Continuous Integration with LabVIEW & Jenkins: Web Service Implementation

Contents

[Introduction 2](#_Toc483166567)

[Setting up the CI Server 3](#_Toc483166568)

[Configuring LabVIEW 3](#_Toc483166569)

[Installing Jenkins 4](#_Toc483166570)

[Launching Jenkins 4](#_Toc483166571)

[Installing Plugins 4](#_Toc483166572)

[Setting up Email Notification 6](#_Toc483166573)

[LabVIEW CI Installation 6](#_Toc483166574)

[Installing the LabVIEW CI Web Service 6](#_Toc483166575)

[Pipeline 8](#_Toc483166576)

[Hello World Pipeline 8](#_Toc483166577)

[Simple LabVIEW Build Test 10](#_Toc483166578)

[Overview of Pipeline Job Architecture 11](#_Toc483166579)

[Building Pipeline Job from Jenkinsfile 13](#_Toc483166580)

[Scripting Jenkinsfile 13](#_Toc483166581)

[Pipeline Stages and Steps 15](#_Toc483166582)

[Configuring Global Library for Groovy Scripts 17](#_Toc483166583)

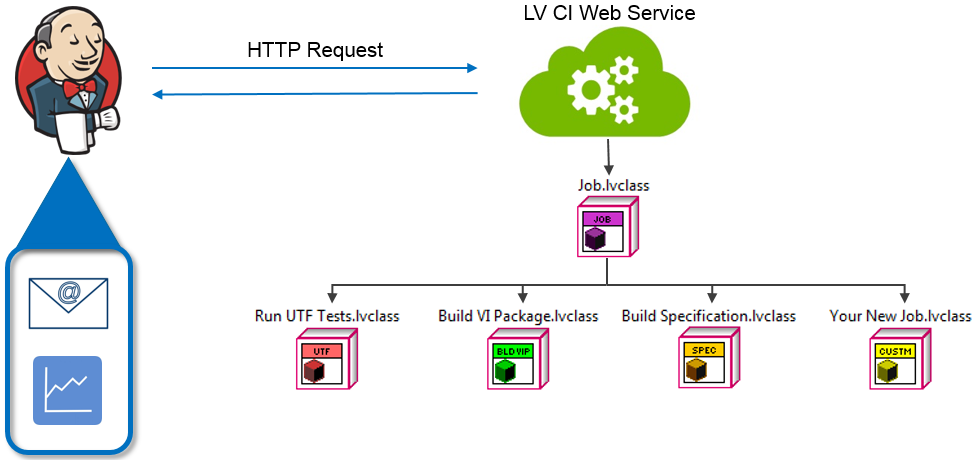
[Pulling It Together 19](#_Toc483166584)

[Known Issues 19](#_Toc483166585)

# Introduction

Continuous Integration (CI) is not a new concept; it is however somewhat new within the LabVIEW ecosystem. There have been several CI approaches over the last couple years, and this web service system is an attempt to merge the best components we have seen into a single extensible framework.

At a high level, this system has two main components: Jenkins and the CI Web Service.



Jenkins is an open source CI program and will handle the following:

* + Communication with your source code management software
  + Organizing build processes through the Pipeline plugin
  + Sending HTTP requests to the CI Web Service
  + Other functionality through Jenkins’ many plugins

The CI Web Service is built using LabVIEW Object Oriented Programming and can be extended by creating a new child of the abstract “Job” class. When the service receives a request, it loads the class requested and executes that classes functionality (build, run a VI, etc), then responds to the HTTP request when complete.

**Note:** The LabVIEW CI Service requires LabVIEW 2014 or later.

The remainder of this guide will walk you through setting up your CI Server (LabVIEW, the CI Web Service, and Jenkins & its plugins) as well as creating some simple Jenkins jobs.

# Setting up the CI Server

## Configuring LabVIEW

The CI Server needs a copy of the LabVIEW development environment to build applications. Once you’ve installed LabVIEW (remember, 2014 or higher), you must change the LabVIEW Application Web Server port. The default port for LabVIEW’s Web Service is 8080, which is also the default port that Jenkins uses. In theory, you could change the Jenkins port, but we find it easier to move LabVIEW.

This port conflict is known to cause problems such as the Jenkins page not being able to load. To rectify this, the port of the LabVIEW Web Manager needs to be changed.

1. Open LabVIEW, and from the toolbar, navigate to Tools>>Options.
2. Under the Web Service tab, click the “Configure Web Application Server” button. This will launch the LabVIEW Application Web Server in an internet explorer window.
3. Under System Web Server and Application Web Server, verify that the HTTP or HTTPS ports are **NOT** 8080. If any are currently set to 8080, change them to another unused port, such as 9090, as seen in Figure 1. If you change the port, you will need to reboot your system for it to take effect.

