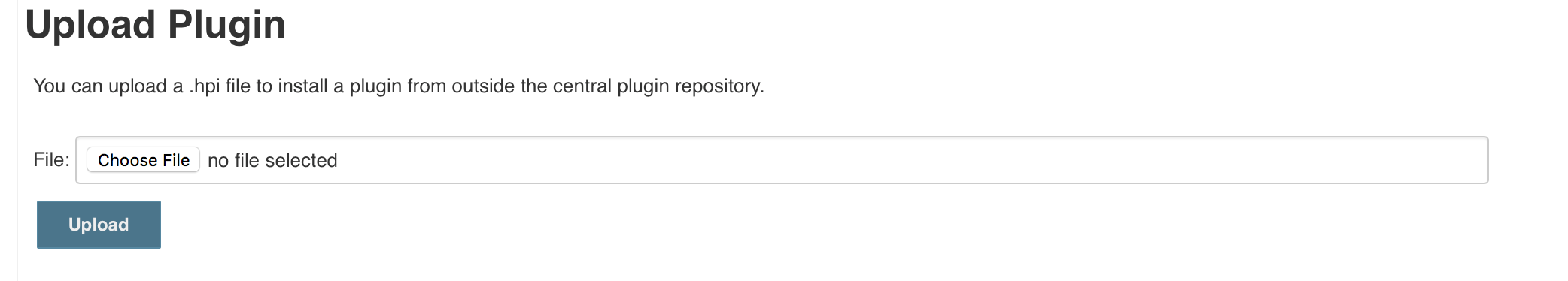
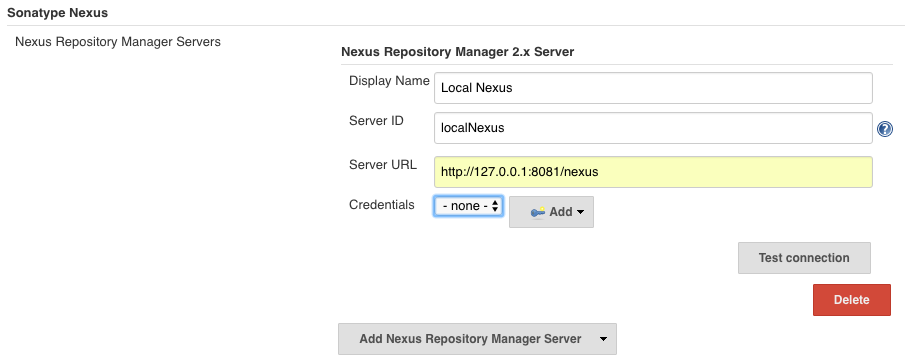
**Plugin Installation**

Before you get started, you must first download and install the Nexus Jenkins Plugin from [Sonatype Downloads](http://download.sonatype.com/nexus/ci/latest.hpi). Then from the Jenkins dashboard, navigate to **Manage Jenkins -> Plugin Manager**, proceed to the **Advanced** tab, and upload the downloaded HPI using the Upload Plugin form shown below.



**Configure System**

Once you have the plugin installed, the next thing you need to do is configure a Nexus Repository Manager to be able to upload your build artifacts. Once again click on the **Manage Jenkins** link from the dashboard and then the **Configure System** link. Under the **Sonatype Nexus** heading select **Nexus Repository Manager 2.x Server** from the **Add Nexus Repository Manager Server**dropdown.

