**Jenkins pipeline**

**Pipeline (Jenkins Pipeline)** is a suite of Jenkins plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. A continuous delivery pipeline is an automated expression of process for getting software from version control right through to users and customers.

Jenkins pipeline adds a powerful set of automation tools onto Jenkins. Setting up a Pipeline project means writing a script that will sequentially or concurrently apply some steps of the process we want to accomplish.

**Jenkinsfile** is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. Pipelines are implemented in code (groovy script) and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline as needed. As Jenkins configuration is maintained in version control, there is no chance of losing it and can easily reverted to stable version. As its written in groovy, it support complex real-world continuous delivery requirements, including the ability to fork/join, loop, and perform work in parallel.

**Problems can be solved using Jenkinsfile**

* Define the CI/CD pipeline as a code to make it self-documented, reproducible and versioned.
* Have a single definition of build steps for any type of build job, be it multi-branch, merge requests or parameterized.
* Get away from a manual configuration of build steps.
* Make the Pipeline easily extendable. E.g., it should not be complicated to add a new static analysis tool report into all the configured build jobs.

**Build Jobs** are the runnable tasks that are controlled and monitored by Jenkins. Examples of jobs include compiling source code, running tests, provisioning a test environment, deploying, archiving, posting build jobs such as reporting, and executing arbitrary scripts.

**Jenkins Pipeline Terms**

*Stage*—  is a step for defining a conceptually distinct subset of the entire Pipeline, for example: “Build”, “Test”, “Deploy “ etc.

*Step* — a single task; fundamentally steps tell Jenkins what to do.

*Node*— most work a Pipeline performs is done in the context of one or more declared node steps. Node selects where the pipeline will be executed. Generally node step 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 where work can be done on files checked out from source control.

**Challenges we face in our current architecture:** Currently we are facing following challenges.

* **No such verification for state of the Jenkins Node.**
* **Automated distribution of job across node was not present**
* **Utilization of node which is become online in-between was not possible**
* **Didn’t follow Pipeline Model as per Industry**
* **Consolidated Reporting was not present**
* **Difficult to manage project and browser details within the job**
* **Automatic Scale up for execution platform was difficult**
* **File sharing across nodes was difficult**

