**Jenkins Pipeline**

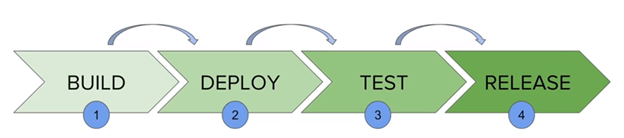
In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

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.

## **What is Continuous Delivery Pipeline?**

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



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

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

**In simple words, a pipeline is a set of interconnected tasks that execute in a specific order**

### **Understanding Jenkinsfile Syntax:**

There are two types of syntax using which we can define a *Jenkinsfile*:

* *Declarative Pipeline syntax*
* *Scripted Pipeline syntax*

#### **Declarative Pipeline Syntax**

The declarative syntax is a new feature that used code for the pipeline. **It provides a limited pre-defined structure**. Thereby, it offers an easy & simple continuous delivery pipeline. Moreover, it uses a *pipeline block*.

#### **Scripted Pipeline Syntax**

Unlike declarative syntax**, the *scripted pipeline syntax* is the old traditional way to write the *Jenkinsfile* on Jenkins web UI.** Moreover, it strictly follows the groovy syntax and helps to develop a complex pipeline as code.

**Understanding Key constructs of Jenkinsfile**

We will now see some critical concepts used in *Pipelines*:

* ***Pipeline****- A pipeline is a set of instructions that includes the processes of continuous delivery. For example, creating an application, testing it, and deploying the same. Moreover, it is a critical element in declarative pipeline syntax, which is a collection of all stages in a Jenkinsfile. We declare different stages and steps in this block.*

pipeline{

}

* ***Node****- A node is a key element in scripted pipeline syntax. Moreover, it acts as a machine in Jenkins that executes the Pipeline.*

node{

}

* ***Stage****- A stage consists of a set of processes that the Pipeline executes. Additionally, the tasks are divided in each stage, implying that there can be multiple stages within a Pipeline. The below snippet shows the different stages that one can define in a Pipeline.*

pipeline{

agent any

stages{

stage('Build'){

........

}

stage('Test'){

........

}

stage('Deploy'){

........

}

stage('Monitor'){

........

}

}

}

* ***Steps****- A step in Jenkins defines what we have to do at a particular step in the process. There can be a series of steps within the stage. Moreover, whatever step we define in a stage would be executed within it.*

pipeline{

agent any

stages{

stage('Stage 1'){

steps{

//Perform steps related to this stage

}

}

stage('Stage 2'){

steps{

//Perform steps related to this stage

}

}

}

}

* ***Agent****- An agent is a directive that enables the users to execute multiple projects within the same Jenkins instance by distributing the load. Moreover, we assign an executor to the build through an agent. You can either use a single agent for the entire pipeline or use a distinct agent for the different stages of the pipeline. Subsequently, some of the parameters used with agents are -*
  + ***Any****- Any of the available agents execute the pipeline.*
  + ***None****- It is used at the pipeline root and implies no global agent, but each stage must specify its own agent.*
  + ***Label****- The labeled agent is used to execute the pipeline or the specific stage.*
  + ***Docker****- One can use the Docker images as the execution environment & specifying the agent as docker.*

## **Create a Jenkins Pipeline**

After installation of Jenkins now we are good to move further 🎊 🎉🎊 🎉

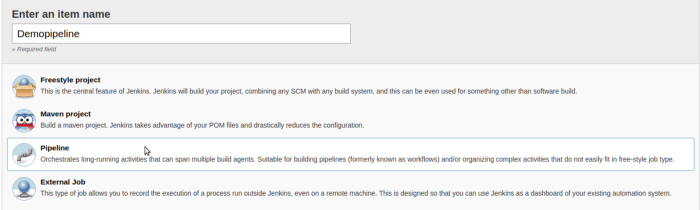
**Step 1)** For Jenkins Pipeline we need to install the **Pipeline** plugin.

Goto**Manage Jenkins > Manage Plugins >Available**section**>**search**Pipeline.**

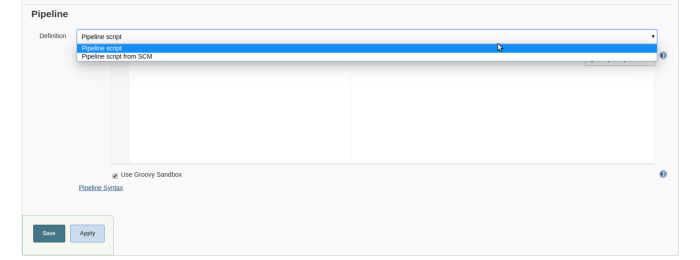
If you are already installed**Pipeline**It will display in **Installed** section.

Now we can create a new project with **Pipeline.**

**Step 2)**Goto Jenkins Home > New Item > Create Project with Pipeline.



After creating Pipeline Project it will display this view



This is the Pipeline view Here you can write your pipeline.