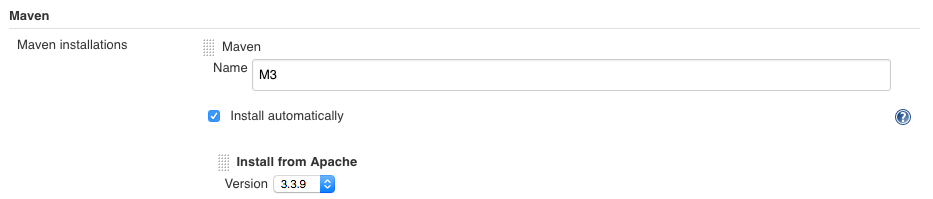


Display Name is the name of the server you would like shown when selecting Nexus Repository Manager instances for build jobs. Server ID is a unique ID which is used to reference the Repository Manager in Build Pipeline scripts, it should be alphanumeric without spaces. The Nexus Jenkins Plugin uses Jenkins credentials provider to manage server credentials. To authenticate the connection, use the Add dropdown to create a credentials entry and then select it from the Credentials dropdown. After entering the information for your Nexus Repository Manager, and click the **Test Connection** button to ensure connectivity to your Nexus Repository Manager installation.



**Tool Configuration (Optional)**

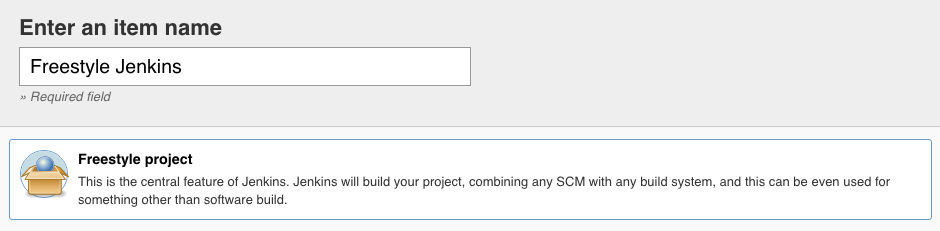
For the following example you'll use Maven to build in your project. In order to do so, you need to ensure that you have a Maven installation configured in Jenkins. To do so, navigate to the **Manage Jenkins -> Global Tool Configuration** and configure a Maven installation as shown below.

****

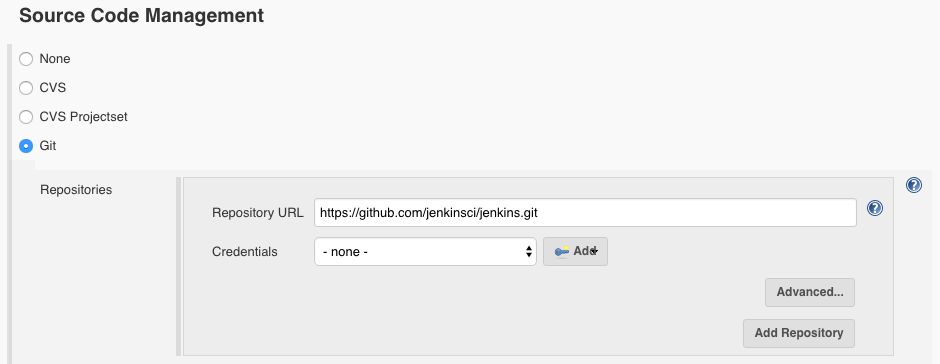
**Freestyle Build**

**Freestyle Build Step Configuration**

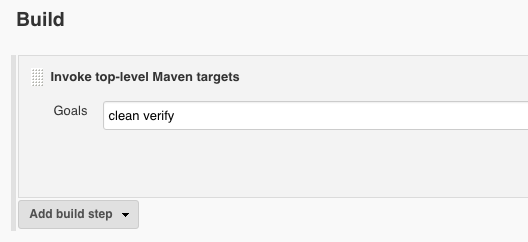
The Nexus Jenkins plugin supports traditional Freestyle jobs. For this example we will build Jenkins within Jenkins (inception!) and publish it to our local Nexus instance setup in the previous steps. To start, create a **Freestyle Project** by clicking on the **New Item** link on the Dashboard. Give the project a name and click **OK** to continue with the configuration.



Setup the Freestyle project as you would normally, in this example we will use Git as our SCM.

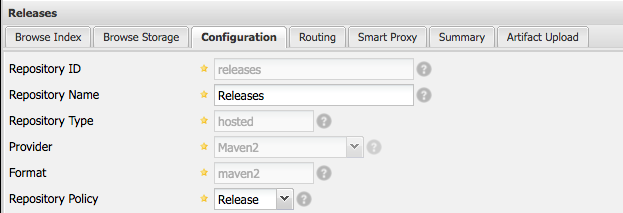


Add a build step for **Invoke top-level Maven targets** and configure maven to build Jenkins as the first step.