****

4

Jenkins Node pool

New Architecture



XD



XD



XD



XD



Stakeholders



XD



XD



Trigger Pipeline Script

Perform Report Consolidation

Execution triggered on available node

**Task details for concurrent test execution in Jenkins pipeline using groovy script.**

|  |
| --- |
| **Executing On Online Node:** In freestyle job, if the targeted node is in offline state, job fails. In pipeline build we can execute different steps of workflow in different nodes. If any node is in offline state during execution, that particular step can be skipped and then next step will be executed without failing whole job. So it’s important to check whether the node is available or not before executing any step on that node. |
| *if(Jenkins.instance.getNode('Win7\_Node1').toComputer().isOnline()){*  *//script Block*  *}* |
| **Set custom Workspace for the Node:** Jenkins downloads source codes from repository to its node’s workspace. Sometimes we require dependent binaries to be present in workspace for the build. In that case binaries should not be checked out from version control. So It is required to specify Jenkins workspace to a shared location where dependent files are also present. |
| *def workSpacePath = "C:\\Jenkins\\Libraries"*  *ws("${workSpacePath}") {*  *//script Block*  *}* |
| **Parallel Execution in multiple nodes:**  **a)** We can execute same task in multiple nodes in parallel or triggering jobs and allowing them to run concurrently.  **b)** We can execute different tasks of pipeline in parallel with the aim of getting feedback sooner and making the most use of available resources. |
| *def parallelTasks = [:]*  *for(i = 0; i < 4; i += 1) {*  *def taskName = "task\_${i}"*  *parallelTasks["task\_${i}"]={*  *stage("task\_${i}"){*  *node('ParallelExecution'){*  *ws(“C:\\Jenkins\\Libraries”) {*  *def envname = "${env.NODE\_NAME}:Executor ${EXECUTOR\_NUMBER}"*  *println "BuildEnv:: ${envname}"*  *println "Before Wait:: ${taskName}"*  *sleep 20*  *println "After Wait:: ${taskName}"*  *}*  *}*  *}*  *}*  *}*  *parallel parallelTasks* |
| **Batch file Execution in nodes:**  A task to execute a .bat file in node. |
| *stage("Batch Execution"){*  *println "ProjectName is: ${ProjName}"*  *bat 'echo "This is a Batch script" '*  *bat returnStatus: true, script: "ant GS\_CI -DFCBP=YY -DOutputFileName=%NODE\_NAME%\_%BUILD\_NUMBER% -Dprojectname=${ProjName} -DBrowserslist=${BrwsrName} -DMaxTimeout\_min=6"*  *}* |
| **Powershell execution in node:** A task to execute .PS1 file in node. “PowerShell” plugin needs to be installed for this. |
| *stage("Executing On Online Node"){*  *if(Jenkins.instance.getNode('KousikNode').toComputer().isOnline()){*  *node('KousikNode'){*  *stage('InvokePowershell\_MergeXML') {*  *powershell returnStatus: true, script: "H:\\WS\\26-04-2018\\xmlmerge.ps1 "*  *}*  *}*  *}*  *}* |
| **Read properties file to get values from keys:** Read a properties file to get the values from keys. “Pipeline Utility Steps” plugin needs to be installed for this task. |
| *stage("Read Properties"){*  *def props = readProperties file:"\\CICD\\report\_path.properties"*  *reportFolder = props['REPORT\_FOLDER']*  *htmlReport = props['HTML\_REPORT']*  *println "REPORT\_FOLDER:: ${reportFolder}"*  *println "HTML Report:: ${htmlReport}"*  *}* |
| **Copy Files to a shared location:** Copy file task. |
| *stage("Copy Files"){*  *String sourceFilePath1 = "H:\\For Kousik\\Report1\\AllModulesMainAutomatedTestResultIE02\_26\_20183\_34\_05PM.html"*  *String destinationFilePath1 = "H:\\For Kousik\\Report1\\AllModulesMainAutomatedTestResultIE02\_26\_20183\_34\_05PM.html"*  *String destinationFilePath1=*  *"H:\\For Kousik\\FinalReport\\AllModulesMainAutomatedTestResultIE02\_26\_20183\_34\_05PM.html"*  *(new AntBuilder()).copy(file: sourceFilePath1, tofile: destinationFilePath1)*  *}* |
| **Stashing & UnStashing to use intermediate files during pipeline execution:** In a case where we want to use artifacts produced in earlier steps we can “stash” and “unstash” pre-made artifacts to save time. Stashing doesn’t keep artifacts permanently. Stash files can be unstash in entire pipeline workflow. |
| *node('Win10Master'){*  *stage("Custom WorkSpace"){*  *ws('H:\\For Kousik') {*  *stage("stash Result"){*  *stash includes: 'H:/For Kousik/FinalReport/\*\*/\*', name: 'TestResultHTML'*  *}*  *}*  *}*  *}*  *stage('MoveResult'){*  *node('KousikNode'){*  *ws('H:\\WS\\Workspace') {*  *dir('Report'){*  *unstash 'TestResultHTML'*  *}*  *}*  *}*  *}* |