There are two options here :

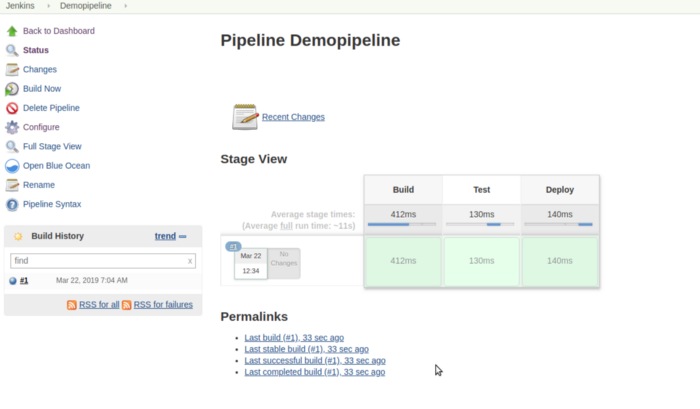
***Pipeline Script****: We can write the Pipeline Directory on Jenkins.*

***Pipeline Script from SCM****: We can write Jenkinsfile and upload it on Github and connect it using Pipeline Script from SCM option.*

This is the demo pipeline script. you can copy demo script and paste it on Jenkins or you can create jenkinsfile and upload it on Github.

Here we go, Now we create our first Jenkins Pipeline.

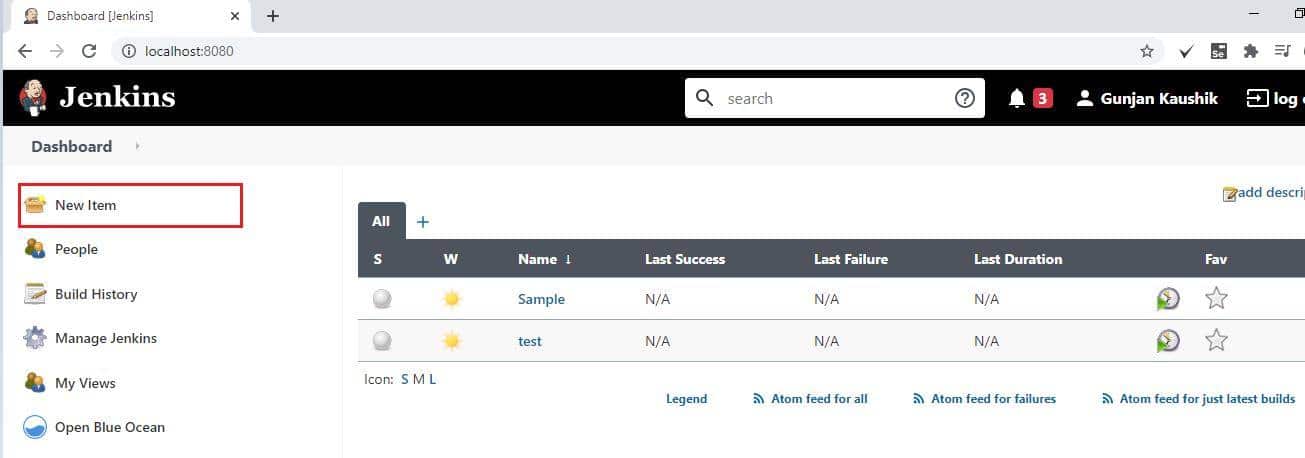
When we run our Jenkins Pipeline job we can see this type of layout



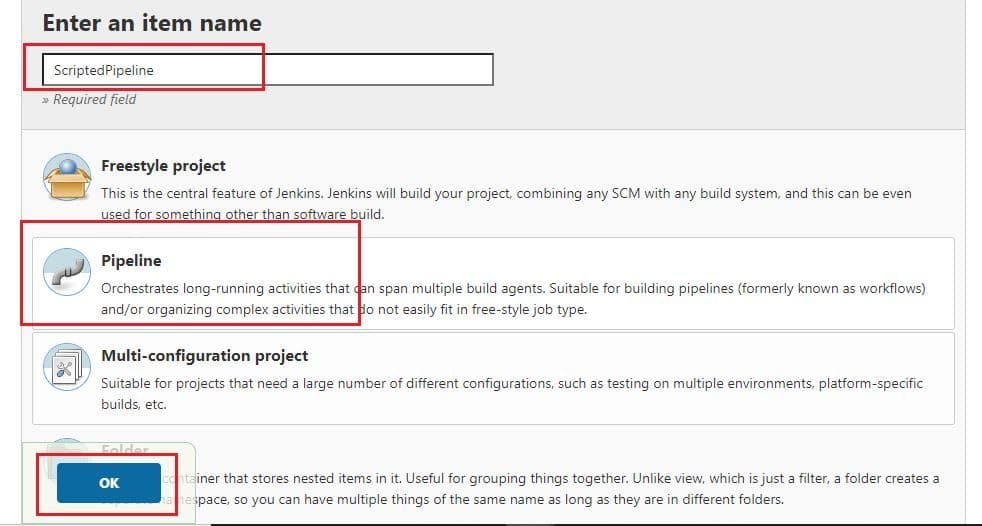
### ***How to create a Scripted Jenkins Pipeline?***