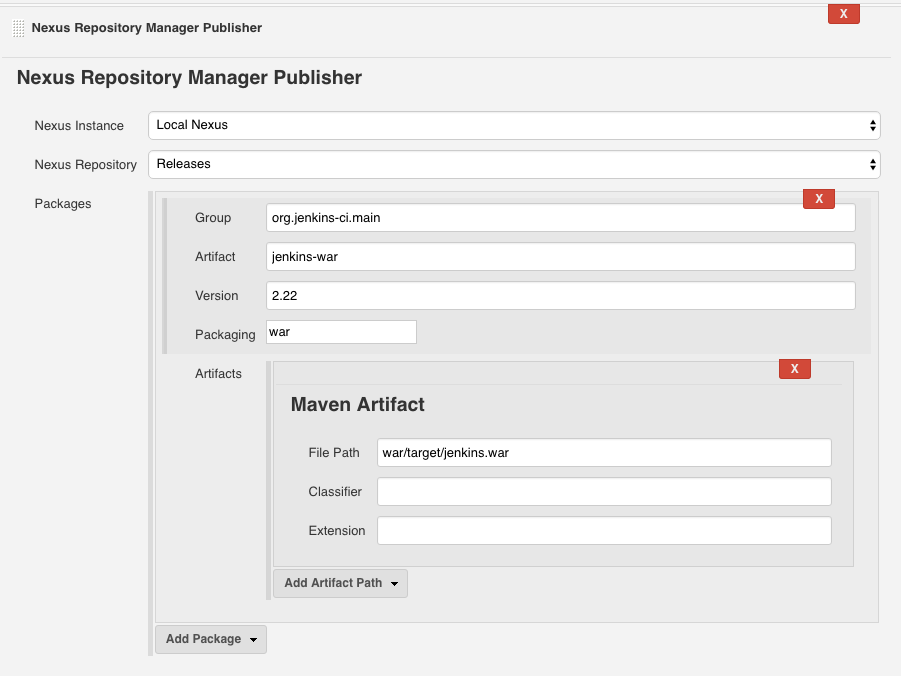
**Nexus Repository Manager Publisher Configuration**

Using the **Add build step** dropdown, select **Nexus Repository Manager Publisher** to add a step to publish your artifacts. This step should occur after the build so that the binaries are available for upload. Once adding the step, configure the Nexus Instance by selecting the Display Name configured in the first steps. Once the instance is selected, you should see a list of all repositories your authenticated user has access to in the **Nexus Repositories** dropdown. Select a repository which has Release Repository Policy and allows for Artifact Uploads.

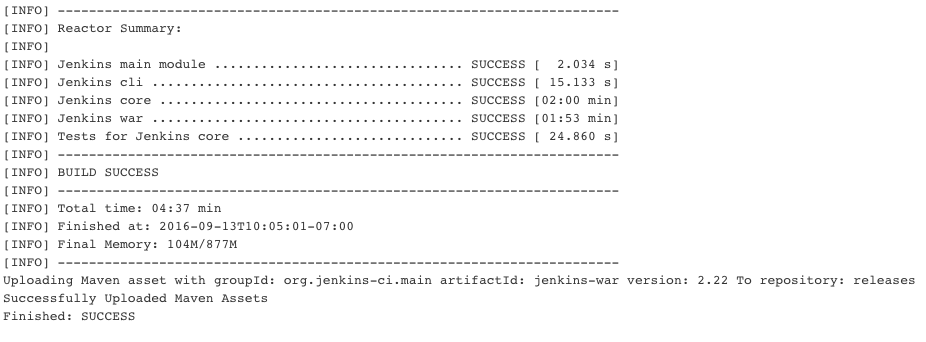


Once a repository is selected, you can select Packages to publish to Nexus Repository Manager during your Freestyle build. Packages are synonymous to Maven artifacts, a group of binaries (jar, -sources.jar) which are described by a component identifier (GAV).

