) platform such as Git. This enables the developers to access, edit and check the code at all times.

- The Jenkinsfile is written using the Groovy Domain-Specific Language and can be generated using a text editor or the Jenkins instance configuration tab.
- **Two** types of **syntax** are used for defining your JenkinsFile.
  1. **Scripted**
  2. **Declarative**

Scripted and Declarative Pipeline syntax

scripted pipeline is a traditional way of writing the code. In this pipeline, the Jenkinsfile is written on the Jenkins UI instance. scripted pipeline is defined within a 'node' block.

- The scripted pipeline uses stricter groovy based syntaxes because it was the first pipeline to be built on the groovy foundation.
- Since this Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax.

Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write.

- This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.
- The declarative pipeline is defined within a 'pipeline' block

Jenkins Pipeline Concepts

Pipeline

This is a user defined block which contains all the processes such as build, test, deploy, etc. It is a collection of all the stages in a Jenkinsfile. All the stages and steps are defined within this block. It is the key block for a declarative pipeline syntax.

```
pipeline {

}
```

Node

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline. It is a key part of the scripted pipeline syntax.

```
node {

}
```

Jenkins Pipeline Concepts

Agent

An agent is a directive that can run multiple builds with only one instance of Jenkins. This feature helps to distribute the workload to different agents and execute several projects within a single Jenkins instance. **It instructs Jenkins to allocate an executor for the builds.**

A single agent can be specified for an entire pipeline or specific agents can be allotted to execute each stage within a pipeline. Few of the parameters used with agents are:

Any
Runs the pipeline/ stage on any available agent.

None
This parameter is applied at the root of the pipeline and it indicates that there is no global agent for the entire pipeline and each stage must specify its own agent.

Label
Executes the pipeline/stage on the labelled agent.

Jenkins Pipeline Concepts

Docker

This parameter uses docker container as an execution environment for the pipeline or a specific stage. In the below example I'm using docker to pull an ubuntu image. This image can now be used as an execution environment to run multiple commands.

```
pipeline {
    agent {
        docker {
            image 'ubuntu'
        }
    }
}
```

Jenkins Pipeline Concepts

Stages

This block contains all the work that needs to be carried out. The work is specified in the form of stages. There can be more than one stage within this directive. Each stage performs a specific task. In the following example, I've created multiple stages, each performing a specific task.

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        ...
                }
                stage ('Test') {
                        ...
                }
                stage ('QA') {
                        ...
                }
                stage ('Deploy') {
                        ...
                }
                stage ('Monitor') {
                        ...
                }
        }
}
```

Jenkins Pipeline Concepts

Steps

A series of steps can be defined within a stage block. These steps are carried out in sequence to execute a stage. There must be at least one step within a steps directive. In the following example I've implemented an echo command within the build stage. This command is executed as a part of the 'Build' stage.

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        steps {
                                echo 'Running build phase...'
                        }
                }
        }
}
```

Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

1. Execute this Pipeline or any of its stages,
   on any available agent.
1. Defines the "Build" stage.
2. Perform some steps related to the "Build" stage.
3. Defines the "Test" stage.\
4. Perform some steps related to the "Test" stage.
5. Defines the "Deploy" stage.
6. Perform some steps related to the "Deploy" stage.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any (1)
    stages {
        stage('Build') { (2)
            steps {
                // (3)
            }
        }
        stage('Test') { (4)
            steps {
                // (5)
            }
        }
        stage('Deploy') { (6)
            steps {
                // (7)
            }
        }
    }
}
```

Scripted Pipeline fundamentals

In Scripted Pipeline syntax, one or more node blocks do the core work throughout the entire Pipeline. a node block does two things:

1. Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an **executor is free on a node, the steps will run**.
2. Creates a workspace (a directory specific to that particular Pipeline)

Jenkinsfile (Scripted Pipeline)

```
node { 1
    stage('Build') { 2
        // 3
    }
    stage('Test') { 4
        // 5
    }
    stage('Deploy') { 6
        // 7
    }
}
```

Example of a JenkinsFile using Declarative Pipeline syntax

1. **pipeline** is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.
2. **agent** is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.
3. **stage** is a syntax block that describes a stage of this Pipeline. stage blocks are optional in Scripted Pipeline syntax.
4. **steps** is Declarative Pipeline-specific syntax that describes the steps to be run in this stage.
5. **sh** is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.
6. **junit** is another Pipeline step (provided by the JUnit plugin) for aggregating test reports.

Jenkinsfile (Declarative Pipeline)

```
pipeline { (1)
    agent any (2)
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') { (3)
            steps { (4)
                sh 'make' (5)
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml' (6)
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Advantages of Jenkins

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.
- All plug-ins are not compatible with the declarative pipeline syntax.
- Sometimes it's very hard to troubleshoot the issues due to improper error handling in plugins.
- Lots of plug-ins have a problem with the updating process.
- Most of the plugins are developed and managed by open-source contributors so they rely on the mercy of collaborators or to develop their own.

THANK YOU

ABHISHEK K L | MCA dept

RAMAIAH
Institute of Technology