Once you have started Jenkins and logged on to its UI, you may follow the below steps to create your first pipeline-

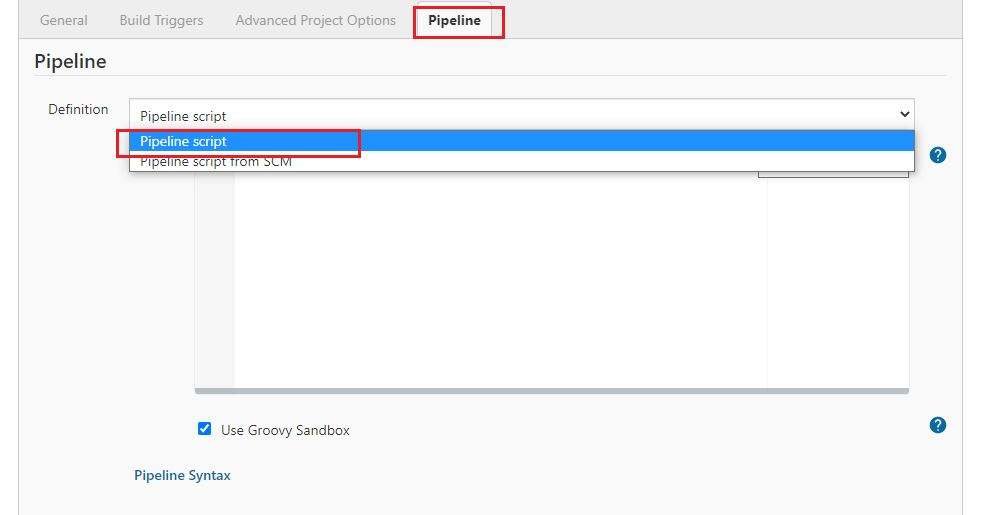
1. **Firstly, from the Jenkins dashboard, click on New Item on the left panel.**



1. **Secondly, enter the *name* for your pipeline, select *Pipeline* from the list. After that, click *OK*.**



1. **After that, go to the *Pipeline* tab, and from the *Definition*, the dropdown selects the *Pipeline* script.**



1. The next step is to write your pipeline code in the web UI provided by Jenkins. Let us see a sample pipeline example as available in Jenkins-

pipeline {

agent any

stages {

stage('One') {

steps {

echo 'Hi, welcome to pipeline demo...'

}

}

stage('Two') {

steps {

echo('Sample testing of Stage 2')

}

}

stage('Three') {

steps {

echo 'Thanks for using Jenkins Pipeline'