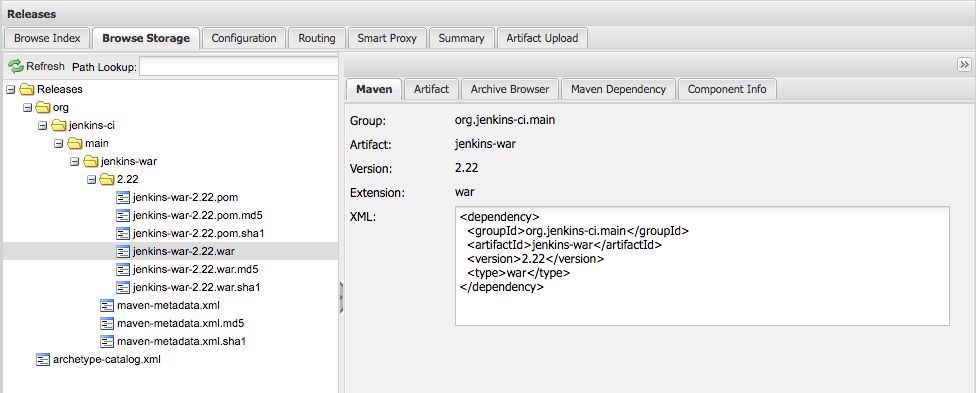
For this example, use the **Add Package** dropdown to select a **Maven Package**. Fill in the Group, Artifact, and Version fields. You will notice the Packaging field is freeform but will provide autocomplete for traditional Maven packages. In this example we will want to publish the Jenkins war and will select war for the package. Lastly, use the **Add Artifact Path** dropdown and select **Maven Artifact**. This allows you to select the artifacts to upload for this particular GAV. Jenkins builds the war to war/target/jenkins.war which we will use for the **File Path**.



Finally save the changes and launch a build for your project. Once the build has finished you should be presented with output similar to what's shown below.

****

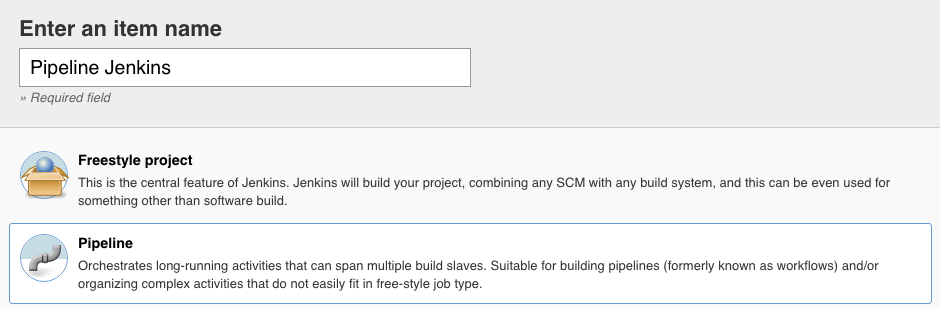
Returning to our Nexus instance, we can see the Jenkins war is now available in the Releases repository.



