multibranch pipeline job type lets you define a job where from a single git repository Jenkins will detect multiple branches and create nested jobs when it finds a Jenkinsfile

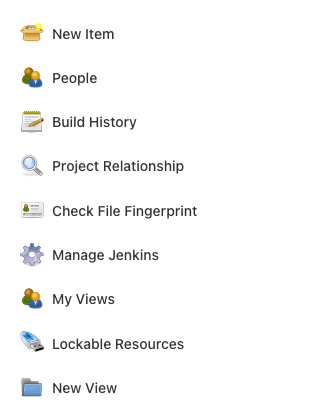
From the above definition, we can understand, Jenkins can scan Git repo for Jenkinsfile & create jobs automatically. All it needs from us is the Git Repo details. In this article, we are going to use a [sample GitHub repository](https://github.com/iamvickyav/spring-boot-h2-war-tomcat). Our sample GitHub repo contains a sample Spring Boot project which can be deployed to Tomcat.

|  |
| --- |
| agent any     stages {         stage('Build Code') {             steps {                 sh """                 echo "Building Artifact"                 """             }         }        stage('Deploy Code') {            steps {                 sh """                 echo "Deploying Code"                 """            }        }     }  } |

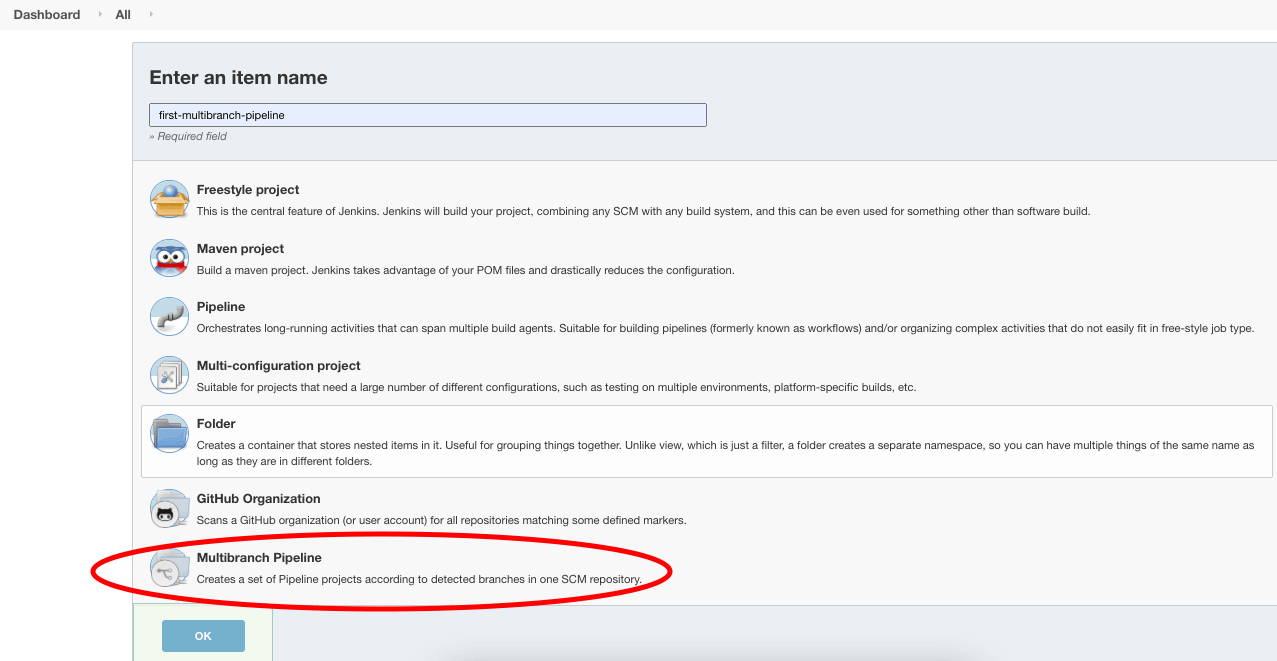
We created two stages “Build Code” & “Deploy Code” in our Jenkinsfile, each of them configured to print appropriate messages. Now we have the Git repo with Jenkinsfile ready.