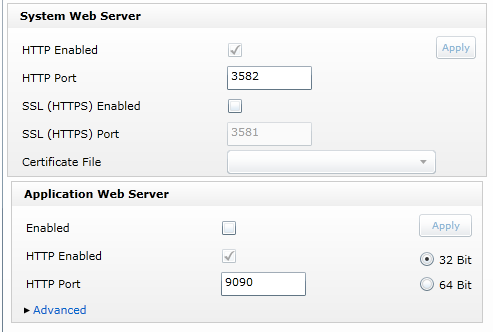


Figure 1: LabVIEW Application Web Server

**Important note:** The LabVIEW CI Server does **not** use the LabVIEW Application Web Server port. The only reason we are changing the Application Web Server’s port number is to clear that port for Jenkins. The CI Server uses port 8002 instead (this is a hard-coded value). Any calls to the CI Server should use port 8002, and not the value you selected for the Application Web Service.

## Installing Jenkins

1. Navigate to <https://jenkins-ci.org/>
2. Download the latest or desired version.
3. Once downloaded, run the installer and use the following directions, https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service.

## Launching Jenkins

Jenkins runs as a Windows service, and is therefore already running. We access the client GUI through a web interface via the following:

1. Open a browser and navigate to the default URL: <http://localhost:8080>
2. If this is the first time that you are setting up Jenkins, copy the password stored in the file C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword and enter that into Jenkins.
3. This will lead you to the Jenkins Setup Wizard. You can choose to install the suggested plugins, but ensure that you also have the plugins listed in the “Installing Plugins” section.

## 

## Installing Plugins

Jenkins, as with most CI applications, is built as a plug-in framework. Jenkins has a large community of active users developing plugins and will be of varying quality. It is recommended you read the plugin’s documentation and understand its intended functionality prior to installation. Below is the list of plugins, and process for installation, that we will discuss throughout the remainder of this guide.

To install the plugins manually:

1. Navigate to Manage Jenkins from the left-side menu:



Figure : Jenkins Dashboard Menu

1. Navigate to Manage Plugins:

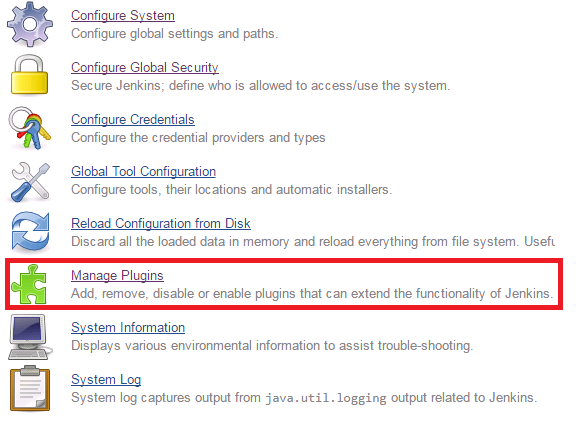


Figure : Manage Jenkins Options

1. Click on the Available Tab.
2. Search for and select:
   1. HTTP Request Plugin
   2. Job DSL Plugin
   3. Run Condition Extras Plugin
   4. Windows Slaves Plugin\*
   5. Workspace Cleanup Plugin
   6. (Whatever Source Code Control you use)

For the Git repository:

* + 1. Git plugin
    2. GitHub Organization
  1. Green Balls
  2. Junit Plugin
  3. Pipeline, and including:
     1. Pipeline: Basic Steps
     2. Pipeline: Groovy
     3. Pipeline: Job
     4. Pipeline: SCM Step
     5. Pipeline: Shared Groovy Library
  4. Pipeline Stage View Plugin
  5. Groovy

\*Note: The Windows Slaves Plugin depends on one or more of these other plugins. You will not be able to find or select this plugin until you have installed the others, and restarted Jenkins.

1. Select “Download now and install after restart.”

## Setting up Email Notification

In a few sections, you’ll start creating jobs and you may want these jobs to be able to report failures. For this to work, you need to provide Jenkins with some basic information by performing the following:

1. Navigate to Manage Jenkins.
2. Select Configure System.
3. Scroll down to the “Email Notification” section (usually all the way at the bottom) and enter your company’s SMTP server and default user email suffix.

**Note**: If adding a default suffix to your username wouldn’t result in a valid email address, you can specify an address for each username within the user settings. In that case just leave the suffix field blank.

1. Test your SMTP server setting by checking the box for “Test configuration by sending test e-mail”.
   1. This will add a “Test E-Mail Recipient” field. Put your address in the field and click the “Test Configuration” button.
   2. Jenkins will tell you if it successfully contacted the server and, if successful, you should receive a simple email from Jenkins. Click Save.

# LabVIEW CI Installation

The LabVIEW CI Web Service runs on the build service machine, and processes build step requests from Jenkins. Since Jenkins and the CI Web Service communicate via HTTP, you could place them on separate machines. This architecture can be advantageous if you need to build across multiple versions of LabVIEW and want to keep those versions separate, or if you want to have multiple build workers available. Jenkins also provides a layer for this type of distributed build functionality called [agents](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds). For simplicity, we’ll set them up on the same machine.

