**Chapter 13. Integrating Artifact Management**

Multiple stages of many pipelines rely on working with an artifact repository—both to publish versions of artifacts created in the pipeline and to retrieve specific versions for use in the pipeline. In this chapter, we’ll examine how to work with one of the most popular artifact managers, JFrog [Artifactory](https://jfrog.com/artifactory/). We’ll explore how to migrate functionality from an existing Freestyle project to a pipeline-as-code. We’ll also see how to do some other common tasks that require extra setup. Next, we’ll look at some challenges when trying to use the Artifactory integration with a Declarative Pipeline.

Finally, we’ll take a quick look at the pipeline steps for archiving artifacts and recording fingerprints (tracking information for which artifacts are associated with which builds).

First, though, for those who may not be familiar with Artifactory, we’ll take a quick look at why we use it and the value it can add.

# Publishing and Retrieving Artifacts

While the rationale for most of the technologies used in our example pipeline so far is obvious, that doesn’t always seem to be the case for leveraging an artifact repository. As such, before diving into how to integrate Jenkins 2 pipelines with Artifactory, it’s worth noting the benefits that warrant making the investment to use it in your pipelines.

Just as a well-structured pipeline should have facilities to manage source code, there should also be a facility to manage binary artifacts and other generated deliverables. Artifact management of this type is not always a given in pipelines, but in the cases where it is not initially included, its utility and the need for it are certain to become apparent quickly when the pipeline’s scope increases.

Some key technical and business drivers for using an artifact versioning and management tool include:

* Avoiding rebuilding from potentially unstable or changing source each time an artifact is needed
* Providing a versioned copy of an artifact (that has undergone some amount of testing), so that everyone knows what they are getting
* Having multiple versions stored and versioned to allow different consumers to use different versions (e.g., current, last release, etc.)
* Integrating with CI servers (such as Jenkins) so that, if a build is clean, the artifact can automatically be published into a repository (optionally with metadata about the build that generated it)
* Allowing virtual repositories that can aggregate multiple well-known or internal repositories, simplifying the search and ordering of artifacts

While there are multiple artifact repository management tools available, we will focus on Artifactory here, since it is one of the most commonly used. Artifactory provides both a Community Edition and a Pro Edition. Continuing the model from our example pipeline in [Chapter 10](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch10.html#CH_Conversions), we are targeting an open source pipeline at its most basic, so the focus here is on the free Community Edition of Artifactory. The Pro Edition may have additional functionality that allows for more easily accomplishing some tasks. However, most of what we do here should be easily transferable.

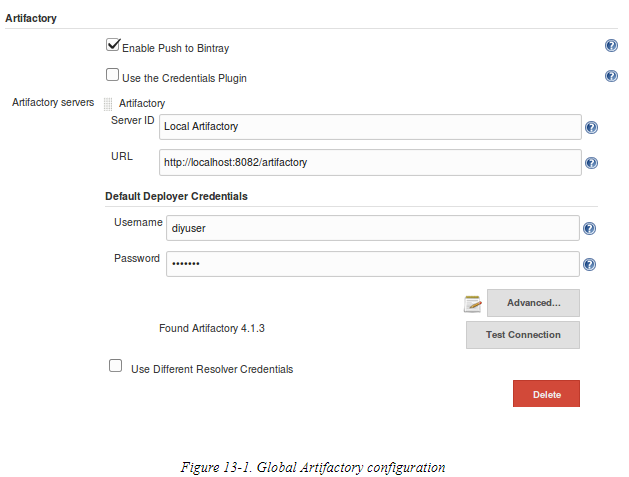
Now, let’s see how an Artifactory CE setup that has been integrated with traditional Jenkins can be integrated with Jenkins 2. We’ll start with the basic setup for using the tool.

# Setup and Global Configuration

As with any other application, we’ll need to have an instance of the application up and running with access for Jenkins. As well, we will want to have a recent version of the [Artifactory plugin](http://bit.ly/2J6w4NO) installed. Any reasonably modern version will have pipeline support built in.

We can search [the Plugin Compatibility with Pipeline page on GitHub](http://bit.ly/2qQ3gT5), we can search for Artifactory and see that, as of plugin version 2.5.0, we had step compatibility. So, as long as we have at least that version of the plugin installed, we should be able to use the basic functionality we need.

Next, we want to make sure that we have the global configuration for the Artifactory server done in Jenkins. This is done on the Configure System page. (If you have difficulty remembering whether to go to Configure System or Global Tool Configuration, think of “system” as being similar to “server”; so, configuring the Artifactory server would be done in the Configure System area.) [Figure 13-1](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch13.html#fig_global_artif_config) shows an example configuration.