| **Execute next stage after completion of all parallel Job:** All the steps of pipeline workflow is sequential except “Parallel” step. Next step after parallel step will be executed after completion of all concurrent jobs. So we can consolidate our test reports produced in all parallel jobs. |
| *stage("AfterParallelJobs"){*  *stage("Executing On Online Node"){*  *if(Jenkins.instance.getNode('KousikNode').toComputer().isOnline()){*  *node('KousikNode'){*  *ws('H:\\For Kousik') {*  *println "Hi All the parallel jobs have completed!!!!"*  *}*  *}*  *}*  *}*  *}* |
| **Merge xml:** After completion of all concurrent executions, output xml of each machine will be merged in a single xml using groovy. This task is already done using ANT and PowerShell. But it should be done in Groovy also to avoid dependency on other scripting languages.  In below script, We call xmlmerge.ps1 to merge all the xml presents in output report path where output xmls are already unstash. |
| *stage('InvokePowershell\_MergeXML') {*  *def ReportPath = "${env.WORKSPACE}\\report"*  *withEnv(['ReportPath = "${ReportPath}"']) {*  *println "env path: ${env:ReportPath}"*  *powershell returnStatus: true, script: "${env.WORKSPACE}\\xmlmerge.ps1 ${env:ReportPath}"*  *}*  *}* |
| **Merge xml Using Groovy:** |
| *import groovy.util.XmlParser*  *import groovy.xml.MarkupBuilder*  *import static groovy.io.FileType.FILES*  *node('master'){*  *def parser = new XmlParser()*  *def writer = new StringWriter()*  *def builder = new MarkupBuilder(writer)*    *def reportXml = []*  *new File("H:\\For Kousik\\Report\\kousik").eachFileRecurse(FILES) {*  *if(it.name.endsWith('.xml')) {*  *reportXml.add(it)*  *println it*  *}*  *}*    *println "reportXML: ${reportXml.get(0)}"*    *String fileContents1 = new File("H:\\For Kousik\\Report\\kousik\\testng-results.xml").text*  *String fileContents2 = new File("H:\\For Kousik\\Report\\Pinaki\\testng-results.xml").text*  *String fileContents3 = new File("H:\\For Kousik\\Report\\Tatha\\testng-results.xml").text*  *def xmlpath = []*  *xmlpath.add(fileContents1)*  *xmlpath.add(fileContents2)*  *xmlpath.add(fileContents3)*  *doc1 = parser.parseText(xmlpath.get(0))*  *def doc2 = null*    *for(i = 1; i < xmlpath.size(); i += 1) {*  *doc2 = parser.parseText(xmlpath.get(i))*  *doc2.test.each { test ->*  *doc1.append(test)*  *}*  *}*      *}*  *}* |
| **Count "PASS", "FAIL", "SKIP" Test Cases and Append in consolidated XML:** It is required to publish final test result in a pie chart. |
| *def passCount = 0*  *def failCount = 0*  *def skipCount = 0*  *def totalCount = 0*    *doc1.test.each { test ->*  *test.'class'.each{ classNode ->*  *classNode.'test-method'.each{ method ->*  *println "${method.@status}"*  *def status = "${method.@status}"*  *if("${method.@status}" == "PASS"){*  *passCount = passCount + 1*  *println "PASS case"*  *}*  *else if("${method.@status}" == "FAIL"){*  *failCount = failCount + 1*  *println "FAIL case"*  *}*  *else{*  *skipCount = skipCount + 1*  *println "SKIP case"*  *}*    *}*  *}*  *}*    *totalCount = failCount + passCount + skipCount*  *println "totalCount: ${totalCount}"*  *println "passCount: ${passCount}"*  *println "failCount: ${failCount}"*  *println "failCount: ${skipCount}"*    *doc1.@TotalCount = "${totalCount}"*  *doc1.@PassCount = "${passCount}"*  *doc1.@FailCount="${failCount}"*  *doc1.@SkipCount="${skipCount}"*    *println doc1*    *def xmlFile = "H:/For Kousik/Report/testng-merge.xml"*  *new XmlNodePrinter(new PrintWriter(new FileWriter(xmlFile))).print(doc1)* |
| **Publish TestNGReport**: Publish testing report using consolidated xml. |
| *stage("Publish TestNGReport"){*  *step([$class:'Publisher',reportFilenamePattern:*  *'\*\*/H:/GTAF\_Framework/GTAF\_SELENIUM\_V6.3/Libraries/CICD/TestNGReport.xml'])*  *}* |
| **Convert consolidated xml to html**: Task to convert XML to HTML using XSLT transform in Groovy or PowerShell |
| *stage("XSLTransform"){*  *node('Win7\_Node1'){*  *ws("${workSpacePath}") {*  *def path = "${env.WORKSPACE}\\report"*  *def NodeFilePath = createFilePath(path)*  *println "Node File Path: ${NodeFilePath}"*  *def xslPath = "${env.WORKSPACE}\\Final.xsl"*  *println "XSL File Path: ${xslPath}"*  *def xmlPath = "${NodeFilePath}\\Final.xml"*  *println "XML File Path: ${xmlPath}"*  *//def htmlPath = "${NodeFilePath}\\GroovyChart.html"*  *//println "HTML File Path: ${htmlPath}"*    *//File works only in the 'master' so path should be exist in 'master' don't depend where its executing*  *//def xslt= new File(xslPath).getText()*  *//readfile takes the path present in the node or master where it's currently executing*  *def xslt = readFile "${xslPath}"*  *println "XSL Content: ${xslt}"*  *// Load xml*  *//def xml= new