The CI Web Service was built as an extensible framework, and components are distributed as VI Packages.

## Installing the LabVIEW CI Web Service

**Before you begin:**

**Installation of CI Web Service:**

1. Open VI Package Manager (as an administrator) and install the “NI CI Server Source.”
2. Once installed, navigate to the installation folder: <applications folder>\National Instruments\LabVIEW Continuous Integration\Server Source\LabVIEW CI Service
3. Open LabVIEW as an administrator and open LabVIEW CI Service.lvproj.
4. Open and run Build Server.vi. This VI builds the various components of the CI Server. Build Server.vi will automatically launch the installer for the CI Web Service. Follow the instructions in the installer
5. After installing web service, you will be presented with a setup dialog, as shown.

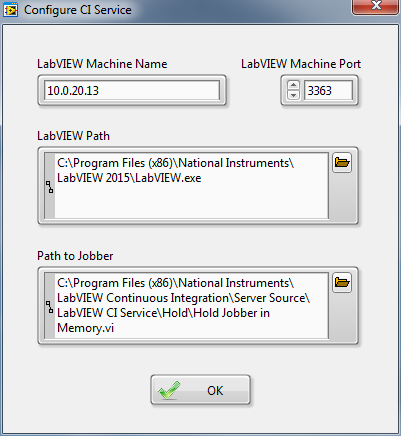


Figure : Configure CI Service Dialogue

**Note:** The service may default your LabVIEW path to 2014. If using a newer version, ensure the path matches your current version. The default location for “The Hold Jobber in Memory.vi” is shown in the dialog. Confirm that this is the location on your computer as well.

1. You should have an icon on your desktop for the LabVIEW CI Web Service. Launch the service. Depending on your security configuration, you may need to launch the service as an administrator.
2. Restart Jenkins.

# Pipeline

Jenkins Pipeline is a suite of plugins, which supports implementing and integrating continuous delivery pipelines into Jenkins. The default interaction model with Jenkins, historically, has been very web UI driven, requiring users to manually create jobs, then manually fill in the details through a web browser. This requires additional effort to create and manage jobs to test and build multiple projects, it also keeps the configuration of a job to build/test/deploy separate from the actual code being built/tested/deployed. This prevents users from applying their existing CI/CD best practices to the job configurations themselves. With the introduction of the [Pipeline plugin](https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Plugin), users now can implement a project’s entire build/test/deploy pipeline in a Jenkinsfile and store that alongside their code, treating their pipeline as another piece of code checked into source control. The plugin contains features such as:

* The ability to suspend/resume of executing jobs.
* Checking the pipeline definition into source control (Jenkinsfile)
* Support for extending the domain specific language with additional, organization specific steps, via the "[Shared Libraries](https://jenkins.io/doc/book/pipeline/shared-libraries/)" feature.[[1]](#footnote-2)[[2]](#footnote-3)

For the reasons stated above, we also chose to use pipelines to create our jobs in Jenkins. For more information on pipelines, please see <https://jenkins.io/doc/book/pipeline/>. We will start this tutorial with a simple "Hello World" example of a Jenkins Pipeline.

## Hello World Pipeline

For this tutorial, we will start out with a simple Jenkins Pipeline job that will display Hello World to the user when it has successfully built. To create a new pipeline job, use the following steps:

1. Choose **New Item** from the Jenkins dashboard on the left side.
2. Title the job, "HelloWorld," and select a Pipeline, as seen in the following figure, and click **Ok.**

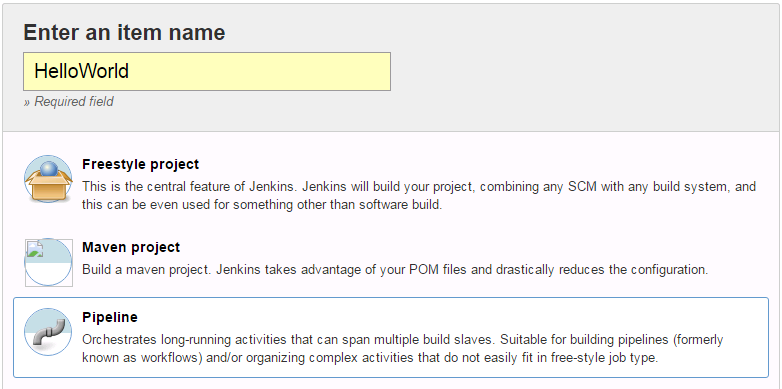


Figure : "HelloWorld" Pipeline Job Setup

1. Underneath the Pipeline heading, you will have the option to either directly write a script or call a Jenkinsfile from a source code repository. For the HelloWorld pipeline, choose the option **Hello World,** as seen in the following figure:

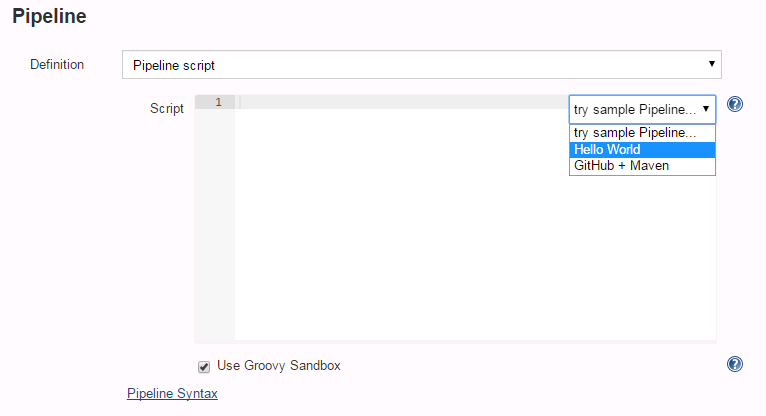


Figure : Pipeline Definition in “HelloWorld” Pipeline Job

1. Click **Save.** This will lead you to the HelloWorld pipeline job's dashboard.
2. To build your Pipeline, select **Build Now** from the left-hand menu. You will see the Build History on the left populate. For your first build, you should see "#1."
3. Click on "**#1**" to see the build's status. From the left side, choose **Console Output.** The Console Output displays the status and any echo commands from the Jenkins pipeline. It is a handy tool for debugging. You should see a similar output as the following, the workspace will be unique to your server:

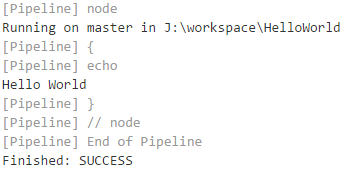


Figure : "HellowWorld" Pipeline Console Output

## Simple LabVIEW Build Test

Now that you are familiar with pipelines, we can test it out with the CI Service. The CI service communicates with LabVIEW through LabVIEW web services. To test whether the CI Service is properly working, use the following steps:

1. In LabVIEW, create a project. Create an object in build specifications. Make sure that it successfully builds.
2. In Jenkins, click a **New Item**, choose **Pipeline**, and title it “LVBuild\_Test.”
3. Use the following script for the pipeline definition:

|  |
| --- |
|  |
|  |  |
|  | #!/usr/bin/env groovy  import groovy.json.JsonOutput  { |
|  | echo 'Build the package' |
|  | //The following line defines the JSON string expected by the LabVIEW Build method of the CI service. Please replace all the variables to match your system and build specification. Executor numbers are defined from an environment variable. |
|  | def lvBuild\_json = JsonOutput.toJson([Executor\_Number: env.EXECUTOR\_NUMBER.toString(), Workspace\_Path : “C:\\ LVBuild\_Test”, 'Build\_Spec': “your\_specification\_name”,'Target': “your\_target\_name”,'Project\_Path': " C:\\LVBuild\_Test\\your\_project\_path","Auto\_Version\_Increment": true]) |
|  | echo lvBuild\_json |
|  |  |
|  | //The following lines post the response from the CI Service. A successful build will return code 200. It will return the error along with error code 500. Error code 408 indicates that the CI service has not been started.  def lvb\_response = httpRequest validResponseCodes: "200,500", url: "http://localhost:8002/LabVIEWCIService/LabVIEW\_Build?JSON="+java.net.URLEncoder.encode(lvBuild\_json, "UTF-8").replaceAll("\\+", "%20") |
|  | println("Status: "+lvb\_response.status) |
|  | println("Content: "+lvb\_response.content) |
|  | if (lvb\_response.status!=200)  { |
|  | error("Call to CI Server method LabVIEW\_Build failed with error: "+lvb\_response.content) |
|  | } |
|  | echo 'Magic wait of 5 seconds...' |
|  | sleep(5) |

1. To test the code, start the LabVIEW CI Service and “Run as Administrator.”
2. Trigger a build in Jenkins, by hitting **Build Now.**
3. Look at the console output to see the result of your build.

This example demonstrates a simple job that calls the LabVIEW CI Service to invoke LabVIEW to build any build specification. If pipelines were not used, a job needs to be created for each single test or build and this code would have to be duplicated over and over. However, leveraging the benefits of pipelines, the codebase is reduced to a single script for each function explained in the following sections.

## Overview of Pipeline Job Architecture

The “Hello World” and “LabVIEW Build” example was a simple introduction into pipelines. All the code necessary for execution was in the pipeline job configuration. We saw that the pipeline example echoed back “Hello World” to the console when the user ran the build. This method can be cumbersome because the user must manually update the configuration on the Jenkins server each time the test code would require maintenance.