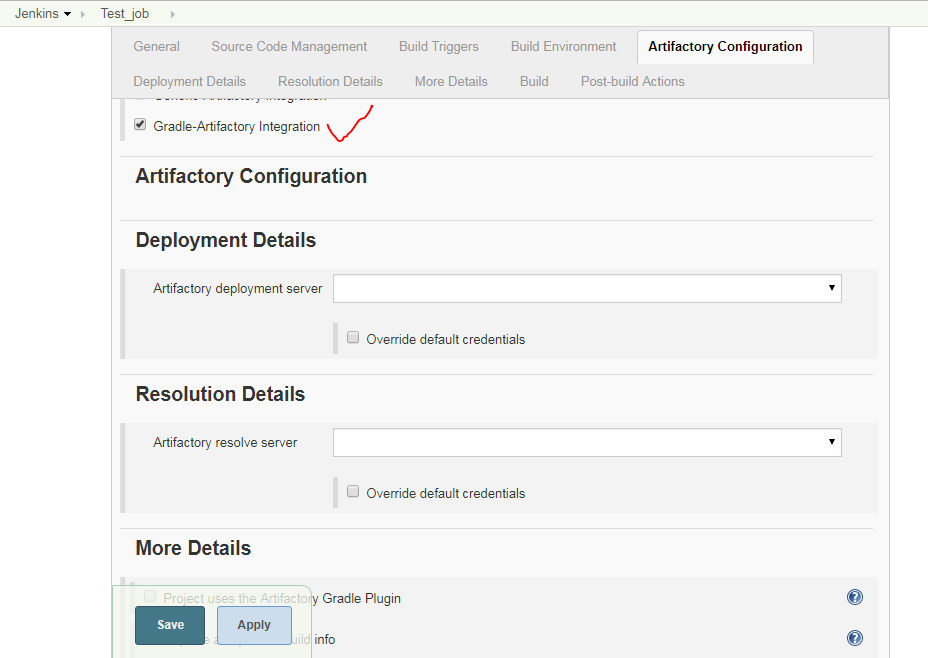


# Using Artifactory in a Scripted Pipeline

In the traditional Jenkins web model for using Artifactory, there were a number of areas to configure (and thus forms to fill in). Typically we would start with defining where in Artifactory to *publish*artifacts we produce (the “deployment server”) and where in Artifactory to *resolve* (look for) dependencies (the “resolution server”).

In the traditional web interface, we would configure those elements by selecting one of the options for Artifactory integration in the Build Environment section. There are several options to choose from, including Ivy, Maven, and Gradle. We’ll focus on a Gradle example, as that’s what’s used in our example pipeline, but this should translate fairly easily to other types of available integration.

Below Figure shows the Gradle-Artifactory Integration option selected. This, in turn, invokes the Artifactory Configuration section with the related forms to fill in for the deployment server and the resolution server.



To translate this to a pipeline environment, we need to define some values to point to the server and repositories that we want. Also, depending on the type of the integration, we have specific objects that represent the functionality of the combined Artifactory and build application. These “compound” objects then allow us to invoke Artifactory functionality with the build application via direct calls.

As an example, here are the related steps we can use to set up the Artifactory/Gradle integration in a typical pipeline.

First, we need to create an instance of an Artifactory object that points to the Artifactory server as we have it configured in Jenkins. This is intended to reference the global name we provided for the server in the configuration, similar to the tool DSL step that we have used for other applications. The basic form here is:

**def** server = Artifactory.server "<name>"

To match what we were using in the Freestyle job of our example pipeline where our server was configured as “Local Artifactory,” we would set this as follows:

**def** server = Artifactory.server "Local Artifactory"

Now, we can create an instance of an object that represents the predefined integration of Artifactory and the build application. At the same time, we can also point it to the installed version of the application. The basic form for this part is:

**def** artifactoryGradle = Artifactory.newGradleBuild()

artifactoryGradle.tool = "<Gradle tool name in Jenkins>"

Adapting from our traditional pipeline example yields this:

**def** artifactoryGradle = Artifactory.newGradleBuild()

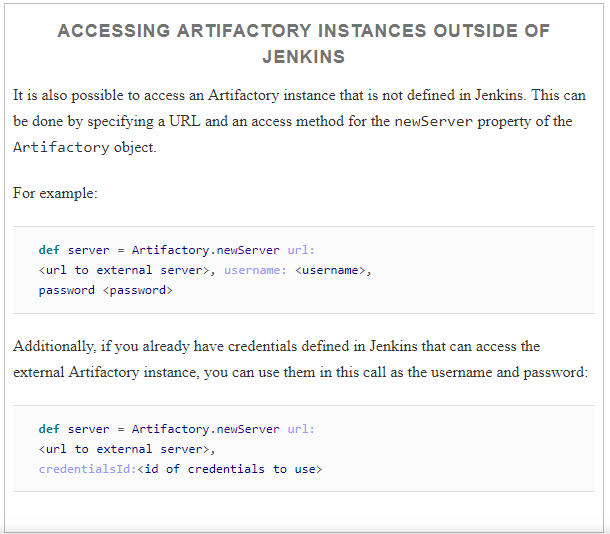
artifactoryGradle.tool = "gradle3"

