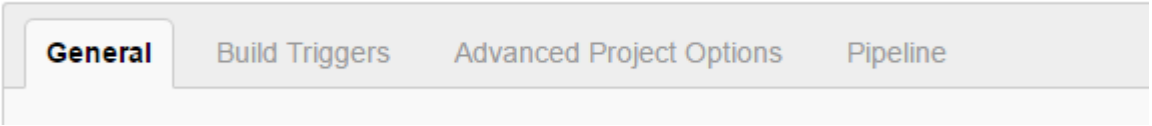
Jenkins Pipeline Job

Select the Pipeline job after giving a name to the job.



The menu is different from the freestyle project, where we configured build, post build action etc. In case of Pipeline job we do not have those option, instead we have a menu item to create a Pipeline itself. Let's look at the option.

The main section of the Pipeline job is the Pipeline section where we write a groovy script to align the steps in a pipeline.

Groovy is based on Java, so one who understands Java can very easily grasp with the groovy syntax.

Jenkins provides a help to create the Groovy script by using the 'Pipeline syntax' option.

Now to start creating a Pipeline job to build using Maven as the build stage, click on the 'Pipeline Syntax ' option, this will open a page to create groovy syntax for the action (steps) in the build job.

Here select the step as 'bat: Windows Batch script'. Here enter the commands that we will use for running the maven build, as 'clean package'

Click on the 'Generate Pipeline script' and this will provide the groovy syntax for build step.

The obvious step would be to archive the artefacts. For this we can use the snippet generator again, and in that select the step as 'step: General build step', in here select the 'Archive the artifact' build step.

Likewise select appropriate sample steps to create the entire build script in Groovy for the Pipeline job.

One very important thing to note here is that the pipeline job as to be assigned to a node to work with the pipeline job. So select the groovy snippet generator and select, 'node:Allocate node'

This functionality in the pipeline job actually allows selecting which node to be used for running the job and thus manage the way we want the Jenkins to run parallel tasks.

This functionality in Jenkins is on the principle that Jenkins works in the Master – slave mode, while most of the jobs are run on the master, if the master remains busy, then the agent is pulled to work on the request. The jobs are distributed to the node (agents) as per their availability.

```
node {
   stage 'checkout'
   git 'https://github.com/ganeshhp/helloworldweb.git'

   def project_path='TestProject/simpleWebApp'

      dir(project_path) {
         stage 'build'
         bat 'mvn clean package'

         stage 'archive'
         archiveArtifacts 'target/*.war'

         stage 'issue log'
         step([$class: 'JiraVersionCreatorBuilder'])
      }

}
```

The Pipeline syntax being a Groovy script, we can modify, manipulate it to a great extent.

Try adding some stages as shown below.,

```
stage('Build') {

      build 'sample'

  }
```

Here the ‘sample’ is a job which is executed as a step in the pipeline job.

```
node {
   stage ('scm checkout') {
   checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs:
   }

   # build job: 'buildjob', quietPeriod: 3

   stage ('build') {
   sh 'mvn -f pom.xml clean package'
   }

   stage ('deploy') {
   sh '''sudo cp /home/vagrant/jenkins/workspace/workspace/build070117/target/*.war /opt/tomcat/apache-tomcat-7.0.78/webapps/
   sudo sh /opt/tomcat/bin/shutdown.sh
   sudo sh /opt/tomcat/bin/startup.sh'''

   }
   stage ('archive') {
   archiveArtifacts 'target/*.war'
   }
}
```

Node statement definition

Build step definition

```
node('Jnode1'){
    stage ('scm checkout') {
    git 'https://github.com/ganeshhp/helloworldweb.git'
    }

    stage ('build') {
    sh 'mvn -f pom.xml clean package'
    }

    stage ('deploy') {
    sh '''sudo cp /home/vagrant/jenkins/workspace/workspace/build070117/target/*.war /opt/tomcat/apache-tomcat-7.0.78/webapps/
    sudo sh /opt/tomcat/apache-tomcat-7.0.78/bin/shutdown.sh
    sudo sh /opt/tomcat/apache-tomcat-7.0.78/bin/startup.sh'''

    }
    stage ('archive') {
    archiveArtifacts 'target/*.war'
    }
}
```

Defining Node name to run a job on a specific node