*1.*

A valid Declarative pipeline must be defined with the “pipeline” sentence and include the next required sections:

* agent
* stages
* stage
* steps

Also, these are the available directives:

* environment (Defined at stage or pipeline level)
* input (Defined at stage level)
* options (Defined at stage or pipeline level)
* parallel
* parameters
* post
* script
* tools
* triggers
* when

**Agent**

Jenkins provides the ability to perform distributed builds by delegating them to “agent” nodes.

**Stages**

This section allows to generate different stages on your pipeline that will be visualized as different segments when the job is run.

At least one “stage” section must be defined on the “stages” section. It will contain the work that the pipeline will execute.

**Steps**

The last required section is “steps”, which is defined into a “stage”. At least one step must be defined in the “steps” section.

**Environment**

This directive can be both defined at stage or pipeline level, which will determine the scope of its definitions. When “environment” is used at the “pipeline” level, its definitions will be valid for all of the pipeline steps. If instead it is defined within a “stage”, it will only be valid for the particular stage.

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building..'

}

}

stage('Test') {

steps {

echo 'Testing..'

}

}

stage('Deploy') {

steps {

echo 'Deploying....'

}

}

}

}

2.node{

stage(Build){

bat"msbuild ${C:\\Jenkins\\my\_project\\workspace\\test\\my\_project.sln}"

}

stage('Selenium tests'){

dir(automation\_path){//changes the path to “automation\_path”

bat"mvn clean test -Dsuite=SMOKE\_TEST -Denvironment=QA"

}

}

}

This script has the following stages:

* Build stage:
  + bat “msbuild…”: Builds the project by specifying a Visual studio solution file.
* Selenium tests stage:
  + dir(automation\_path): Changes the current directory to the value set on the automation\_path variable.
  + bat “mvn clean test … “: Invokes maven to perform tests specified in the suite “SMOKE\_TEST” and using the values defined on “QA”. Also, the “clean” flag  will clean the build.

**Orchestration** is the automated configuration, coordination, and management of computer systems and software. A number of **tools** exist for automation of server configuration and management ..

Ex: Terraform, ansible, chef etc..

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose.

* Easy Installation
* Continuous Integration
* Continuous Delivery
* Easy Configuration
* Plug-in, Distributed and Extensible

Once you have defined Jenkins give an example, you can refer the below mentioned use case:

* First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull 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 deploys the built in the test server.
* After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
* It will continue to check the  source code repository for changes made in the source code and the whole process keeps on repeating.

To start Jenkins manually open Console/Command line, then go to your Jenkins installation directory. Over there you can use the below commands:

TostartJenkins: **jenkins.exe start**  
TostopJenkins: **jenkins.exe stop**  
To restart Jenkins: **jenkins.exe restart**

**What is a DSL** Jenkins

The Domain Specific Language (DSL) itself that allows users to describe jobs using a Groovy-based language, and a Jenkins plugin which manages the scripts and the updating of the Jenkins jobs which are created and maintained as a result.

**Continuous Integration (CI)** is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

The continuous delivery pipeline is the automated expression of the process for version control from different users or customers.

A **continuous integration** and deployment pipeline (**CD/CI**) is such an important aspect of a software project. It saves a ton of manual, error-prone deployment work. It results in higher quality software for continuous integration, **automated tests**, and code metrics.

**What is build** pipeline **in Jenkins?**

Answer # Job chaining in **Jenkins** is the process of automatically starting other job(s) after the execution of a job. This approach lets you build **multi-step build pipelines** or trigger the rebuild of a project if one of its dependencies is updated.

**create a backup and copy files in Jenkins**

To create a backup all you need to do is to periodically back up your JENKINS\_HOME directory. This contains all of your build jobs configurations, your slave node configurations, and your build history. To create a back-up of your Jenkins setup, just copy this directory. You can also copy a job directory to clone or replicate a job or rename the directory.