**Steps To Create A Jenkins Multibranch Pipeline**

**Step 1**: Open Jenkins home page (http://localhost:8080 in local) & click on New Item from the left side menu.



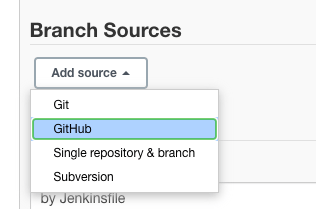
**Step 2:** Enter **Jenkins job name** & choose the style as **multibranch pipeline** and click **OK**.



**Step 3**: In the **configure** page, we need to configure only one thing – **The Git Repo source**.

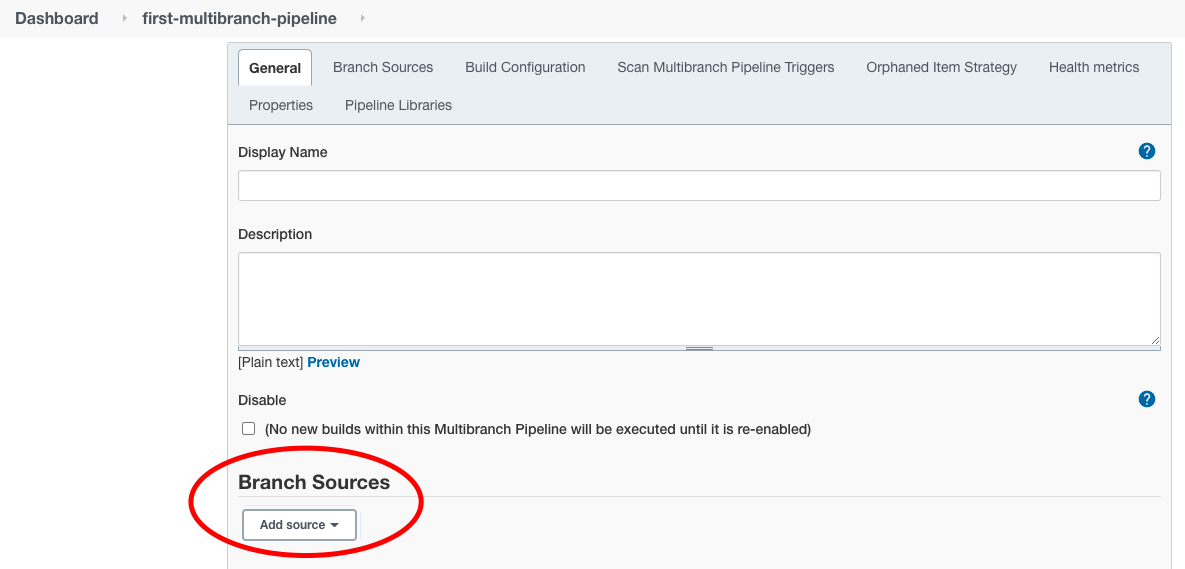
Scroll down to **Branch Sources** section & click on **Add Source** dropdown.

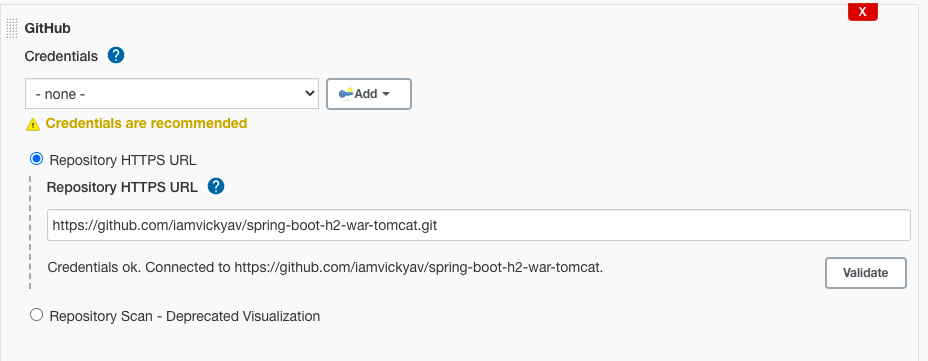
Choose **GitHub** as Source as our Sample GitHub repo is hosted there.