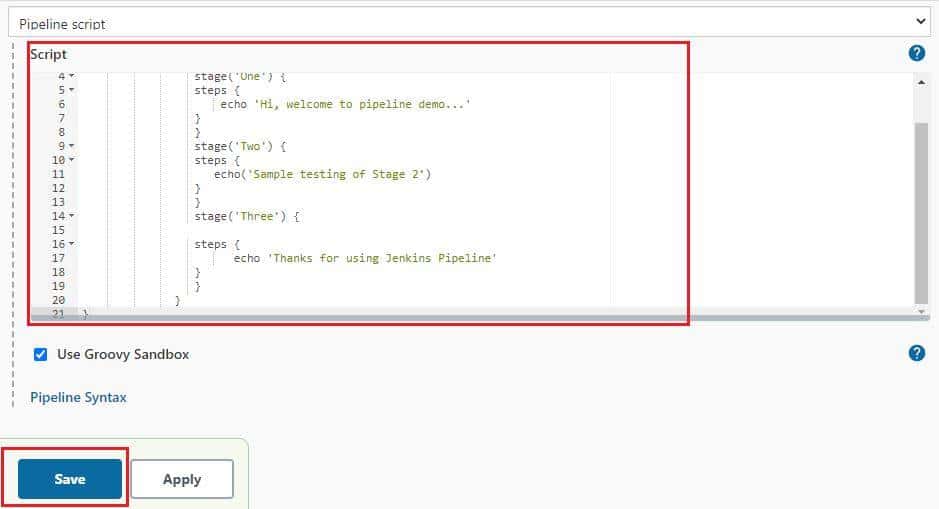
}

}

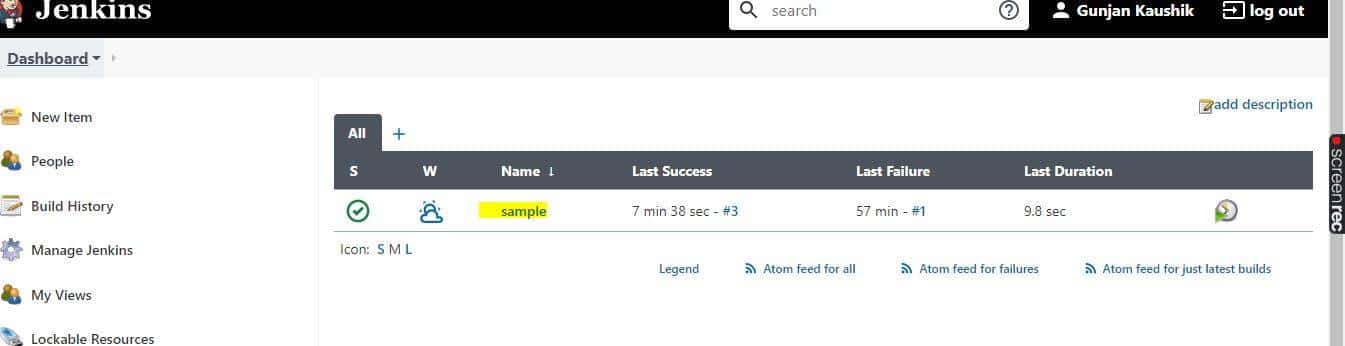
}

}

You need to copy and paste the same in UI as shown below-



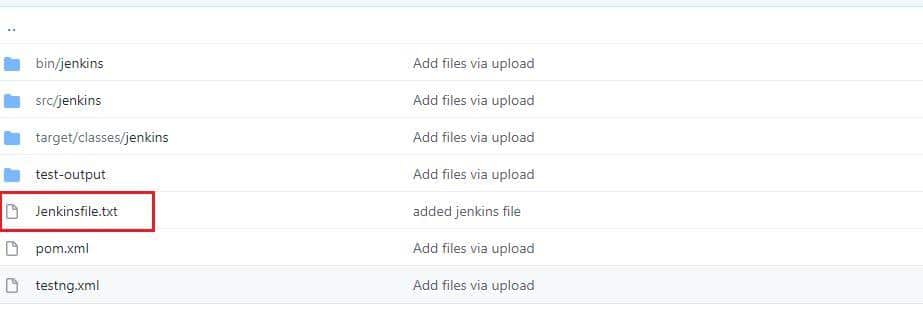
1. **After that, click on *Save*. Conclusively, this finishes the process.**



We will see running this pipeline in the next section; before that, let us see how to create a Declarative Pipeline in Jenkins.

### ***How to create a Declarative Jenkins Pipeline?***

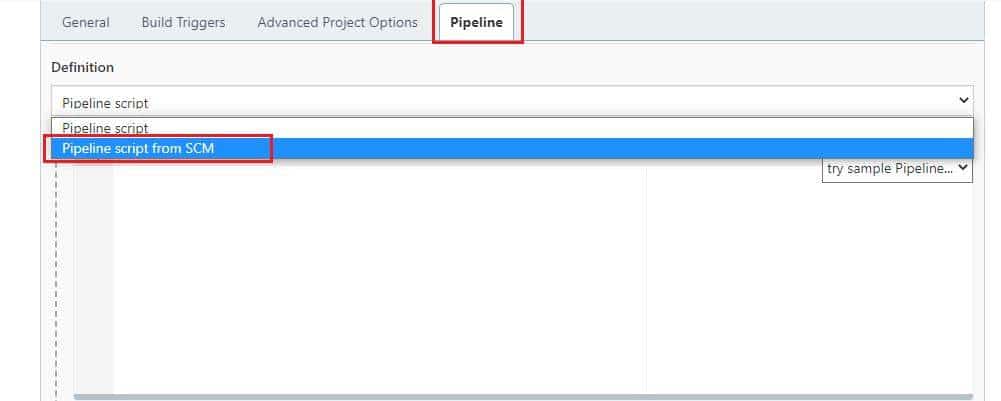
To create a declarative pipeline, you need to have a ***Jenkinsfile*** in place. Since I will be using the project from my Github account, I have already placed the ***Jenkinsfile*** in my project.