### **How will you secure Jenkins?**

The way I secure Jenkins is mentioned below, if you have any other way to do it than mention that:

* Ensure global security is on.
* Ensure that Jenkins is integrated with my company’s user directory with appropriate plugin.
* Ensure that matrix/Project matrix is enabled to fine tune access.
* Automate the process of setting rights/privileges in Jenkins with custom version controlled script.
* Limit physical access to Jenkins data/folders.
* Periodically run security audits on same.

**What you do when you see a broken build for your project in Jenkins?**

I will open the console output for the broken build and try to see if any file changes were missed. If I am unable to find the issue that way, then I will clean and update my local workspace to replicate the problem on my local and try to solve it.

**how you can move or copy Jenkins from one server to another?**

You can:

* Move a job from one installation of Jenkins to another by simply copying the corresponding job directory.
* Make a copy of an existing job by making a clone of a job directory by a different name.
* Rename an existing job by renaming a directory. Note that if you change a job name you will need to change any other job that tries to call the renamed job.

Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

**What is Jenkinsfile and what it does?**

1. The definition of a Jenkins Pipeline is typically written into a text file called a Jenkinsfile which in turn is checked into a project’s source control repository.

simple Jenkins Pipeline Code for Java:

Jenkinsfile (Declarative Pipeline)  
pipeline {  
    agent { docker 'maven:3.3.3' }  
    stages {  
        stage('build') {  
            steps {  
                sh 'mvn --version'  
            }  
        }  
    }  
}

### **Archive Build Output Artifacts**

##### **Synopsis**

This is a simple demonstration of how to archive the build output artifacts in workspace for later use.

// This shows a simple example of how to archive the build output artifacts.

node {

stage "Create build output"

// Make the output directory.

sh "mkdir -p output"

// Write an useful file, which is needed to be archived.

writeFile file: "output/usefulfile.txt", text: "This file is useful, need to archive it."

// Write an useless file, which is not needed to be archived.

writeFile file: "output/uselessfile.md", text: "This file is useless, no need to archive it."

stage "Archive build output"

// Archive the build output artifacts.

archiveArtifacts artifacts: 'output/\*.txt', excludes: 'output/\*.md'

}

### **Artifactory Maven Build**

##### **Synopsis**

This is a simple demonstration of how to run a Maven build, that resolves dependencies, upload artifacts and publish build info to Artifactory.   
Read the full documentation [here](https://www.jfrog.com/confluence/display/RTF/Working+With+Pipeline+Jobs+in+Jenkins).

node {

// Get Artifactory server instance, defined in the Artifactory Plugin administration page.

**def** server = Artifactory.server "SERVER\_ID"

// Create an Artifactory Maven instance.

**def** rtMaven = Artifactory.newMavenBuild()

**def** buildInfo

stage('Clone sources') {

git url: 'https://github.com/jfrogdev/project-examples.git'

}

stage('Artifactory configuration') {

// Tool name from Jenkins configuration

rtMaven.tool = "Maven-3.3.9"

// Set Artifactory repositories for dependencies resolution and artifacts deployment.

rtMaven.deployer releaseRepo:'libs-release-local', snapshotRepo:'libs-snapshot-local', server: server

rtMaven.resolver releaseRepo:'libs-release', snapshotRepo:'libs-snapshot', server: server

}

stage('Maven build') {

buildInfo = rtMaven.run pom: 'maven-example/pom.xml', goals: 'clean install'

}

stage('Publish build info') {

server.publishBuildInfo buildInfo

}

}

### **Push Git Repo**

##### **Synopsis**

This demonstrates how to push a tag (or branch, etc) to a remote Git repository from within a Pipeline job. The authentication step may vary between projects. This example illustrates injected credentials and also username / password authentication.

##### **Note**

If you inject a credential associated with your Git repo, use the Snippet Generator to select the plain Git option and it will return a snippet with this gem:

java stage('Checkout') { git branch: 'lts-1.532', credentialsId: '82aa2d26-ef4b-4a6a-a05f-2e1090b9ce17', url: 'git@github.com:jenkinsci/maven-plugin.git' } This is not ideal - there is an open JIRA, https://issues.jenkins-ci.org/browse/JENKINS-28335, for getting the GitPublisher Jenkins functionality working with Pipeline.