Instead the test developer can opt to define a pipeline from an SCM. This pipeline definition is known as a Jenkinsfile. A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. This is the foundation of "Pipeline-as-Code"; treating the continuous delivery pipeline a part of the application to be version and reviewed like any other code.[[3]](#footnote-4) Therefore, the Jenkinsfile is treated like any other piece of code.

A Jenkinsfile is contained in the root of each repository. In this tutorial’s example, it is simple and it does not contain the low-level code for a pipeline build and execution. It calls into pipeline function, which contains the stages for the job (There will be more detail provided on this in the next sections). Each stage then calls the low-level code to execute an action, such as checkout from source code manager, invoke a build specification or run unit framework tests. This approach is seen in figure 9. There are three distinct levels in a pipeline job – a Jenkinsfile than calls the pipeline configuration for the repository, the pipeline code that separates the job into multiple steps, and the low-level code to execute the job’s action.

Figure : Single Repository CI Architecture

UTF\_Test

Jenkinsfile

Run\_VI

Define what steps to run

Pipeline is hard-coded for one repository

Define a step

Lowest-level building block

Reused across many companies

LV\_Build

However, if there is only one repository in the organization, it is not necessary to have the three different levels. The Jenkinsfile can define the stages and call the low-level code, seen in figure 8. For the multiple repository case, the purpose of separating the stages into a separate script from the Jenkinsfile is to make it reusable among many Jenkinsfiles and repositories. It also becomes easier to edit and update; the user will only have to modify one location and have the change propagate to all repositories, rather than modify multiple instance of the same code.

UTF Test

Pipeline

SCM Checkout

Jenkinsfile

Jenkinsfile

Jenkinsfile

Sets pipeline parameters

(Project path, Build Specification, etc.)

Trivially Simple

One per repository

Define what steps to run

Reused across many projects

Define a step

Lowest-level building block

Reused across many companies

LV Build

Figure 9: Multiple Repository CI Architecture

## Building Pipeline Job from Jenkinsfile

As discussed earlier, a Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile, is the same, it’s generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.[[4]](#footnote-5)

In this tutorial, GitHub was used (<http://github.com/roxanakarami/TestRepository>) as a source code management. However, a Jenkins Pipeline is configurable with any source code control. For a more in depth look at Jenkinsfile, please visit [here](https://jenkins.io/doc/book/pipeline/jenkinsfile/). Use the following steps to configure a pipeline job with a Jenkinsfile:

1. Create a new Pipeline job by choosing **New Item** from the main Jenkins dashboard. Choose Pipeline and name the job accordingly.
2. When configuring a pipeline job, under the Pipeline definition, instead of writing a Groovy script in the script editor, we can choose to use a Jenkinsfile script from a source code management, seen in the following figure:

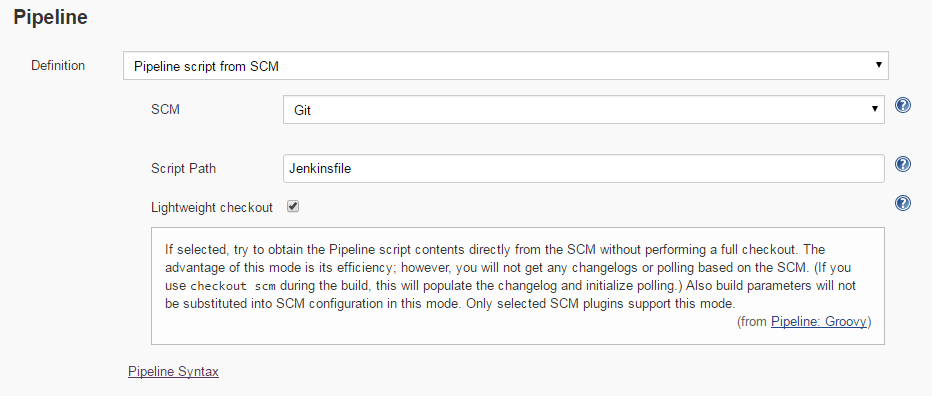


Figure : Jenkinsfile Configuration for a Pipeline Jobs

**Note:** This Jenkins file must be in the root of the SCM repository.

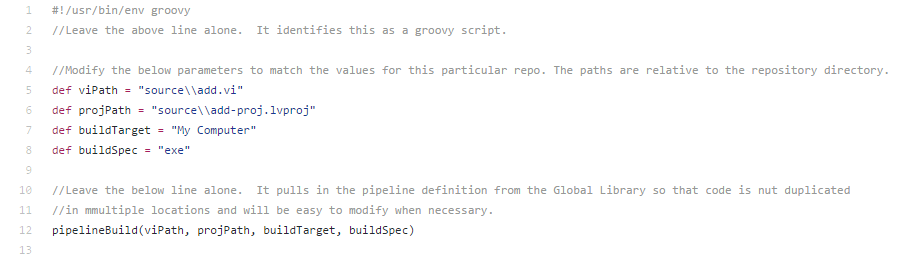
1. The pipeline job created will use the Jenkinsfile located in your SCM repository to carry out the build.