At this point, we can instantiate (choose) our deployment repository and/or our resolver repository. The context here is fairly straightforward, so jumping to an implementation that matches our traditional pipeline looks like this:

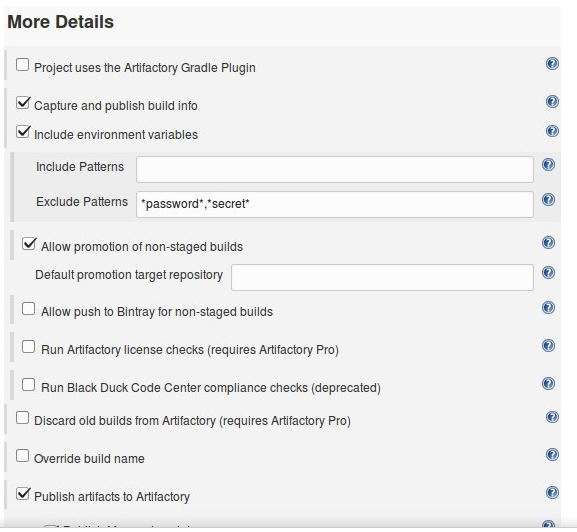
artifactoryGradle.deployer repo:'libs-snapshot-local', server:server

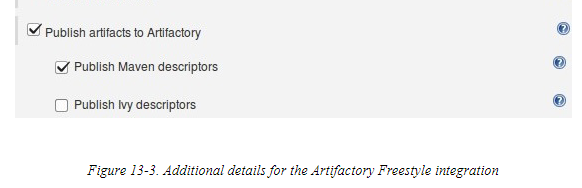
artifactoryGradle.resolver repo:'libs-release', server:server

The server:server reference here is actually a parameter and a value. The parameter name is server:, and the value we are passing in is the server instance object (from the def server...code) we defined earlier.



Beyond the basic configuration of the server and repositories, the traditional Jenkins interface includes a large number of options for Artifactory integration. [Figure 13-3](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch13.html#fig_add_dets_4artif_int) shows the first part of these options, as shown in the More Details section of the job.





If you are working with Artifactory, most of these will be well understood already. As such, we won’t go into detail on each one. We’ll just list a few examples, followed by the code that can be used to set them in a pipeline script. In the More Details section, we can:

* Tell Jenkins whether or not Gradle is already including the Artifactory plugin.

artifactoryGradle.usesPlugin = **true** | **false**

* Set options to capture build information:
  1. Define an instance variable to hold an Artifactory buildInfo object.
  2. Set the buildInfo environment capture switch to true.
  3. **def** buildInfo = Artifactory.newBuildInfo(),

buildInfo.env.capture = **true** | **false**

* Set deploy/publish options:
  1. Set a flag to indicate whether to deploy Maven descriptors.
  2. Define any patterns to be excluded from being deployed to Artifactory.
  3. artifactoryGradle.deployer.deployMavenDescriptors = **true** | **false**
  4. artifactoryGradle.deployer.artifactDeploymentPatterns.addExclude(
  5. "<file pattern>")

Once we have set the appropriate options, we can invoke the object to actually do the work, such as running the Gradle build and publishing the results. First, we invoke the artifactoryGradle object as we would for Gradle:

artifactoryGradle.run rootDir: "/", buildFile: 'build.gradle',

tasks: ...

Then we publish the build info:

server.publishBuildInfo buildInfo

A similar approach can be taken with other build tools and Artifactory, such as Maven. For example, instead of an artifactoryGradle object that is created by invoking newGradleBuild(), you might have a new artifactoryMaven object defined this way:

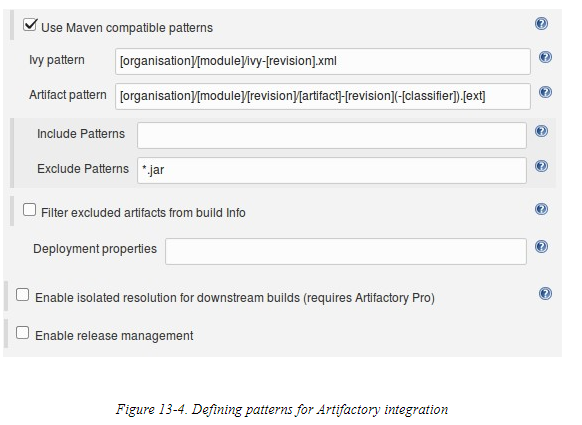
**def** artifactoryMaven = Artifactory.newMavenBuild()