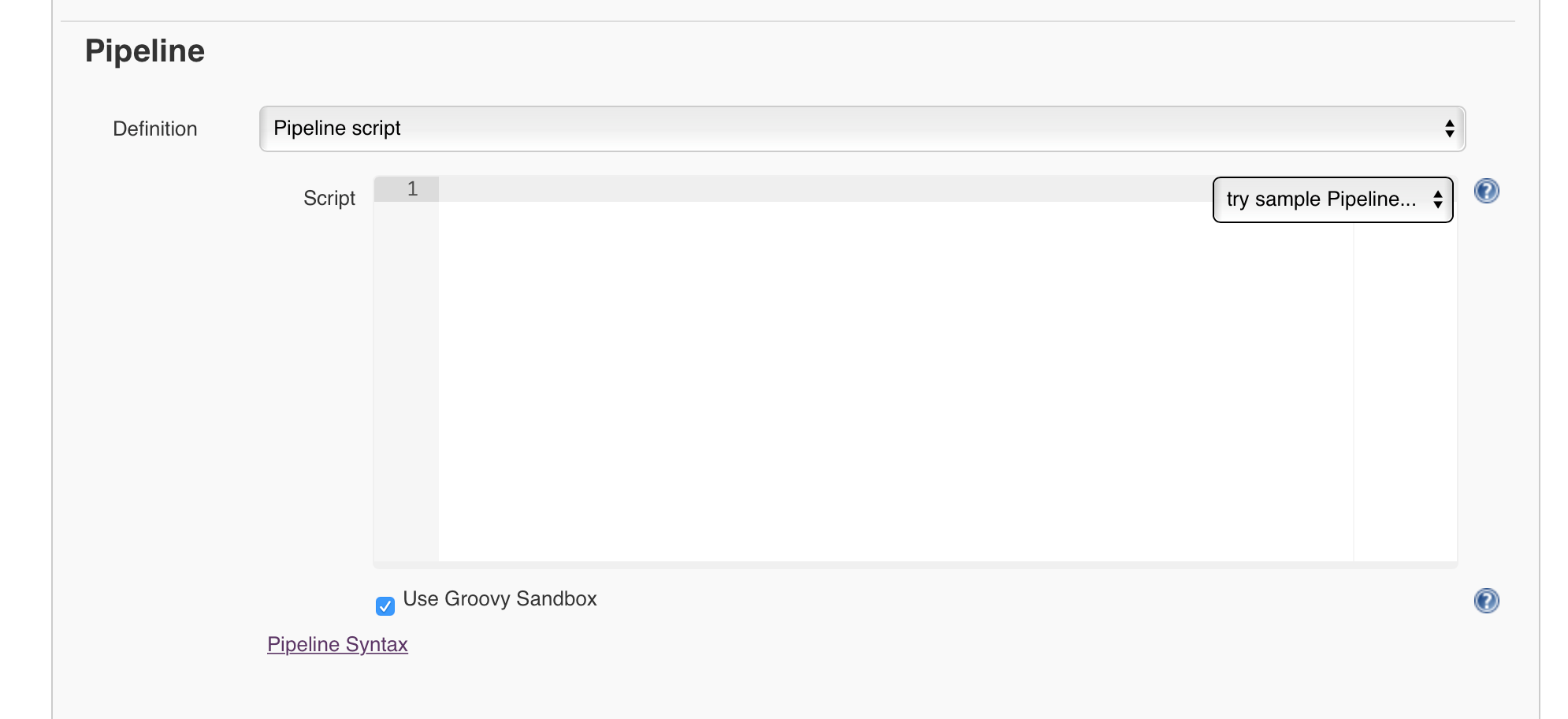
**Pipeline Build**

**Pipeline Build Step Configuration**

Jenkins Pipeline builds allow for precise control and configuration over your build process. In this example we will, again, build Jenkins within Jenkins and publish to our local Nexus Repository Manager instance. Some of the configuration steps will be repeated from the Freestyle build instructions, so feel free to skim over parts that you have already read. To start, create a **Pipeline** job by clicking on the **New Item** link on the Dashboard. Give the project a name and click **OK** to continue with the configuration.



Pipeline jobs allow you to configure the job via a Groovy script. The script is editable in the section with the **Pipeline** header. In this example we will use the sample **Github + Maven** script by selecting this option from the dropdown that reads **try sample Pipeline...**



Once the sample script is selected, the text area will be populated with the Preparation, Build and Results stages. Note that the Preparation stage makes reference to the M3 Maven tool which we setup in the **Tool Configuration (Optional)** step of this documentation. Below is the script with the Jenkins GitHub repository substituted for the sample repository.