## Scripting Jenkinsfile

The Jenkinsfile and all other scripts for the pipeline jobs are written in [Apache groovy](http://groovy-lang.org/), an object-oriented programming language for the Java platform. Therefore, every Jenkinsfile must have “#!/usr/bin/env groovy” in the first line.

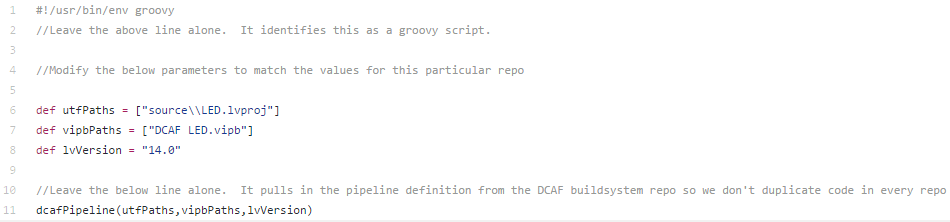
The Jenkinsfile can be easily be customized for each different repository and project. The method that was used for the tutorial example and the LabVIEW DCAF project.

The Jenkinsfile we have created calls the appropriate functions that can perform the builds, rather than the code for the pipeline job. For example, the following Jenkinsfile makes a call to the pipelineBuild method with defined parameters seen in the following figure:



The pipelineBuild() method, which is contained in the global library, will be called by Jenkins when this pipeline job is initiated. The defined parameters will be unique for each Jenkinsfile, i.e. each repository. When the pipeline job begins, Jenkins will pass in the parameters to the pipelineBuild() method.

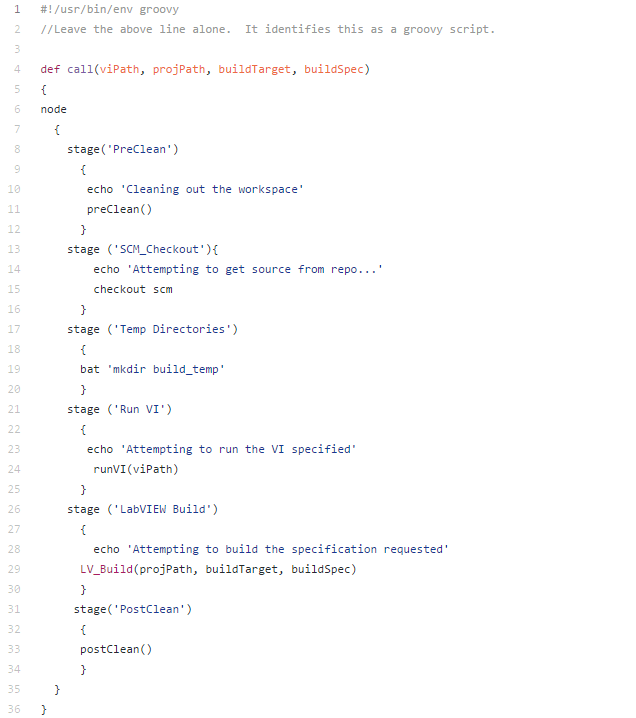
The purpose of reserving the Jenkinsfile exclusively for function calls is to avoid unnecessary code duplication and minimize developer efforts. When the code for the pipeline job exists within the global library, any changes to it will propagate to any Jenkinsfile that relies on that function. The test developer will not have to duplicate efforts by modifying the code in all Jenkins files, but rather the code contained in the global library. Another example of a Jenkinsfile is the following used in the LabVIEW DCAF LED repository, seen in the following figure:



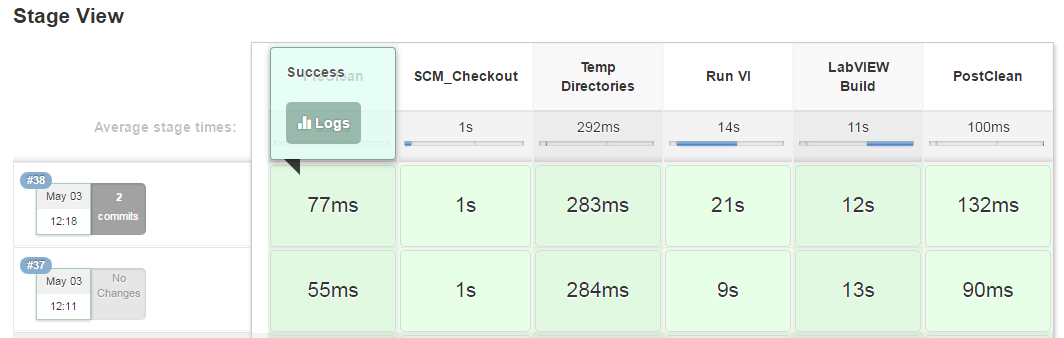
Again, the same principle was used here. This Jenkinsfile calls into the dcafPipeline() method with parameters specific to the LED repository.