From there, you could proceed to set options for the Artifactory/Maven integration based on the object just created. For example, to add a configuration item that both includes certain files and excludes others, you could do:

artifactoryMaven.deployer.artifactDeploymentPatterns.addInclude(

"<paths to include>").addExclude("<paths to exclude>")

This is similar to how we might define patterns in a Freestyle project, as shown in [Figure 13-4](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch13.html#fig_def_pats_artif_int).



You could also turn off deployment as follows:

artifactoryMaven.deployArtifacts = **false**

Next, we’ll look at a few examples of some other common tasks that require some extra setup.

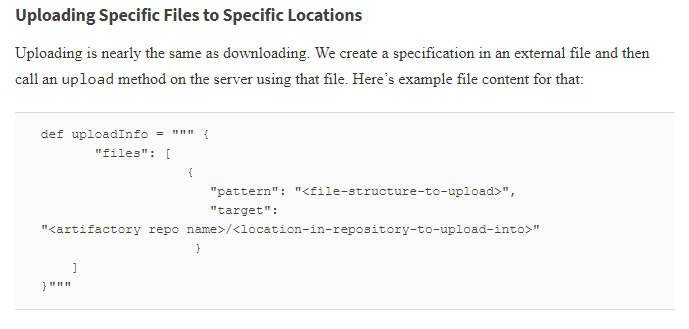
# Performing Other Tasks

Once you have the basic integration with Artifactory set up in your pipeline, there are likely to be other operations you need or want to do with it, such as uploading/downloading specific files, promoting builds, etc. In this section, we take a look at how you can accomplish some of those tasks.

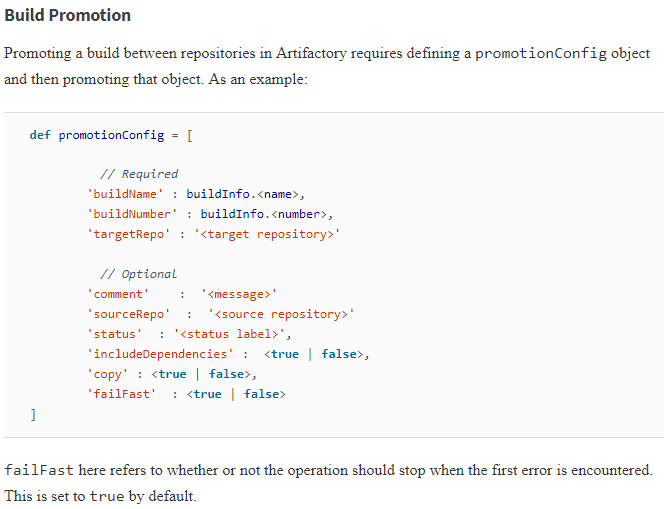
## Downloading Specific Files to Specific Locations

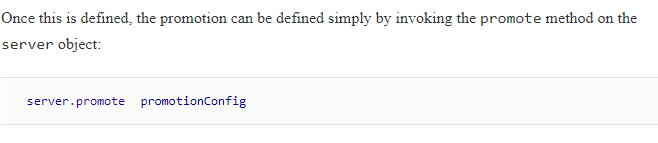
To download particular files, you create a specification in an external file. An example would be a JSON file that lists what files to download and where to put them when downloaded, like this:











**Integration with a Declarative Pipeline**

As outlined previously, Artifactory integration in a Jenkins pipeline currently depends on the ability to define instances of objects to point to the server, the integration object, etc. In a pipeline that is created using declarative syntax, such declarations are not allowed, and trying to use them directly in the pipeline will result in an error.

How, then, do we make use of the Artifactory integration in a Declarative Pipeline? There are several options, including:

* Placing code in a script block in the Declarative Pipeline
* Placing code outside of the larger pipeline block
* Creating a shared library to handle the Artifactory interactions

See [Chapter 7](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch07.html#CH_Declarative_Pipelines) for more details on the first two options. While doable, these have trade-offs, especially if you intend to try to manage your pipeline through the Blue Ocean interface. Details about developing shared libraries (in support of the last option) can be found in [Chapter 6](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch06.html#CH_Extending_Your_Pipeline).

One other note here: it is possible at some point in the future that JFrog or someone else will develop a plugin that provides more direct support for Artifactory integration with Declarative Pipelines. If the current situation presents a challenge for you, you may want to periodically check for newer versions of the plugin that might offer better direct support.