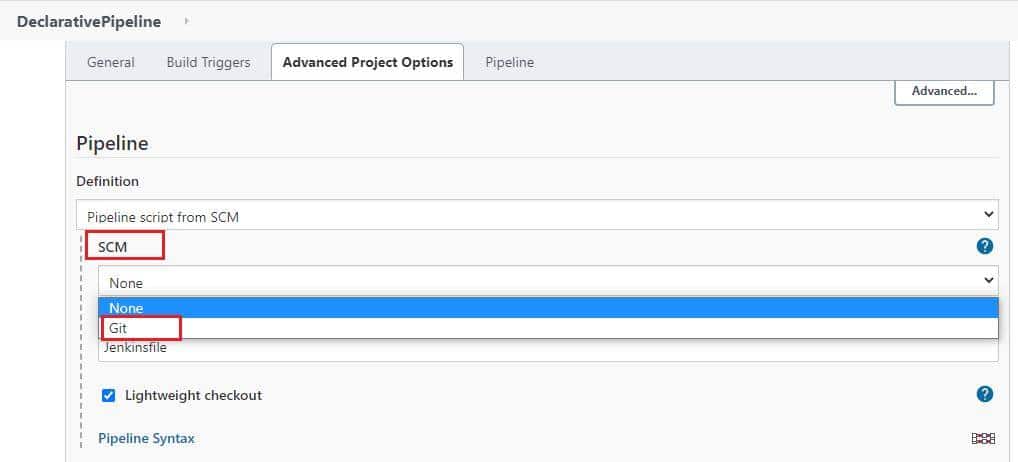
***Note***: For the tutorial, I am using the same code as used in the scripted pipeline in the Jenkinsfile.

For creating a Declarative Pipeline, you may follow step#1 and Step#2 from the scripted pipeline creation steps stated above and then follow the below steps-

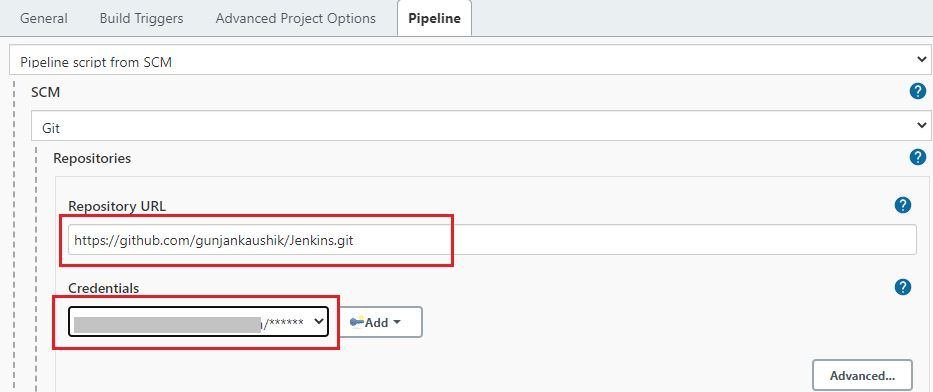
1. Go to the Pipeline tab, and from the ***Definition***, the dropdown selects the ***Pipeline script from SCM***.



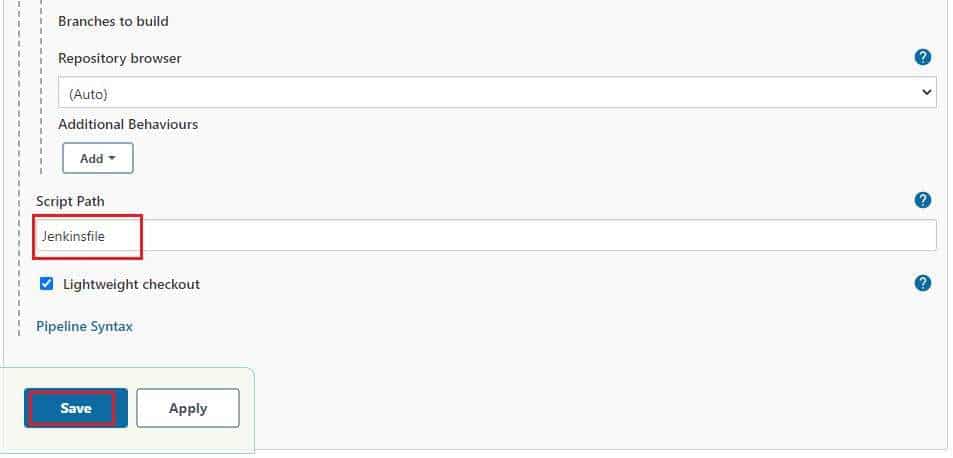
1. You need to select Git from the SCM dropdown now.