## Pipeline Stages and Steps

As seen in the example for TestRepository, the Jenkinsfile invokes the pipelineBuild() method. The code is seen in the following figure:



The pipelineBuild script does not contain the code for the build either – the build is defined in three layers. However, it defines the stages for the Jenkins build job and calls the methods for the stage to execute. A [stage](https://jenkins.io/doc/book/pipeline/syntax/) step is a primary building block in Pipeline, dividing the steps of a Pipeline into different sections and creates a visualization of the progress using the "Stage View" plugin. When the pipeline builds, it will divide your test into the different stages as seen in the figure below:



In the Stage View, seen above, each block corresponds to a stage defined in the piplinebuild() script.

Again, note that each step contains another function call rather than containing the code that performs each step. This is like the Jekinsfile structure. Each stage calls into the necessary separate functions, making the pipeline job easy to maintain and avoids code duplications. Multiple jobs could access these functions that exist within the global library. It is possible to include all the code within this pipeline job as well. However, for the reasons stated above, we chose to only include function calls in the pipeline build script. The separate functions, such as RunVI and LVBuild, are all different scripts that exist in the global library.

These functions are wrappers for the functions that the LabVIEW CI Service can perform. The following sections cover these methods. Again, for the DCAF pipeline, a similar methodology was used.

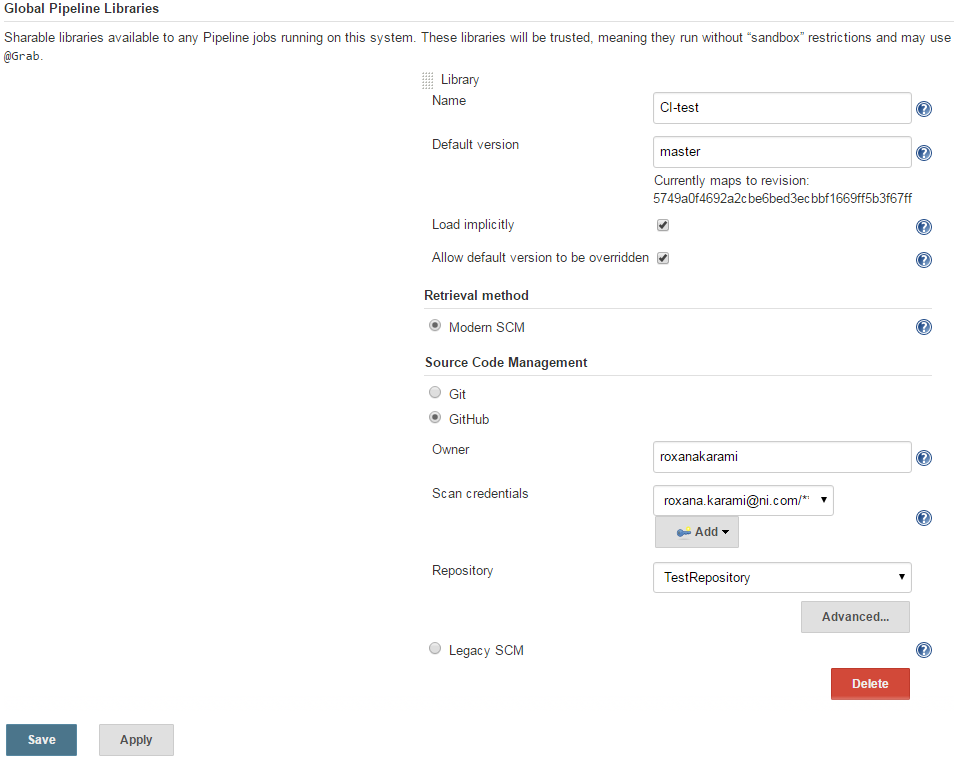
## Configuring Global Library for Groovy Scripts

**Note**: “Pipeline: Shared Groovy Libraries” is the required plugin for this functionality. z

The global library contains all the groovy scripts necessary for each separate method that a Jenkinsfile or any other groovy scripts may call. These are methods to Run VIs or Build any pre-defined items in a project’s build specification. Multiple Jenkinsfiles or other functions can invoke these scripts if Jenkins has access to these files from a source code repository.

Jenkins will need to be configured so that it can load these scripts automatically and use them for any Jenkinsfile:

1. From the main Jenkins Dashboard, navigate to **Manage Jenkins>>Configure System.**
2. Underneath the heading, Global Pipeline Libraries, you can point to the library of these functions. The following figure shows the setup in Jenkins for the example repository located at github.com/roxanakarami/TestRepository.