**Step 3**: Enter the **Repository HTTPS URL** as https://github.com/iamvickyav/spring-boot-h2-war-tomcat.git and click on **Validate .**

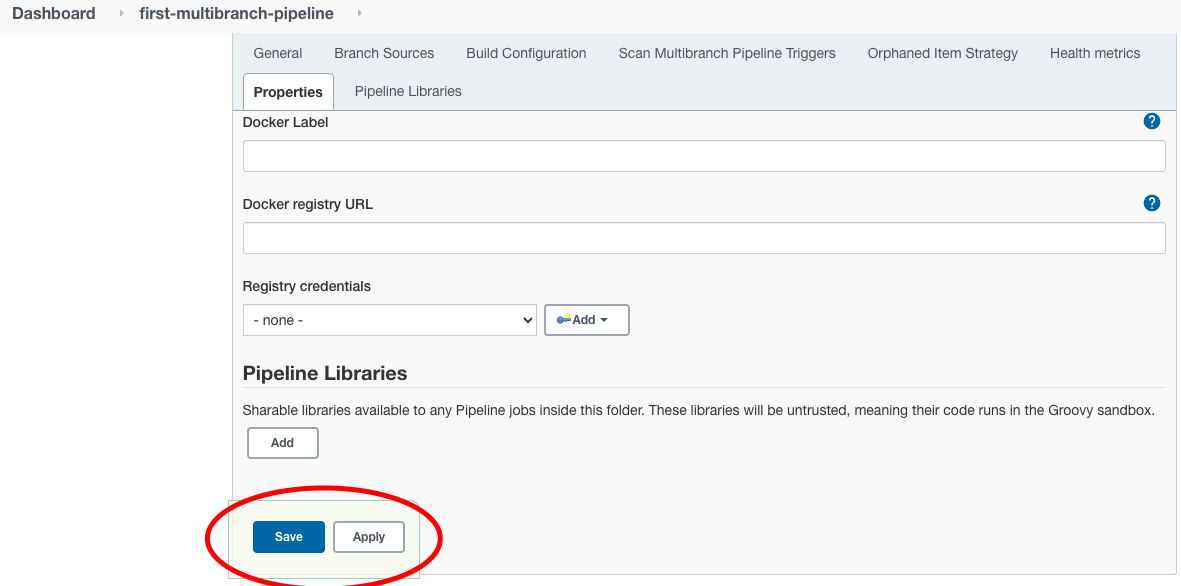
Since our GitHub repo is hosted as a public repo, we don’t need to configure credentials to access it. For enterprise/private repos, we may need credentials to access them.





The “**Credentials ok**” message represents the connection between Jenkins server & the Git repo is successful.

**Step 4**: Leave the rest of the configuration sections as such for now and click on the **Save** button at the bottom.



On saving, Jenkins will perform the following steps automatically.

**Scan Repository Step**

* Scan the Git repo we configured
* Look for the list of branches available in the Git repo
* Select branches which has Jenkinsfile

**Running Build Step**

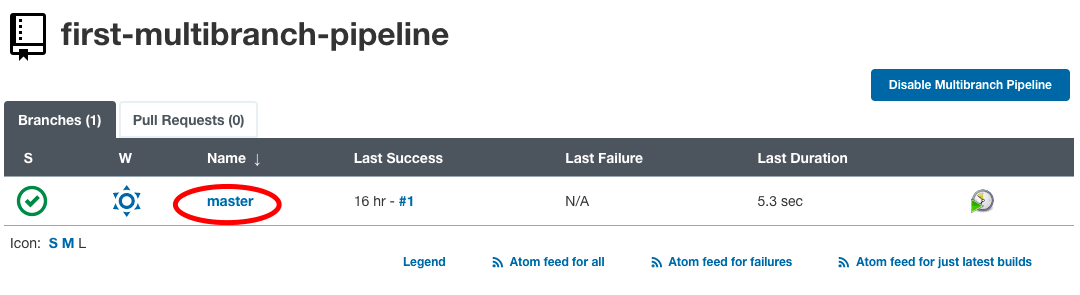
* Run build for each of the branches found in previous step with steps mentioned in Jenkinsfile

From the Scan Repository Log section, we can understand what happened during the Scan Since we have an only a master branch in our git repo, Scan Repository Log says **1** branches were processed