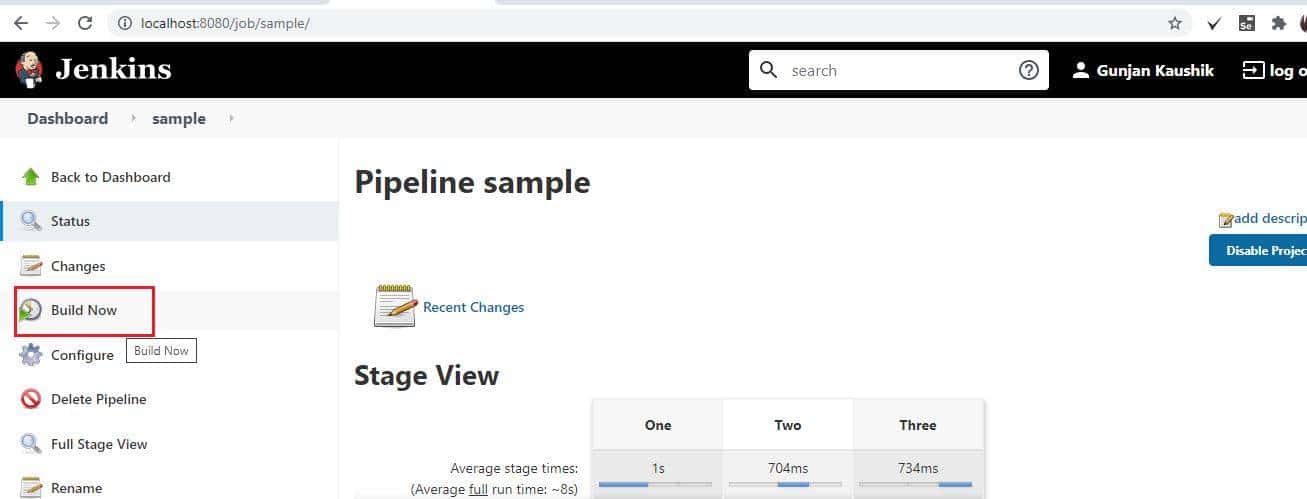
1. Now, you will get an option to input your Repository URL and credentials.



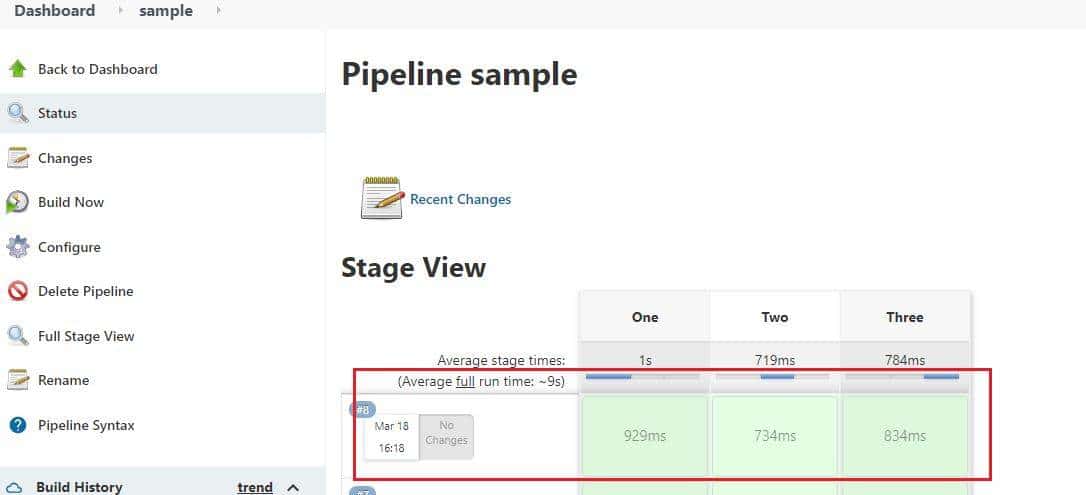
1. Next, you may set the ***branch*** or let it be blank for any branch. In the script path, you need to write the ***Jenkinsfile*** name that exists in your repository. Click on ***Save***, and there you go, your declarative pipeline is ready for use.



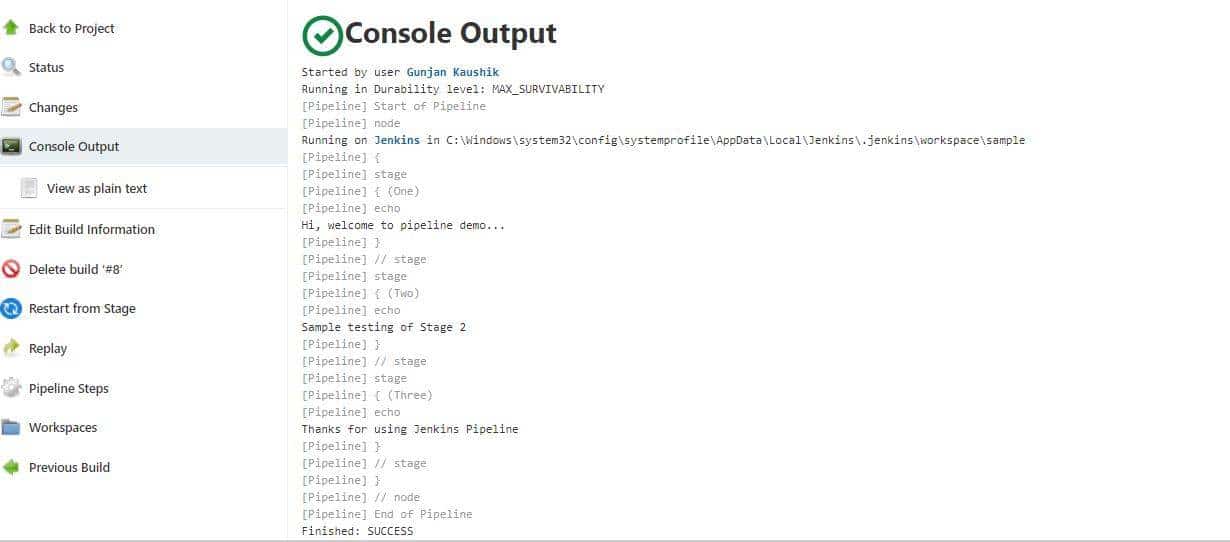
Now that you are all set with your pipelines, you can execute the same from your Jenkins UI. All you need to do is select your pipeline and click on ***Build Now*** link on the left panel.