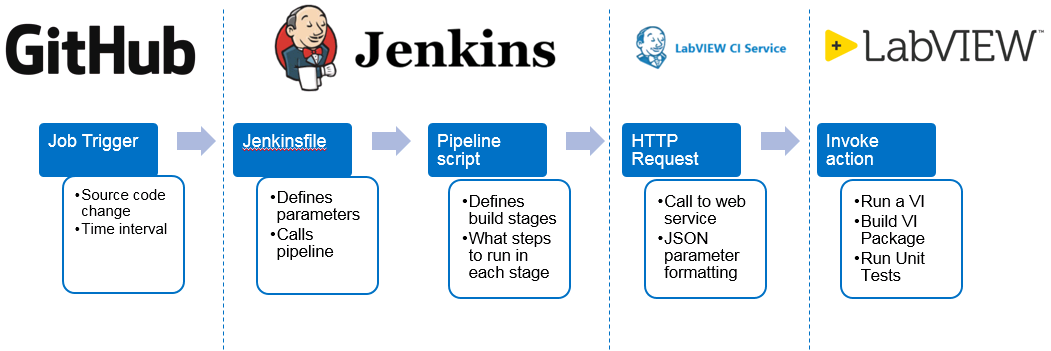


1. It is recommended that you leave the Retrieval method as Modern SCM.
2. Jenkins will autopopulate with your SCM if not using Git or Github with the correct plugins. Configure these settings so that Jenkins points correctly to your repository.
3. You may add multiple sets of global libraries, but beware of namespace conflicts.

**Note**: that these global scripts must be contained in a folder entitled “vars” or “src” in a source code repository. You will receive a build error otherwise.

Examples of this can be seen in the example tutorial, github.com/roxanakarami/TestRepository, or at <https://github.com/LabVIEW-DCAF/buildsystem>.

## Pulling It Together



The figure above outlines the entire the continuous integration process. Github, or any SCM, triggers a pipeline job a change is committed. A job can also be manually triggered or trigged at a specified time interval. The Jenkinsfile defines the build parameters an invokes pipeline scripts when a build is initiated. The low-level code uses http requests to communicate with the LabVIEW CI Service, which invokes LabVIEW to run VIs, run unit tests, etc.

Once you have your code written, use the following steps to run your builds.

1. Configure a pipeline job in Jenkins and create a Jenkinsfile in your repository.
2. Configure a shared library and use any groovy scripts that are necessary for your build from https://github.com/LabVIEW-DCAF/buildsystem/tree/lv-ci-service/vars.
3. Start the LabVIEW CI Service and “Run as Administrator.”
4. Initiate the job in Jenkins, and look at the console output for the build results.

## Known Issues

**Error 56**

This is a timeout error that the LabVIEW CI Service. Please restart your computer. Once your computer has restarted, give the computer a few minutes to start up. If you do not, this can lead to the time out error. Then start the service.

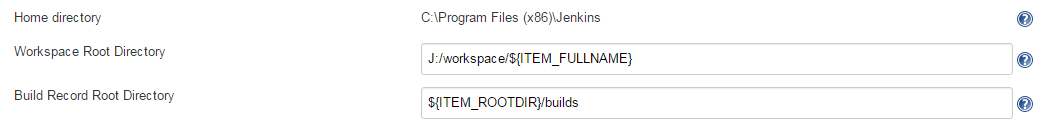
If this does not work, reinstall the service. Navigate to section “Installation of CI Web Service” and start at number 2.

**JSON String longer than 512 characters**

If the JSON string is longer than 512 characters, then the build will fail. This is a known issue with LabVIEW web services. The JSON string must be less than 512 characters.

One way to map a drive to the Jenkins workspace directory. For example, you can map the J: drive to the Jenkins workspace directory to shorten the path and to decrease the amount of characters assed to the SI Service in the http request. To do this, use the following steps:

1. Open regedit.exe.
2. Navigate to KEY\_LOCAL\_MACHINE \ SYSTEM \ CurrentControlSet \ Control \ Session Manager \ DOS Devices
3. Add a new REG\_SZ value and name it J: where J is your drive letter
4. The value should be the path in this form: \DosDevices\C:\Program Files (x86)\Jenkins.
5. Restart your computer. Confirm that the J: drive points to C:\Program Files (x86)\Jenkins.
6. Navigate to Jenkins Dashboard and click on **Manage Jenkins>>Configure System.**
7. Under the “Home Directory” function, Click Advanced.
8. Modify your Workspace Root Directory to match the following:



1. Now the workspace path will be reduced to just “J:” rather than C:\Program Files (x86)\Jenkins\workspace, reducing the length of your JSON string.

1. https://jenkins.io/solutions/pipeline/ [↑](#footnote-ref-2)
2. https://jenkins.io/doc/book/pipeline/ [↑](#footnote-ref-3)
3. Jenkins reference [↑](#footnote-ref-4)
4. https://jenkins.io/doc/ [↑](#footnote-ref-5)