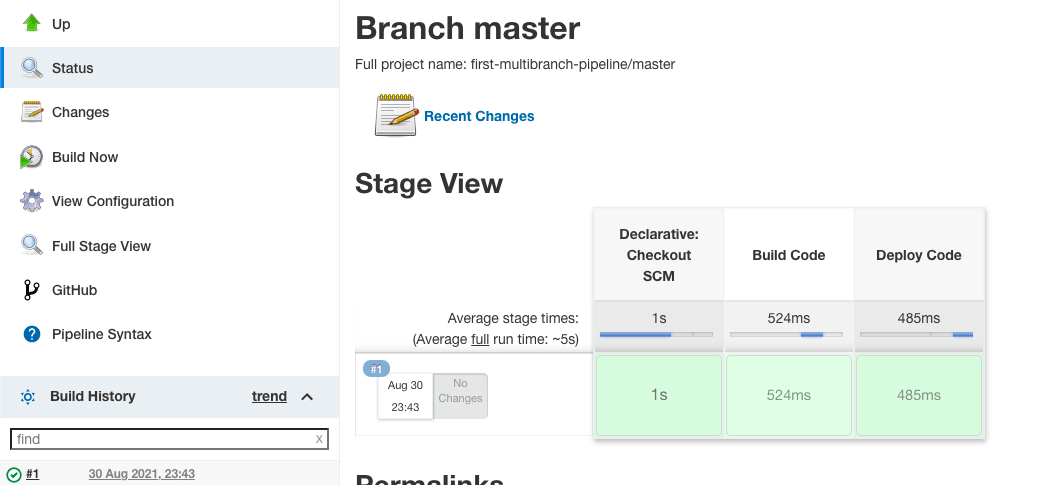
After the scan is complete, Jenkins will create & run a build job for each processed branch separately.

In our case, we had only one branch called master. Hence build will run for master branch alone. We can check the same by clicking on **Status** in the left side menu.

We can see a build job created for the master branch in the status section.



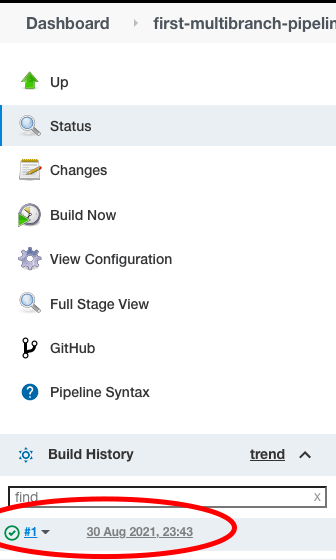
Click on the branch name to see the build job log & status.



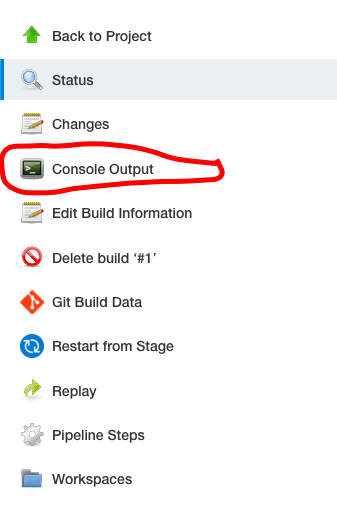
**Stage View** gives a visual representation on how much time each stage took to execute & status of the build job.