Once you run the pipeline, you will see the results displayed on the stage view as shown below-



You can view the console output for the build as well, which would display the print statements and errors(if any) that will be helpful for your debugging.



See how easy it is to create and build your pipeline.

# How to build & deploy application on web server using Jenkins CI/CD pipeline

**Introduction**

In this article we will discuss automation of application deployment on a web server (tomcat) using Jenkins pipeline. We will use [Jenkins pipeline as a script](https://www.jenkins.io/doc/book/pipeline/getting-started/). Using the pipeline we will build the application first and then deploy the build artifacts into a web server. We will use a sample Java project with maven.

**Prerequisite**

To complete this tutorial, you will need the below:

Prerequisite Knowledge

* Git, GitHub
* Building java application using maven
* Jenkins
* Web server like tomcat, JBoss etc

Prerequisite setup

* You should have Jenkins installed on a Linux machine, and it is up & running. Also, Jenkins dashboard should be accessible from your browser.
* We will use a Java project with maven, so we need maven to be installed on the Jenkins machine and MAVEN\_HOME path should be set properly.
* You will need a Java project on GitHub.
* You will need an JFrog artifactory repository to store the build artifacts. If you don’t have an account, you can create on (it is free for limited usage).
* Finally, you will need a machine where tomcat server is installed. Tomcat web server should be up & running.

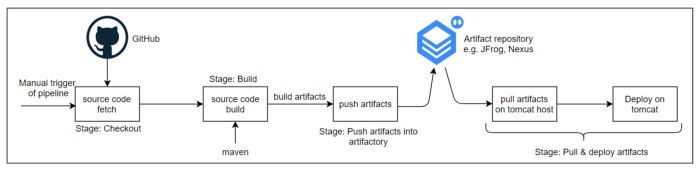
**Step 1 — Getting Visual Idea about the Pipeline**

There are two option -

* Option 1: Push the build artifacts to repository and then download it from repository on the tomcat host.
* Option 2: Send the build artifacts to tomcat host using SCP(Secure Copy Protocol (SCP) is a protocol for securely transferring files between a local and a remote host or between two remote hosts.).

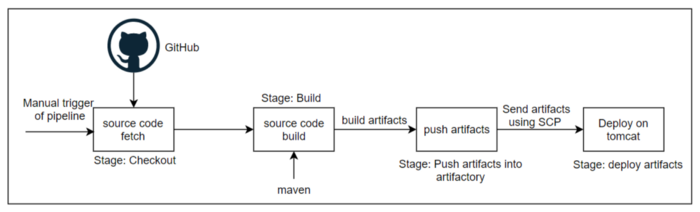
In this article we discussed Option 1.

Option 1: Push the build artifacts to repository and then download it from repository on the tomcat host



In this case, after build we are pushing the artifacts into repository and downloading it on the tomcat host.

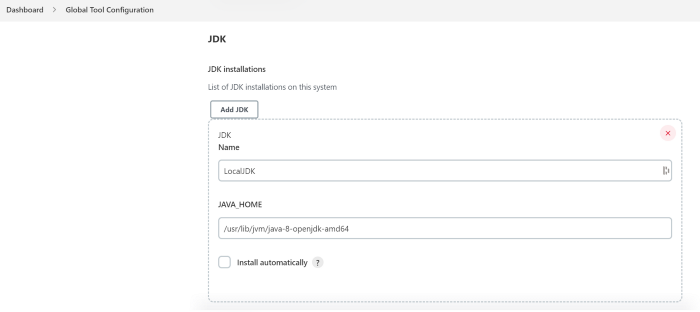
Option 2: Send the build artifacts to tomcat host using SCP



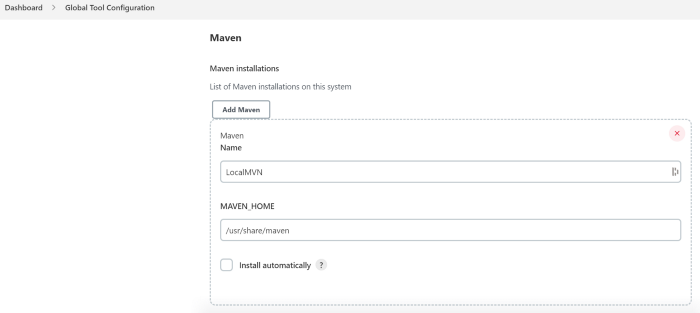
In this case, after build we are sending the artifacts to tomcat host using SCP.