##### **Credit**

Based on Stackoverflow answer at http://stackoverflow.com/questions/33570075/tag-a-repo-from-a-jenkins-workflow-script Injected credentials gist at https://gist.github.com/blaisep/eb8aa720b06eff4f095e4b64326961b5#file-jenkins-pipeline-git-cred-md

// This is currently the best way to push a tag (or a branch, etc) from a

// Pipeline job. It's not ideal - https://issues.jenkins-ci.org/browse/JENKINS-28335

// is an open JIRA for getting the GitPublisher Jenkins functionality working

// with Pipeline.

// credentialsId here is the credentials you have set up in Jenkins for pushing

// to that repository using username and password.

withCredentials([usernamePassword(credentialsId: 'git-pass-credentials-ID', passwordVariable: 'GIT\_PASSWORD', usernameVariable: 'GIT\_USERNAME')]) {

sh("git tag -a some\_tag -m 'Jenkins'")

sh('git push https://${GIT\_USERNAME}:${GIT\_PASSWORD}@<REPO> --tags')

}

// For SSH private key authentication, try the sshagent step from the SSH Agent plugin.

sshagent (credentials: ['git-ssh-credentials-ID']) {

sh("git tag -a some\_tag -m 'Jenkins'")

sh('git push <REPO> --tags')

}

### **Artifactory Generic Upload Download**

##### **Synopsis**

This is a simple demonstration of how to download dependencies, upload artifacts and publish build info to Artifactory.   
Read the full documentation [here](https://www.jfrog.com/confluence/display/RTF/Working+With+Pipeline+Jobs+in+Jenkins).

node {

git url: 'https://github.com/jfrogdev/project-examples.git'

// Get Artifactory server instance, defined in the Artifactory Plugin administration page.

**def** server = Artifactory.server "SERVER\_ID"

// Read the upload spec and upload files to Artifactory.

**def** downloadSpec =

'''{

"files": [

{

"pattern": "libs-snapshot-local/\*.zip",

"target": "dependencies/",

"props": "p1=v1;p2=v2"

}

]

}'''

**def** buildInfo1 = server.download spec: downloadSpec

// Read the upload spec which was downloaded from github.

**def** uploadSpec =

'''{

"files": [

{

"pattern": "resources/Kermit.\*",

"target": "libs-snapshot-local",

"props": "p1=v1;p2=v2"

},

{

"pattern": "resources/Frogger.\*",

"target": "libs-snapshot-local"

}

]

}'''

// Upload to Artifactory.

**def** buildInfo2 = server.upload spec: uploadSpec

// Merge the upload and download build-info objects.

buildInfo1.append buildInfo2

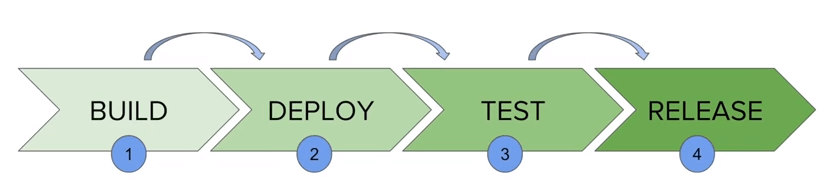
// Publish the build to Artifactory

server.publishBuildInfo buildInfo1

}

### What is Continuous Delivery Pipelines? How it Works?

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

[](https://www.guru99.com/images/1/091318_0510_JenkinsPipe1.png)

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

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

**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 J**enkinsFile are**:

* You can create pipelines automatically for all branches and execute pull requests with just one **JenkinsFile.**
* You can review your 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 JenkinsFile.

**Declarative versus Scripted pipeline syntax:**

There are two types of 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** |
| Pipeline | The pipeline is a set of instructions given in the form of code for continuous delivery 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. |