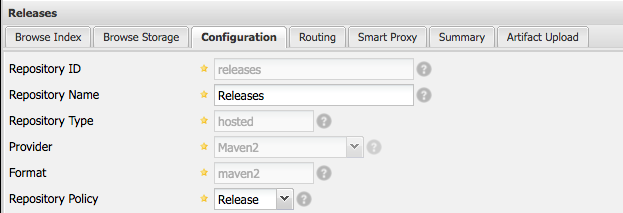
|  |
| --- |
| node {     def mvnHome     stage('Preparation') {        git '[https://github.com/jenkinsci/jenkins.git'](https://github.com/jenkinsci/jenkins.git%27)        mvnHome = tool 'M3'     }     stage('Build') {        if (isUnix()) {           sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"        } else {           bat(/"${mvnHome}\bin\mvn" -Dmaven.test.failure.ignore clean package/)        }     }     stage('Results') {        junit '\*\*/target/surefire-reports/TEST-\*.xml'        archive 'target/\*.jar'     }  } |

**Nexus Repository Manager Publisher Configuration**

Now, we will add a **Nexus Repository Manager Publisher** step to the build pipeline script. This step should occur after the build so that the binaries are available for upload. For best reporting during the build process, the publish step should exist in its own stage. Below the Results stage, add a new Publish stage to the script.

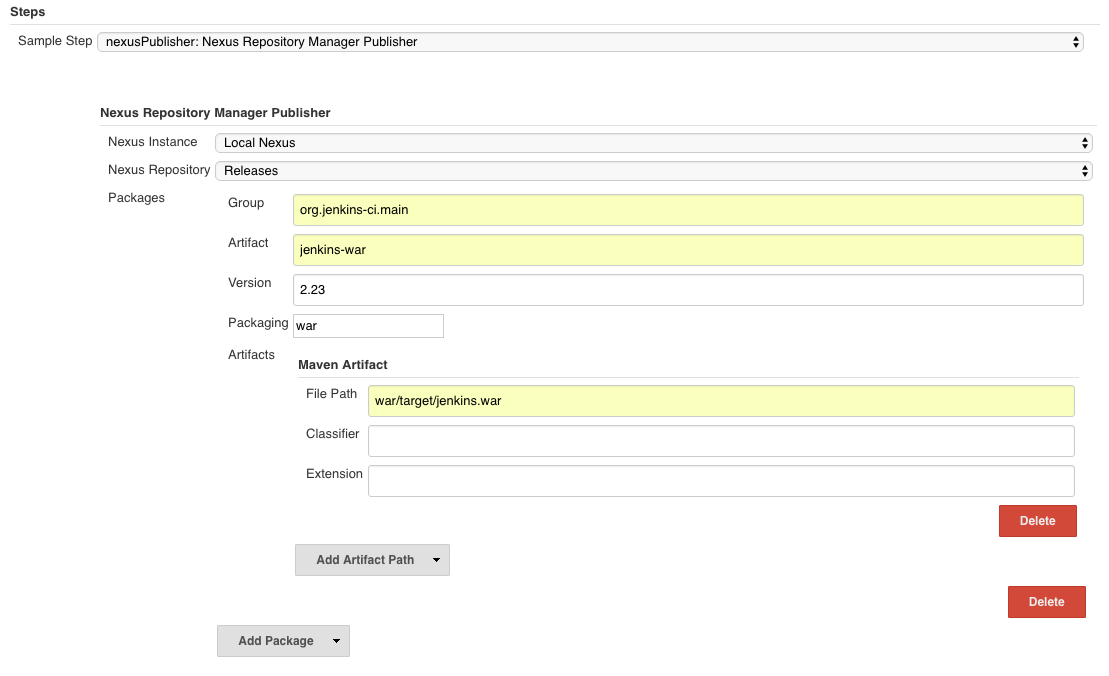
|  |
| --- |
| stage('Publish') {    } |

Jenkins Pipeline makes getting started with their scripting easy using the **Pipeline Syntax** wizard to generate the necessary Groovy code to publish your artifact. Click the **Pipeline Syntax** link in the **Pipeline** section to open the wizard in a new window. Under **Steps** open the **Sample Step** dropdown and find the **nexusPublisher: Nexus Repository Manager Publisher** step in the dropdown. Selecting this step will bring up a configuration section that will look familiar to those who completed the **Freestyle Build** tutorial in this documentation. We will repeat the process here, first configure the Nexus Instance by selecting the Display Name configured in the first steps. Once the instance is selected, you should see a list of all repositories your authenticated user has access to in the **Nexus Repositories**dropdown. Select a repository which has Release Repository Policy and allows for Artifact Uploads.