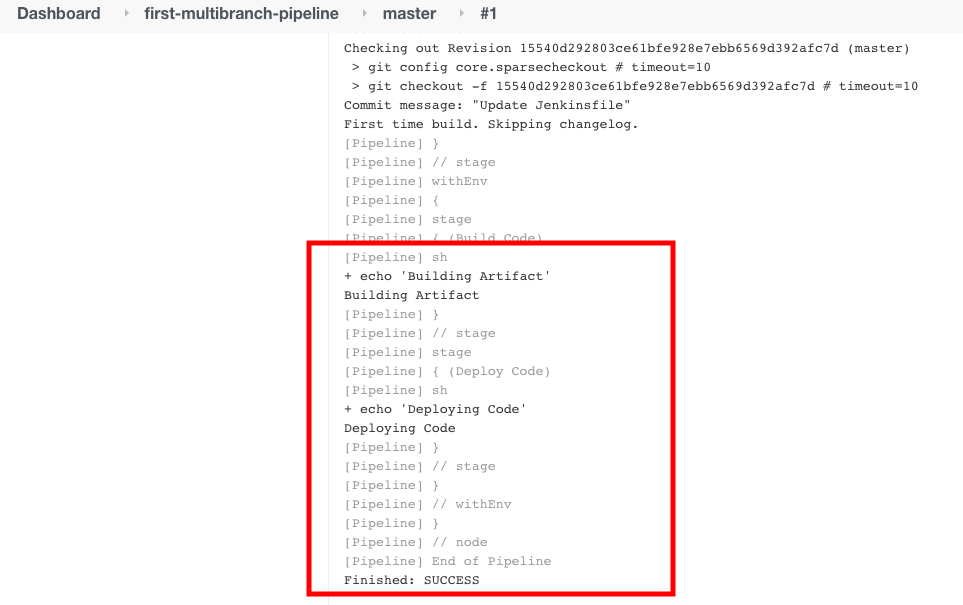
### **Access Build Job Run Logs**

**Step 1**: Click on the build number under the build history section.



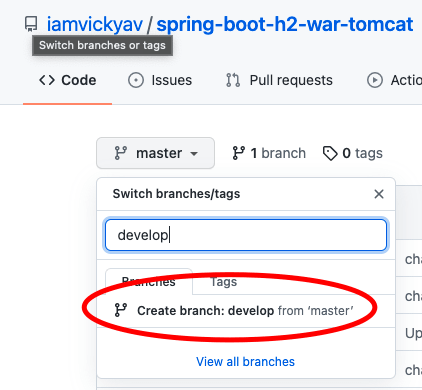
**Step 2**: Then choose the **Console Output** from the left side menu to see the logs.





What happens if we have more than one branch in our Git repo? Let’s check that now.

In the Git repo, a new branch called **develop** is created.



To differentiate **develop** branch build, we made small changes in echo commands in Jenkinsfile.

**Jenkinsfile in master branch**:



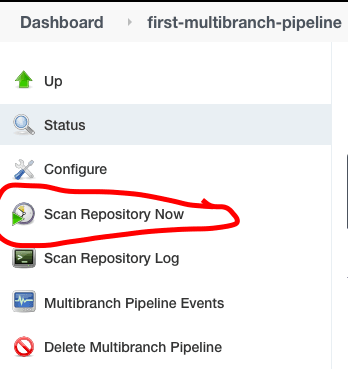
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | pipeline {     agent any     stages {         stage('Build Code') {             steps {                 sh """                 echo "Building Artifact"                 """             }         }        stage('Deploy Code') {            steps {                 sh """                 echo "Deploying Code"                 """            }        }     }  } |

**Jenkinsfile in develop branch**:

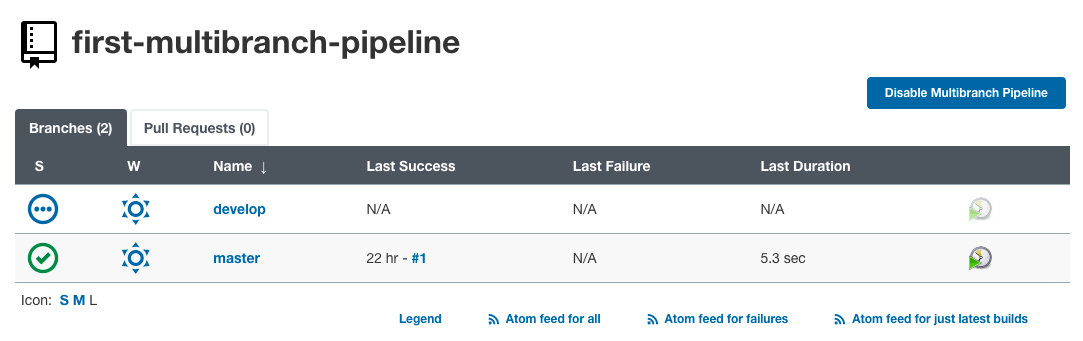


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | pipeline {     agent any     stages {         stage('Build Code') {             steps {                 sh """                 echo "Building Artifact from Develop Branch"                 """             }         }        stage('Deploy Code') {            steps {                 sh """                 echo "Deploying Code from Develop Branch"                 """            }        }     }  } |

Now we have two Jenkinsfile in two different branches. Let’s rerun the repository scan in Jenkins to see the behaviour.



We can see the new branch got detected by Jenkins. Hence a new job was created separately for develop branch.



On clicking ‘**develop**’, we can see the log for the develop branch’s build job.