**Step 2 — Configure Jenkins with JAVA\_HOME, MAVEN\_HOME**

We need to provide JAVA\_HOME, MAVEN\_HOME in the 'Global Tool Configuration' of Jenkins (Manage Jenkins → Global Tool Configuration).



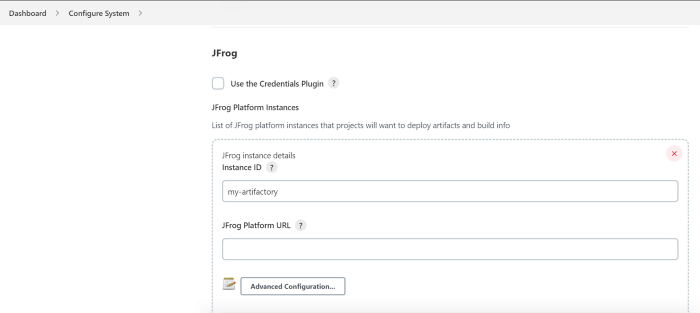
providing JAVA\_HOME path



providing MAVEN\_HOME path

**Step 3 — Configure repository credential with Jenkins**

1. Since we are using JFrog artifactory to store the build artifacts, so we need to configure credential of JFrog artifactory with Jenkins. For this we installed JFrog plugin in Jenkins and configured the details under ‘Mange Jenkins → Configure System’.



configuring details of JFrog artifactory in Jenkins

2. If the source code repository is a private repository then you need to configure the credential.

**Step 4— Enable passwordless communication between Jenkins host & Tomcat host**

To achieve this we will configure the SSH keys. On the Jenkins server we need to generate SSH keys, and we need to place the public key of Jenkins server into the tomcat host so that Jenkins can connect with the tomcat host without password. And on the Jenkins server we need to make StrictHostKeyChecking no OR StrictHostKeyChecking accept-new in the /etc/ssh/ssh\_config file.

**Step 5— Prepare Jenkins Pipeline Script**

We are using Jenkins declarative type pipeline script. Below is the Jenkins pipeline script

pipeline {  
 agent any  
 tools {  
 maven "LocalMVN"  
 }  
 stages {  
 stage('Checkout') {  
 steps {  
 withCredentials([string(credentialsId: 'GitHub\_Token', variable: 'github\_token')]) {  
 checkout([$class: 'GitSCM',  
 branches: [[name: '\*/master']],  
 extensions: [[$class: 'CleanCheckout']],  
 userRemoteConfigs: [[url: 'https://' + env.github\_token + '@' + 'github.com/sk617/simple-java-maven-app.git']]  
 ])  
 }  
 }  
 }  
 stage('Build') {  
 steps {  
 sh "mvn -Dmaven.test.failure.ignore=true clean package"  
 }  
 }  
 stage('Push artifacts into artifactory') {  
 steps {  
 rtUpload (  
 serverId: 'my-artifactory',  
 spec: '''{  
 "files": [  
 {  
 "pattern": "\*.war",  
 "target": "example-repo-local/build-files/"  
 }  
 ]  
 }'''  
 )  
 }  
 }  
 stage('Pull artifacts & deploy on tomcat') {  
 steps{  
 withCredentials([usernamePassword(credentialsId: 'my-artifactory-cred',  
 usernameVariable: 'USERNAME',  
 passwordVariable: 'PASSWORD')]) {  
 sh 'ssh ubuntu@149.158.89.34 curl -u ' + USERNAME + ':' + PASSWORD + ' -X GET "Your\_JFrog\_Artifactory\_URL\_of\_file" --output /opt/tomcat9/webapps/deploy.war'  
 }  
 }  
 }  
 }  
}

Stage: Checkout

In this stage source code will be fetched from GitHub. If it is a private repo then we need to provide the credential. In the above script we have provided credential.

Stage: Build

In this stage source code will be built using maven. At the end of build it produces a war file.

Stage: Push artifacts into artifactory

In this stage, build artifacts are pushed into artifactory.

Stage: Pull artifacts & deploy on tomcat

In this stage, artifacts are pulled from artifactory and deployed on tomcat web server.

**Stage 6 — Trigger the pipeline**

This is the last step where we need to trigger the pipeline. Once we trigger the pipeline, it will execute all the tasks sequentially.