Jenkins Shared Library

Most of applications have the same steps to build, test & deploy the application. So instead of repeating those steps for every application, we can actually create a shared library, which can be used across applications.

Creating Jenkins Shared Library

Jenkins Shared Libraries are usually pushed into a separate Git repository & checked out by Jenkins job during the job run.

In brief, here are steps to create & use the Jenkins shared library:

Step 1: Create vars folder

Create a separate project directory. Inside the project directory, create a directory called vars, which will host the shared library's source code

Step2: Create shared library files with extension .groovy

Step 3: Create call() function inside Groovy file

```
def call(String name) {
        echo "Welcome, ${name}."
}
```

When a shared library is referred from the Jenkins job, Jenkins, by default, will invoke the call() function within our Groovy file. Consider the call() function like the main() method in Java. We can also specify parameters for the call() function if we want to.

Steps to configure Jenkins pipeline library:

Step 1: Open Jenkins in your browser

Step 2: Click on manage jenkins on the left side menu.

Step 3: Click on configure system. Scroll down until you find the Global Pipeline Libraries section.

Step 4: Under the Library section, configure values as below.

Name first-shared-lib (remember, we will refer to this shared library from Jenkins pipeline using this name).

Default version (branch name of our Shared Library git repo).

Under the Retrieval method, choose Modern SCM.

Under Source Code Management, choose Git.

Enter your Pipeline Shared Libraries repo URL under Project Repository

Configure credentials if your repo requires credentials to checkout code

Step 5: Click on the save button.

Referring Jenkins Shared Library from Pipeline

Steps to refer Jenkins pipeline library:

Step 1: In Jenkins, click on New Item on the left side menu, enter Jenkins job name & choose the pipeline as style & click on the OK button.

Under the pipeline section, you can see a script textbox where we can manually copy, paste the contents of our pipeline.

```
@Library('first-shared-lib') _
    welcomeJob 'RAM'
```

Note: _ (underscore) is a must after the @Library annotation.

@Library will import the shared library to our Jenkins job.

Remember first-shared-lib is the name we gave while configuring shared libraries in Manage Jenkins.

We can access multiple libraries with one statement

```
@Library(['my-shared-library', 'first-shared-lib']) _
```

welcomeJob will invoke the call() function in welcomeJob.groovy created under vars folder.

The 'RAM' string will be sent as a parameter to the call() function

Step 2: Click on the Save button.

Step 3: Click on Build Now button on the left side menu to trigger the build.

Step 4: Once the build is complete, check the logs by clicking on the Console Output from the left side menu.

You should see the Welcome, RAM message in the Console Output.

You can use multiple functions in shared library file filename is multiplefunctions

```
def getCurrentTimestamp() {
   return new Date().format("yyyy-MM-dd HH:mm:ss")
}
def getCurrentBuildNumber() {
   return env.BUILD_NUMBER
}
def addNumbers(int a, int b) {
return a + b
}
```

<u>You can call multiple functions inside the script blog</u>

```
@Library('libraries') _
    pipeline {
       agent any
           stages {
              stage('Stage 1') {
                 steps {
                    script {
                          multiplefunctions.getCurrentTimestamp()
                       }
                     }
                }
              stage('Stage 2') {
                         def result = multiplefunctions.addNumbers(10, 20)
                         echo "Sum of 10 and 20 is: ${result}"
                       }
                   }
              }
```

multiplefunctions is the file name and getCurrentTimestamp is the method name

<u>You can add entire pipeline into this groovy file</u>

Create a jenkins shared library file with name test.groovy

```
def call(Map pipelineParams) {
    pipeline {
      agent {
        label pipelineParams.agent
      }
      stages {
          stage('Stage 1') {
                steps {
                      script {
                            gitCheckout(branch: pipelineParams.branch,  url: "pipelineParams.url")
                       }
                 }
            }
        }

      }

  }
```

Call this file insidethe pipelinecode

```
@Library('libraries') _

test(branch:'main',url:'https://github.com/aadityapatill/PracticeRepo')
```

This way the the pipeline code can adapt to different situations and be reused effectively.

- The provided code demonstrates the flexibility of a Jenkins shared library by allowing users to pass parameters based on their specific requirements.
- Users can easily customize the behavior of the shared library functions by providing different parameter values when calling them within their pipelines.
- This parameterization ensures that the code remains adaptable and reusable across various projects and scenarios.