File(xmlPath).getText()*  *def xml = readFile "${xmlPath}"*  *println "XML Content: ${xml}"*  *// Set output file*  *def html = new FileOutputStream("${workSpacePathMaster}\\Final.html")*  *//def html = new FileOutputStream(htmlPath)*  *//def html = new hudson.FilePath(channel, manager.build.workspace.toString() + "\\test.properties*    *// Create transformer*  *def transformer = TransformerFactory.newInstance().newTransformer(new StreamSource(new StringReader(xslt)))*    *//Out-Put html will only be created in master.So provided html path should be present in master*  *//Perform transformation*  *transformer.transform(new StreamSource(new StringReader(xml)), new StreamResult(html))*  *//transformer.transform(new StreamSource(new StringReader(xml)), new StreamResult(htmlPath))*  *}*  *}* |
| **PublishHTML**: Task to publish converted HTML report. |
| *stage("PublishHTML"){*  *node('master'){*  *publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: "${workSpacePathMaster}", reportFiles: 'Final.html', reportName: 'HTML Report', reportTitles: 'Test Result'])*  *//Execute this command in script console as sometimes browser security block the pie chart to show*  *System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "default-src 'self'; style-src 'self' 'unsafe-inline';")*  *}*  *}* |
| **Send email**: Senda finalHTML report to stake holders. |
| *node('master'){*  *stage('Send email') {*  *def mailRecipients = "Kousik.Manna@cognizant.com"*  *def jobName = currentBuild.fullDisplayName*  *println "${jobName}"*  *//emailext body: '''${SCRIPT, template="groovy-html.template"}''',*  *//emailext**body***:** *'''${FILE,path="FinalReport/AllModulesMainAutomatedTestResultIE02\_26\_20183\_34\_05PM.html"}''',*  *//mimeType: 'text/html',*  *//subject: "[Jenkins] ${jobName}",*  *//to: "${mailRecipients}",*  *//replyTo: "${mailRecipients}",*  *//recipientProviders: [[$class: 'CulpritsRecipientProvider']]*  *}*  *}* |
| **Start Jenkins slave in offline node remotely:**   1. UseJenkins API to make offline node online. 2. If Jenkins API is not able to make node online, we use power shell remote execution to start slave agent in remote node. |
| **Write XSLT in proper format using CSS:** |
| **Use google chart for pie chart in xslt:** Test result will be shown in a pie chart. For this task we will implement google chart inside xslt. |
| *<head>*  *<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>*  *<script type="text/javascript">*  *google.charts.load('current', {'packages':['corechart']});*  *google.charts.setOnLoadCallback(drawChart);*  *function drawChart() {*  *var data = google.visualization.arrayToDataTable([*  *['Task', 'Test Result for suite'],*  *['Total Passed Test Count::<xsl:value-of select="suite/@PassCount" />', <xsl:value-of select="suite/@PassCount" />],*  *['Total Failed Test Count::<xsl:value-of select="suite/@FailCount" />', <xsl:value-of select="suite/@FailCount" />],*  *['Total Skipped Test Count::<xsl:value-of select="suite/@SkipCount" />', <xsl:value-of select="suite/@SkipCount" />]*  *]);*  *var options = {*  *title: 'Test Result for suite',*  *pieHole: 0.4*  *//is3D: true*  *};*  *//var chart = new google.visualization.PieChart(document.getElementById('piechart'));*  *//var chart = new google.visualization.PieChart(document.getElementById('piechart\_3d'));*  *var chart = new google.visualization.PieChart(document.getElementById('donutchart'));*  *chart.draw(data, options);*  *}*  *</script>*  *</head>* |
| **One end to end execution:** end to end execution of POC by integrating all the tasks. |
| **Crete Node label in Jenkins**: This is a way to group similar slave nodes according to the test environment required by the build process. If we tag a job to a particular node and that node is not available then the job will not trigger. But if we tie a job to a group of nodes using label expression then Jenkins will trigger the job in the currently available node. |
| *//////Crete Node label ParallelExecution in Jenkins to group similar slave nodes according*  *//////to the test environment required by the build process.*  *node('ParallelExecution'){*  *//Script Block*  *}* |
| **Check if #ofJobs Triggered > #of available nodes in a label:** Wecan triggerall the parallel jobs without knowing the number of available nodes in a particular label expression. If we trigger more jobs than available nodes, Jenkins will take care the execution without failing. |
| **Passing argument to PS1 file from groovy:** We need to pass test output path location to a PowerShell file as argument from pipeline groovy script. |

**HTML Out-put**