Once a repository is selected, you can select Packages to publish to Nexus Repository Manager during your Freestyle build. Packages are synonymous to Maven artifacts, a group of binaries (jar, -sources.jar) which are described by a component identifier (GAV).

For this example, use the **Add Package** dropdown to select a **Maven Package**. Fill in the Group, Artifact, and Version fields. You will notice the Packaging field is freeform but will provide autocomplete for traditional Maven packages. In this example we will want to publish the Jenkins war and will select war for the package. Lastly, use the **Add Artifact Path** dropdown and select **Maven Artifact**. This allows you to select the artifacts to upload for this particular GAV. Jenkins builds the war to war/target/jenkins.war which we will use for the **File Path**.



Clicking the **Generate Groovy** button below the step configuration will generate the configured pipeline script. Copy the generated script and return to the Pipeline job configuration.

**Nexus Publisher Script**

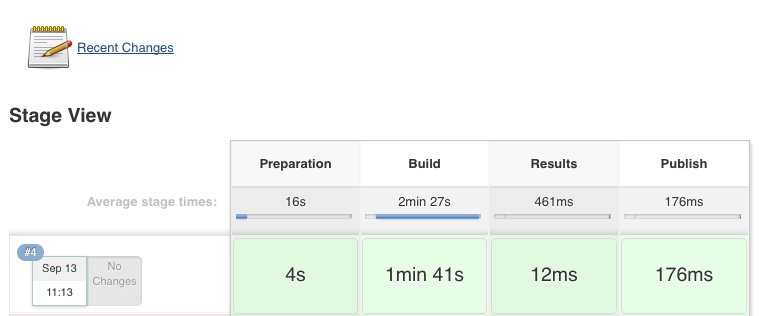
|  |
| --- |
| nexusPublisher nexusInstanceId: 'localNexus', nexusRepositoryId: 'releases', packages: [[$class: 'MavenPackage', mavenAssetList: [[classifier: '', extension: '', filePath: 'war/target/jenkins.war']], mavenCoordinate: [artifactId: 'jenkins-war', groupId: 'org.jenkins-ci.main', packaging: 'war', version: '2.23']]] |

After copying the script, paste it into the previously added Publish stage. Once completed, your entire script should look like the following.

**Example Pipeline Build Script**

|  |
| --- |
| node {     def mvnHome     stage('Preparation') {        git '[https://github.com/jenkinsci/jenkins.git'](https://github.com/jenkinsci/jenkins.git%27)        mvnHome = tool 'M3'     }     stage('Build') {        if (isUnix()) {           sh "'${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"        } else {           bat(/"${mvnHome}\bin\mvn" -Dmaven.test.failure.ignore clean package/)        }     }     stage('Results') {        junit '\*\*/target/surefire-reports/TEST-\*.xml'        archive 'target/\*.jar'     }     stage('Publish') {       nexusPublisher nexusInstanceId: 'localNexus', nexusRepositoryId: 'releases', packages: [[$class: 'MavenPackage', mavenAssetList: [[classifier: '', extension: '', filePath: 'war/target/jenkins.war']], mavenCoordinate: [artifactId: 'jenkins-war', groupId: 'org.jenkins-ci.main', packaging: 'war', version: '2.23']]]     }  } |

Save the configuration and run the build. Once completed, you should be presented with a successful build similar to what's shown below.



Returning to our Nexus instance, we can see the Jenkins war is now available in the Releases repository.

